Lecture Notes in Computer Science 4077

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison
Lancaster University, UK

Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler
University of Surrey, Guildford, UK

Jon M. Kleinberg
Cornell University, Ithaca, NY, USA

Friedemann Mattern
ETH Zurich, Switzerland

John C. Mitchell
Stanford University, CA, USA

Moni Naor
Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz
University of Bern, Switzerland

C. Pandu Rangan
Indian Institute of Technology, Madras, India

Bernhard Steffen
University of Dortmund, Germany

Madhu Sudan
Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos
University of California, Los Angeles, CA, USA

Doug Tygar
University of California, Berkeley, CA, USA

Moshe Y. Vardi
Rice University, Houston, TX, USA

Gerhard Weikum
Max-Planck Institute of Computer Science, Saarbruecken, Germany

Myung-Soo Kim Kenji Shimada (Eds.)

Geometric Modeling and Processing – GMP 2006

4th International Conference
Pittsburgh, PA, USA, July 26-28, 2006
Proceedings

Springer

Volume Editors

Myung-Soo Kim
Seoul National University, School of Computer Science and Engineering
Seoul 151-742, Korea
E-mail: mskim@snu.ac.kr

Kenji Shimada
Carnegie Mellon University, Mechanical Engineering
Pittsburgh, PA 15213, USA
E-mail: shimada@cmu.edu

Library of Congress Control Number: 2006929220

CR Subject Classification (1998): I.3.5, I.3.7, I.4.8, G.1.2, F.2.2, I.5, G.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-36711-X Springer Berlin Heidelberg New York
ISBN-13 978-3-540-36711-6 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2006
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11802914 06/3142 5 4 3 2 1 0

Preface

This book contains the proceedings of Geometric Modeling and Processing 2006, the fourth in a biennial international conference series on geometric modeling, simulation and computing, which was held July 26–28, 2006 in Pittsburgh, USA. The previous conferences were in Hong Kong (2000), Tokyo (2002), and Beijing (2004). The next conference (GMP 2008) will be held in China.

GMP 2006 received 84 paper submissions, covering various areas of geometric modeling and processing. Based on the recommendations of 114 reviewers, 36 regular papers were selected for conference presentation, and 21 short papers were accepted for poster presentation. The authors of these proceedings come from Austria, Belgium, Canada, Chile, China, Colombia, Greece, Indonesia, Israel, Japan, Korea, Lebanon, Singapore, the UK, and USA.

We are grateful to the authors who submitted to GMP 2006 and to the many dedicated reviewers. Their creativity and hard work substantially contributed to the technical program of the conference. We would also like to thank the members of the Program Committee for their strong support. We also wish to thank David Gossard, GMP2006 Conference Chair, and past GMP Program Co-chairs Shimin Hu, Ralph Martin, Helmut Pottmann, Hiromasa Suzuki, and Wenping Wang, whose current and previous work has helped to establish this conference as a major event in geometric modeling and processing. Further, we wish to thank the members of the Computer Integrated Engineering Laboratory at Carnegie Mellon University, in particular Soji Yamakawa, for their invaluable assistance throughout the conference preparation.

We gratefully acknowledge the financial support of Carnegie Mellon University. Finally, we wish to thank all conference participants for making GMP 2006 a success.

We hope that the readers will enjoy this book. In our view, it impressively demonstrates the rapid progress in geometric modeling and processing. It shows the importance and range of this field, with its impact in such areas as computer graphics, computer vision, machining, robotics, and scientific visualization. Finally, we hope that the conference and its proceedings will stimulate further exciting research.

Myung-Soo Kim
Kenji Shimada

Conference Committee

Conference Chair

David Gossard (Massachusetts Institute of Technology, USA)

Program Co-chairs

Myung-Soo Kim (Seoul National University, Korea)
Kenji Shimada (Carnegie Mellon University, USA)

Steering Committee

Shimin Hu (Tsinghua University, China)
Ralph Martin (Cardiff University, UK)
Helmut Pottmann (Institut für Geometrie, TU Wien, Austria)
Hiromasa Suzuki (University of Tokyo, Japan)
Wenping Wang (Hong Kong Univsersity, Hong Kong)

Program Committee

Chandrajit Bajaj (University of Texas at Austin, USA)
Hujun Bao (Zhejiang University, China)
Alexander Belyaev (Max-Planck-Institut für Informatik, Germany)
Wim Bronsvoort (Delft University of Technology, The Netherlands)
Stephen Cameron (Oxford University, UK)
Fuhua (Frank) Cheng (University of Kentucky, USA)
Falai Chen (University of Science and Technology, China)
Eng Wee Chionh (National University of Singapore, Singapore)
Jian-Song Deng (University of Science and Technology, China)
Gershon Elber (Technion, Israel)
Rida Farouki (University of California, Davis, USA)
Gerald Farin (Arizona State University, USA)
Anath Fischer (Technion, Israel)
Michael Floater (SINTEF Applied Mathematics, Norway)
Xiao-Shan Gao (Chinese Academy of Sciences, China)
Ron Goldman (Rice University, USA)
Craig Gotsman (Technion, Israel)
Xianfeng Gu (State University of New York, Stony Brook, USA)
Baining Guo (Microsoft Research Asia, China)
Satyandra K. Gupta (University of Maryland, USA)

Soonhung Han (KAIST, Korea)
Shimin Hu (Tsinghua University, China)
Christoph Hoffmann (Purdue University, USA)
Leo Joskowicz (The Hebrew University of Jerusalem, Israel)
Tao Ju (Washington University, St. Louis, USA)
Bert Jüttler (Johannes Kepler Universität Linz, Austria)
Satoshi Kanai (Hokkaido University, Japan)
Takashi Kanai (RIKEN, Japan)
Deok-Soo Kim (Hanyang University, Korea)
Tae-Wan Kim (Seoul National University, Korea)
Young J. Kim (Ewha Womans University, Korea)
Leif Kobbelt (RWTH Aachen, Germany)
Haeyoung Lee (Hongik University, Korea)
In-Kwon Lee (Yonsei University, Korea)
Seungyong Lee (POSTECH, Korea)
Ligang Liu (Zhejiang University, China)
Weiyin Ma (City University of Hong Kong, Hong Kong)
Takashi Maekawa (Yokohama National University, Japan)
Ralph Martin (Cardiff University, UK)
Hiroshi Masuda (University of Tokyo, Japan)
Kenjiro Miura (Shizuoka University, Japan)
Ahmad H. Nasri (American University of Beirut, Lebanon)
Ryutarou Ohbuchi (Yamanashi University, Japan)
Yutaka Ohtake (RIKEN, Japan)
Alexander Pasko (Hosei University, Japan)
Martin Peternell (Institut für Geometrie, TU Wien, Austria)
Helmut Pottmann (Institut für Geometrie, TU Wien, Austria)
Hartmut Prautzsch (Universitaet Karlsruhe, Germany)
Hong Qin (State University of New York, Stony Brook, USA)
Stephane Redon (INRIA Rhone-Alpes, France)
Maria Cecilia Rivara (Universidad de Chile, Chile)
Nicholas Sapidis (University of the Aegean, Greece)
Vadim Shapiro (University of Wisconsin-Madison, USA)
Hayong Shin (KAIST, Korea)
Yoshihisa Shinagawa (University of Illinois at Urbana-Champaign, USA)
Yohanes Stefanus (University of Indonesia)
Kokichi Sugihara (University of Tokyo, Japan)
Hiromasa Suzuki (University of Tokyo, Japan)
Chiew-Lan Tai (Hong Kong University of Science and Technology, Hong Kong)
Shigeo Takahashi (University of Tokyo, Japan)
Kai Tang (Hong Kong University of Science and Technology, Hong Kong)
Changhe Tu (Shandong University, China)
Tamas Várady (Geomagic Hungary, Hungary)
Johannes Wallner (Institut für Geometrie, TU Wien, Austria)
Charlie Wang (The Chinese University of Hong Kong)

Guojin Wang (Zhejiang University, China)
Jiaye Wang (Shandong University, China)
Michael Wang (The Chinese University of Hong Kong)
Wenping Wang (Hong Kong University, Hong Kong)
Joe Warren (Rice University, USA)
Soji Yamakawa (Carnegie Mellon University, USA)
Hong-Bin Zha (Peking University, China)
Kun Zhou (Microsoft Research Asia, China)

Additional Reviewers

Sigal Ar
Oscar Kin-Chung Au
Sergei Azernikov
Anna Vilanova i Bartroli
Silvia Biasotti
Jung-Woo Chang
Yoo-Joo Choi
Hongbo Fu
Iddo Hanniel
Yaron Holdstein
Martin Isenburg
David Johnson
Sujeong Kim
Ji-Yong Kwon
Shuhua Lai
Yu-Kun Lai
Torsten Langer
Jae Kyu Lee
Jieun Jade Lee
Zhouchen Lin
Yang Liu
Yong-jin Liu
Alex Miropolsky
Muthuganapathy Ramanathan
Malcolm Sabin
Oliver Schall
Guy Sela
Olga Sorkine
Raphael Straub
Han-Bing Yan
Yong-Liang Yang
Xu Yang
Min-Joon Yoo
Jong-Chul Yoon
Seung-Hyun Yoon
Weiwei Xu
Xinyu Zhang
Qian-Yi Zhou

Table of Contents

Shape Reconstruction

Automatic Extraction of Surface Structures in Digital Shape Reconstruction
Tamas Várady, Michael A. Facello, Zsolt Terék 1

Ensembles for Normal and Surface Reconstructions
Mincheol Yoon, Yunjin Lee, Seungyong Lee, Ioannis Ivrissimtzis, Hans-Peter Seidel 17

Adaptive Fourier-Based Surface Reconstruction
Oliver Schall, Alexander Belyaev, Hans-Peter Seidel 34

Curves and Surfaces I

Least–Squares Approximation by Pythagorean Hodograph Spline Curves Via an Evolution Process
Martin Aigner, Zbynek Šír, Bert Jüttler 45

Geometric Accuracy Analysis for Discrete Surface Approximation
Junfei Dai, Wei Luo, Shing-Tung Yau, Xianfeng David Gu 59

Quadric Surface Extraction by Variational Shape Approximation
Dong-Ming Yan, Yang Liu, Wenping Wang 73

Geometric Processing I

Tracking Point-Curve Critical Distances
Xianming Chen, Elaine Cohen, Richard F. Riesenfeld 87

Theoretically Based Robust Algorithms for Tracking Intersection Curves of Two Deforming Parametric Surfaces
Xianming Chen, Richard F. Riesenfeld, Elaine Cohen, James Damon 101

Subdivision Termination Criteria in Subdivision Multivariate Solvers
Iddo Hanniel, Gershon Elber 115

Towards Unsupervised Segmentation of Semi-rigid Low-Resolution Molecular Surfaces
Yusu Wang, Leonidas J. Guibas 129

Curves and Surfaces II

Piecewise Developable Surface Approximation of General NURBS Surfaces, with Global Error Bounds
Jacob Subag, Gershon Elber 143

Efficient Piecewise Linear Approximation of Bézier Curves with Improved Sharp Error Bound
Weiyin Ma, Renjiang Zhang 157

Approximate μ-Bases of Rational Curves and Surfaces
Liyong Shen, Falai Chen, Bert Jüttler, Jiansong Deng 175

Shape Deformation

Inverse Adaptation of Hex-dominant Mesh for Large Deformation Finite Element Analysis
Arbtip Dheeravongkit, Kenji Shimada 189

Preserving Form-Features in Interactive Mesh Deformation
Hiroshi Masuda, Yasuhiro Yoshioka, Yoshiyuki Furukawa 207

Surface Creation and Curve Deformations Between Two Complex Closed Spatial Spline Curves
Joel Daniels II, Elaine Cohen 221

Shape Description

Computing a Family of Skeletons of Volumetric Models for Shape Description
Tao Ju, Matthew L. Baker, Wah Chiu 235

Representing Topological Structures Using Cell-Chains
David E. Cardoze, Gary L. Miller, Todd Phillips 248

Constructing Regularity Feature Trees for Solid Models
M. Li, F.C. Langbein, R.R. Martin 267

Insight for Practical Subdivision Modeling with Discrete Gauss-Bonnet Theorem
Ergun Akleman, Jianer Chen 287

Shape Recognition

Shape-Based Retrieval of Articulated 3D Models Using Spectral Embedding
Varun Jain, Hao Zhang 299

Separated Medial Surface Extraction from CT Data of Machine Parts
Tomoyuki Fujimori, Yohei Kobayashi, Hiromasa Suzuki 313

Two-Dimensional Selections for Feature-Based Data Exchange
Ari Rappoport, Steven Spitz, Michal Etzion 325

Geometric Modeling

Geometric Modeling of Nano Structures with Periodic Surfaces
Yan Wang 343

Minimal Mean-Curvature-Variation Surfaces and Their Applications in Surface Modeling
Guoliang Xu, Qin Zhang 357

Parametric Design Method for Shapes with Aesthetic Free-Form Surfaces
Tetsuo Oya, Takenori Mikami, Takanobu Kaneko, Masatake Higashi 371

Curves and Surfaces III

Control Point Removal Algorithm for T-Spline Surfaces
Yimin Wang, Jianmin Zheng 385

Shape Representations with Blossoms and Buds
L. Yohanes Stefanus 397

Manifold T-Spline
Ying He, Kexiang Wang, Hongyu Wang, Xianfeng Gu, Hong Qin 409

Subdivision Surfaces

Composite $\sqrt{2}$ Subdivision Surfaces
Guiqing Li, Weiyin Ma 423

Tuned Ternary Quad Subdivision
Tianyun Ni, Ahmad H. Nasri 441

Geometric Processing II

Simultaneous Precise Solutions to the Visibility Problem of Sculptured Models
Joon-Kyung Seong, Gershon Elber, Elaine Cohen 451

Density-Controlled Sampling of Parametric Surfaces Using Adaptive Space-Filling Curves
J.A. Quinn, F.C. Langbein, R.R. Martin, G. Elber 465

Engineering Applications

Verification of Engineering Models Based on Bipartite Graph Matching for Inspection Applications
Fabricio Fishkel, Anath Fischer, Sigal Ar 485

A Step Towards Automated Design of Side Actions in Injection Molding of Complex Parts
Ashis Gopal Banerjee, Satyandra K. Gupta 500

Finding All Undercut-Free Parting Directions for Extrusions
Xiaorui Chen, Sara McMains 514

Short Papers

Robust Three-Dimensional Registration of Range Images Using a New Genetic Algorithm
John Willian Branch, Flavio Prieto, Pierre Boulanger 528

Geometrical Mesh Improvement Properties of Delaunay Terminal Edge Refinement
Bruce Simpson, Maria-Cecilia Rivara 536

Matrix Based Subdivision Depth Computation for Extra-Ordinary Catmull-Clark Subdivision Surface Patches
Gang Chen, Fuhua (Frank) Cheng 545

Hierarchically Partitioned Implicit Surfaces for Interpolating Large Point Set Models
David T. Chen, Bryan S. Morse, Bradley C. Lowekamp, Terry S. Yoo 553

A New Class of Non-stationary Interpolatory Subdivision Schemes Based on Exponential Polynomials
Yoo-Joo Choi, Yeon-Ju Lee, Jungho Yoon, Byung-Gook Lee, Young J. Kim 563

Detection of Closed Sharp Feature Lines in Point Clouds for Reverse Engineering Applications
Kris Demarsin, Denis Vanderstraeten, Tim Volodine, Dirk Roose 571

Feature Detection Using Curvature Maps and the Min-cut/Max-flow Algorithm
Timothy Gatzke, Cindy Grimm 578

Computation of Normals for Stationary Subdivision Surfaces
Hiroshi Kawaharada, Kokichi Sugihara 585

Voxelization of Free-Form Solids Represented by Catmull-Clark Subdivision Surfaces
Shuhua Lai, Fuhua (Frank) Cheng 595

Interactive Face-Replacements for Modeling Detailed Shapes
Eric Landreneau, Ergun Akleman, John Keyser 602

Straightest Paths on Meshes by Cutting Planes
Sungyeol Lee, Joonhee Han, Haeyoung Lee 609

3D Facial Image Recognition Using a Nose Volume and Curvature Based Eigenface
Yeunghak Lee, Ikdong Kim, Jaechang Shim, David Marshall 616

Surface Reconstruction for Efficient Colon Unfolding
Sukhyun Lim, Hye-Jin Lee, Byeong-Seok Shin 623

Spectral Sequencing Based on Graph Distance
Rong Liu, Hao Zhang, Oliver van Kaick 630

An Efficient Implementation of RBF-Based Progressive Point-Sampled Geometry
Yong-Jin Liu, Kai Tang, Joneja Ajay 637

Segmentation of Scanned Mesh into Analytic Surfaces Based on Robust Curvature Estimation and Region Growing
Tomohiro Mizoguchi, Hiroaki Date, Satoshi Kanai, Takeshi Kishinami 644

Finding Mold-Piece Regions Using Computer Graphics Hardware
Alok K. Priyadarshi, Satyandra K. Gupta 655

A Method for FEA-Based Design of Heterogeneous Objects
Ki-Hoon Shin, Jin-Koo Lee 663

Time-Varying Volume Geometry Compression with 4D Lifting Wavelet Transform
Yan Wang, Heba Hamza 670

A Surface Displaced from a Manifold
Seung-Hyun Yoon 677

Smoothing of Meshes and Point Clouds Using Weighted Geometry-Aware Bases
Tim Volodine, Denis Vanderstraeten, Dirk Roose 687

Author Index 695

Automatic Extraction of Surface Structures in Digital Shape Reconstruction

Tamas Várady[1], Michael A. Facello[2], and Zsolt Terék[1]

[1] Geomagic Hungary, Ltd., Budapest, Hungary
[2] Geomagic, Inc., Research Triangle Park, North Carolina, USA

Abstract. One of the most challenging goals in digital shape reconstruction is to create a high-quality surface model from measured data with a minimal amount of user assistance. We present techniques to automate this process and create a digital model that meets the requirements in mechanical engineering CAD/CAM/CAE. Such a CAD model is composed of a hierarchy of different types of surfaces, including primary surfaces, connecting features and vertex blends at their junctions, and obey a well-defined topological structure that we would like to reconstruct as faithfully as possible. First, combinatorially robust segmentation techniques, borrowed from Morse theory, are presented. This is followed by an algorithm to create a so-called feature skeleton, which is a curve network on the mesh that represents the region structure of the object. The final surface structure comprises the optimally located boundaries of edge blends and setback vertex blends, which are well aligned with the actual geometry of the object. This makes the surface structure sufficient for an accurate, CAD-like surface approximation including both quadrangular and trimmed surface representations. A few representative industrial objects reconstructed by Geomagic systems illustrate the efficiency and quality of the approach.

Keywords: digital shape reconstruction, segmentation, combinatorial Morse theory, curve tracing, vertex blends.

1 Introduction

Digital Shape Reconstruction (formerly reverse engineering) deals with converting physical objects into a computer representation. DSR is a particular chapter within a general discipline called Digital Shape Sampling and Processing (DSSP) that integrates all point cloud related computations emerging in various fields [Marks05, Geom06]. There are well-established techniques to create polygonal meshes from measured data, which need to be further converted to a representation suitable for CAD, CAM, and CAE. The biggest challenge is to automate this conversion process while producing a model that meets the requirements of downstream applications, including good structure and high quality surfaces.

In Computer Aided Design and – in particular – in mechanical engineering, the majority of objects are composed of (i) relatively large, primary surfaces

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 1–16, 2006.
© Springer-Verlag Berlin Heidelberg 2006

connected by (ii) highly-curved connecting features, such as edge blends or free-form steps, and (iii) vertex blends at their junctions. Segmentation is the process of partitioning the polygonal mesh into an accurate and consistent region structure, where each region is a *pre-image* of the final CAD model [VarMar02]. The quality of segmentation fundamentally determines the quality of the final surface model [VarFac05]; a faithful and geometrically well-aligned region structure is a necessary condition to accurately approximate regions by standard implicit and parametric surfaces according to the above surface hierarchy.

In the last few years several approaches have been reported to segment triangular meshes. One group of methods cover the shape by a collection of quadrangles in a consistent manner, that are created by tiling a strongly decimated triangular mesh, or using Voronoi diagrams [EckHop96, HecGar97, LeSSCD98]. Unfortunately, these surface models – due to their uniformity and four-sidedness – cannot properly reproduce standard CAD objects which require general topology and different types of surfaces. Another group of approaches puts the main emphasis on extracting connected regions whose triangles are likely to belong together based on their geometric characteristics. These approaches – including region growing [SapBes95, LeoJaS97], watershed methods [ManWhi99, RazBae03] – are strong in collecting matching geometric data, but they face difficulties in creating a full, consistent topological structure and representing smooth transitions between primary surfaces. A third group of approaches limits the class of bounding surfaces to simple surfaces only [FitEgF97, BenVar01]. This – of course – helps to automate the segmentation process, however, excludes a large set of objects where conventional prismatic parts are combined with complex free-form surfaces.

The majority of the segmentation methods use curvature estimations or other indicators to locally qualify the vertices of a triangular mesh using local point neighborhoods [CsaWal00, BenVar04, HuaMen01]. It is a hard problem to find an appropriate threshold to segment a given object. Difficulties are partly due to differences in dimensions, level of measurement noise and unevenly distributed triangulations. It is also hard to find a single, global threshold due to great variations in curvature – take for example the simultaneous detection of very small and large radius fillets.

In this paper we introduce a new segmentation approach which combines results from combinatorial Morse Theory with special geometric modeling algorithms that have been adapted for digital shape reconstruction. We would like to create CAD-like structures that reflect the original design intent and make high-quality surface approximation possible. Our main goals are the following:

- separate primary regions and highly curved transition regions
- create complete and consistent region structures without topological limitations
- avoid threshold setting, but offer alternative segmentations, if necessary
- provide an automatic procedure with no or minimal user assistance
- develop a computationally efficient and robust procedure, that can be used for large scanned data sets and objects with high complexity.

The paper is structured as follows. In Section 2 we present an overview of the overall process. In Sections 3, 4 and 5 we discuss three particular topics in more detail – Morse complex-based segmentation, curve tracing for building feature skeletons and using setback type vertex blends to complete face loops in the final surface model. Before the concluding remarks, the results of the algorithm are illustrated using an industrial part in Section 6.

2 Overview of the Process

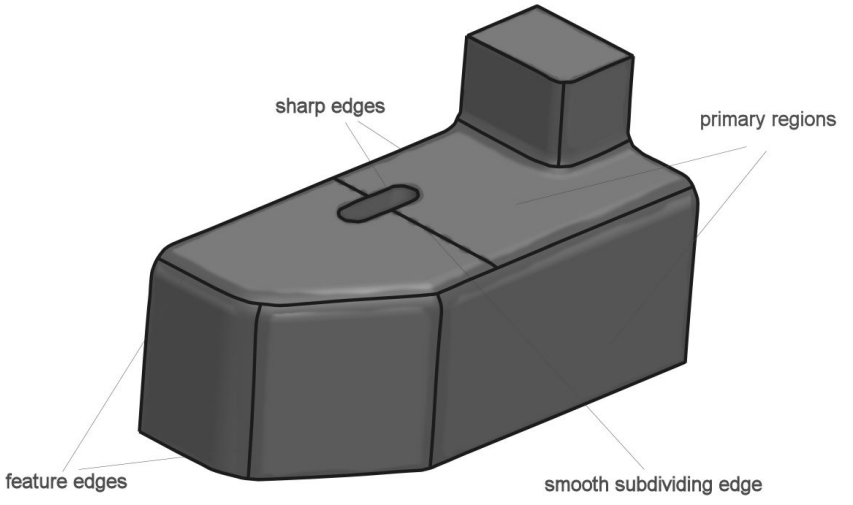
In this section we summarize our shape reconstruction algorithm. A simple schematic part was chosen to illustrate the consecutive phases, as shown in Figures 1(a) and 1(b). The input is a triangulated mesh that was created previously by approximating the measured data points. There are five phases:

1. Hierarchical Morse Complex Segmentation
2. Feature Skeleton Construction
3. Computing Region Boundaries
4. Surface Structure Creation
5. Surface Fitting

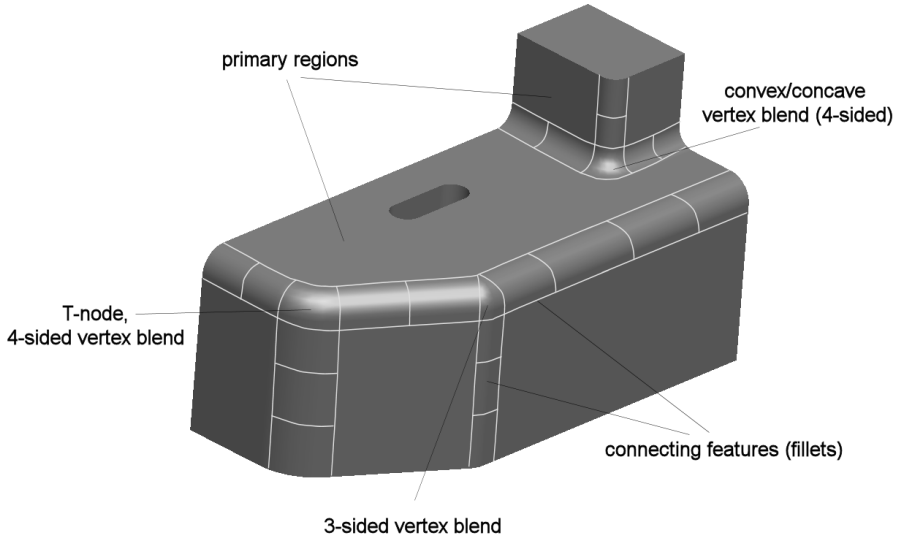
Phase 1: Hierarchical Morse Complex Segmentation. Morse theory studies smooth functions over manifolds [Miln63]. A Morse complex partitions the manifold into a collection of regions by a network of curves that connect the non-degenerate, critical points (minima, maxima, and saddles) of a given function. In our context, we use curvature indicator values estimated at each vertex of the polygonal mesh, and create a piecewise linear function to highlight the highly curved parts of the shape. Utilizing concepts of persistence and prioritization [EdeLeZ02], a hierarchy of topologically simplified segmentations is obtained. Each segmentation consists of monotonic regions, whose boundaries form a combinatorially correct curve network. These curves are composed of polylines of connected triangle edges and are typically ragged. We also perform an additional mesh operation to *thicken* these boundaries on the mesh. As a result, strips of triangles – called *separator sets* – are created; simultaneously, the original monotonic regions shrink (see red triangles in Figure 1(a)). Thus, the result of Phase 1 is a set of relatively flat regions having clearly separated by triangles of highly curved transitions. Further details follow in Section 3.

Phase 2: Feature Skeleton Construction. In this phase we construct an intermediate data structure called a *feature skeleton*, which is a smooth curve network of edges running in the middle of separator sets. Special curve tracing algorithms are applied using an estimated translational vector field (Section 4). As shown in Figure 1(a), a separator set may indicate (i) a smooth connecting feature between two adjacent regions, such as a fillet, (ii) a sharp feature, where there is tangential discontinuity between two surfaces, or (iii) a smooth subdividing curve that is defined, or computed, to cut large regions into smaller ones.

The above three types of edges are classified in advance, since they require different treatments. *Feature edges* are only temporary entities and will later



(a)



(b)

Fig. 1. (a) Separator sets and feature skeleton. (b) Features, vertex blends and the final surface structure.

be replaced by a pair of boundary curves. *Sharp edges* are precisely extracted from the polygon model, and become boundaries in the final surface structure. Smooth *subdividing curves* lie in the interior of regions where there is no curva-

ture variation; and similarly to sharp edges, they are smoothed and preserved. We also classify the vertices of the skeleton. In particular, we identify *collinear* edge pairs that approach a vertex from opposite directions; as an example take T-nodes that are degree-3 vertices with one pair of collinear edges.

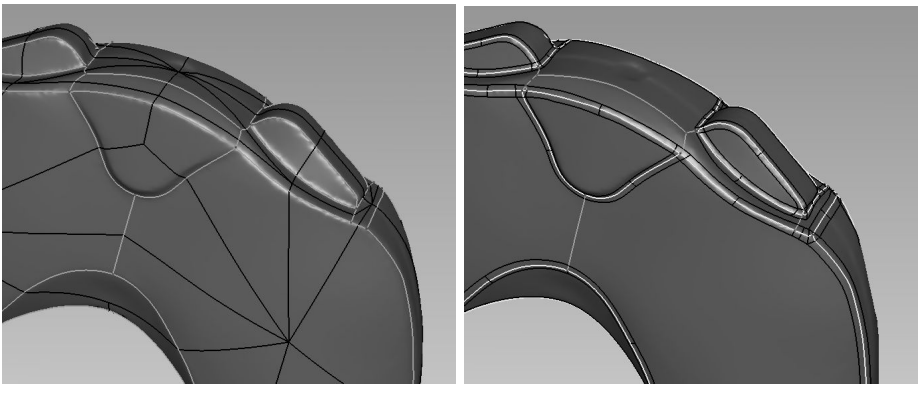
Phase 3: Computing Region Boundaries. In this phase we create boundary edges for the individual connecting features and vertex blends, see Figure 1(b). First we replace each feature edge of the skeleton by a pair of *longitudinal boundaries*, then replace each vertex of the skeleton by a loop of a *setback vertex blend* [Braid97, VarHof98]. As will be discussed in Section 4, the vertex loop may consist of an alternating sequence of *profile* curves and *spring* curves. Profile curves terminate the corresponding connecting features, spring curves connect pairs of adjacent feature boundaries lying on the same primary region. Spring curves must be inserted due to various reasons – see the T-node, or the convex-concave vertex blend in Figure 1(b); they may also degenerate with zero length.

Phase 4: Surface Structure Creation. The *thickened feature skeleton* comprises the previously determined boundaries of the connecting features and the loops of the vertex blends that also determine the loop structure of the primary regions, as shown in Figure 1(b). The connecting features are generally four-sided lying between two primaries and terminated by two vertex blends. The vertex blends terminate the features, but they may also share spring curves with adjacent primary regions. For a given primary region, the thickened edges of the corresponding loop may contain self-intersections, which must be removed for a valid structure. This step may modify the topology of the original thickened feature skeleton, as will be discussed in Section 5.

Phase 5: Surface Fitting. The above algorithm leads to a geometrically well-aligned structure whose existence is necessary for high-quality surface fitting. By construction the relatively large primary regions are empty of highly curved features, and the feature regions are free from possible artifacts coming from the primaries. There are two alternative techniques to approximate this surface structure. (i) A collection of smoothly connected quadrangular tiles can be used for rapid surfacing, or (ii) trimmed surfaces and special features can be fitted to obtain conventional CAD models. The latter yields much better surface quality, but requires more extensive computations. Alternative concepts of surface fitting techniques have been analyzed in [VarFac05], however, in this paper our only interest is the extraction of surface structures.

3 Hierarchical Morse Complex Segmentation

Our segmentation utilizes results from Combinatorial Morse Theory, which analyzes functions defined over manifolds [EdeHaZ03]. These manifolds are represented as polygonal meshes, produced from point clouds. In our context a function approximates an unknown, smooth function in a piecewise linear form; often the term *indicator function* will be used. In the following paragraphs we introduce basic concepts from Morse Theory in a nutshell, and simplification



(a) (b)

Fig. 2. (a) Morse Complex segmentation and related quadrilaterals. (b) Feature-based segmentation using the Morse regions.

algorithms applied to Morse complexes that eventually lead to the feature-based region structures.

Morse Theory. Classical Morse theory studies critical points of generic, smooth functions on manifolds [Miln63]. Let M be a 2-manifold and $f : M \rightarrow R$ a smooth function. At a point $x \in M$, assume a local orthonormal coordinate system; the *gradient* $\nabla f(x)$ is the vector of the local partial derivatives. The point x is *critical* if its gradient is zero, and *regular* otherwise. A critical point is *non-degenerate* if its *Hessian* is nonsingular, which is a property independent of the coordinate system. There are three types of non-degenerate critical points: minima, saddles, and maxima. It can be shown that the non-degenerate critical points are isolated. Technically f is considered a Morse function if (i) all critical points are non-degenerate; and (ii) the critical points have pair-wise different heights.

The gradient forms a smooth vector field on M. An *integral line* is a curve traced on the surface whose tangent coincides with the local gradient of M for all of its points. It always starts at a critical point and ends at another critical point without containing the endpoints. Because f is smooth, two integral lines are either disjoint or identical. A *descending manifold* $D(x)$ of a critical point x is the set of points that flow toward x, i.e. the point x and all points of the integral lines ending at x. For a minimum, $D(x)$ is identical to point x; for a saddle, $D(x)$ contains two connecting integral lines; for a maximum, $D(x)$ is an open disk. The Morse Complex is the collection of these disjoint manifolds: the boundary curves connect the saddles and the minima, and thus form a curve network that separates all simple monotonic regions of the surface, each assigned to a maximum.

Piecewise linear functions. The main effort in combinatorial topology is to turn the mathematical ideas of Morse theory into algorithms that operate on piecewise linear functions. Our assumption is that we have a triangulation K homeomor-

phic to a 2-manifold M. Our indicator function f is explicitly defined at the vertices and linearly interpolated over the edges and triangles of K.

For a good, global segmentation, we assume that our indicator function distinguishes highly curved and relatively flat parts. The majority of practical indicator functions are related – directly or indirectly – to surface curvature or its reciprocal value, including planarity, mean or Gaussian curvature, dimensionality, slippage, and others, which are estimated using a point neighborhood around a given vertex; for details see [CsaWal00, BenVar04, GelGui04]. Our experience is that the choice of the indicator function is not so crucial, and most of them provide reasonable segmentation when Morse theory is applied. The simplest indicator is planarity, defined by the error term of a local best-fit-plane; alternatively, curvature estimations based on local, low-degree implicit functions were also found reasonably stable and computationally acceptable. Locally estimated mean curvature was used for the examples of this paper; see Figure 8(c).

Constructing and simplifying a Morse complex. Our goal is to construct the descending manifolds of the local maxima, and create "rivers" which separate the regions and flow through the local minima. The details of the algorithm can be found in [EdeLeZ02, EdeHaZ03, AEHW04]; here we present a simplified version. The vertices are classified as regular or critical by taking the local star of triangles, and computing the function values at the related vertices. Starting at maximum points and taking the ordered vertices by function values we can merge triangles into descending 2-manifolds. The algorithm creates open disks, and it can be proven that the topology of the regions is the same as for the smooth case.

Due to the nature of piecewise linear functions and numerical estimations, the initial Morse complex is likely to consist of too many regions, which needs to be simplified. A possible procedure, which guarantees that the regions always remain simple, merges adjacent regions by removing a pair of critical points from the structure on M. Either a *saddle with a maximum* is cancelled by erasing two curves from M, or a *saddle with a minimum* by contracting curves with degree-1 endpoints. The sequence of cancellations creates a hierarchy of progressively coarser segmentations. Although there are many possible strategies the mathematically most elegant method to prioritize cancellations is based on the idea of persistence, as introduced in [EdeLeZ02]. Here the minima and maxima are paired with saddles, and the persistence of a pair is the absolute difference in function values between the two critical points.

Figure 2(a) shows a Morse Complex segmentation after simplification, the indicator function takes its maxima at the flat parts of the mesh. As can be observed, each monotonic region is bounded by a loop of orange arcs, alternating between minimum and saddle points; there are also black arcs that connect the maximum points to the saddles. In other words – at a saddle always four arcs meet connecting two minima and two maxima. The orange loop represents the final boundary of the Morse regions and serves as the basis of our further computations. This loop will be smoothed, aligned, thickened and transformed into a feature-based region structure (Figure 2(b)), as will be described in the next sections.

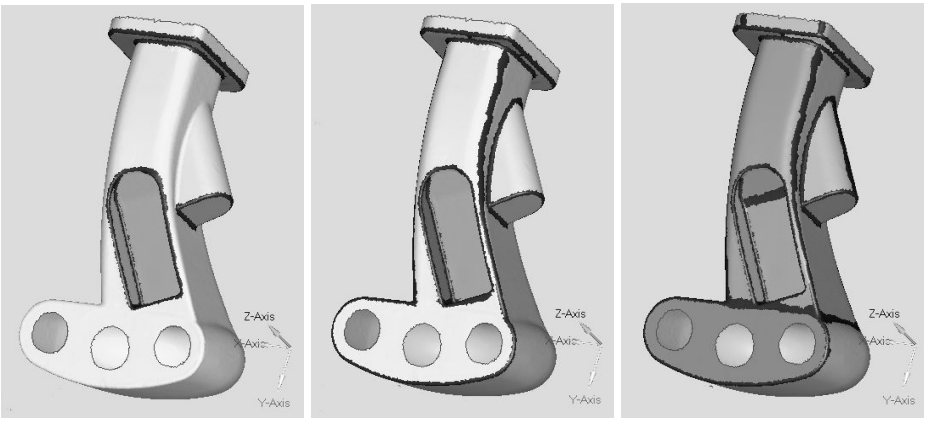


Fig. 3. Morse complex segmentation and separator sets; setting low, medium or high sensitivity defines different surface structures

Separator sets. Morse segmentation provides a clear topological structure; however, it ignores the geometry of the "flat" regions and the highly curved transitions. Based on a local threshold we can identify the triangles that belong to the connecting features. The zigzagged polylines will be thickened and we obtain triangle strips that likely belong to transitional regions, see red triangles in Figure 3. At the same time, the original Morse regions shrink and the remaining triangles now represent the primary regions in a "feature-free" form.

The three pictures in Figure 3 illustrate the previously mentioned concept of hierarchical segmentation. Based on different sensitivity values three different region structures have been created; less sensitive segmentations can always be embedded into more sensitive ones.

4 Feature Based Curve Extraction

In the previous phase, a topologically valid and consistent structure has been created, where the triangles of the mesh have been labeled to belong to one of the primary regions or the separator set. In the second and third phases we focus on creating a topologically identical, but geometrically correct curve network. The ragged region boundaries will be replaced by smoothed polylines crossing through the triangles. The feature skeleton consists of *mid-curves* running in the middle of the feature surfaces, while the thickened feature skeleton is composed of pairs of *feature boundary curves.*

Translational vector field. An important indicator that can be assigned to the vertices of the mesh is the *translational vector*, which characterizes the strength of the local extrusion in a given point neighborhood. Take vertices P_i that surround a given vertex P, and estimate the related normal vectors N_i. Mapping these normals to the Gaussian sphere, fit a plane which goes through the origin and approximates endpoints N_i. The normal vector of the fitted plane defines the

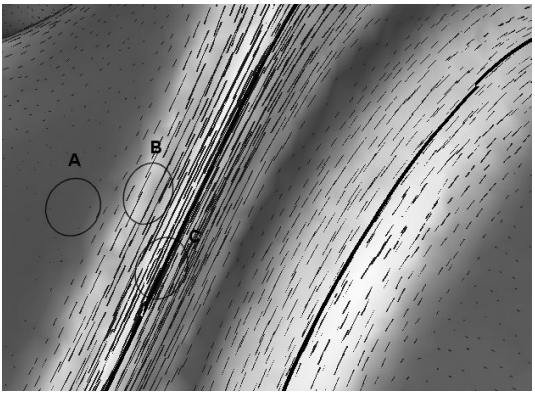


Fig. 4. Three points with different translational and similarity indicators

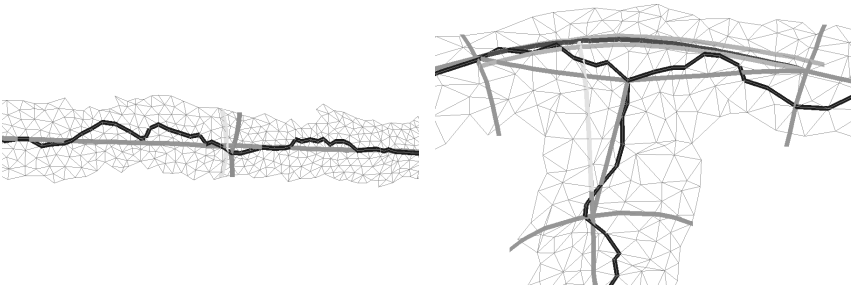
direction of the translation; the error of the least-squares fit characterizes its strength. Clearly, for the points of the connecting features translation will be strong (see Figure 4), while for the points of primary regions and vertex blends it will become weak or vanish.

Similarity filters. For the computation of mid-curves and feature boundaries another indicator called *similarity* proved to be useful, see [BenVar04]. Assume that f denotes an indicator function. Let us take $f(P)$ and n indicator values in the neighborhood, and measure the sum of differences expressed as

$$s(P) = \frac{1}{n\,|f(P)|} \sum_i |f(P) - f(P_i)| \quad . \tag{1}$$

If the indicator values are similar to that of the central point, $s(P)$ will be very close to zero. In Figure 4 three points are marked. At points A and C the similarity will be strong, being in the middle of a primary or a feature region. At point A, translation is weak, while at point C strong translation is indicated. The strongest similarity value will indicate the most likely location of the mid-curve. At point B, similarity vanishes and the translational strength is divided by the two halves of the neighborhood. These criteria help to determine likely locations of feature boundaries.

Curve tracing. In computer aided geometric design there are well-established techniques to trace curves including the computation of intersection curves or boundaries of various features, such as edge blends. In these cases, the surfaces and the procedures are fully defined, and it is possible to trace exact points on the curve using derivatives of the surfaces. In digital shape reconstruction the situation is different since the surfaces have not yet been created. Fortunately, the underlying mesh helps, and using the local indicators, we can compute the related feature curves.



(a) (b)

Fig. 5. (a) Curve tracing in the middle of a feature. (b) Guided curve tracing within a vertex blend area.

It is worth distinguishing three types of curve tracing.

1. *Feature tracing* is applied when the translational vector field is strong. The translational vectors are naturally defined in the interior of triangles as well by weighting the three related vectors at the vertices. Tracing trajectories of the vector field is analogous to solving an ordinary differential equation on the mesh using a Runge–Kutta-like numerical integration. An example is shown in Figure 5(a), where feature tracing started at the middle point of a separator set. This point was determined longitudinally by halving the arc-length of the related polyline. To locate its cross-sectional position we searched for the extreme value of the similarity indicator; see Point C in Figure 4. After defining the middle point we move towards the two vertex blends on the separator set. The translational vectors become weaker and feature tracing is terminated at the estimated setback position of the vertex blends; this will be explained in the next section. Figure 5(a) shows the original polyline (blue) coming from the Morse segmentation, the estimated cross section (yellow) at the middle, and the traced mid-curve (green). Note that a mid-curve – by construction – always remains within the separator set. It is terminated when it (i) leaves the separator set, (ii) gets closed into itself, or (iii) reaches the boundary of the mesh.
2. *Guided tracing* is used when the translational vector field becomes weak or "ill-defined", but we have a rough, guiding polyline and constraints to satisfy at the endpoints. This situation occurs along smooth subdividing edges that may partition a large smooth region, or in vertex areas where the direction of the translational vector field cannot be robustly estimated. Tracing is performed in a step-wise manner extending the current tracing direction, enforcing a smoothing term and satisfying constraints from the given polyline. As an example, take Figure 5(b), where feature tracing stopped at the setbacks of the vertex blend. The initial tangents of the connecting curves are thus given, and the curves (in green) must smoothly run into the vertex where the three blue polylines meet. In fact, in a later phase of feature skeleton construction, the vertex positions of the original polylines are also relocated.

3. For many parts "vanishing" features also occur, where the strength of the translational field gradually disappears. For example, in many car body panels there are several features which gradually and smoothly flow into the larger primary surfaces. In these cases the so-called *balanced tracing* is used, which combines the benefits of feature and guided tracing by taking an affine combination of the translational direction and a direction estimated at a point of the guiding polygon.

5 Vertex Blends with Setback

In the previous section we dealt with the generation of longitudinal boundary curves of connecting features that are necessary for the surface structure. In this section we focus on the junctions where they run together or interfere with each other. Generating vertex blends is a complex issue from both the topological and geometrical points of view. This topic was in the focus of geometric modeling research a decade ago, but now we revisit and adopt these techniques in digital shape reconstruction.

Setback type vertex blends. The well-known "suitcase corner" connects three edge blends with the same radii, see Figure 6(a). This vertex blend is a 3-sided patch, which can be represented by an octant of a sphere. The naive approach to create a vertex blend is to intersect the boundaries (trimlines) of two edge blends meeting on the same primary face and use these points as corner points for this 3-sided patch. The general situation, however, is much more complicated and may require forming complex blends where an arbitrary number of edges meet. These edges can be locally convex or concave, the angles between them are not necessarily close to 90 degrees, and the edge blends may vary from high to low cross-sectional curvature. We need to handle tangential and cuspate edges, as well, and allow keeping an edge sharp without replacing it by a blend.

To deal with these complex cases, the concept of *setbacks* was introduced by [Braid97, VarHof98], in which the boundaries of the edge blends are terminated before they reach the intersection points and a larger surface piece is inserted as shown in Figure 6(b). Setbacks help to avoid difficult shape configurations: compare Figures 6(c) and (d) with the aesthetically pleasing setback vertex blends in Figures 6(e) and (f).

A setback type vertex blend has maximum $2n$-sides, where in the most general case n spring curves and n profile curves alternate. Once again, profile curves terminate the edge blends and spring curves connect two corner points lying on the same primary surfaces. Depending on the geometric configuration, the length of any of these curves can be chosen to be zero, and the resulting vertex blend may be treated as a degenerate form of the $2n$-sided blend [VarHof98]. As illustrated in Figure 6(g), a single spring curve and two zero-length spring curves can make this vertex blend four-sided. Figure 6(h) shows another example where a profile curve has zero length, since the corresponding edge is sharp and is not replaced by a blend.

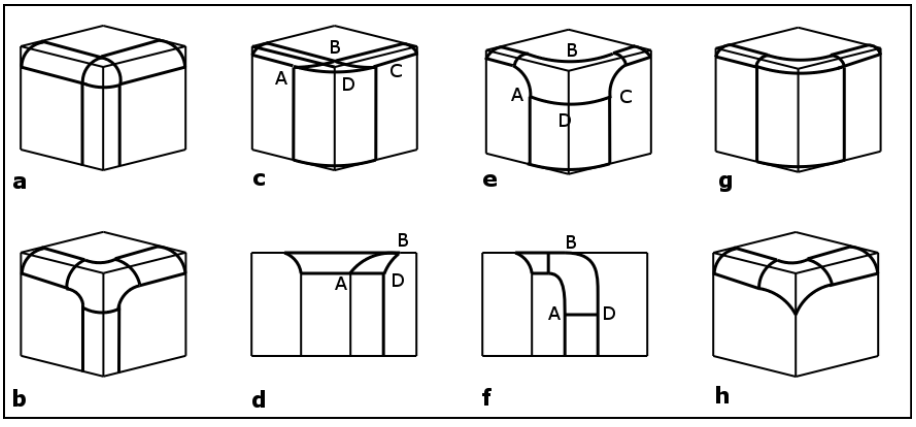


Fig. 6. Different vertex blend configurations

Setbacks. The setback concept is heavily used when we intend to determine the best configuration of unknown vertex blends. Let sb_i denote the distance between the original unblended vertex and the cross-sectional termination of an edge blend, and r_{i-1} and r_{i+1} denote the range constraints computed from the widths of the previous and next blended edges. To push setbacks further off either for aesthetic reasons or handling degenerate situations we introduce a correction term s_i. Compare Figure 6(g) and 6(e): in the latter case s_i has a positive value, and a six-sided vertex blend is created instead of a four-sided one. The final setback value can be expressed by

$$sb_i = s_i + \max(r_{i-1}, r_{i+1}) \ . \tag{2}$$

As explained earlier, in the feature skeleton building phase we determine only approximate values for the setbacks by detecting that the translational strength falls under a certain level; see Figure 5(b). The exact setback values are computed after tracing the exact feature boundary curves, which determine the above range values.

Spring curves. Special care is needed to handle vertex blends at T-nodes, or when convex and concave edge blends meet, see examples in Figure 1(b). The feature skeleton in these cases will connect two mid-curve pairs with *collinear* tangents. This situation can be detected by matching the ingoing and outgoing curve trajectories within the separator set of a vertex blend. Such an example is shown in Figure 5(b), where two close trajectories and two nearly equal estimated radii on the left and the right sides confirm the hypothesis that we are dealing with a T-node.

Once collinear edge pairs are detected the insertion of a spring curve is compulsory; in the remaining cases we compute the corners by intersecting the related two features boundaries running on the mesh. At the end, the number of sides of the vertex blend will be equal to the number of profile curves plus the number of spring curves.

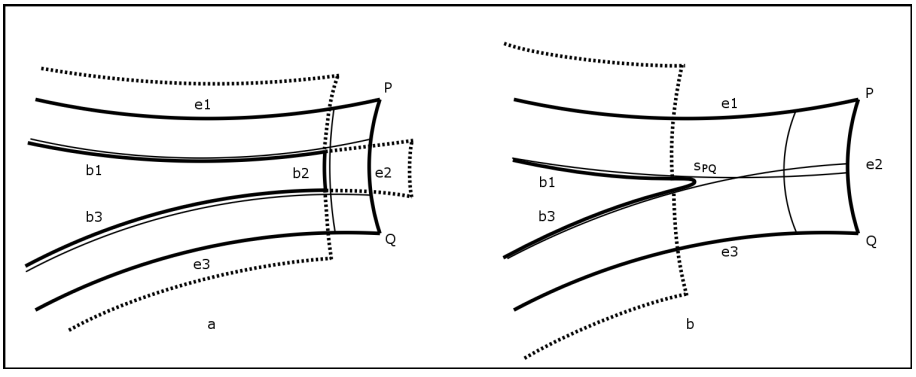
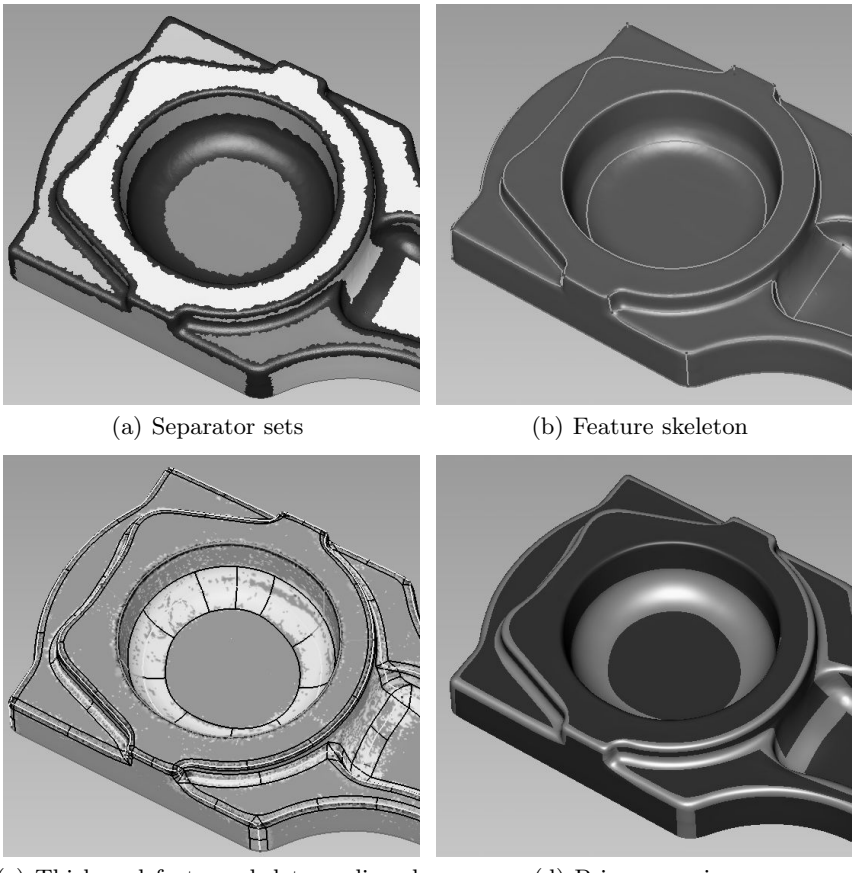


Fig. 7. Shrinking primary edge loops

Complex vertex blends. Without going into a detailed analysis we remark that building consistent loops for the primary faces may often require further computations. The basic problem is that the independently generated feature boundaries may interfere with each other, or become degenerate. A simple example shows how a primary region shrinks in Figure 7. On the left side, boundary $b1$ is intersected with boundary $b2$, which is intersected with $b3$, i.e. the internal region loop inherits the loop structure of the feature skeleton, see $e1 - e2 - e3$ and $b1 - b2 - b3$. In the other case on the right side, the thickened feature boundary $b2$ vanishes due to width of the transitions and the originally disjoint vertex blends at points P and Q merge. As a result, an "artificial" spring curve s_{PQ} is inserted to connect the shrunk boundaries $b1$ and $b3$ to complete the loop.

6 An Example

An industrial object using real measured data has been chosen to illustrate the proposed process of creating surface structures. Figure 8(a) shows the automatically generated primary regions and the separator sets that correspond to a particular Morse segmentation. Figure 8(b) shows the extracted feature skeleton structure that runs in the middle of the separator sets and has already been smoothed. The thickened feature skeleton, including edge and vertex blend boundaries, is shown in Figure 8(c). As can be seen the initial estimation are well-aligned with the numerical curvature map computed using the polygonal mesh. In the last Figure 8(d) the created primaries (red) and the connecting feature regions (grey) can be seen. Note that there is no connecting feature between the top face and the adjacent cylinder, since the common edge was classified as sharp and it is computed by surface-surface intersection. This model contains 253030 triangles. Using a Pentium 2 GHz processor and the Geomagic Studio shape reconstruction system, the following computation times were measured.



(a) Separator sets

(b) Feature skeleton

(c) Thickened feature skeleton, aligned with curvatures

(d) Primary regions

Fig. 8. An example

Creating the polygonal mesh	12 sec
Computing separator sets	7 sec
Generating a feature skeleton	3 sec
Computing the thickened surface structure	3 sec

Time measurements for the final surface fitting have not been included into the table, since it strongly depends on whether rapid surfacing or trimmed surface fitting is applied.

In this example there was no need for user intervention, however, for complex parts or noisy data sets the user may want to enhance the results of the automatic algorithms.

7 Conclusion

An automatic process to create a CAD-like surface structure over a polygonal mesh was presented. The consistent topology of the structure is assured by applying results from combinatorial Morse theory, while the correct geometric location of the segmenting curve network is the result of tracing methods that utilize local indicators estimated at the vertices of the mesh. The final loop structures were created by applying setback type vertex blends. This process ends with a well-aligned, feature-based structure; however, further steps are necessary to create a complete CAD model. Different issues emerge when quadrilateral or trimmed surface models are fitted. To enhance the quality of the reconstructed models further research and development efforts are needed; these include exact feature boundary relocation, stitching issues, surface fairing with dependencies and enforcing various engineering constraints.

Acknowledgements

This algorithm has been developed and implemented by Geomagic's international engineering team residing in North Carolina and Hungary. The authors would like to acknowledge the important contribution of Herbert Edelsbrunner concerning the fundamentals of Morse Complex segmentation, and that of Tobias Gloth and Dmitry Nekhayev, who implemented the initial modules of the above algorithm. This research has been supported by two NSF–SBIR grants, namely Award #0450230, "Creating functionally decomposed surface models from measured data" and Award #0521838, "Applications of Morse theory in reverse engineering."

References

[AEHW04] P. Agarwal, H. Edelsbrunner, J. Harer, Y. Wang: *Extreme elevation on a 2-manifold,* Proc. 20th Ann. Symp. on Computational Geometry, 2004, 357–365

[BenVar01] P. Benko, T. Varady: *Algorithms for reverse engineering boundary representation models,* Computer-Aided Design, **33** (11), 2001, 839–851

[BenVar04] P. Benko, T. Varady: *Segmentation methods for smooth point regions of conventional engineering objects,* Computer-Aided Design, **36**, 2004, 511–523

[Braid97] I. Braid: *Non-local blending of boundary models, Computer-Aided Design,* **29** (2), 1997, 89–100

[CsaWal00] P. Csakany, A. M. Wallace: *Computation of local differential parameters on irregular meshes,* In: The Mathematics of Surfaces IX, Eds.: R. Cipolla, R. R. Martin, Springer-Verlag, 2000, 19–33

[EckHop96] M. Eck, H. Hoppe: *Automatic reconstruction of B-spline surfaces of arbitrary topological type,* Computer Graphics, SIGGRAPH'96, 1996, 325–334

[EdeLeZ02] H. Edelsbrunner, D. Letscher, A. Zomorodian: *Topological persistence and simplification,* Discrete Computational Geometry, **28**, 2002, 511–533

[EdeHaZ03] H. Edelsbrunner, J. Harer, A. Zomorodian: *Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds,* Discrete Computational Geometry, **30**, 2003, 87–107

[FitEgF97] A. W. Fitzgibbon, D. W. Eggert, R.B. Fisher: *High-level CAD model acquisition from range images,* Computer-Aided Design, **29** (4), 1997, 321–330

[GelGui04] N. Gelfand, L. J. Guibas: *Shape Segmentation Using Local Slippage Analysis,* Eurographics Symposium on Geometry Processing, 2004, 214–223

[Geom06] http://www.geomagic.com/en/dssp_resources/

[HecGar97] P. S. Heckbert, M. Garland: *Survey of Polygonal Surface Simplification Algorithms,* Multiresolution Surface Modeling Course, SIGGRAPH'97, 1997

[HuaMen01] J. Huang, C.H. Menq: *Automatic data segmentation for geometric feature extraction from unorganized 3D coordinate points,* IEEE Trans. Rob. Aut., **17** (3), 2001, 268–279

[LeoJaS97] A. Leonardis, A. Jaklic, F. Solina, *Superquadrics for segmenting and modeling range data,* IEEE PAMI, **19** (11), 1997, 1298–1295

[LeSSCD98] A. Lee, W. Sweldens, P. Schroeder, L. Cowsar, D. Dobkin: *MAPS: multiresolution adaptive parameterization of surfaces,* Comput. Graphics, Proc., SIGGRAPH'98, 1998, 95–104

[ManWhi99] A. P Mangan, R. T. Whitaker: *Partitioning 3D surface meshes using watershed segmentation,* IEEE Trans. on Visualization and Computer Graphics, 1999

[Marks05] P. Marks: *Capturing a Competitive Edge through Digital Shape Sampling & Processing (DSSP),* SME, Blue Book Series, 2005

[Miln63] J. Milnor: Morse Theory, Princeton Univ. Press, New Jersey, 1963

[RazBae03] A. Razdan, M. S. Baeb: *A hybrid approach to feature segmentation of triangle meshes,* Computer-Aided Design, **35**, 2003, 783–789

[SapBes95] N. S. Sapidis, P. J. Besl: *Direct Construction of Polynomial Surfaces from Dense Range Images through Region Growing,* ACM Trans. on Graphics, **14** 1995, 171–200

[VanBru04] M. Vanco, G. Brunnett: *Direct Segmentation of Algebraic Models for Reverse Engineering,* Computing, Springer, **72** (1–2), 2004, 207–220

[VarHof98] T. Varady, C. M. Hoffmann: *Vertex blending: problems and solutions,* Mathematical Methods for Curves and Surfaces II, Eds: M. Daehlen, T. Lyche, L. L. Schumaker, Vanderbilt University Press, 1998, 501–527

[VarMar02] T. Varady, R. R. Martin: *Reverse Engineering, Chapter 26,* In: Handbook of Computer Aided Geometric Design, Eds.: G. Farin, J. Hoschek, M.–S. Kim, North-Holland, 2002, 651–681

[VarFac05] T. Varady, M. A. Facello: *New trends in digital shape reconstruction,* The Mathematics of Surfaces XI, Eds.: R. R. Martin, H. Bez, M. Sabin, Springer, 2005, 395–412

[VieShi05] M. Vieira, K. Shimada: *Surface mesh segmentation and smooth surface extraction through region growing,* Computer Aided Geometric Design, **22**, 2005, 771–792

Ensembles for Normal and Surface Reconstructions

Mincheol Yoon[1], Yunjin Lee[1], Seungyong Lee[1], Ioannis Ivrissimtzis[2], and Hans-Peter Seidel[3]

[1] POSTECH
[2] Coventry University
[3] MPI Informatik

Abstract. The majority of the existing techniques for surface reconstruction and the closely related problem of normal estimation are deterministic. Their main advantages are the speed and, given a reasonably good initial input, the high quality of the reconstructed surfaces. Nevertheless, their deterministic nature may hinder them from effectively handling incomplete data with noise and outliers. In our previous work [1], we applied a statistical technique, called *ensembles*, to the problem of surface reconstruction. We showed that an ensemble can improve the performance of a deterministic algorithm by putting it into a statistics based probabilistic setting. In this paper, with several experiments, we further study the suitability of ensembles in surface reconstruction, and also apply ensembles to normal estimation. We experimented with a widely used normal estimation technique [2] and Multi-level Partitions of Unity implicits for surface reconstruction [3], showing that normal and surface ensembles can successfully be combined to handle noisy point sets.

1 Introduction

Creating a 3D model of a real world object is a lengthy and complicated process. Despite recent progress, the whole procedure is still far from being optimal and may also need some manual input (e.g., see [4,5]). As a result, 3D content is still relatively scarce, slowing the spreading pace of 3D in critical applications like e-commerce.

The 3D modeling pipeline starts with the acquisition stage. We scan the physical object acquiring geometric information, usually in the form of a point cloud. The next task, which is the topic of this paper, is processing the geometric information to create a surface representation of the boundary of the scanned object. This problem is known in the literature as *surface reconstruction*.

Surface reconstruction is closely related to the problem of *normal reconstruction* for an unorganized point cloud (also referred in the literature as normal estimation). The reason is that the fastest and more robust surface reconstruction algorithms require points with normals as input, instead of unorganized points. Clearly, good normal reconstructions are necessary for good surface reconstructions. In our experiments [1], outlier normal noise was the most likely

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 17–33, 2006.
© Springer-Verlag Berlin Heidelberg 2006

source of problems when using the state-of-the-art surface reconstruction techniques, such as [3].

Most of the proposed surface reconstruction techniques are deterministic and control the quality of the surface through a user specified error bound. However, noisy or incomplete data often lead to reconstructions that do not faithfully represent the input, even when the error bound is set to zero. In this case, further manual input may be necessary for an acceptable result.

As we argued in [1], the robustness of a deterministic algorithm can be improved by putting the algorithm in a probabilistic setting with repetitive random subsampling of the input and averaging of the different outputs. The trade-off for the improvement in the robustness is the extra computational cost. We believe that this is well justified in the case of surface reconstruction, given that the process is not yet fully automated and the extra spent computational time can save human labor time. In addition, surface reconstruction is usually an one-off process, justifying more computation to obtain a better quality result.

To put a deterministic algorithm into a probabilistic setting, first, we randomly subsample the input data to create several subsets of input which are not necessarily distinct. Next, we use the deterministic algorithm to process each one of the subsets separately. The result is a set of several different outputs which is called an *ensemble*. In the last step, the members of the ensemble are combined into a single output. The latter is expected to have higher quality compared to the individual members of the ensemble.

In this paper, we use ensembles for two closely related problems, surface and normal reconstructions. The deterministic algorithm we use for normal reconstruction was proposed by Hoppe *et al.* [2] and is still widely used (e.g., [6]). The deterministic algorithm we chose for surface reconstruction is the Multi-level Partition of Unity (MPU) implicits [3], which is one of the fastest and most up-to-date techniques available.

The main contributions can be summarized as follows;

- We demonstrate the effectiveness of the ensemble framework by experimenting with specific surface [3] and normal [2] reconstruction techniques.
- We investigate how the final reconstruction is affected by the
  - averaging method that creates the final output
  - sampling rate
  - number of ensemble members

2 Preliminaries

2.1 Ensembles

The ensemble technique is one of the central themes of statistical learning. In a general setting, a probabilistic algorithm, running many times on the same input data, generates several different outputs which are then combined into a single model. To optimize this process, it is important to use a robust averaging formula for combining the members of the ensemble.

The tools and the methodology for the study of the averaging rules depend heavily on the categorization of the algorithm as *supervised* or *unsupervised*. In the case of a supervised algorithm, we have some knowledge of the properties of the outputs. For example, we may know the error of each output. This knowledge can be used to find a combined output which will provably converge to zero error under mild conditions [7,8].

In an unsupervised algorithm, which is the case of this paper, we do not have any knowledge of the properties of the outputs. Thus, mean averaging, or a majority vote in the discrete case, is the only available option from the theoretical point of view. Nevertheless, we may still be able to devise a more sophisticated averaging rule that will be more robust in practice. We address this important issue in Sections 3.2 and 4.2.

In [9], the ensemble technique has been applied to the problem of surface reconstruction, showing a considerable improvement over the corresponding single reconstruction algorithm [10]. However, in [9], the ensemble technique was used over a naturally probabilistic algorithm, while here we impose a probabilistic setting on a deterministic algorithm and show that this improves its robustness.

2.2 Normal Estimation Techniques

The estimation of normals for an unorganized point cloud is usually done in two steps. The first step is the estimation of a tangent plane at each point, which will give the direction of the normal at that point. The second step is to determine a consistent orientation for the tangent planes at all points.

Regarding the estimation of the tangent plane, Hoppe *et al.* [2] use principal component analysis on the k-neighborhood of a point. As a result, the estimated tangent plane is the minimum least square fitting of the k-neighborhood. Its normal is the eigenvector corresponding to the smallest eigenvalue of the covariance matrix of the k-neighborhood. Pauly *et al.* [11] improve on this by minimizing a weighted least square distance, where the weights are given by a sigmoidal function. Gopi *et al.* [12] use singular value decomposition to minimize the dot product between the estimated normal and the vectors from the point to its k-neighbors. Hu *et al.* [13] proposed a bilateral normal reconstruction algorithm which combines estimations obtained at different sampling rates. Mitra *et al.* [14] use the least square fitting of the points inside a sphere of a given radius. Dey *et al.* [15] compute normals pointing to the center of the largest Delaunay ball incident to that point. Techniques for normal smoothing [16] are also related to the problem of normal estimation.

The second step in normal reconstruction is the consistent orientation of the tangent planes. It has attracted relatively little research interest and the pioneering work [2] is still widely used as the state-of-the-art. In [2], they start from an arbitrarily oriented tangent plane and propagate its orientation to its neighbors, guided by a minimum spanning tree. The edge weights of the tree are the angles between the tangent planes.

This second step looks much simpler as it only adds a binary attribute to the non-oriented tangent plane. Moreover, given a good point sample from a smooth

orientable surface, one would expect very few inconsistencies. Nevertheless, by wrongly orienting an exact tangent plane, we obtain a normal vector opposite to the correct. That is, the wrong orientation of the exact tangent plane produces the highest possible error, which means that at the second step of the normal estimation, we might introduce outlier normal noise. In our experiments, we found that this second step is more likely to introduce the kind of error that most affects the visual quality of the final reconstruction.

2.3 Surface Reconstruction Techniques

A large category of the proposed surface reconstruction algorithms can directly process points without normals. α-shapes are used in [17] and B-spline patches are fitted with detail displacement vectors in [18]. In [19,20,21,22], the surface is reconstructed from the Delaunay tetrahedrization of the point set.

Another category of algorithms requires points with normals as input. In this case, an implicit function $f : \mathbf{R}^3 \rightarrow \mathbf{R}$ is fitted to the input data and then the surface is extracted as the zero level set of f. This volumetric approach was pioneered in [2]. The current state-of-the-art in this approach includes a radial basis function based technique [23] and the MPU implicits [3] which uses quadratics.

In this paper, we use the MPU implicits [3], which is suitable for reconstructing a surface from a large point set equipped with normals. The algorithm divides a bounded domain into cells, creating an octree-based partition of the space, and approximates the local shape of the points in each cell with a piecewise quadratic function. If the local approximation for a cell is not accurate enough, the cell is subdivided and the approximation process is repeated until a user-defined accuracy is achieved. Finally, all the local approximations are blended together using local weights, giving a global approximation of the whole input point set.

3 Review of Surface Ensemble

The use of ensembles for surface reconstruction was proposed in [1]. The input of the algorithm is a point cloud P with normals and the output is a surface S. The pseudocode of the algorithm is

Surface Reconstruction Ensemble

Input: Point cloud P with normals.
Output: Surface S.

1. Create several random subsets of P. These subsets may overlap.
2. Process the subsets separately with a deterministic surface reconstruction algorithm.
3. Use a surface averaging method to combine the reconstructions obtained in Step 2 into a single surface S.

3.1 Surface Ensemble Generation

For simplicity, the m subsets of P are created by random subsampling of P. In the case of surface reconstruction, each ensemble member gives one function value to each point of the space, regardless of the distribution of randomly sampled points. In contrast, as we will see in Section 4.1, a slightly more complicated procedure of random sampling without repetition is required for a normal ensemble, in order to guarantee an adequate number of normal estimates at every point.

After obtaining the random subsets, we use the MPU algorithm to generate an implicit surface representation for each subset. In the experiments, we used the MPU implementation available on the internet.

3.2 Surface Averaging

As a result of the ensemble generation in Section 3.1, we obtain a set of m functions

$$f_j : \mathbf{R}^3 \to \mathbf{R}, \quad j = 1, \ldots, m. \tag{1}$$

The zero level set of each function f_j defines a surface S_j. The combined ensemble surface S is the zero level set of a function f obtained by averaging the member functions f_j.

The simplest way to define f at a point $\mathbf{x}$ is to take the mean average

$$f(\mathbf{x}) = \frac{1}{m} \sum_{j=1}^{m} f_j(\mathbf{x}). \tag{2}$$

In some cases, this simple average may be satisfactory. However, as it was shown in [1], the robustness of the method can be improved by removing probable outliers from the set of functions we average. Without loss of generality, assume that at a given point $\mathbf{x}$, we have

$$f_1(\mathbf{x}) \leq f_2(\mathbf{x}) \leq \ldots \leq f_m(\mathbf{x}). \tag{3}$$

A more robust averaging function is given by

$$f(\mathbf{x}) = \frac{1}{m - 2r} \sum_{j=r+1}^{m-r} f_j(\mathbf{x}). \tag{4}$$

In other words, we compute a mean average of the functions f_j at $\mathbf{x}$ after excluding the r smallest and the r largest values. The function f in Eq. (4) is continuous [1]. It was also shown that $r = \lfloor m/4 \rfloor$ works nicely in practice, especially for the handling of outlier noise [1].

4 Normal Ensemble

The input of the ensemble for normal reconstruction is a set P of points without normals. We first generate random subsets of P and use them to compute the members of the normal ensemble. We then combine the members into a single normal reconstruction. The process is summarized by the following pseudocode;

Normal Reconstruction Ensemble

Input: Unorganized point cloud P.
Output: Point cloud with normals P_N.

1. Create several overlapping random subsets of P.
2. Process each subset separately estimating normals for its points.
3. For each point of P, estimate a single normal by combining all the normals estimated for this point at Step 2.

4.1 Normal Ensemble Generation

First, we subsample P to create the subsets P_i, $i = 1, \ldots, k$. Among different possible ways to perform the subsampling, we choose the simplest solution, as long as the simplicity does not compromise the quality of the results.

In our experiments, the sets P_i, $i = 1, \ldots, k$, have the same number of points,

$$|P_1| = |P_2| = \cdots = |P_k|, \tag{5}$$

where $|P|$ denotes the number of points in P. The sampling rate $d = |P_i|/|P|$ is the density of the subsampling. Obviously, the value of d affects the quality of the normal estimation. For example, if $d = 1$, the algorithm becomes deterministic and no improvement over the single reconstruction is possible. On the other hand, a very small d may again yield bad estimates because points that are far away from P_i may become its closest neighbors.

The choice of k, i.e., the number of members of the ensemble, is a trade-off between speed and quality. The normal estimation algorithm will run k times on a set of size $|P_i|$ and thus, a large k will slow the process down. On the other hand, there will be about $m = k \cdot d$ different normal estimations for each point of P and the higher this number, the more accurate the estimates.

During this process, it is important that each point is sampled several times, because the goal is to obtain good normal estimations for all the points of P. However, if the random sampling algorithm does not explicitly avoid repetitions, there is a possibility that some points of P will be sampled very few times, or never. To solve this problem, we create the subsets P_i using an algorithm for sampling without repetition and when we exhaust all the points of P, we start the sampling without repetition all over again.

After constructing the random subsets P_i, we use the algorithm described in [2] to obtain normal estimations. In the experiments, we adopted the implementation of the algorithm available on the internet and used the default parameters in most cases.

4.2 Normal Averaging

In the previous step, for each point of P, we obtained several different normal estimates. Next, we have to combine them into a single estimate with a less expected error. Similarly to Eq. (2), we can use the normalized mean average

$$\mathbf{n} = \sum_{i=1}^{m} (\mathbf{n}_i) / |\sum_{i=1}^{m} (\mathbf{n}_i)|. \tag{6}$$

To improve on the results obtained with mean averaging, we have to find a robust normal averaging analogous to Eq. (4). One possible way is to start with Eq. (6) and then find the estimates that deviate most from $\mathbf{n}$. That is, for each $\mathbf{n}_j$, we compute the angle θ_j between $\mathbf{n}$ and $\mathbf{n}_j$. Without loss of generality, assume that

$$\theta_1 \leq \theta_2 \leq \ldots \leq \theta_m. \tag{7}$$

We can exclude the r estimates with the larger deviations from $\mathbf{n}$ and average the rest of them using [24]. That is, we finally obtain

$$\mathbf{n}_f = \text{Aver}(\mathbf{n}_1, \ldots, \mathbf{n}_{m-r}), \tag{8}$$

where Aver() is the average of normals proposed in [24].

We experimented with $r = \lfloor m/2 \rfloor$, excluding about a half of the estimates. It is highly unlikely that outlier noise, e.g., wrongly oriented normals, covers more than a half of the samples. However, we still witnessed some problems with the quality of the normal reconstruction, which we attributed to the inaccurate ordering of the normals caused by the low accuracy of the initial mean average in Eq. (6).

To improve the accuracy of the normal estimation, we notice that Eq. (7) orders the normals according to their total variance

$$Var(\mathbf{v}) = \frac{1}{m} \sum_{j=1}^{m} (1 - \mathbf{v} \cdot \mathbf{n}_j)^2. \tag{9}$$

Then, instead of directly using Eq. (9) to order the normals, we compute the sum of total variances

$$Var(N) = \frac{1}{m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} (1 - \mathbf{n}_i \cdot \mathbf{n}_j)^2 \tag{10}$$

and use its constant multiple as the threshold for outlier detection. That is, if $Var(\mathbf{n}_i)$ is larger than $c \cdot Var(N)$, we consider $\mathbf{n}_i$ to be an outlier. In this approach, at different points, a different number of outliers may be removed. We consider this as an important improvement, because we do not expect the outliers to be evenly spread over P. In our experiments, we found that a value for c between 1.1 and 1.3 achieves the best results. Notice that a value $c = 1$ means that a normal is outlier when its total variance is larger than the average of the total variances.

The same approach to outlier removal can also be applied to surface ensembles. However, in that case, we also have to consider the continuity of the final surface. Thus, the total variance approach becomes too complex on surfaces, which is why we did not use it in this paper.

5 Ensemble Experiment with an Implicit Model

To validate the ensemble algorithms, we tested them on a surface with a known analytic formula. We used the *tangle cube* given by the equation

$$x^4 - 5x^2 + y^4 - 5y^2 + z^4 - 5z^2 + 11.8 = 0, \tag{11}$$

which has a fairly complex shape and non-trivial topology (see Fig. 1(a)).

In the set up of the validation experiment, we followed the approach used in [22]. We first sampled a large number of points from the tangle cube, here $N = 244,936$ points. In addition, we sampled subsidiary points to which we added noise or outlier noise by randomly perturbing the positions from the original. We created five point sets with different amounts of noise or outlier points. In the first point set, we added some outliers with much larger amount of perturbation as well as noisy points. Other point sets contain only noisy points. Table 1 shows the maximum displacement from the original position used for generating noisy and outlier points, measured as the percentage of the diagonal of the bounding box. The ratio of the number of subsidiary points to N, i.e., the number of points without noise, is inside the parentheses.

Table 1. Five point sets representing the tangle cube with different amounts of noise

Model	Noise	Outlier	Model	Noise	Outlier
Tangle A	2%(30%)	14%(8%)	Tangle D	5%(60%)	none
Tangle B	2%(30%)	none	Tangle E	5%(90%)	none
Tangle C	5%(30%)	none			

5.1 Error Measurement

To evaluate the normal reconstructions, we measured the errors in all points except those with added noise, where we do not have an analytical formula for the exact normal. The average error is

$$RMS = \sqrt{\frac{1}{N}\sum_j (1 - \mathbf{n}_j^e \cdot \mathbf{n}_j)^2} \tag{12}$$

and the maximum error is

$$MAX = \max_j (1 - \mathbf{n}_j^e \cdot \mathbf{n}_j), \tag{13}$$

where $\mathbf{n}_j^e$ is the estimated normal and $\mathbf{n}_j$ is the exact normal computed from the analytical formula of the surface.

To evaluate the surface reconstructions, we measured the distances of the N original points sampled on the tangle cube from the reconstructed surfaces. The distances were computed by the Metro tool [25], where we adjusted the tool to measure the distances of points from a mesh.

5.2 Effectiveness of the Ensembles

We validated the ensemble technique with experiments on tangle A and tangle B. They have the same amount of noise but the former also has outliers. In the

experiments, for normal ensembles, the sampling rate d is 0.1 and the number of ensemble members m is 6, which implies that the number of subsets k is 60. For surface ensembles, the sampling rate is 0.2 and the number of ensemble members is 11.

Table 2 shows the results of normal reconstructions. As expected, the error of the normal reconstruction always decreases with the use of ensemble. Table 2 also shows a comparison of the three normal averaging methods described in Section 4.2. In the case of tangle A, which contains outliers, the more sophisticated averaging rule using Eq. (10) achieves the best results. However, in the case of tangle B, which does not contain outliers, the simple averaging of Eq. (6) produces the best results. From these results, we can see that the averaging rule with Eq. (10) is most effective when the given point set contains some outliers.

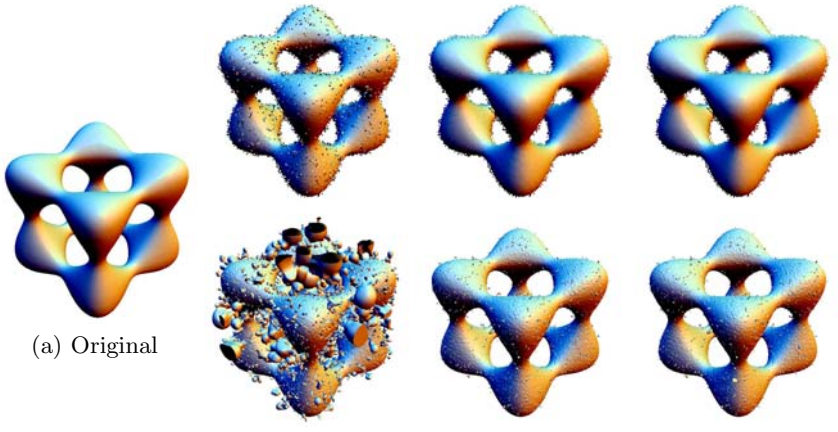
Table 3 shows the results of surface reconstructions. In tangle A, the use of more accurate normal estimates decreases the RMS error of the surface reconstruction. We also notice that the error of the corresponding reconstructed surfaces is almost the same, regardless of the normal averaging method used. Regarding the use of surface ensemble, it always decreases the MAX error of the reconstructed surface. Surface ensemble decreases the RMS error in tangle A,

Table 2. Normal Ensembles: "S. Avg." is the simple average of Eq. (6). "O. Avg." is the average with normal ordering of Eq. (8). "V. Avg." is the average with the total variance of Eq. (10). The computation time is measured in seconds on a PC running Windows with a Pentium 4 630 processor with 2GB memory.

Model	# Points		Single Est.	Normal Ens.		
				S. Avg.	O. Avg.	V. Avg.
Tangle A	337,698	Time (sec)	34.73	251.31	252.70	253.28
		RMS ($\times 10^{-4}$)	44.15	8.08	8.29	7.45
		MAX ($\times 10^{-4}$)	853.4	188.98	208.66	180.8
Tangle B	317,934	Time (sec)	30.44	226.70	228.23	228.45
		RMS ($\times 10^{-4}$)	26	3.15	4.04	4.09
		MAX ($\times 10^{-4}$)	420.65	73.19	104.07	90.09

Table 3. Surface Reconstruction: For surface ensembles, "S. Avg." is the simple average of Eq. (2). "R. Avg." is the robust average of Eq. (4). The normals obtained by the normal ensemble with "V. Avg." were used for surface ensembles. The computation time is measured as in Table 2.

Model	# Points		Single MPU				Surface Ens.	
			Single N.	S. Avg.	O. Avg.	V. Avg.	S. Avg.	R. Avg.
Tangle A	337,698	Time (sec)	48.05	30.19	30.06	30.08	114.19	114.26
		RMS ($\times 10^{-4}$)	3.56	2.98	3.01	2.99	2.46	1.84
		MAX ($\times 10^{-4}$)	40.12	50.76	47.16	46.88	26.42	21.03
Tangle B	317,934	Time (sec)	12.53	12.55	12.75	12.70	51.75	54.0
		RMS ($\times 10^{-4}$)	1.78	1.88	1.58	1.93	1.78	1.77
		MAX ($\times 10^{-4}$)	11.28	11.72	12.21	12.77	8.58	8.67



(a) Original (b) Single normal (c) Simple average (d) Eq. (10) average

Fig. 1. Tangle Cube: Comparison of different normal estimates and the surfaces obtained from them. At the top of (b), (c) and (d), each point is rendered with illumination determined by the normal vectors. At the bottom of (b), (c) and (d), the surfaces are reconstructed using a single MPU.

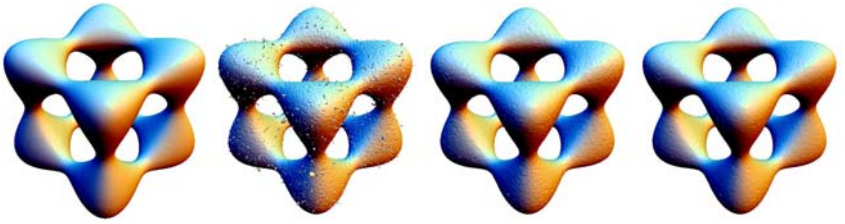
while it slightly affects the RMS error of tangle B. In addition, in tangle A which contains outlier noise, the robust averaging is better than the simple averaging in surface ensembles. In the experiments of surface ensembles, we used the normals obtained by the normal ensemble with the total variance averaging in Eq. (10).

Fig. 1 visualizes the results of normal ensembles on tangle A, which have higher visual quality than those with single normal estimation. In contrast, numerically, the error of a single MPU using single normal estimation is not much worse than that of a single MPU using normals from ensembles (see Table 3). However, as the large difference in the visual quality reveals, this is due to the fact that we measure the error as the distances of the points on the original surface from the reconstruction. That is, all these artifacts in the reconstruction are not fully penalized by the error metric. If we computed distances from the points on the reconstruction to the original tangle cube instead, then the error measure would be more faithful to the visual quality. Nevertheless, the former distances, from the tangle cube to the reconstruction, are more consistent with the error metric for normal estimation and we used them in this paper.

Fig. 2 shows that the visual quality of the surface ensemble is better than that of a single surface reconstruction.

5.3 Influence of the Sampling Rate

In this section, normal and surface ensembles are experimented with different sampling rates. The number of ensemble members is fixed, which is 6 and 11 for normal and surface ensembles, respectively. For normal and surface averaging, we used the robust techniques with Eq. (4) and Eq. (10), respectively. In the



(a) Original (b) Single MPU (c) Simple average (d) Robust average

Fig. 2. Tangle Cube: Comparison of surfaces obtained from single MPU and surface ensembles

experiments of surface ensembles, we used the normals obtained by the normal ensemble with the sampling rate of 0.1.

Table 4 shows the relationship between sampling rate and noise. In the case of normal ensembles, a larger amount of noise in the point set leads to smaller optimal sampling rates. Indeed, tangle C has 0.1 as optimal sampling rate, tangle D has 0.05-0.1, and tangle E has 0.05. In contrast, in the case of surface ensemble, all of three models have the optimal sampling rate in the range of 0.2-0.25.

Table 4. Experimental results with various sampling rates

Model	($\times 10^{-4}$)	Normal Ensemble					Surface Ensemble				
		0.01	0.05	0.1	0.2	0.3	0.1	0.15	0.2	0.25	0.3
Tangle C	RMS	5529.59	9.32	7.97	13.12	20.27	2.5	2.38	2.37	2.36	2.41
	MAX	19999.62	208.73	143.22	255.5	320.18	18.21	22.59	18.15	23.03	23.25
Tangle D	RMS	43.54	10.26	13.01	26.39	42.96	3.53	3.33	3.3	3.26	3.7
	MAX	1256.24	215.42	141.6	400.76	731.48	23.41	22.36	22.46	22.58	23.77
Tangle E	RMS	35.38	11.84	18.72	40.92	69.02	3.96	3.79	3.67	3.65	3.7
	MAX	782.66	205.21	227.26	758.52	1079.01	22.83	23.72	21.93	22.24	23.77

5.4 Influence of the Number of Ensemble Members

The ensembles studied here have their members created by the same probabilistic process. Thus, all the members of an ensemble have the same expected error, which can be written as the sum of the bias and the variance. When the size of the ensemble increases, that is, when more members are added to it, the variance component of the expected error of the ensemble decreases. Therefore, if we use larger ensembles, we can expect more accurate reconstructions. Table 5 shows the experimental validation of this theoretical reasoning. As a trade-off, the computational time also increases with the size of the ensemble.

In Table 5, the sampling rates are fixed as 0.1 and 0.2 for normal and surface ensembles, respectively. Robust averaging with Eq. (4) and Eq. (10) were used for normal and surface ensembles, respectively. For surface ensembles, we used the normals obtained by the normal ensemble when the number of ensemble members is 6.

Table 5. Experimental results with various numbers of ensemble members

Model	# Points		Normal Ens.			Surface Ens.		
			6	12	18	5	11	17
Tangle A	337,698	Time (sec)	253.28	523.76	802.22	53.86	114.26	182.39
		RMS ($\times 10^{-4}$)	7.45	4.14	3.19	2.39	1.84	1.62
		MAX ($\times 10^{-4}$)	180.8	74.9	52.4	33.7	21.0	14.0

6 Experimental Results from Real Data

In this section, we show experimental results from two well-known point sets; the bierkrug model from the ViHAP3D project and the armadillo model from the Stanford 3D scanning repository. Throughout this section, we always use Eq. (10) for normal averaging and the robust averaging with Eq. (4) for surface ensembles.

Fig. 3 shows normal estimations with and without ensemble, and the corresponding single MPU reconstructions for the bierkrug model. In the experiments, for normal ensembles, the sampling rate d is 0.2 and the number of ensemble members m is 6. For surface ensembles, the sampling rate is 0.2 and the number of ensemble members is 11.

In the visualization of the point sets in Figs. 3(a) and 3(b), the drawing inscribed on the bierkrug looks more detailed in the single normal estimation. However, this does not mean that the single estimated normals are more accurate. In fact, in this area of the bierkrug, we have overlapping range images and most of the single estimated normal information, even though visually pronounced, is of low quality. This is confirmed by Figs. 4(c) and 4(d), where in both cases the single MPU reconstructions smoothed out the inscription.

In Figs. 4(a), 4(c), 4(e), and 4(f), we show close-ups of the two single MPU reconstructions of Fig. 3. In Figs. 4(b), 4(d), 4(g), and 4(h), we show close-ups of the corresponding surface ensemble reconstructions using single estimated and ensemble normal sets. The combination of normal and surface ensembles clearly

(a) (b) (c) (d)

Fig. 3. Bierkrug: (a) Normals obtained by single estimation. (b) Normals obtained by ensemble. (c) and (d) Single MPU reconstructions of (a) and (b), respectively.

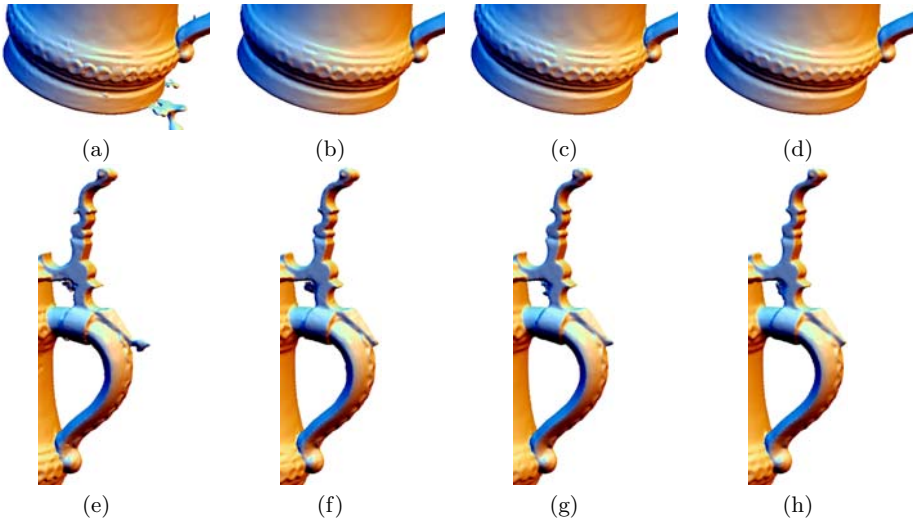


Fig. 4. Bierkrug: (a) and (e) zoom-ins of Fig. 3(c). (c) and (f) zoom-ins of Fig. 3(d). (b) and (g) show the result of the surface ensemble for Fig. 3(a). (d) and (h) show the result of the surface ensemble for Fig. 3(b).

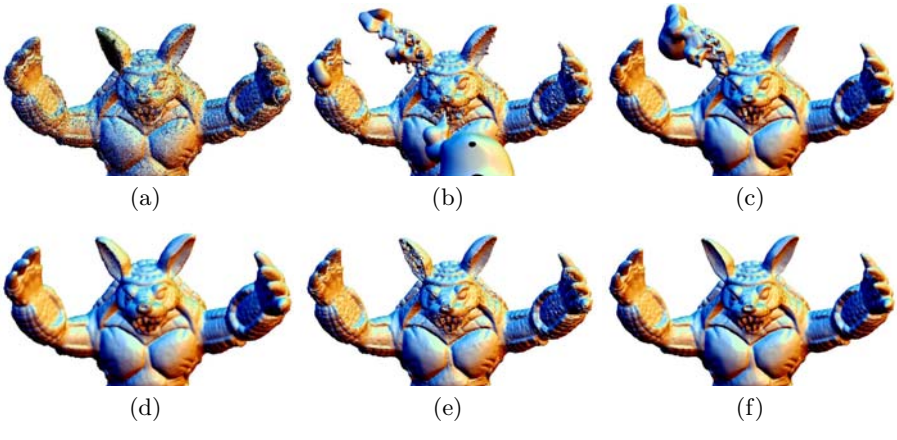


Fig. 5. Armadillo: (a)-(c) Single estimated normals. (d)-(f) Ensemble estimated normals. Left: Points with normals. Center: Single MPU. Right: MPU ensemble.

outperforms all the other methods. In particular, it is the only method that resolves the artifacts on the base of the bierkrug, as shown in Fig. 4(d). It is also the one that gives the best reconstruction of the smaller handle, as shown in Fig. 4(h).

Fig. 5 shows the point set and the single and ensemble MPU reconstructions of the armadillo model. The normals are obtained either by single or by ensemble estimation. In the experiments, for normal ensembles, the sampling rate d is

0.1 and the number of ensemble members m is 6. For surface ensembles, the sampling rate is 0.2 and the number of ensemble members is 11.

The surface ensembles in Figs. 5(c) and 5(f) outperform the single reconstructions in Fig. 5(b) and 5(e). Similarly to the experimental results for the bierkrug, the model in Fig. 5(f), obtained using ensembles both for normal estimation and surface reconstruction, has the highest visual quality. Indeed, the artifacts have been effectively removed, while all the important geometric detail is preserved. Note that the input of this experiment is the original raw data from the Stanford 3D scanning repository and the results should not be compared with reconstructions using filtered clean data. In the MPU reconstructions, we did not use the confidence values provided with the point data.

Table 6 shows timing statistics of the ensembles with information about the model sizes.

Table 6. Timing statistics: The computation time is measured as in Table 2

Model	# Points	Normal estimation		Surface reconstruction	
		Single Normal	Normal ensemble	Single MPU	Surface ensemble
Bierkrug	500,690	63.91	540.57	33.14	170.27
Armadillo	1,394,271	410.07	2102.1	107.89	304.55

7 Discussion and Future Work

Ensemble is a powerful statistical tool with a wide range of applications. It facilitates the handling of large noisy data and thus it is well suited for the problem of surface reconstruction from scanned data, as well as the closely related problem of normal estimation.

In our previous short paper [1], we used the ensemble technique to enhance the performance of a surface reconstruction algorithm. In this paper, we apply the ensemble to the problem of normal estimation. We show that ensembles can increase the resilience of existing normal estimation algorithms in the presence noise and outliers.

Normal and surface ensembles can be naturally combined by using the output of the normal ensemble as the input of the surface ensemble. We found that this combination maximizes the quality of the final reconstructed surface because it deals with both kinds of outliers that might create unwanted artifacts. The normal ensemble deals with normal outliers in the form of wrongly oriented normals, while the surface ensemble deals with spatial outliers.

On the technical side, we propose a new method for normal averaging in the presence of noise and outliers. Compared to the mean averaging, our method produces more accurate normal estimates on inputs with outlier points. It also outperforms the averaging by Eq. (8), which is a simple extension of the robust averaging we used for surface ensembles. The reason is that our method does not only remove outlier normals but also averages as many inlier normals as possible.

To validate the ensemble technique, we made experiments with several point sets with different noise profiles, sampled from the same implicit surface. Naturally, we found that our algorithm shows considerable improvements on inputs with outliers, while it shows small improvements on inputs with only moderate noise. In addition, we experimented with different sampling rates and numbers of ensemble members. While the optimal sampling rate for a surface ensemble is almost always the same, the optimal sampling rate for normal ensembles decreases when the noise of the input increases. The experiment with different numbers of ensemble members showed the trade-off between the accuracy of the estimated normals and computational time. As the number of ensemble members increases, so does the accuracy of the results and the computational cost.

As the ensemble algorithms naturally filter out the noise of the data, they have several similarities with smoothing. For example, the size of the local neighborhoods used in most smoothing algorithms is related to the density of the random subsampling of the ensembles. We believe that random sampling reflects the probabilistic nature of the noise better, compared to the deterministically defined k-neighborhoods. Another advantage of the ensembles over smoothing is their ability to cope with outliers. By filtering out outliers before averaging, ensembles can prevent a wrongly oriented normal or a misplaced point from affecting the final surface. In contrast, surface reconstruction without proper handling of outliers can generate a highly distorted shape which may not be remedied by smoothing.

Compared to the deterministic approach, one drawback of the method is the higher computational cost. However, accurate normal estimations enhance the performance of the surface reconstruction algorithms and further increase the robustness of the surface ensemble technique. This way, we also increase the possibility of obtaining an accurately reconstructed surface which will require less human labor for postprocessing.

In terms of memory cost, an ensemble technique may seem to need more memory than the corresponding single reconstruction technique because we should keep several outputs for averaging to generate the final output. However, in practice, the dominant memory consumption usually happens when we run the reconstruction technique, which may need additional data structures for processing. By applying the reconstruction technique to the subsets of input in turn and storing the results to files, we can separate the ensemble generation and averaging steps. In this case, each subset is much smaller than the input and the memory cost will be reduced in the step of ensemble generation. For averaging step, the memory cost may not increase if we can compactly represent the reconstruction results by removing additional data structures not used anymore.

We notice that the way we have put the MPU and Hoppe's method for normal estimation into a probabilistic setting is extremely simple, i.e., random subsampling of the data set. We expect that the same framework can be readily applied to improve the performance of other normal and surface reconstruction algorithms. It is our future plan to experiment with other algorithms and verify

the validity of that claim. A more challenging future work will be a theoretical analysis of the effects of the sampling rate and the number of ensemble members on the quality of the reconstructed surface.

Acknowledgements

The authors would like to thank Yutaka Ohtake for great help on the implementation of MPU ensembles. The armadillo model is courtesy of the Stanford Computer Graphics Lab. This research was supported in part by the BK21 program, the ITRC support program, and KOSEF (F01-2005-000-10377-0).

References

1. Lee, Y., Yoon, M., Lee, S., Ivrissimtzis, I., Seidel, H.P.: Ensembles for surface reconstruction. In: Proc. Pacific Graphics 2005. (2005) 115–117
2. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W.: Surface reconstruction from unorganized points. In Computer Graphics (Proc. ACM SIGGRAPH 1992) (1992) 71–78
3. Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., Seidel, H.P.: Multi-level partition of unity implicits. ACM Transactions on Graphics **22**(3) (2003) 463–470
4. Bernardini, F., Rushmeier, H.: The 3D model acquisition pipeline. Computer Graphics Forum **21**(2) (2002) 149–172
5. Weyrich, T., Pauly, M., Keiser, R., Heinzle, S., Scandella, S., Gross, M.: Post-processing of scanned 3D surface data. In: Proc. Eurographics Symposium on Point-Based Graphics 2004. (2004) 85–94
6. Sainz, M., Pajarola, R., Mercade, A., Susin, A.: A simple approach for point-based object capturing and rendering. IEEE Computer Graphics and Applications **24**(4) (2004) 24–33
7. Schapire, R.E.: The strength of weak learnability. Machine Learning **5**(2) (1990) 197–227
8. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: Machine Learning: Proc. the 13th International Conference. (1996) 148–156
9. Ivrissimtzis, I., Lee, Y., Lee, S., Jeong, W.K., Seidel, H.P.: Neural mesh ensembles. In: Proc. 3D Data Processing, Visualization, and Transmission, 2nd International Symposium on (3DPVT'04). (2004) 308–315
10. Ivrissimtzis, I., Jeong, W.K., Lee, S., Lee, Y., Seidel, H.P.: Neural meshes: Surface reconstruction with a learning algorithm. Technical Report MPI-I-2004-4-005, Max-Planck-Institut für Informatik, Saarbrücken (2004)
11. Pauly, M., Keiser, R., Kobbelt, L.P., Gross, M.: Shape modeling with point-sampled geometry. ACM Transactions on Graphics **22**(3) (2003) 641–650
12. Gopi, M., Krishnan, S., Silva, C.: Surface reconstruction based on lower dimensional localized Delaunay triangulation. Computer Graphics Forum (Proc. Eurographics 2000) **19**(3) (2000) 467–478
13. Hu, G., Xu, J., Miao, L., Peng, Q.: Bilateral estimation of vertex normal for point-sampled models. In: Proc. Computational Science and Its Applications (ICCSA 2005). (2005) 758–768
14. Mitra, N., Nguyen, A., Guibas, L.: Estimating surface normals in noisy point cloud data. Special Issue of International Journal of Computational Geometry and Applications **14**(4–5) (2004) 261–276

15. Dey, T.K., Li, G., Sun, J.: Normal estimation for point clouds : A comparison study for a voronoi based method. In: Proc. Eurographics Symposium on Point-Based Graphics 2005. (2005) 39–46
16. Jones, T.R., Durand, F., Zwicker, M.: Normal improvement for point rendering. IEEE Computer Graphics and Applications **24**(4) (2004) 53–56
17. Bajaj, C., Bernardini, F., Xu, G.: Automatic reconstruction of surfaces and scalar fields from 3D scans. In Proc. ACM SIGGRAPH 1995 (1995) 109–118
18. Krishnamurthy, V., Levoy, M.: Fitting smooth surfaces to dense polygon meshes. In Proc. ACM SIGGRAPH 1996 (1996) 313–324
19. Amenta, N., Bern, M., Kamvysselis, M.: A new Voronoi-based surface reconstruction algorithm. In Proc. ACM SIGGRAPH 1998 (1998) 415–421
20. Amenta, N., Choi, S., Kolluri, R.K.: The power crust, unions of balls, and the medial axis transform. Computational Geometry: Theory and Applications **19**(2-3) (2001) 127–153
21. Dey, T.K., Goswami, S.: Tight cocone: A water-tight surface reconstructor. Journal of Computing and Information Science in Engineering **3**(4) (2003) 302–307
22. Kolluri, R., Shewchuk, J.R., O'Brien, J.F.: Spectral surface reconstruction from noisy point clouds. In: Proc. Symposium on Geometry Processing 2004 (SGP 2004). (2004) 11–21
23. Carr, J., Beatson, R., Cherrie, J., Mitchell, T., Fright, W., McCallum, B., Evans, T.: Reconstruction and representation of 3D objects with radial basis functions. In Proc. ACM SIGGRAPH 2001 (2001) 67–76
24. Buss, S.R., Fillmore, J.P.: Spherical averages and applications to spherical splines and interpolation. ACM Transactions on Graphics **20**(2) (2001) 95–126
25. Cignoni, P., Rocchini, C., Scopigno, R.: Metro: Measuring error on simplified surfaces. Computer Graphics Forum **17**(2) (1998) 167–174

Adaptive Fourier-Based Surface Reconstruction

Oliver Schall, Alexander Belyaev, and Hans-Peter Seidel

Computer Graphics Group
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
{schall, belyaev, hpseidel}@mpi-inf.mpg.de

Abstract. In this paper, we combine Kazhdan's FFT-based approach to surface reconstruction from oriented points with adaptive subdivision and partition of unity blending techniques. The advantages of our surface reconstruction method include a more robust surface restoration in regions where the surface bends close to itself and a lower memory consumption. The latter allows us to achieve a higher reconstruction accuracy than the original global approach. Furthermore, our reconstruction process is guided by a global error control achieved by computing the Hausdorff distance of selected input samples to intermediate reconstructions.

1 Introduction

Many of today's applications make use of 3D models reconstructed from real-world objects such as machine parts, terrain data, and cultural heritage. Furthermore, digital scanning devices for acquiring high-resolution 3D point clouds have recently become affordable and commercially available. This has increased the demand for techniques for the robust reconstruction of accurate models from point cloud data. Therefore, surface reconstruction has been a field of intensive research addressed in various fields and many approaches have been proposed.

An important group of surface reconstruction algorithms are computational geometry approaches [1,2,3,4]. Those algorithms usually involve the computation of Delaunay or dual structures from the input data and reconstruct the surface by extraction from the previously computed Delaunay complex. One significant advantage of these methods is that they are usually supported by rigorous mathematical guarantees. On the other hand, most computational geometry techniques rely on clean data. Therefore, recent research trends focus on making those methods more robust on noisy data [5,6,7].

Another class of surface reconstruction algorithms approximate the input by the zero-level set of a trivariate function which is usually extracted to obtain the resulting surface. Hoppe et al. [8] locally estimate the signed distance function as the distance to the tangent plane of the closest point. Other methods use globally or locally supported radial basis functions (RBFs) to define the implicit function [9,10,11,12]. Ohtake et al. [13] define the implicit surface by locally fitting quadratic primitives to the input point set. Another powerful class of algorithms are Moving-Least-Squares (MLS) techniques [14,15,16]. They define

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 34–44, 2006.
© Springer-Verlag Berlin Heidelberg 2006

the implicit function by applying a projection operator that moves points in the vicinity of the MLS surface onto the surface itself. The surface is thus defined by all fixpoints of the projection operator. Steinke et al. [17] propose a machine learning approach based on Support Vector Machines (SVMs) to approximate an implicit surface from a point cloud which can be considered as an extension of RBF-based methods.

Recently, Kazhdan introduced a novel and elegant FFT-based reconstruction technique [18]. His approach is able to reconstruct a solid, watertight model from an oriented point set. He approaches the reconstruction problem indirectly by first determining the integral of the characteristic function of the domain bounding the input point set instead of the function itself. Using Stokes' Theorem this volume integral can be transformed into a surface integral which is dependent on positions on the boundary of the volume and the corresponding normal directions. As the oriented input point set can be seen as a sampling of this boundary, it can be used to approximate the surface integral and with it the integral of the characteristic function. To finally obtain the characteristic function of the dataset itself, the integration is conducted using the inverse FFT. This method allows a robust and fast reconstruction of a solid and watertight model from noisy samples. On the other hand, the approach has a high memory requirement due to its global nature. The integral of the characteristic function has to be sampled on a uniform grid for the whole volume in order to be able to apply the inverse FFT. This limits the maximal reconstruction resolution of the approach on today's computers to a level where the reconstruction of fine details of the input data is not possible. Furthermore, the approach has no global error control and its globality prevents the accurate reconstruction of regions where the input data bends close to itself. Our work proposes a simple solution to overcome these limitations while preserving the advantages of the global approach.

The general idea of our technique is to employ an error-guided subdivision of the input data. For this, we compute the bounding box of the input and apply an octree subdivision. In order to decide whether an octree leaf cell needs to be subdivided, we compute a local characteristic function for the points inside the cell using Kazhdan's global approach. This is a non-trivial task since the points inside a cell do in general not represent a solid. We propose a solution to that problem to avoid that surface parts are created which are not represented by points. If the resulting local approximation inside the cell is not accurate enough, the cell needs to be subdivided. By iterating this procedure, we compute overlapping local characteristic functions at the octree leaves for each part of the input with a user-defined accuracy. We obtain the final reconstruction by combining the local approximations using the partition of unity approach and extracting the surface using a polygonization algorithm. One advantage of our adaptive approach is that the characteristic function is only determined close to the surface and not for the whole volume. As the reconstruction accuracy is mainly limited by memory requirements, this allows us to obtain higher reconstruction resolutions. Additionally, the adaptiveness allows us a more accurate reconstruction of strongly bended regions of the input.

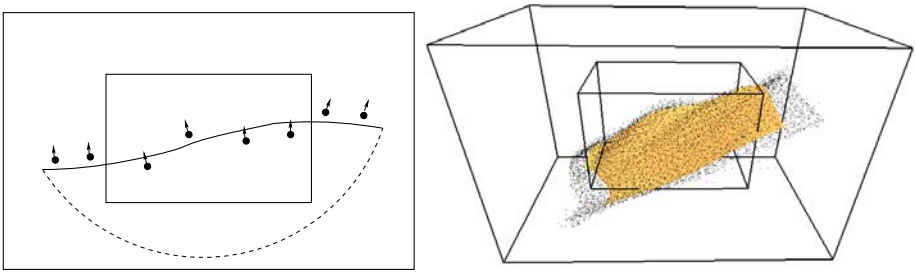


Fig. 1. Left: Local curve approximation for points inside and in the vicinity of a leaf cell (inner rectangle). The dashed line indicates the irrelevant region of the reconstructed solid. Right: Real 3D example of the sketch in the left image after pruning meaningless regions of the solid.

The rest of this paper is organized as follows. Section 2 presents details of our surface reconstruction technique. The data partitioning step and the computation of a local characteristic function for each cell is described in Section 2.1. Their integration and the extraction of the final surface is presented in Section 2.2. We show results of our technique in Section 3 before we conclude and describe future work in Section 4.

2 Adaptive FFT-Based Surface Reconstruction

In this section, we present our adaptive FFT-based surface reconstruction technique (in the following denoted as AdFFT) in detail. We first describe the error-controlled subdivision of the adaptive octree structure and the computation of overlapping local surface approximations for the input points associated with the octree leaves. We then integrate the local approximations using the partition of unity approach to reconstruct the final model.

2.1 Adaptive Octree Subdivision

The general idea of the partition of unity approach is to divide the data domain into several pieces and to approximate the data in these domains separately. The resulting local approximations are then blended together using smooth and local weighting functions which sum up to one over the whole domain.

In order to find local characteristic functions of the domain bounding the input point cloud, we first compute the axis-aligned bounding box of the input data. We then apply an adaptive octree subdivision of this bounding box. In order to decide whether a cell needs to be subdivided, we compute the characteristic function of this cell and its vicinity with a fixed accuracy. If the surface extracted from this characteristic function approximates the points in the cell sufficiently close according to a user-defined accuracy, the cell has not to be subdivided further.

How to compute the characteristic function for a cell of the octree is not obvious, as a straightforward application of the global FFT-based method always

determines a characteristic function representing a solid, whereas the points in a cell form in general non-closed surface patches. To avoid that irrelevant surface parts occur in the local characteristic function, we use the construction shown in Figure 1. We embed the octree cell including its oriented input samples at the center of a larger cell with doubled edge lengths. In order to allow a smooth transition between adjacent local characteristic functions later in the integration step, we add points in the vicinity of the original octree cell to the construction. In our implementation, we choose all points in the octree leaf cell scaled by a constant factor around its center for the computation of the local approximation. According to our experiments a constant factor of $c = 1.8$ works well for all performed tests. By using the global FFT-based method with a fixed resolution (2^5 in our implementation) on the larger volume, we then compute its characteristic function at regular grid positions. As the shape of the octree cells is usually not cubical, we transform all candidate data points and normals to fit into a cube to enable the use of the FFT. Figure 1 sketches the idea behind this construction. The surface patch inside and in the vicinity of the octree cell is correctly reconstructed and the irrelevant surface part of the solid is outside of the inner cell. This works in the majority of the cases as the irrelevant surface part always has an ellipsoidal shape. Additionally, adding sufficient samples in the vicinity of the octree cell increases the diameter of the shape that it does not cross inside the smaller cell. In rare cases, the crossing cannot be avoided due to very different alignment of octree cell and local surface approximation. But since the resulting unwanted surface parts are small and distant to the real surface they can be pruned easily during the polygonization. The right image of Figure 1 shows a real example of a local surface approximation for an octree cell and its vicinity.

To measure the accuracy of the resulting local approximation, we construct a mesh from the computed characteristic function using the Marching Cubes algorithm [19] and compute the Hausdorff distance of selected samples inside the cell to the mesh. If the average computed distance is above the user-defined error, the cell needs to be subdivided further. If a cell is empty, no approximation needs to be computed and we leave it untreated. In order to guarantee an efficient computation of the Hausdorff error, we use only a subset of points inside the octree cell. In our implementation we select 10% of the cell points to obtain a stable estimation.

In the presence of noise it might happen that the error criterion cannot be reached everywhere on the dataset. This leads to an oversubdivision of octree cells in very noisy regions until subcells contain not enough samples to allow a robust local reconstruction. To avoid this, we introduce a stopping criterion to ensure a minimum number of samples in non-empty cells. We fix this lower bound to be 0.5% of the number of input points.

2.2 Integration

After the subdivision step, we obtain an octree with leaves on different depths which are either empty or contain a sampling of a local characteristic function. In order to obtain a common global resolution for all local characteristic functions,

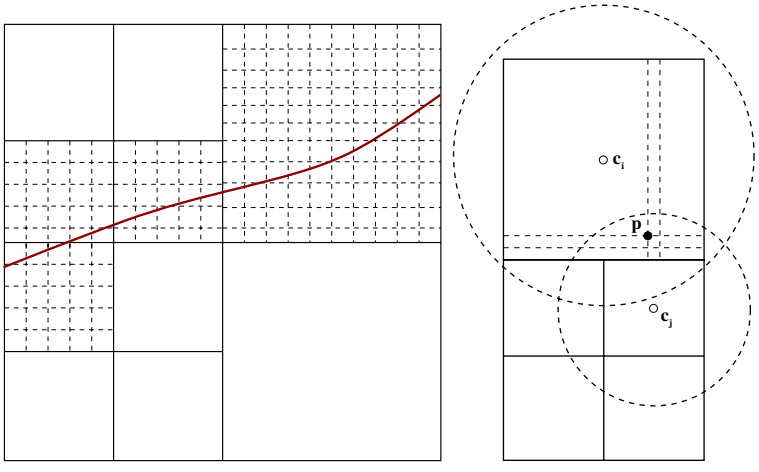


Fig. 2. Left: All local characteristic functions of octree cells containing the final iso-surface have a common resolution. This allows an easy interpolation between adjacent cells. Right: Example octree configuration for partition of unity blending. A corner point $\mathbf{p}$ of a Marching Cubes cell and radial kernels of octree cells with centers $\mathbf{c}_i$ and $\mathbf{c}_j$ are shown.

we reconstruct leaves with lower tree depths using a higher resolution inside each cell (see left illustration of Figure 2). This allows us to blend and to interpolate between adjacent cells and to apply the Marching Cubes algorithm on this uniform grid. To obtain the final reconstruction, we interleave the extraction of the iso-surface and the combination of the local characteristic functions. In order to be able to extract an iso-surface of a characteristic function which has a value of one inside the surface and zero outside of the surface, we need to choose an appropriate iso-value. We follow the global approach and choose it as the average value of the characteristic function values obtained at the input samples.

Our Marching Cubes implementation processes all octree cells for which local characteristic functions have been computed. As the local characteristic functions overlap each other, cubes close to the boundary of octree cells have more than one characteristic function value associated with its corners. To merge them into one value, we use partition of unity blending. More precisely, if we denote the corner position $\mathbf{p}$ and use our octree data structure to find all local function values $\{f_0, \ldots, f_N\}$ at this position which are associated with the cells $\{c_0, \ldots, c_N\}$, we determine the global characteristic function value as

$$f_g = \frac{\sum_{i=0}^{N} w_i f_i}{\sum_{i=0}^{N} w_i}$$

where

$$w_i = G_i \left(||\mathbf{c}_i - \mathbf{p}||_2 \right),$$

and $\mathbf{c}_i$ is the center of the cell c_i. The center of the radial Gaussian weighting function $G_i(\cdot)$ is fixed at $\mathbf{c}_i$. The bandwidth is chosen such that grid positions

Fig. 3. Zoomed parts of reconstructions of the Thai Statuette created using the global Kazhdan approach (left) and our technique (right). Both reconstructions were computed using the maximal possible resolution for each technique. Note that our approach accurately preserves fine details like the grapes beside the woman figures and sharp features like the eyes of the elephant which are lost in the left reconstruction.

more distant than the radius plus overlap of the cell c_i are assigned weights close to zero. For illustration see the right image of Figure 2.

After determining the global characteristic function values for the corners of the cubes, we can interpolate them across the cube edges to compute the position of the chosen iso-value. Our Marching Cubes implementation interpolates the resulting global function quadratically.

3 Results

In this section, we present results of our reconstruction algorithm. We compare our method with several state-of-the-art reconstruction techniques. Furthermore, we apply our method to real-world laser scanner data as well as large and complex point cloud data and discuss computation times and memory consumption.

Results of our reconstruction algorithm are shown in Figures 3-5. The reconstructions in Figures 4 and 5 are shown in flat shading to illustrate faceting. The mesh in Figure 3 is rendered in Phong shading to bring out small details on the surface as single triangles are not visible. Tables 1 and 2 summarize details for the presented reconstructions.

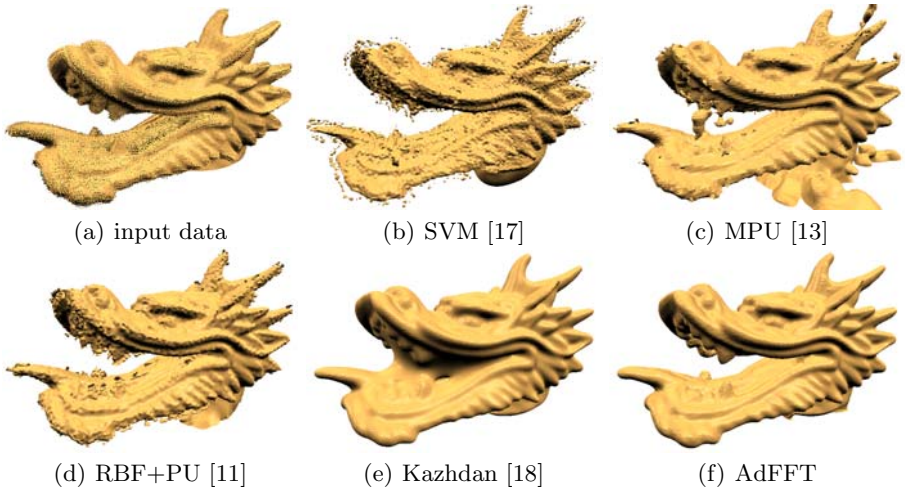
Figure 4 shows a comparison of recent state-of-the-art surface reconstruction techniques with our approach. As input data we choose the head of the original Dragon range scans from the Stanford 3D Scanning Repository. We compare our approach with the recently proposed learning-based reconstruction tech-

Table 1. Timings and memory consumption for the reconstructions shown in Figure 4. All results were computed on a 2.66 GHz Pentium 4 with 1.5 GB of RAM.

method	recon.	polyg.	memory	user-defined error (res.)
SVM	41.86s	315s	304M	$1 \cdot 10^{-4}$
MPU	77s		101M	$5 \cdot 10^{-3}$
RBF+PU	153s	34s	98M	$1 \cdot 10^{-5}$
Kazhdan	8.87s	1.4s	179M	(256^3)
Kazhdan	102s	9.2s	1.1G	(512^3)
AdFFT	41.63s	63s	119M	$1.7 \cdot 10^{-3}$ (256^3)
AdFFT	188s	298s	462M	$1.2 \cdot 10^{-3}$ (512^3)

Table 2. Reconstruction information for the models presented in this paper and computed using our method. The character N denotes the number of input samples and M the number of used patches. The results were computed on a 2.66 GHz Pentium 4 with 1.5 GB of RAM (only the Statuette was computed on a 2.4 GHz AMD Opteron with 3 GB of RAM).

model	N	recon.	polyg.	mem.	error	res.	M
Thai Statuette	5M	406s	675s	2.2G	$2.4 \cdot 10^{-4}$	1024^3	2260
Dragon head scans	485K	188s	298s	462M	$1.2 \cdot 10^{-3}$	512^3	874
Dragon head scans	485K	41.63s	63s	119M	$1.7 \cdot 10^{-3}$	256^3	626
Armadillo scans	2.4M	82s	55s	273M	$1.1 \cdot 10^{-3}$	256^3	565



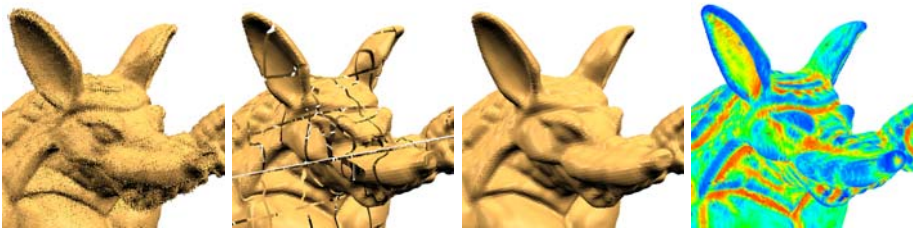
(a) input data (b) SVM [17] (c) MPU [13]

(d) RBF+PU [11] (e) Kazhdan [18] (f) AdFFT

Fig. 4. Comparison of our reconstruction approach (f) with other state-of-the-art techniques illustrated on the Dragon head composed of registered range scans from the Stanford 3D Scanning Repository. Notice that our technique is more robust on noisy data than previous approaches (b)-(d) and generates a more faithful reconstruction in highly bended regions than the global Kazhdan method (e). For a fair comparison no scanning confidence values were used to create the reconstructions (c) and (d). Corresponding timings are reported in Table 1.

nique using Support Vector Machines (SVMs) [17], MPU [13], RBF+PU [11] and Kazhdan's global FFT-based method [18]. For a fair comparison of our method with MPU and RBF+PU, we take no scanning confidences into account while applying them. The figure shows that without confidence measures SVM, MPU and RBF+PU create noisy reconstructions of the Dragon head scans and produce additional zero-level sets around the surface. Due to the global nature of Kazhdan's approach the algorithm is robust reconstructing noisy real-world data but has problems capturing regions where the surface bends close to itself. By localizing the global approach using adaptive decomposition and partition of unity blending, our algorithm is capable of accurately reconstructing those regions while retaining the robustness of the global approach. Note that methods like MPU and RBF+PU have a better performance on real-world data when they utilize scanning confidence values, while our approach is robust on noisy data without using additional scanning information.

In Figure 3 we compare our technique with the global Kazhdan approach according to reconstruction accuracy. We reconstructed the highly detailed Thai Statuette with both techniques using their maximal possible resolutions. Due to the lower memory consumption of our method (see Table 1) we are able to reach higher reconstruction resolutions. This allows us to faithfully represent fine details, for instance, on the trunk of the elephant and the necklace of the woman model (right image) which are lost using the global approach (left image). Note, that although the Thai Statuette was decomposed into 2260 and the Armadillo



(a) input data (b) approximated patches (c) AdFFT (d) mean curvature

Fig. 5. The original Armadillo dataset composed of 114 registered range scans from the Stanford Scanning Repository (a) and a reconstruction from the noisy data using our method (c). Image (b) illustrates the patches without overlap used to reconstruct the Armadillo model. Figure (d) shows the mean curvature of our reconstruction (red represents negative and blue positive mean curvature values). Although the final reconstruction is composed of many patches, the mean curvature plot does not show blending artifacts.

model in Figure 5 into 565 patches and reconstructed with a lower resolution (see Table 2), the reconstructions show no blending artifacts. For all models in this paper, we used an overlap of 5 cells to blend adjacent reconstructions.

Figure 5 analyzes the effect of blending on the results of our surface reconstruction algorithm in more detail. For this, we computed a reconstruction of the original Armadillo range scans using our method. Figure 5(b) illustrates the patches without overlap used to create the integrated reconstruction shown in (c). Figure 5(d) shows a mean curvature plot of (c). The results indicate that no visible blending artifacts close to the cell boundaries are introduced by our approach.

4 Conclusion and Future Work

In this paper, we localized Kazhdan's global FFT-based reconstruction algorithm by using adaptive subdivision and partition of unity blending. We showed that our method preserves the resilience of the global approach and is more robust against noise than previous state-of-the-art reconstruction techniques. Furthermore, it is capable to reconstruct noisy real-world data and allows a precise reconstruction of highly bended regions of the input data which are connected by the global approach. We demonstrated the lower memory consumption of our technique, which allows higher reconstruction resolutions and enables to capture fine and small details in large and complex point clouds.

In the future, we want to consider scanning confidence values in the reconstruction process to further increase the robustness of our approach on real-world data. Furthermore, we plan to combine our technique with the Dual Contouring algorithm [20] allowing for an adaptive polygonization of our reconstructions.

Acknowledgements

We would like to thank Michael Kazhdan and Yutaka Ohtake for making their surface reconstruction software available. The Dragon and Armadillo dataset are courtesy of the Stanford 3D scanning repository. The Thai Statuette is courtesy of XYZ RGB. This work was supported in part by the European FP6 NoE grant 506766 (AIM@SHAPE).

References

1. Amenta, N., Bern, M., Kamvysselis, M.: A new Voronoi-based surface reconstruction algorithm. In: Proceedings of ACM SIGGRAPH 1998. (1998) 415–421
2. Amenta, N., Choi, S., Kolluri, R.: The power crust. In: Proceedings of 6th ACM Symposium on Solid Modeling. (2001) 249–260
3. Boissonnat, J.D.: Geometric structures for three-dimensional shape representation. ACM Transactions on Graphics **3**(4) (1984) 266–286
4. Dey, T.K., Goswami, S.: Tight Cocone: A water-tight surface reconstructor. In: Proc. 8th ACM Sympos. Solid Modeling Applications. (2003) 127–134
5. Dey, T.K., Goswami, S.: Provable surface reconstruction from noisy samples. In: Proc. 20th ACM Sympos. Comput. Geom. (2004)
6. Mederos, B., Amenta, N., Velho, L., de Figueiredo, L.H.: Surface reconstruction from noisy point clouds. In: Eurographics Symposium on Geometry Processing 2005. (2005) 53–62
7. Schall, O., Belyaev, A.G., Seidel, H.P.: Robust filtering of noisy scattered point data. In: Eurographics Symposium on Point-Based Graphics 2005. (2005) 71–77
8. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W.: Surface reconstruction from unorganized points. In: Proceedings of ACM SIGGRAPH 1992. (1992) 71–78
9. Carr, J.C., Beatson, R.K., Cherrie, J.B., Mitchell, T.J., Fright, W.R., McCallum, B.C., Evans, T.R.: Reconstruction and representation of 3D objects with radial basis functions. In: Proceedings of ACM SIGGRAPH 2001. (2001) 67–76
10. Dinh, H.Q., Turk, G., Slabaugh, G.: Reconstructing surfaces using anisotropic basis functions. In: International Conference on Computer Vision (ICCV) 2001. Volume 2. (2001) 606–613
11. Ohtake, Y., Belyaev, A.G., Seidel, H.P.: 3D scattered data approximation with adaptive compactly supported radial basis functions. In: Shape Modeling International 2004, Genova, Italy (2004)
12. Turk, G., Dinh, H.Q., O'Brien, J., Yngve, G.: Implicit surfaces that interpolate. In: Shape Modelling International 2001, Genova, Italy (2001) 62–71
13. Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., Seidel, H.P.: Multi-level partition of unity implicits. ACM Transactions on Graphics **22**(3) (2003) 463–470 Proceedings of SIGGRAPH 2003.
14. Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Silva, C.T.: Point set surfaces. IEEE Visualization 2001 (2001) 21–28
15. Amenta, N., Kil, Y.J.: Defining point-set surfaces. ACM Transactions on Graphics **23**(3) (2004) 264–270 Proceedings of SIGGRAPH 2004.
16. Fleishman, S., Cohen-Or, D., Silva, C.T.: Robust moving least-squares fitting with sharp features. ACM Transactions on Graphics **24**(3) (2005) 544–552 Proceedings of ACM SIGGRAPH 2005.

17. Steinke, F., Schölkopf, B., Blanz, V.: Support vector machines for 3D shape processing. Computer Graphics Forum **24**(3) (2005) 285–294 Proceedings of EUROGRAPHICS 2005.
18. Kazhdan, M.: Reconstruction of solid models from oriented point sets. In: Eurographics Symposium on Geometry Processing 2005. (2005) 73–82
19. Lorensen, W.E., Cline, H.E.: Marching Cubes: a high resolution 3D surface construction algorithm. Computer Graphics **21**(3) (1987) 163–169 Proceedings of ACM SIGGRAPH '87.
20. Ju, T., Losasso, F., Schaefer, S., Warren, J.: Dual contouring of hermite data. ACM Transactions on Graphics **21**(3) (2002) 339–346 Proceedings of ACM SIGGRAPH 2002.

Least–Squares Approximation by Pythagorean Hodograph Spline Curves Via an Evolution Process

M. Aigner, Z. Šír*, and B. Jüttler

Johannes Kepler University Linz, Austria
{martin.aigner, zbynek.sir, bert.juettler}@jku.at
http://www.ag.jku.at

Abstract. The problem of approximating a given set of data points by splines composed of Pythagorean Hodograph (PH) curves is addressed. In order to solve this highly non-linear problem, we formulate an evolution process within the family of PH spline curves. This process generates a one–parameter family of curves which depends on a time–like parameter t. The best approximant is shown to be a stationary point of this evolution. The evolution process – which is shown to be related to the Gauss–Newton method – is described by a differential equation, which is solved by Euler's method.

1 Introduction

Curves with simple closed form descriptions of their parametric speed and arc–length are useful for various applications, such as NC machining. They greatly facilitate the control of the tool along a curved trajectory with constant (or user–defined) speed. In addition, these curves admit a simple exact representation of their offset curves.

This motivated the investigation of the interesting class of Pythagorean Hodograph (PH) curves, see [Far02] and the references cited therein. This class consists of (piecewise) polynomial curves with a (piecewise) polynomial parametric speed, see Fig. 1 for an example. Various constructions for PH curves were developed. Due to the non–linear nature of PH curves, these are mainly based on local techniques, such as the interpolation of Hermite boundary data [MW97, MFC01, FMJ98, ŠJ05][1].

In many situations, it is more appropriate to use global approximation techniques, such as least–squares fitting, since this generally reduces the data volume and produces a more compact representation. In the case of PH curves, very few global methods are available, dealing with interpolation and least–squares fitting [FST98, FKMS01]. In the latter paper, the authors use non–linear optimization to generate a PH quintic which interpolates two boundary points and approximates additional points, where the parameter values assigned to them are kept constant.

Even for simple curve representations, such as polynomial spline curves, curve fitting is a non–linear problem, due to the influence of the parameterization. Different approaches for dealing with the effects of this non–linearity have been developed [AB01, HL93, RF89, PL03, PLH$^+$05, SKH98, WPL06], such as 'parameter correction' or the use of quasi–Newton methods. Clearly the choice of a good initial solution

* Supported by the Austrian Science Fund (FWF) through project P17387–N12.

[1] Similar techniques for space curves exist also, see [Far02].

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 45–58, 2006.
© Springer-Verlag Berlin Heidelberg 2006

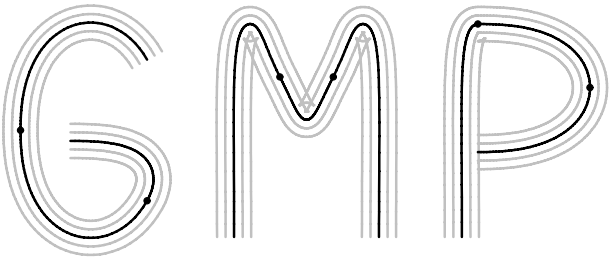


Fig. 1. Examples of piecewise polynomial Pythagorean hodograph curves (black) and their piecewise rational offsets (grey). Each character is composed of three PH quintics.

is of outmost importance for the success of the optimization. Geometrically motivated optimization strategies [PL03, PLH02, PLH$^+$05, WPL06], where the initial solution is replaced by an initial curve and the formulation of the problem uses some geometric insights, may lead to more robust techniques.

Due to the iterative nature of the techniques for non–linear optimization, one may view the intermediate results as a time–dependent curve which tries to adapt itself to the target shape defined by the unorganized point data [PLH02, WPL06]. This is related to the idea of 'active curves' used for image segmentation in Computer Vision [KWT87]. Recently we formulated a general framework for evolution–based fitting of hybrid objects [AJ05].

In this work we generalize this framework and analyze its relation to the Gauss–Newton method. In addition, we apply it to the problem of least–squares approximation by Pythagorean hodograph spline curves.

The remainder of this paper is organized as follows. In the next two sections we recall some basics about PH curves, and we introduce a general framework for abstract curve fitting. Then, this framework will be applied to the special case of Pythagorean hodograph curves, and its relation to Gauss–Newton iteration will be analyzed. Finally we conclude the paper.

2 Pythagorean Hodograph Curves

The *hodograph* of a planar polynomial curve $\mathbf{c}(u) = [x(u), y(u)]^\top$ of degree n is the vector $\mathbf{h}(u) = [x'(u), y'(u)]^\top$ of degree $n-1$, where $'$ denotes the first derivative. Recall that a polynomial curve is called *Pythagorean Hodograph (PH)* if the length of its tangent vector is a (piecewise) polynomial of the parameter u. More precisely, $\mathbf{c}(u) = [x(u), y(u)]^\top$ is called *planar PH curve* if there exists a polynomial $\sigma(u)$ such that

$$x'(u)^2 + y'(u)^2 = \sigma^2(u). \tag{1}$$

Three polynomials x', y' and σ satisfy equation (1)[2] if and only if there exist three polynomials α, β, ω such that

[2] They are said to form a Pythagorean triplets in the ring of polynomials.

$$x' = \omega(\alpha^2 - \beta^2), \quad y' = \omega(2\alpha\beta), \quad \sigma = \omega(\alpha^2 + \beta^2), \tag{2}$$

see [Kub72]. As a major advantage of PH curves, compared to 'ordinary' polynomial curves, they possess a (piecewise) polynomial arc length function

$$s(u) = \int_{u_0}^{u} |\sigma(v)| \, \mathrm{d}v \tag{3}$$

and (piecewise) rational offset curves (parallel curves)

$$\mathbf{o}_d(u) = \mathbf{c}(u) + \frac{d}{|\sigma(u)|}[y'(u), -x'(u)]^\top, \tag{4}$$

where d is the (oriented) offset distance.

Throughout the remainder of this paper we will assume that $\omega = 1$, restricting ourselves to curves with hodographs of the form

$$x'(u) = \alpha^2(u) - \beta^2(u), \quad y'(u) = 2\alpha(u)\beta(u). \tag{5}$$

As to be justified by Proposition 1, these PH curves will be called *regular PH curves*. They form a subset of all PH curves distinguished by the property that $\gcd(x'(u), y'(u))$ is a square of a polynomial.[3]

Regular PH curves can be constructed as follows: First we choose two polynomials $[\alpha(u), \beta(u)]^\top$ which define the so–called *preimage* curve. We generate the hodograph using (5) and integrate the two components. This gives the parametric representation of the PH curve.

Since two curves $\mathbf{c}(u)$, $\tilde{\mathbf{c}}(u)$ have the same hodograph if and only if they differ only by translation, a regular planar PH curve $\mathbf{p}(u)$ is fully determined by the preimage $[\alpha(u), \beta(u)]^\top$ and by the location of its starting point $\mathbf{c}(0)$ (which is specified by choosing the integration constant).

While the 'ordinary' PH curves may have cusps (namely for all parameter values of u which are roots of ω), regular PH curves are always tangent continuous.

Proposition 1. *Any regular (i.e., generated using* (5)*) Pythagorean hodograph curve, where the two polynomials $\alpha(u)$ and $\beta(u)$ defining the preimage are not both identically to zero, $(\alpha(u), \beta(u)) \not\equiv (0, 0)$, has a smooth field of unit tangent vectors for all values $u \in \mathbb{R}$ of the curve parameter. Moreover its parametric speed and arc-length are polynomial functions, and its offsets are rational curves.*

Proof. Clearly, $\sigma(u) = \alpha(u)^2 + \beta(u)^2$ is a non-negative polynomial representing the speed function of $\mathbf{c}(u)$. The absolute value can be omitted in (3) and the arc-length function is a polynomial defined on $\mathbb{R}$. Consider

$$\mathbf{q}(u) = \left[\frac{\alpha(u)^2 - \beta(u)^2}{\alpha(u)^2 + \beta(u)^2}, \frac{2\alpha(u)\beta(u)}{\alpha(u)^2 + \beta(u)^2}\right]^\top. \tag{6}$$

Except for the roots of $\alpha(u)^2 + \beta(u)^2$, the vector $\mathbf{q}(u)$ is a unit vector tangent to $\mathbf{c}$ at $\mathbf{c}(u)$ and it has the same orientation as $[x'(u), y'(u)]^\top$. Moreover, any root u_0 of

[3] This includes the generic case $\gcd(x'(u), y'(u)) = 1$.

$\alpha(u)^2 + \beta(u)^2$ has an even multiplicity $2k$, since $\alpha(u)^2 + \beta(u)^2$ is non-negative. Also $(u - u_0)^k$ must divide both $\alpha(u)$ and $\beta(u)$ and therefore $(u - u_0)^{2k}$ divides the numerators and the denominator of (6). After eliminating all common factors of numerators and denominators, we can therefore extend $\mathbf{q}(u)$ smoothly to $u \in \mathbb{R}$ and we obtain a smooth unit vector field along $\mathbf{c}(u)$.

Finally we note that the offset formula (4) simplifies to

$$\mathbf{o}_d(u) = \mathbf{c}(u) + d\mathbf{q}(u)^\perp, \tag{7}$$

and it defines a rational curve on $\mathbb{R}$. □

Remark 1. The observation formulated in Proposition 1 can be seen as another advantage of PH curves, compared to the more general class of standard polynomial (Bézier) curves. The more general curves are not necessarily tangent continuous, since cusps may be present.

3 An Abstract Framework for Curve Fitting Via Evolution

We describe a general framework for the evolution-based approximation of a given data set by a curve. Later we will apply it to the special case of PH spline curves.

3.1 Families of Parametric Curves and Evolution of Shape Parameters

We consider a parameterized family of planar parametric curves $(\mathbf{s}, u) \mapsto \mathbf{c}_\mathbf{s}(u)$. Two different kinds of parameters appear in the representation of the curve; the curve parameter u and a vector of shape parameters $\mathbf{s} = (s_1, \ldots, s_n)$.

For instance, one may consider a family of spline curves, where the shape parameters are both the control points and the knots. Later, in the case of PH spline curves, the shape parameters will be the control points defining the preimage curve and the integration constants.

We assume that the curve parameter varies within a fixed interval $I = [a, b]$ (the parameter domain of the curve), and that the vector of shape parameters $\mathbf{s}$ is contained in some domain $\Omega \subset \mathbb{R}^n$. For all $(\mathbf{s}, u) \in \Omega \times I$ the curve $\mathbf{c}_\mathbf{s}(u)$ shall depend continuously on the parameters. We assume that the curve has a well–defined normal vector at all points. Due to Proposition 1, this assumption is satisfied in the case of regular PH curves.

Among the curves of this family, we identify a curve that approximates a given set of (unordered) data points $\{\mathbf{p}_j\}_{j=1..N}$ in the least–squares sense. More precisely, we are looking for the vector of shape parameters that defines this curve.

We let the shape parameters $\mathbf{s}$ depend smoothly on an evolution parameter t, $\mathbf{s}(t) = (s_1(t), \ldots, s_n(t))$. The parameter t can be identified with the time. Starting with certain initial values, these parameters are modified continuously in time such that a given initial curve moves closer to the data points.

This movement will be governed by a system of differential equations of the form $\dot{\mathbf{s}} = F(\mathbf{s})$. By numerically solving this system and (approximately) computing the limit $\lim_{t\to\infty} \mathbf{s}(t)$, we obtain a curve $\mathbf{c}_\mathbf{s}(u)$ which has minimal distance from the data points.

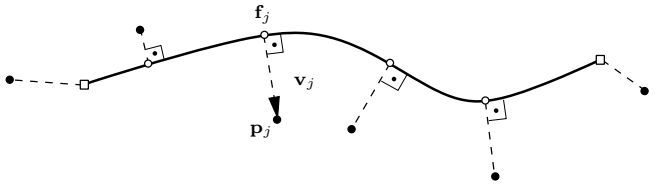


Fig. 2. Closest points and derived (normal) velocities

During the evolution of a curve $\mathbf{c}_{\mathbf{s}(t)}(u)$, each point travels with the velocity

$$\mathbf{v}_{\mathbf{s}(t)}(u) = \dot{\mathbf{c}}_{\mathbf{s}(t)}(u) = \sum_{i=1}^{n} \frac{\partial \mathbf{c}_{\mathbf{s}}(u)}{\partial s_i}\, \dot{s}_i(t). \tag{8}$$

The dot denotes the derivative with respect to the time variable t. Since the tangential component of the velocity (8) can be seen as a reparameterization of the curve, we consider mainly the normal velocity

$$v_{\mathbf{s}(t)}(u) = \mathbf{v}_{\mathbf{s}(t)}(u)^{\top} \mathbf{n}_{\mathbf{s}(t)}(u) = \sum_{i=1}^{n} \left(\frac{\partial \mathbf{c}_{\mathbf{s}}(u)}{\partial s_i}\, \dot{s}_i(t) \right)^{\top} \mathbf{n}_{\mathbf{s}(t)}(u), \tag{9}$$

where $\mathbf{n}_{\mathbf{s}(t)}(u)$ denotes the unit normal of the curve in the point $\mathbf{c}_{\mathbf{s}}(u)$. Note that the normal velocity depends linearly on the derivatives $\dot{s}_i(t)$ of the shape parameters.

3.2 Evolution for Approximation

We will derive the evolution equation by specifying suitable velocities for some points of the curve. We assume that a set of data points $\{\mathbf{p}_j\}_{j=1,\ldots,N}$ is given. For each point, we consider the associated closest point $\mathbf{f}_j = \mathbf{c}(u_j)$ of the curve,

$$u_j = \arg \min_{u \in [a,b]} \|\mathbf{p}_j - \mathbf{c}_{\mathbf{s}}(u)\|. \tag{10}$$

During the evolution, these points are expected to travel towards their associated data points. Consequently, the normal velocity $v(u_j)$ of a curve point $\mathbf{c}_{\mathbf{s}}(u_j) = \mathbf{f}_j$ should be

$$d_j = (\mathbf{p}_j - \mathbf{f}_j)^{\top} \mathbf{n}_{\mathbf{s}(t)}(u_j). \tag{11}$$

If a closest point is one of the two boundary points ($u_j \in \{a, b\}$), then we consider the velocity (8), see Fig. 3.2.

Following (8) and (9) we can compute for each point $\mathbf{f}_j$ the velocity or normal velocity on the one hand and the expected velocity on the other hand. In general, the number of data points exceeds the degrees of freedom of the curve to be fit to these data ($N \gg n$). Hence the conditions for the velocities in the foot points cannot be fulfilled exactly. We choose the time derivatives of the shape parameters such that the conditions for the velocities are satisfied in the least–squares sense,

$$\dot{\mathbf{s}} = \arg\min_{\dot{\mathbf{s}}} = \omega_{\perp} \sum_{\substack{j=1 \\ u_j \notin \{a,b\}}}^{N} (v(u_j) - d_j)^2 + \omega_{\mathbf{v}} \sum_{\substack{j=1 \\ u_j \in \{a,b\}}}^{N} \|\mathbf{v}(u_j) - (\mathbf{p}_j - \mathbf{f}_j)\|^2 + \omega_R R, \tag{12}$$

see (8), (9), (10) and (11). The non–negative weights $\omega_\perp \neq 0, \omega_{\mathbf{v}}$ and ω_R are used to control the influence of the three different terms. In order to ensure that a unique minimizer for the least–squares problem (12) exists, a regularization term R is added in (12). As a possibilty one may use Tikhonov regularization, where $R = \|\dot{\mathbf{s}}\|^2$.

As a necessary condition for a minimum, the derivatives of the right–hand side in (12) with respect to the $\dot{s}_i$ vanish. Since these factors enter linearly in (8) and (9), the optimality condition yields a system of linear equations

$$M(\mathbf{s})\dot{\mathbf{s}} = r(\mathbf{s}). \tag{13}$$

In general, this ODE cannot be solved exactly. Nevertheless, the vector $\dot{\mathbf{s}}$ can easily be computed for each given vector $\mathbf{s}$ by solving the linear system,

$$\dot{\mathbf{s}} = F(\mathbf{s}) = M^{-1}(\mathbf{s})\, r(\mathbf{s}). \tag{14}$$

Using explicit Euler-steps $\mathbf{s}_i \to \mathbf{s}_i + h\dot{\mathbf{s}}_i$, with a suitable step-size h, one can trace the evolving curves. This method for the numerical solution of the ODE corresponds to a discretization of the evolution in time.

Remark 2. In order to reduce the computational effort needed for computing the closest points on the curve (especially when the curve is still relatively far from the data), one can proceed as follows. As a preprocessing step, the distance field of the target shape is computed. This can be done efficiently using the graphics hardware, see [HKL$^+$99]. Starting with some equally spaced sensor points on some initial shape, the velocities (or normal velocities in the case of vertex points) can be defined with the help of the distance field. Finally, the sensor points are replaced by the closest points, if the distance to the data points drops below a certain threshold. Similarly, one may use velocities derived from other data, such as images, in order to deal with applications such as image segmentation.

3.3 Stationary Points of the Evolution

The solutions of the least–squares problem

$$\arg\min_{\mathbf{s}} \sum_{j=1}^{N} \min_{u_j \in [a,b]} \|\mathbf{p}_j - \mathbf{c}_{\mathbf{s}}(u_j)\|^2 \tag{15}$$

are closely related to the evolution process defined by (14). In order to establish this connection, we need some technical assumptions. We assume, that the curve is non–singular ($\mathbf{c}'_{\mathbf{s}}(u_j) \neq \mathbf{0}$) at the closest points $\mathbf{c}_{\mathbf{s}}(u_j)$ to the data points. In addition, we exclude certain singular cases, e.g., when the number of degrees of freedom exceeds the number of data points when the data points lie in some degenerate position. This is made precise in the following definition.

Definition 1. *For a given curve $\mathbf{c}_{\mathbf{s}}(u)$, consider a set $U = \{u_j\}_{j=1..N}$ of parameter values such that $\mathbf{c}'_{\mathbf{s}}(u) \neq \mathbf{0}$ and $\{a, b\} \cap U = \emptyset$. The corresponding unit normal vectors are $\mathbf{n}_j = \mathbf{n}_{\mathbf{s}}(u_j)$. The set of parameters U is said to be* **regular** *if the $N \times n$ matrix*

$$A_{j,k} = \mathbf{n}_j^\top \frac{\partial \mathbf{c}_{\mathbf{s}}(u_j)}{\partial s_k} \tag{16}$$

has maximal rank.

Lemma 1. *In a regular case and if all closest points are neither singular nor boundary points, then any solution of the usual least–squares fitting* (15) *of a curve* $\mathbf{c_s}(u)$ *is a stationary point of the differential equation derived from the evolution process.*

Proof. As a necessary condition, the first derivatives of F with respect to the curve parameters $\{u_j\}_{j=1..N}$ and the shape parameters $\{s_i\}_{i=1..n}$ vanish, where F is the sum of squared errors in (15),

$$\frac{\partial F}{\partial u_j} = 2\left(\mathbf{p}_j - \mathbf{c_s}(u_j)\right)^\top \frac{\partial \mathbf{c_s}(u_j)}{\partial u_j} = 0, \tag{17}$$

and

$$\frac{\partial F}{\partial s_i} = 2\sum_{j=1}^{N}\left(\mathbf{p}_j - \mathbf{c_s}(u_j)\right)^\top \frac{\partial \mathbf{c_s}(u_j)}{\partial s_i} = 0. \tag{18}$$

On the other hand, the ODE defining the curve evolution is found by computing the first derivatives of

$$\sum_{j=1}^{N}\Big(\left(\mathbf{v}_j - (\mathbf{p}_j - \mathbf{f}_j)\right)^\top \mathbf{n}_j\Big)^2$$

with respect to the derivatives of the shape parameters $\dot{s}_k$. This yields

$$2\sum_{j=1}^{N}\left[\left(\mathbf{v}_j - (\mathbf{p}_j - \mathbf{f}_j)\right)^\top \mathbf{n}_j\mathbf{n}_j^\top \frac{\partial \mathbf{c_s}(u_j)}{\partial s_k}\right] = 0 \qquad \forall k \tag{19}$$

Due to (17), the error vectors $\mathbf{p}_j - \mathbf{f}_j$ are perpendicular to the tangent vectors, hence $(\mathbf{p}_j - \mathbf{f}_j)^\top \mathbf{n}_j\mathbf{n}_j^\top = (\mathbf{p}_j - \mathbf{f}_j)^\top$. Taking (18) into account, (19) simplifies to

$$\sum_{j=1}^{N}\left[\mathbf{v}_j^\top \mathbf{n}_j\mathbf{n}_j^\top \frac{\partial \mathbf{c_s}(u_j)}{\partial s_k}\right] = \sum_{j=1}^{N}\left[\left(\sum_{i=0}^{n}\frac{\partial \mathbf{c_s}(u_j)}{\partial s_i}\dot{s}_i\right)^\top \mathbf{n}_j\mathbf{n}_j^\top \frac{\partial \mathbf{c_s}(u_j)}{\partial s_k}\right] = 0 \qquad \forall k.$$

Rewriting this equation we get

$$\sum_{i=0}^{n}\sum_{j=1}^{N}\left[\left(\frac{\partial \mathbf{c_s}(u_j)}{\partial s_i}\right)^\top \mathbf{n}_j\mathbf{n}_j^\top \frac{\partial \mathbf{c_s}(u_j)}{\partial s_k}\right]\dot{s}_i = 0 \qquad \forall k,$$

or in matrix notation $A^\top A\dot{\mathbf{s}} = 0$, where the components of A are defined as in (16). This system has only the trivial solution if the matrix $A^\top A$ is regular which corresponds to rank$(A)= n + 1$. In a regular case this condition holds. □

We will continue this discussion in Section 6, where we prove that the evolution is equivalent to a Gauss–Newton step for the least–squares problem (15).

4 Evolution of PH Splines

In this section we apply the general framework of the previous section to the case of PH splines. More precisely, we will represent the preimage $[\alpha(u), \beta(u)]$ as an open integral B-spline curve [HL93, p. 176]. Let

$$(u_0 = u_1 = \ldots = u_{k-1}, u_k, u_{k+1}, \ldots, u_m, u_{m+1} = u_{m+2} = \ldots = u_{m+k}) \quad (20)$$

be a given knot vector and $N_{i,k}(u)$, $(i = 0, \ldots, m)$ the associated B-spline functions of order k. Then $N_{i,k}(u)$ form a basis of the linear space of piecewise polynomials of degree $k-1$ on the interval $[u_{k-1}, u_{m+1}]$ which are C^{k-2} at the points $\{u_i, i = k, \ldots, m\}$. We choose the components $\alpha(u), \beta(u)$ of the preimage from this space of functions,

$$\alpha(u) = \sum_{i=0}^{m} \alpha_i N_{i,k}(u) \quad \text{and} \quad \beta(u) = \sum_{i=0}^{m} \beta_i N_{i,k}(u). \quad (21)$$

The resulting PH spline is obtained as

$$\mathbf{c}(u) = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \int_{u_{k-1}}^{u} \begin{bmatrix} \alpha^2(\tilde{u}) - \beta^2(\tilde{u}) \\ 2\alpha(\tilde{u})\beta(\tilde{u}) \end{bmatrix} \mathrm{d}\tilde{u} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \sum_{i=0}^{m} \sum_{j=0}^{m} \begin{bmatrix} \alpha_i\alpha_j - \beta_i\beta_j \\ 2\alpha_i\beta_j \end{bmatrix} K_{i,j}(u)$$

where the piecewise polynomials $K_{i,j}(u)$ of degree $2k-1$ are defined as

$$K_{i,j}(u) := \int_{u_{k-1}}^{u} N_{i,k}(\tilde{u}) N_{j,k}(\tilde{u}) \mathrm{d}\tilde{u}. \quad (22)$$

As shape parameters – in the sense of the previous section – we can consider the spline end point coordinates x_0, y_0, the spline coefficients α_i, β_i and even the knots u_i. In our implementation we have kept the knot vector fixed and considered only an evolution with respect to the following $n = 2m + 4$ shape parameters

$$\mathbf{s} = \{x_0, y_0, \alpha_0, \ldots, \alpha_m, \beta_0, \ldots, \beta_m\}. \quad (23)$$

We compute the quantities occuring in (12). The partial derivatives of $\mathbf{c}(u)$ with respect to the shape parameters are

$$\frac{\partial \mathbf{c}(u)}{\mathrm{d}x_0} = [1, 0]^\top, \qquad \frac{\partial \mathbf{c}(u)}{\mathrm{d}y_0} = [0, 1]^\top,$$

$$\frac{\partial \mathbf{c}(u)}{\mathrm{d}\alpha_i} = 2 \sum_{j=0}^{m} [\alpha_j, \beta_j]^\top K_{i,j}(u) \quad \text{and} \quad \frac{\partial \mathbf{c}(u)}{\mathrm{d}\beta_i} = 2 \sum_{j=0}^{m} [-\beta_j, \alpha_j]^\top K_{i,j}(u).$$

The velocity (8) of any curve point $\mathbf{c}(u)$ equals

$$\mathbf{v}_{\mathbf{s}(t)}(u) = [\dot{x}_0, \dot{y}_0]^\top + 2 \sum_{i=0}^{m} \sum_{j=0}^{m} [\alpha_j\dot{\alpha}_i - \beta_j\dot{\beta}_i, \beta_j\dot{\alpha}_i + \alpha_j\dot{\beta}_i]^\top K_{i,j}(u), \quad (24)$$

which is linear in the derivatives $\dot{x}_0, \dot{y}_0, \dot{\alpha}_i, \dot{\beta}_i$ of shape parameters. The unit normals are

$$\mathbf{n}_{\mathbf{s}(t)}(u) = \frac{\mathbf{c}'(u)^{\perp}}{\alpha(u)^2 + \beta(u)^2} = \frac{\displaystyle\sum_{i=0}^{m}\sum_{j=0}^{m}\begin{bmatrix} 2\alpha_i\beta_j \\ \beta_i\beta_j - \alpha_i\alpha_j \end{bmatrix} N_{i,k}(u)N_{j,k}(u)}{\displaystyle\sum_{i=0}^{m}\sum_{j=0}^{m}(\alpha_i\alpha_j + \beta_i\beta_j)N_{i,k}(u)N_{j,k}(u)} \tag{25}$$

which makes it simple to evaluate the normal speed (9).

In each time step of the discretized evolution, we need to find the closest point. For instance, this can be formulated as a polynomial root–finding problem, since

$$\mathbf{c}'(u)^{\top}(\mathbf{c}(u) - \mathbf{p}_i) = 0 \tag{26}$$

is piecewise polynomial in u. For each $\mathbf{p}_i$ we can find all solutions of (26) and compare the distance of the closest one with the distance of $\mathbf{p}_i$ to the end-points.[4] Due to Proposition 1, the normal direction is well defined at all inner points of the curve.

The length of the PH spline has the particularly simple expression

$$L_{\mathbf{s}(t)} = \int_{u_{k-1}}^{u_{m+1}} \left(\alpha(u)^2 + \beta(u)^2\right) \mathrm{d}u = \sum_{i=0}^{m}\sum_{j=0}^{m}(\alpha_i\alpha_j + \beta_i\beta_j)K_{i,j}(u_{m+1}). \tag{27}$$

Clearly, the $K_{i,j}(u_{m+1})$ are constant numbers which have to be computed only once, and $L_{\mathbf{s}(t)}$ is a quadratic function in the shape parameters with partial derivatives

$$\frac{\partial L_{\mathbf{s}(t)}}{\mathrm{d}\alpha_i} = 2\sum_{j=0}^{m}\alpha_j K_{i,j}(u) \quad \text{and} \quad \frac{\partial L_{\mathbf{s}(t)}}{\mathrm{d}\beta_i} = 2\sum_{j=0}^{m}\beta_j K_{i,j}(u). \tag{28}$$

The simple expression of the length of the PH spline inspired us to use the regularization term

$$R := \left(L_e - L_{\mathbf{s}(t)} - \dot{L}_{\mathbf{s}(t)}\right)^2, \tag{29}$$

which forces the curve length $L_{\mathbf{s}(t)}$ to converge to some estimated constant value L_e.

5 Examples of PH Splines Evolution

We apply the procedure described in Section 4 to two examples. In both cases we use piecewise PH cubics defined by piecewise linear C^0 preimages. The resulting PH spline consists of polynomial pieces of degree 3 joined with C^1 continuity.

In order to obtain an evolution which converges to a good PH approximation of the input points, it is necessary to choose a suitable initial position of the evolving curve. For instance, one might assume that the user sketches a polynomial spline curve which is then converted to PH form via Hermite interpolation [MW97]. Alternatively, certain (semi–) automatic estimation procedures can be developed.

[4] Note that the frequent closest point computation can be avoided during the first part of the evolution, when the curve is still relatively far from the data, see Remark 2.

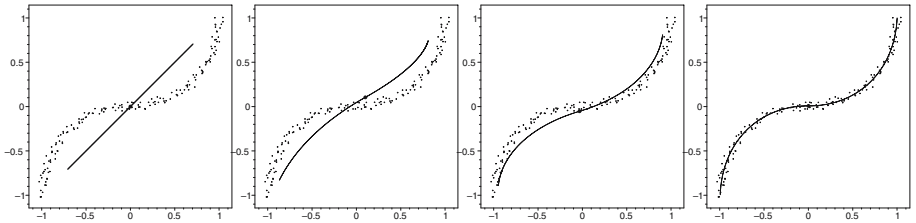


Fig. 3. Approximation of noisy data

In the following examples we have used a different approach. We start with a PH spline which is in a rather poor initial position but, it consists only of a small number of cubic segments. Therefore, only few shape parameters s_i are involved, and the danger of an evolution towards a local minimum is reduced. After several evolution steps, we raise the number of spline segments (via knot insertion) without modifying the shape of the curve $\mathbf{c}(u)$. Then we continue the evolution until some stable situation is reached. This procedure can be repeated until the maximum error is sufficiently low.

Example 1. In this example (see Figure 3), the input points were obtained from two circular arcs with radius 1. We added additional random noise to sampled points ranging from -0.05 to 0.05 in both x and y point coordinates. We evolved a PH spline composed of two cubic PH segments depending on 8 shape parameters: 3 for each of piecewise linear preimage components u, v and 2 integration constants determining the position of the start point $\mathbf{c}(0)$ of the PH spline. In the initial postion, the spline degenerates into a straight line.

Since the target shape is quite simple, no special adjustment of the evolution control values $\omega_\perp, \omega_\mathbf{v}, \omega_R$ and L_e is necessary. The length of the spline was estimated as $L_e = \pi$ and the regularization term (29) was kept unchanged during the whole evolution. Also, the weights occuring in (12) were all set to 1 and the maximal permitted change of the curve to 0.2 during the whole evolution.[5]

Figure 3 shows the evolution of the spline from its initial position towards a stationary solution, which is reached after 8 steps. The maximum error is then $6.02\ 10^{-2}$ which corresponds to the magnitude of the noise.

Example 2. In this example (see Figure 4) the input points were taken from a more complicated free-form curve. For this reason the evolution had to be controlled in a more sophisticated way. Again, we started with a PH spline composed of two straight line segments. The maximal permitted change was again kept equal to 0.2 through the whole evolution. In order to match the global shape of the curve we started with a small imposed curve length $L_e = 8$ and with weights $\omega_\perp = \omega_\mathbf{v} = \omega_R = 1$. After step 30 the global shape of the curve is already well matched and the actual curve length is already 9.89.

Through steps 31 to 45 we gradually raised L_e length up to 14, the real length being at this moment only slightly greater. At this stage of the evolution it was necessary to

[5] At each step the step-size $h \leq 1$ was estimated so that no point of the curve changes more then 0.2. When the curve is sufficiently close to a stationary point, then $h = 1$.

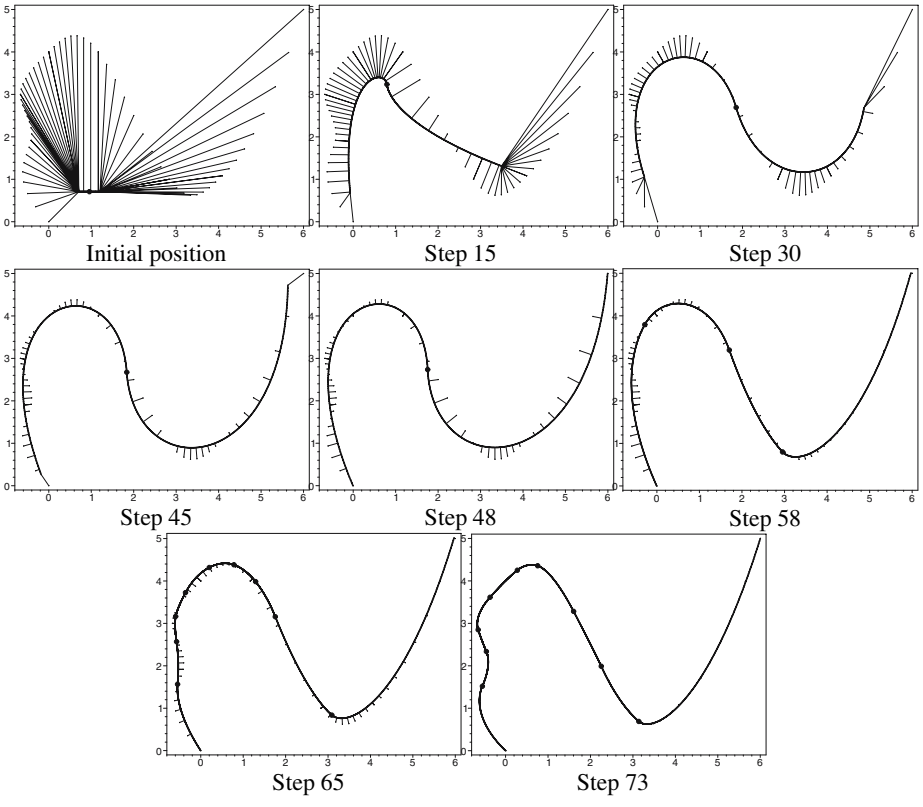


Fig. 4. Approximation of points taken from a spline curve

fix the end points. For this purpose we relaxed the curve length condition by putting the weight w_R equal to 0.1 (while keeping the required length $L_e = 14$) and we set the end-point weight $\omega_{\mathbf{v}} = 100$. After only three steps the end points were fixed (see Step 48). At this moment the length was 15.5 and the maximum error 0.328.

Then we started the knot insertion. For a spline composed of 4 segments we reached a maximum error of 0.227 at step 58. Then we inserted 6 knots in the intervals where the error was large (at the left part of the curve). The non-uniform spline composed of 10 parts converged at a stationary postion in step 73. The length equals 15.3, and the maximum error is $1.63\ 10^{-2}$.

6 Speed of Convergence

We analyze the convergence speed of the PH spline evolution. More precisely, via comparing the evolution method with the Gauss-Newton method we show the quadratic convergence in the zero-residual case.

Lemma 2. *The Euler update of the shape parameters* $\mathbf{s}$ *for the curve evolution with step size* h *is equivalent to a Gauss-Newton with the same step size* h *of the problem*

$$\sum_{j=1}^{N} \|\mathbf{p}_j - \mathbf{c}_\mathbf{s}(u_j)\|^2 \to \min_\mathbf{s} \quad \textit{where} \quad u_j = \arg\min_{u\in[a,b]} \|\mathbf{p}_j - \mathbf{c}_\mathbf{s}(u)\|, \tag{30}$$

provided that $\{a,b\} \cap \{u_j \mid j = 1,\dots,N\} = \emptyset$[6] *and* $\omega_R = 0$.

Proof. Recall that $d_j := (\mathbf{p}_j - \mathbf{c}_\mathbf{s}(u_j))^\top \mathbf{n}_\mathbf{s}(u_j)$, see (11). In order to solve

$$f = \sum_{j=1}^{N} d_j^2 = \sum_{j=1}^{N} \|\mathbf{p}_j - \mathbf{c}_\mathbf{s}(u_j)\|^2 \to \min_\mathbf{s} \quad \text{where} \quad u_j = \arg\min_{u\in[a,b]} \|\mathbf{p}_j - \mathbf{c}_\mathbf{s}(u)\|,$$

one may use a Gauss-Newton iteration. The new iterate $\mathbf{s}^+ = \mathbf{s} + h\Delta\mathbf{s}$ is found by solving

$$\sum_{j=1}^{N} [d_j(\mathbf{s}) + (\nabla d_j(\mathbf{s}))^\top \Delta\mathbf{s}]^2 \to \min_{\Delta\mathbf{s}}. \tag{31}$$

In our case, the components[7] of the gradients ∇d_j are found from

$$2d_j[\nabla d_j]_i = [\nabla(d_j^2)]_i = [\nabla\|\mathbf{p}_j - \mathbf{c}_\mathbf{s}(u_j)\|^2]_i =$$
$$= -2\left(\frac{\partial \mathbf{c}_\mathbf{s}(u_j)}{\partial s_i} + \mathbf{c}'_\mathbf{s}(u_j)\frac{\partial u_j}{\partial s_i}\right)^\top (\mathbf{p}_j - \mathbf{c}_\mathbf{s}(u_j)) = -2\left(\frac{\partial \mathbf{c}_\mathbf{s}(u_j)}{\partial s_i}\right)^\top (\mathbf{p}_j - \mathbf{c}_\mathbf{s}(u_j)),$$

where we exploited the orthogonality of the tangent vectors $\mathbf{c}'_\mathbf{s}(u_j)$ at the closest points and the error vectors $(\mathbf{p}_j - \mathbf{c}_\mathbf{s}(u_j)) = 0$. Hence,

$$[\nabla d_j]_i = -\left(\frac{\partial}{\partial s_i}\mathbf{c}_\mathbf{s}(u_j)\right)^\top \mathbf{n}_\mathbf{s}(u_j), \tag{32}$$

and Gauß-Newton reads as

$$\sum_{j=1}^{N} \left[(\mathbf{p}_j - \mathbf{c}_\mathbf{s}(u_j))^\top \mathbf{n}_\mathbf{s}(u_j) - \sum_{i=1}^{n} \left(\frac{\partial}{\partial s_i}\mathbf{c}_\mathbf{s}(u_j)^\top \mathbf{n}_\mathbf{s}(u_j)\Delta s_i\right)\right]^2 \to \min_{\Delta\mathbf{s}}. \tag{33}$$

Due to (9) and (11), the time derivatives $\dot{s}_i$ obtained from the optimization problem (12), which defines the evolution of the curve, are equal to the Gauss–Newton updates Δs_i obtained from (33)[8]. Hence, for stepsize $h = 1$, the Euler method for the evolution and the Gauss–Newton iteration for (30) are equivalent. □

Gauss–Newton methods exhibit quadratic convergence, provided that the residuum vanishes (i.e., all errors vanish for the final solution). Indeed, it can be seen as a Newton iteration, where the second part of the expansion

$$\nabla^2 f = \sum_{j=1}^{N} \nabla d_j (\nabla d_j)^\top + \sum_{j=1}^{N} d_j \nabla^2 d_j \tag{34}$$

of the Hessian has been omitted. If $d_j = 0$, then this part vanishes.

[6] This technical assumption ensures that none of the closest points appears at the boundary. It could be avoided by considering closed curves instead of open ones.

[7] Here, $[\mathbf{v}]_i$ denotes the i–th component of a vector $\mathbf{v} = (v_1, \dots . v_n)^\top$.

[8] The second and third term in (12) are not present, since no closest points at the curve boundaries were assumed to exist, and $\omega_R = 0$.

Table 1. Approximation errors during the evolution

Step	Error	Step	Error	Step	Error	Step	Error	Step	Error
1	$1.02\ 10^{-1}$	3	$3.48\ 10^{-2}$	5	$5.52\ 10^{-3}$	7	$6.50\ 10^{-9}$	9	$1.10\ 10^{-30}$
2	$6.50\ 10^{-2}$	4	$1.67\ 10^{-2}$	6	$4.95\ 10^{-5}$	8	$1.37\ 10^{-16}$	10	$2.78\ 10^{-60}$

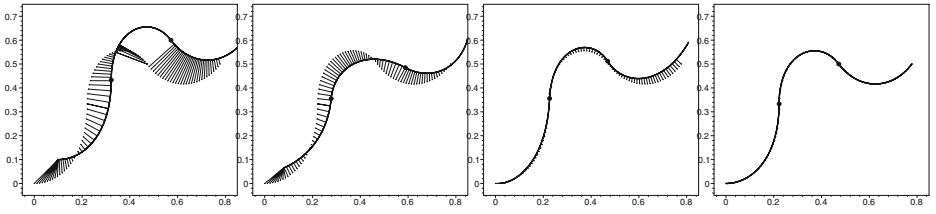


Fig. 5. Approximation of points taken from a PH spline

Example 3. In order to demonstrate the speed of convergence, we consider an example where the input points were taken from a PH spline, see Figure 5. The initial position of the evolution has been obtained by only slightly perturbing the coefficients of the input curve. Through the first five steps of the evolution, the curve evolved to a good approximant - see Table 1 for approximation errors at different evolution steps. For all remaining steps, the approximation error at any step is essentially a square of the error at the previous step, which demonstrates the quadratic convergence of the method.

7 Concluding Remarks

We developed and analyzed an evolution–based fitting procedure for Pythagorean hodograph spline curves. It was shown that this problem can efficiently be dealt with, provided that a good initial solution is available. In this sense, least–squares fitting by PH spline curves is not necessarily more complicated than the same problem for standard curve represention. Indeed, the special properties of PH curves make it even easier to use certain geometrically motivated regularization terms, such as the length of the curve. Future research will be devoted to using the approximation procedure in order to obtain more compact representation of NC tool paths (currently often specified as G-code), where we will cooperate with one of our industrial partners, and on least–squares approximation by surfaces with rational offsets.

References

[AB01] M. Alhanaty and M. Bercovier. Curve and surface fitting and design by optimal control methods. *Computer–Aided Design*, 33:167–182, 2001.

[AJ05] M. Aigner and B. Jüttler. Hybrid curve fitting. FSP Industrial Geometry, Report no. 2 (2005), available at www.ig.jku.at, 2005.

[Far02] R. T. Farouki. Pythagorean-hodograph curves. In *Handbook of computer aided geometric design*, pages 405–427. North-Holland, Amsterdam, 2002.

[FKMS01] R. T. Farouki, B. K. Kuspa, C. Manni, and A. Sestini. Efficient solution of the complex quadratic tridiagonal system for C^2 PH quintic splines. *Numer. Algorithms*, 27(1):35–60, 2001.

[FMJ98] R. T. Farouki, J. Manjunathaiah, and S. Jee. Design of rational cam profiles with pythagorean-hodograph curves. *Mech. and Mach. Theory*, 33(6):669–682, 1998.

[FST98] R. T. Farouki, K. Saitou, and Y.-F. Tsai. Least-squares tool path approximation with Pythagorean-hodograph curves for high-speed CNC machining. In *The mathematics of surfaces, VIII (Birmingham, 1998)*, pages 245–264. Info. Geom., Winchester, 1998.

[HKL+99] Kenneth E. Hoff, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. In *SIGGRAPH '99*, pages 277–286, New York, 1999. ACM Press/Addison-Wesley.

[HL93] J. Hoschek and D. Lasser. *Fundamentals of computer aided geometric design*. A K Peters, Wellesley, MA, 1993.

[Kub72] K.K. Kubota. Pythagorean triplets in unique factorization domains. *Amer. Math. Monthly*, 79:503–505, 1972.

[KWT87] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: active contour models. *Int. J. Comp. Vision*, 1:321–331, 1987.

[MFC01] H. P. Moon, R. T. Farouki, and H. I. Choi. Construction and shape analysis of PH quintic Hermite interpolants. *Comput. Aided Geom. Design*, 18(2):93–115, 2001.

[MW97] D. S. Meek and D. J. Walton. Geometric Hermite interpolation with Tschirnhausen cubics. *J. Comput. Appl. Math.*, 81(2):299–309, 1997.

[PL03] H. Pottmann and S. Leopoldseder. A concept for parametric surface fitting which avoids the parametrization problem. *Comp. Aided Geom. Design*, 20:343–362, 2003.

[PLH02] H. Pottmann, S. Leopoldseder, and M. Hofer. Approximation with active B-spline curves and surfaces. In *Proc. Pacific Graphics*, pages 8–25. IEEE Press, 2002.

[PLH+05] H. Pottmann, S. Leopoldseder, M. Hofer, T. Steiner, and W. Wang. Industrial geometry: recent advances and appl. in CAD. *Comp.-Aided Design*, 37:751–766, 2005.

[RF89] D. Rogers and N. Fog. Constrained B-spline curve and surface fitting. *Computer–Aided Design*, 21:641–648, 1989.

[ŠJ05] Z. Šír and B. Jüttler. Constructing acceleration continuous tool paths using Pythagorean hodograph curves. *Mech. and Mach. Theory*, 40(11):1258–1272, 2005.

[SKH98] T. Speer, M. Kuppe, and J. Hoschek. Global reparametrization for curve approximation. *Comput. Aided Geom. Design*, 15:869–877, 1998.

[WPL06] W. Wang, H. Pottmann, and Y. Liu. Fitting B-spline curves to point clouds by squared distance minimization. *ACM Transactions on Graphics*, 25(2), 2006.

Geometric Accuracy Analysis for Discrete Surface Approximation

Junfei Dai[1], Wei Luo[1], Shing-Tung Yau[2], and Xianfeng David Gu[3]

[1] Center of Mathematical Sciences, Zhejiang Univeristy
[2] Mathematics Department, Harvard University
[3] Center for Visual Computing, Stony Brook University

Abstract. In geometric modeling and processing, computer graphics and computer vision, smooth surfaces are approximated by discrete triangular meshes reconstructed from sample points on the surface. A fundamental problem is to design rigorous algorithms to guarantee the geometric approximation accuracy by controlling the sampling density.

This theoretic work gives explicit formula to the bounds of Hausdorff distance, normal distance and Riemannian metric distortion between the smooth surface and the discrete mesh in terms of principle curvature and the radii of geodesic circum-circle of the triangles. These formula can be directly applied to design sampling density for data acquisition and surface reconstructions.

Furthermore, we prove the meshes induced from the Delaunay triangulations of the dense samples on a smooth surface are convergent to the smooth surface under both Hausdorff distance and normal fields. The Riemannian metrics and the Laplace-Beltrami operators on the meshes are also convergent. These theoretic results lay down the theoretic foundation for a broad class of reconstruction and approximation algorithms in geometric modeling and processing.

1 Introduction

In geometric modeling and processing, computer graphics and computer vision, smooth surfaces are often approximated by polygonal surfaces, which are reconstructed from a set of sample points. One of the fundamental problems is to measure the approximation accuracy in terms both position, normal fields and Riemannian metrics. It is highly desirable to design practical reconstruction algorithms with approximation errors fully controlled by the sampling density and triangulation method. This work accomplishes this goal by establishing the relation between the Hausdorff distance, normal field distance and the sampling density.

Different surface reconstruction algorithms have been discussed by many researchers. Hoppe et al. [1], [2] represented the surface by the zero set of a signed distance function. Amenta et. al developed a series of algorithms based on Voroni diagram in [3], [4], [5]. Bernardini and Bajaj used α shapes for manifold sampling and reconstruction [6,7]. Recently Ju et. al introduced the dual contour method for reconstruction [8]. Floater and Reimers reconstructed surfaces based on parameterizations [9]. Surface reconstruction has been applied to reverse engineering [10], geometric modeling [11], mesh optimization and simplification [12] and many other important applications.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 59–72, 2006.
© Springer-Verlag Berlin Heidelberg 2006

It is a common belief that by increasing the sampling density, the reconstructed discrete mesh will approximate the smooth surface with any desired accuracy. This work aims at precisely formulating this common belief and rigorously prove it in an appropriate setting. This result will offer the theoretic guarantee for the general algorithms in geometric modeling and processing, where the measurements on smooth surface are calculated on its discrete approximations and the physical phenomena on original surface are simulated on the discrete counterpart.

Geometric Accuracy. There are different levels of accuracy when approximating a smooth surface by discrete meshes,

1. *Topological consistency*, it requires the surface and mesh are homeomorphic to each other;
2. *Positional consistency*, measured by Hausdorff distance between the surface and the mesh;
3. *Normal consistency*, it requires the normal fields on the surface and on the mesh are close to each other.

Many previous works address the theoretic guarantee of topological consistency. Leibon et al. proved that if the sample density is high enough, the smooth surface and the triangle mesh induced by the Delaunay triangulation is homeomorphic in [13]. Amenta et. al proved similar result in [5].

In terms of positional consistency, Amenta et al. invented a series of algorithms which reconstruct the meshes from sample points based on Voronoi diagrams. Assume the diameter of the circum-circle of the triangles is ε and the normal error is small enough, the Hausdorff distance between the mesh and the surface is bounded by the ε^2 in [5]. In [14], Elber introduced an algorithm to approximate freeform surfaces by discrete meshes with bounded Hausdorff distance.

Positional consistency does not guarantee the normal consistency. It is very easy to find a sequence of meshes, which converge to a smooth surface under the Hausdorff distance, but the normal field does not converge. In [15] and [16], Morvan and Thibert established theoretic results to estimate the normal error and area difference in terms of Hausdorff distance and angles on the triangulation.

In geometric modeling and processing, many algorithms require calculating the geodesics [17]. Many parameterization works require accurately approximating the Riemannian metrics [18], spectrum compression also needs good approximation of Laplacian-Beltrami operators [19]. These important applications demand the theoretic guarantee for accurate approximation for Riemannian metric and differential operators. It has been shown in [20], Hausdorff convergence and normal field convergence guarantees the convergence of area, Riemannain metric tensor and laplace-Beltrami operator

Therefore, our work focuses on estimating both Hausdorff distance and normal field distance at simultaneously, the only assumption is the sampling density.

Triangulations. Triangulations play vital roles in surface reconstruction. There are different ways to measure the refinement of a triangulation,

1. The bound l of the longest lengths of the edges of triangles in the mesh.
2. The bound d of the diameters of the circum-circles of triangles in the mesh.

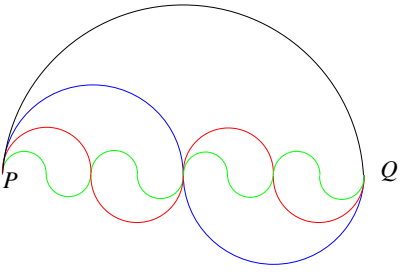


Fig. 1. Hausdorff convergence doesn't guarantee normal convergence and length convergence. The black curve is a half circle with radius r, the blue curve is composed by two half circles with radii $\frac{r}{2}$; the red curve is composed by 4 half circles with radii $\frac{r}{4}$. A sequence of curves can be constructed, they converge to the diameter PQ under the Hausdorff distance. But the length of each of them equals to πr, which do not converge to the length of the diameter $2r$.

It is obvious that the diameter bounds the edge length, but the edge length does not bound the diameter. In the following discussion, we will demonstrate that the Hausdorff distance is bounded by the square of the edge length, whereas the normal error is bounded by the diameter of the circum-circle.

In figure 1, we demonstrate a one dimensional example, where a family of curves converge to a straight line segment under Hausdorff distance. The lengths and normals do not converge.

In figure 2, we demonstrate an example, where for the same sets of sample points, the bounds of edge lengths go to zero, but the bounds of the diameters of circum-circles remain constant. Therefore, the area, the metrics on the meshes do not converge to those on the smooth surface.

Given a dense set of point samples, it is highly desirable to find a triangulation such that the circum-circles are as small as possible. For point samples on the plane, Delaunay triangulation is a good candidate for such a triangulation. Leibon generalizes Delaunay triangulation to arbitrary Riemannian manifolds [13]. In the following discussion, we use Delaunay triangulation to refer Delaunay triangulation on surfaces. The Delaunay triangulation is determined solely by the point samples. In the following discussion, we will show that the meshes induced by the Delaunay triangulations are convergent both under Hausdorff distance and normal distance.

In practice, there is no prior knowledge of the smooth surface, only the dense point samples are available. The connectivity induced by the surface Delaunay triangulation can be best approximated using Voronoi diagram in R^3 as described in [3,5]. We have not fully proven the consistency between the two triangulations.

Factors Affecting Geometric Accuracy. In order to achieve bounded Hausdorff error and normal error, the following factors play crucial rules, the sampling density should be carefully designed. The major factors determine the sampling density are as followings,

– **Principle curvature.** For regions with higher principle curvature, the samples should be denser.

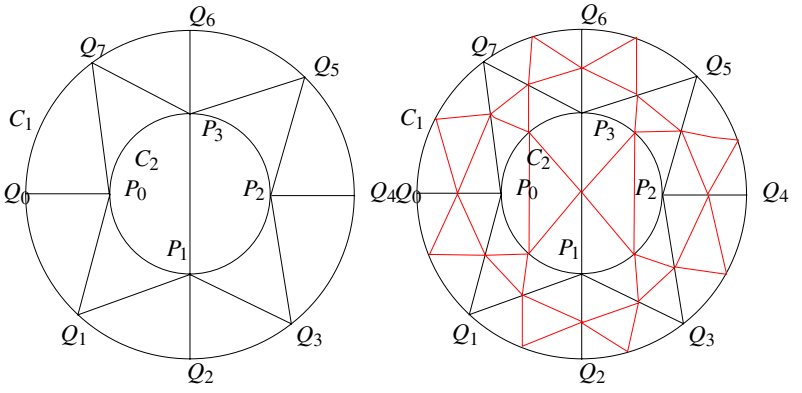


Fig. 2. Hausdorff convergence vs. normal convergence. In the left frame, the center is the north pole $(0,0,1)$ of the unit hemisphere. C_1 is the equator $x^2+y^2=1$, C_2 is the intersection circle between the sphere with the plane $z=\frac{1}{2}$. All the arcs Q_iQ_j and P_iQ_j are geodesics, the arcs P_iP_j are arcs along C_2. The right frame shows one step subdivision: insert the middle points of all the arcs in the left frame, split each triangle to 4 smaller ones, such that if an edge connecting two points on C_2, the edge is the arc on C_2, otherwise the edge is a geodesic segment. Repeating this subdivision process to get a sequence of triangulations $\{T_n\}$, and a sequences meshes M_n induced by the triangulations. The longest edge length of T_n goes to zero, M_n converge to the hemi-sphere under Hausdorff distance. For any M_n there is one triangles f_0 adjacent to P_0 and contained in the curved triangle $P_1P_0P_3$. Because all three vertices of f_0 are on C_2, its circumscribe circle is C_2, the normal of f_0 is constant which differs from the normal at P_0 to the sphere. Therefore, $\{M_n\}$ doesn't converge to the sphere under normal distance.

- **Distance to medial axis** For regions closer to the medial axis, the samples should be denser to avoid topological ambiguity during the reconstruction process. It is also called local feature size.
- **Injectivity radius** Each point p on the surface M has a largest radius r, for which the geodesic disk $B(p,r)$ is an embedding disk. The injectivity radius of M is the infinum of the injectivity radii at each point. Each geodesic triangle on the surface should be contained in a geodesic disk with radius less than the injectivity radius.

These factors are not independent, but closely related. Suppose k is the bound of principle curvature on the surface, then the distance to the medial axis is no greater than $\frac{1}{k}$ as proved in [21].

Comparisons to previous theoretic results. Hildebrandt et.al's work [20] focuses on the equivalence between convergences of polyhedral meshes under different metrics, such as Hausdorff, normal, area and Laplace-Beltrami. Assuming the Hausdorff convergence and the homeomorphism between the surface and the mesh, all the error estimation is based on the homeomorphism.

Leibon et al.'s work [13] focuses on the existence of Delaunay triangulation for dense sample set. It only estimates the Riemannian metric error without considering Hausdorff error and normal error.

Amenta et al.'s work [5] only demonstrates the estimation of Hausdorff error under the two assumptions, first the sampling density is sufficiently high, second the normal field error is given and bounded.

Morvan and Thibert [15] [16] estimate the normal error and area difference in terms of Hausdorff distance and angles on the triangulation. In practice, in order to control both the Hausdorff distance and angles of triangulation, Chew's algorithm is applied to progressively add samples to reduce the Hausdorff distance and improve the triangulation.

Previous works either assume the normal error is bounded and estimate the Hausdorff distance or assume the Hausdorff distance is given and estimate the normal field error. In contrast, our work shows that solely the radii of geodesic circum-circles of faces on the triangulations are enough to guarantee the convergence of both Hausdorff distance and the normal fields. To the best of our knowledge, our work is the first one to bound both the Hausdorff error and the normal error (therefore the Riemannian metric distortion) only by the sampling density.

The main theorem of the work is that if the sample density is ε, then the Hausdorff distance is no greater $4k\varepsilon^2$, the normal error is no greater than $9k\varepsilon$, where k is the upper bound of the principle curvature on the smooth surface to be approximated. The metric distortion is measured by the infinitesimal length ratio, which is bounded by $1-4k^2\varepsilon^2$ and $\frac{1+4k^2\varepsilon^2}{1-9k\varepsilon}$.

The paper is organized as the following, the next section 2 introduces the preliminary concepts and theorems proven in previous works; our new theoretic results are explained in details in section 3. This section is the most technical part of the work, the main focus is the proofs of three major theorems; experimental results are demonstrated in section 4; Finally the paper is concluded in section 5, where the future works are briefly discussed.

2 Definitions and Preliminaries

In this section, we review the preliminary concepts necessary for our further theoretic arguments. We adapt the definitions from [13],[20], [3] and [5].

We assume that the surface S is closed without any boundary, at least C^2 smooth with bounded principle curvature, embedded in R^3.

2.1 Medial Axis, ε-Sampling and Delaunay Triangulation

The *medical axis* of a surface S embedded in R^3 is the closure of the set of points with more than one nearest neighbor in S. The *local feature size* $f(p)$ at point $p \in S$ is the least distance of p to the medial axis.

A geodesic disk $B(p,r)$ centered at p with radius r is the the point sets

$$B(p,r) = \{q \in S | d(p,q) \leq r\},$$

d is the geodesic distance on the surface. The *injectivity radius* at a point $p \in S$ is the largest radius $\tau(p)$, for which the geodesic disk $B(p,\tau(p))$ is an embedding on S.

Suppose $\varepsilon : S \to R$ is a positive function defined on the surface S, a point set X is an ε-*sample*, if for any point $p \in S$, there is at least one sample inside the geodesic disk $B(p, \varepsilon(p))$.

The definition of Delaunay triangulations of X on S is the same as it is in R^2. They are defined as having the empty circumscribing circle property: the circum-circle for each geodesic triangle contains no vertices of the triangulation in its interior. In order to gurantee the uniqueness and embedness of the circum-circles, X should be dense enough.

Leibon et al. proved in [13] that, suppose X is a generic ε-sample, ε satisfies the following conditions:

$$\varepsilon(p) \leq \min\{\frac{2\tau(p)}{5}, \frac{2\pi}{5k(p)}\}, \tag{1}$$

where $k(p)$ is the upper bound of the principle curvature, $k(p) = \max_{q \in B(p,\tau(p))} |k(q)|$, then the Delaunay triangulation of X exists and is unique.

2.2 Hausdorff Distance, Normal Distance and Shortest Distance Map

Let $M_1, M_2 \subset R^3$ be non empty point sets, the *Hausdorff distance* between M_1 and M_2 is defined as

$$d_H(M_1, M_2) = \inf\{\varepsilon > 0 | M_1 \subset U_\varepsilon(M_2), M_2 \subset U_\varepsilon(M_1)\}, \tag{2}$$

where $U_\varepsilon(M) = \{x \in R^3 | \exists y \in M : d(x,y) < \varepsilon\}$.

Suppose S and M are two surfaces embedded in R^3, the shortest distance map $g : M \to S$ is defined to map $p \in M$ to its nearest point $g(p)$ on S. It is proved that the line connecting p to $g(p)$ is along the normal direction at $g(p)$ on S. It has been proven in [13], if the sample density ε satisfies the Delaunay triangulation condition equation 1 and the following

$$\varepsilon(p) \leq \frac{f(p)}{4}, \tag{3}$$

then the g is a homeomorphism between the mesh M and S induced by the Delaunay triangulation. Then we denote the inverse of g as $\Phi = g^{-1} : S \to M$ and call it the *inverse shortest distance map*, then

$$\Phi(p) = p + \phi(p)\mathbf{n}(p), p \in S \tag{4}$$

where $\mathbf{n}(p)$ is the normal vector at p on S, $\phi(p)$ measures the distance from p to $\Phi(p)$ on the mesh. The normal distance between S and M is defined as

$$d_n(S, M) = \max_{p \in S} |\mathbf{n}(p) - \mathbf{n} \circ \Phi(p)|.$$

Suppose $\gamma : t \to S$ is a curve on S, then $\Phi \circ \gamma : t \to M$ is a curve on M. It is proven in [20], the infinitesimal distortion of length satisfies

$$\min_i(1 - \phi k_i) \leq \frac{dl_M}{dl} \leq \max_i \frac{1 - \phi k_i}{< \mathbf{n}, \mathbf{n} \circ \Phi >}, \tag{5}$$

where $dl = \sqrt{< d\gamma, d\gamma >}$ is the length element on S, $dl_M = \sqrt{< d\gamma \circ \Phi, d\gamma \circ \Phi >}$ is the corresponding length element on M, k_i is the principle curvature.

3 Geometric Accuracy Analysis

In this section, we analyze the geometric accuracy of reconstructed meshes. Suppose X is an ε-sample on S, if ε satisfies equation 1 then X induces a unique Delaunay triangulation T, where all edges are geodesics. Each face on T has a unique geodesic circumscribed circle, the bound of all the radii $r(X)$ is determined by the sampling density ε. Then by replacing geodesic triangles on T to Euclidean triangles, a piecewise linear complex $M(X)$ is produced, denoted as the *Delaunay mesh* induced by X. Our goal is to estimate the Hausdorff error, normal error and Riemannian metric error between S and M, in terms of the $r(X)$ and sampling density ε.

The following is the major steps of our proof,

1. We first estimate the Hausdorff distance between a geodesic triangle and the planar triangle through its vertices.
2. Then we estimate the normal deviation between the normal at an arbitrary point in a geodesic triangle and the normal of the planar triangle.
3. Finally we discuss the Hausdorff distance and normal distance between S and M, then we estimate the metric distortion.

3.1 Hausdorff Distance Between a Geodesic Triangle and a Planar Triangle

Lemma 1. *Let $R(t)$ be an arc length parameterized smooth space curve with curvature bound $\kappa > 0$, $0 \le a,b,t,t' \le \pi/\kappa$, $m = \frac{R(b)-R(a)}{|R(b)-R(a)|}$ then the following estimates hold*

$$t \ge |R(t)| \ge 2\sin(\kappa t/2)/\kappa \tag{6}$$

$$|R'(t) \times m| \le \frac{1}{4}\kappa(b-a), \;\; t \in [a,b], \kappa(b-a) < \sqrt{6} \tag{7}$$

$$|R'(t') - R'(t)| \le |2\sin(\kappa(t-t'))| \tag{8}$$

$$\angle R(a)R(t)R(b) \ge \pi/2 \tag{9}$$

$$\mathrm{dist}(R(t), R(a)R(b)) \le \frac{\kappa(b-a)\min(t-a, b-t)}{4}, \;\; 0 < a < t < b < \sqrt{6}/\kappa \tag{10}$$

$$0 < (R(t) - R(a), m) < |R(b) - R(a)|, \quad t \in (a,b) \tag{11}$$

where $\mathrm{dist}(\cdot,\cdot)$ *denote the distance from a point to a line, and* $(\cdot,\cdot)$ *denotes the inner product of two vectors.*

Proof. Consider function $f(t) = (R'(t), R'(0))$, then since $R'(t) \perp R''(t)$,

$$\begin{aligned} f'(t) &= (R''(t), R'(t_0)) = (R''(t), R'(t_0) - (R'(t_0), R'(t))R'(t)) \\ &\ge -\kappa |R'(t) \times R'(t_0)| = -\kappa\sqrt{1 - f^2(t)} \end{aligned}$$

$f(t)$ satisfies $f'(t) \ge -\kappa\sqrt{1-f^2(t)}$, then

$$\frac{\partial}{\partial t}(\arccos f(t)) \le \kappa$$

$$f(t) \ge \cos(\kappa t), \quad t \in [0, \pi/\kappa]$$

Now the estimates follows by integration:

$$\begin{aligned}
|R(t)|^2 &= \int_0^t \int_0^t (R'(t_1), R'(t_2))\, dt_1 dt_2 \\
&\geq \int_0^t \int_0^t \cos(\kappa(t_2 - t_1))\, dt_1 dt_2 \\
&= 4\kappa^{-2} \sin^2(\kappa t/2) \quad \Rightarrow (6)
\end{aligned}$$

$$\begin{aligned}
|R(b) - R(a)|(R'(t), m) &\geq \int_a^b \cos\kappa(s-t)\, ds \\
&= \frac{1}{\kappa}(\sin\kappa(b-t) + \sin\kappa(t-a)) \qquad (12) \\
&\geq (b-a) - \frac{\kappa^2 (b-a)^3}{6} \\
|R'(t) \times m| &= \sqrt{1 - (R'(t), m)^2} \\
&\leq \sqrt{1 - (1 - \kappa^2 (b-a)^2/6)^2}, \ \text{ if } (b-a) < \sqrt{6} \\
&\leq \frac{\sqrt{2}}{6} \kappa (b-a) \quad \Rightarrow (7)
\end{aligned}$$

Equation (12) implies $(R'(t), R(b) - R(a)) > 0$ when $b - a < \pi$, hence $(R(t), R(b) - R(a))$ is an increasing function of t, hence (11) is proved.

$$\begin{aligned}
(R(b) - R(t), R(t) - R(a)) &\geq \int_a^t \int_t^b \cos\kappa(u - v)\, du dv \\
&= \cos\kappa(b-t) + \cos\kappa(t-a) - 1 - \cos(b-a) \\
&\geq 0, \quad \text{if } \kappa(b-a) < \pi \ \Rightarrow (9).
\end{aligned}$$

$$(R'(t), R'(t')) \geq \cos\kappa(t' - t), \ \ |R(t)| = |R'(t')| = 1 \ \Rightarrow (8)$$

Assume $t - a < b - t$, then (7) implies that

$$\text{dist}(R(t), R(a)R(b)) = |(R(t) - R(a)) \times m)| \leq \kappa(b-a)(t-a)/4 \quad \Rightarrow \text{Eqn(10)}.$$

□

Notation: We use $\widetilde{\ }$ to denote an geometric object on a surface in geodesic sense, such as $\widetilde{AB}$, $\widetilde{\Delta ABC}$ to denote a geodesic or a geodesic triangle.

Lemma 2. *P, Q are two points in a geodesic convex region of a smooth surface with principal curvature bounded by* κ*, then the normal at P, Q differs by at most* $\kappa|\widetilde{PQ}|$.

Proof. Bound of principal curvature implies $|\nabla n| \leq \kappa$, where ∇ is the covariant derivative and n is the normal. Hence the estimate. □

The following theorem estimate the distance of points inside a geodesic triangle to the plane through the vertices, independent of the shape of the triangle.

Theorem 1. *Let $\Delta\widetilde{ABC}$ be a geodesic triangle on a smooth surface embedded in $\mathbb{R}^3$ where the principal curvature is bounded by κ and the maximal length d of edges of $\Delta\widetilde{ABC}$ is bounded by $1/\kappa$. P is any point inside the triangle, P_{ABC} is the plane through A,B,C, then the* $\mathrm{dist}(P,P_{ABC}) \leq \kappa d^2/4$.

Proof. Assume A is the vertex farthest from P, $\widetilde{AP}$ intersects $\widetilde{BC}$ at Q, P' is the projection of P onto AQ. By (10),

$$\mathrm{dist}(Q,P_{ABC}) \leq \mathrm{dist}(Q,BC) \leq \kappa d^2/8.$$
$$|PP'| \leq \kappa d^2/8$$

(11) implies that P' is inside AQ and $\mathrm{dist}(P',P_{ABC}) \leq \mathrm{dist}(Q,P_{ABC})$,

$$\mathrm{dist}(P,P_{ABC}) \leq |PP'| + \mathrm{dist}(P',P_{ABC}) \leq \kappa d^2/4$$

□

3.2 Normal Error Estimation Between a Geodesic Triangle and a Planar Triangle

Lemma 3. *Let $\widetilde{ABC}$ be a geodesic triangle with maximal length of edge d, the principal curvature is bounded by κ, $d < 2/\kappa$, $\angle BAC = \alpha$, then the normal n_A to the surface at A and the normal n to P_{ABC} satisfies*

$$|n_A \times n| \leq \max\left(\frac{\kappa d}{4\sin(\alpha/2)}, \frac{\kappa d}{4\cos(\alpha/2)}\right) \tag{13}$$

Proof. Denote by T_1 the tangent vector at A to $\widetilde{AB}$, T_2 tangent to $\widetilde{AC}$, V_1 the unit vector along AB, V_2 along AC. Then by (7)

$$|T_1 \times V_1| \leq \kappa d/2, |T_2 \times V_2| \leq \kappa d/4$$

So

$$\begin{aligned} |(n_A,V_1)| &= |(n_A,V_1 - (V_1,T_1)T_1)| \\ &\leq |V_1 - (V_1,T_1)T_1| = |T_1 \times V_1| \leq \kappa d/4 \\ |(n_A,V_2)| &\leq \kappa d/4 \end{aligned}$$

The projection of n_A onto P_{ABC} falls into the parallelogram with both width $\kappa d/2$ and inner angle $\alpha, \pi - \alpha$, centered at A. Now (13) follows by simple trigonometry. □

Lemma 4. *Let $l(t)$ be a geodesic circle radius r, parameterized by arc length. Suppose the principal curvature is bounded by κ in the disk and $r \leq 1/(4\kappa)$. $N(t)$ is the tangent vector at $l(t)$ normal to $l'(t)$. Then for $t < r$,*

$$(l(t) - l(0), N(0)) \geq \frac{t^2}{5r} \tag{14}$$

Proof. Let $n(t)$ be the normal to the surface at $l(t)$. The curvature condition implies $|(l''(t),n(t))| \le \kappa$. Hessian comparison theorem [22] implies

$$(l''(t),N(t)) \ge \kappa\cot(\kappa r) \ge \frac{19}{20r}, \quad \text{if } \kappa r < 1/4$$
$$|l''(t)| \le \kappa\sqrt{\coth\kappa r^2 + 1} \le \frac{11}{10r}, \quad \text{if } \kappa r < 1/4 \tag{15}$$

Lemma 3 implies $|n(t) - n(0)| \le \kappa t$. (8) and (15) implies $|l'(t) - l'(0)| \le 11t/10r$, then for $t \le r$

$$\begin{aligned}(l''(t),N(0)) &= (l''(t),N(t)) + (l''(t),N(0) - N(t)) \\ &= (l''(t),N(t)) + (l''(t),n(0)\times l'(0) - n(t)\times l'(t)) \\ &\ge (l''(t),N(t)) - |l''(t)|(|n(0) - n(t)| + |l'(0) - l'(t)|) \\ &\ge \frac{1}{r}(\frac{19}{20} - \frac{11\kappa t}{10} - \frac{121t}{100r})\end{aligned} \tag{16}$$

Use $l'(0) \perp N(0)$, integrate (16) to get

$$(l(t) - l(0),N(0)) \ge \frac{9t^2}{40r}$$

□

Theorem 2. *D is a geodesic disk of radius r of a smooth surface embedded in $\mathbb{R}^3$ with principal curvature bounded by κ, $r < 1/(4\kappa)$. A,B,C are three distinct points on the boundary of D, P_{ABC} is the plane through A,B,C, ϕ is the projection map from D onto P_{ABC}. For any point $p \in D$, $v \in T_p$ is a tangent vector, we have*

$$|n_p - n_{ABC}| \le 4.5\kappa r \tag{17}$$
$$|v| \ge |\phi_*(v)| \ge |v|(1 - 4.5\kappa r) \tag{18}$$
$$\mathrm{dist}(p,P_{ABC}) \le 9\kappa r^2 \tag{19}$$

Proof. Consider the intersection angle between the radial geodesic connecting center O of D and the vertices A,B,C.

If two such intersection angle is less than $9/10$, say $\angle\widetilde{AOB}, \angle\widetilde{BOC}$, then comparison theorem shows that the arc between A, B or between B, C along boundary of D is less than

$$\frac{9}{10}\cdot\frac{e^{\kappa r} - e^{-\kappa r}}{2\kappa} \le r \tag{20}$$

Let d_1, d_2 be the length of line segment AB, BC respectively, then (11) implies $\angle ABC > \pi/2$ while Lemma 4 implies

$$\angle ABC \le \pi - \arcsin(d_1/5r) - \arcsin(d_2/5r)$$

Then apply Lemma 3 to get

$$|n_B - n_{ABC}| \le \frac{\kappa(d_1 + d_2)}{4\cos(\angle ABC)/2} \le \frac{\kappa(d_1 + d_2)}{2d_1/5r + 2d_2/5r} = 2.5\kappa r$$

If only one such intersection angle is less than $9/10$, say $\angle\widetilde{AOB}$. For the triangle AOC, by (6) and (7)

$$\begin{gathered}\angle AOC \geq 9/10 - 2\arcsin(\kappa r/4) \geq 0.77 \\ r > |OA|, |OC| > .99r \\ |AC| \geq 2 * .99\sin(0.77/2)r > 0.74r \qquad (21) \\ \angle CAO \leq \arccos((.99^2 + .74^2 - 1)/(2 * .99 * .74)) < 1.21 \\ \angle BAO \leq \pi/2 + \arcsin(\kappa r/4) \leq 1.64 \\ 4\cos(\angle(CAB)/2) \geq 4\cos(1.425) \geq 0.58 \qquad (22)\end{gathered}$$

On the other hand for ΔABC, $r > |AC|, |BC| \geq 0.74r$ by (21), $|AB| \leq r$ by (20), so

$$\begin{gathered}\angle BAC \geq \min(\arccos(.5/.74), 2\arcsin(.74/2)) \geq .75 \\ 4\sin(\angle BAC/2) \geq 1.4 \qquad (23)\end{gathered}$$

Now apply Lemma 3 with estimate (22) and (23) to give an estimate of the difference of the normal at A and to the plane ABC, then use Lemma 2 to get (17).

Given (17) proved, (18) easily follows as

$$|v - \phi_*(v)| = |(v, n_{ABC})| = |(v, n_{ABC} - n_A)| \leq 4.5\kappa r|v|$$

and for (19), let $l(t)$ be the geodesic connecting A and p

$$\mathrm{dist}(p, P_{ABC}) = |\int_l (l'(t), n_{ABC})dt| \leq \widetilde{\mathrm{dist}}(p, A) * 4.5\kappa r \leq 9\kappa r^2$$

□

3.3 Geometric Accuracy for Delaunay Meshes

Combining the theoretic results in section 2 and the estimations on a single geodesic triangle theorem 1 and theorem 2, we can easily get the following theorem.

Theorem 3. *Suppose S is a closed C^2 smooth surface embedded in R^3. The principle curvature upper bound is k, the injective radius lower bound is τ, the lower bound of local feature size is f. Suppose X is an ε-sample set on S, such that constant ε satisfies the following conditions,*

$$\varepsilon \leq \min\{\frac{2\tau}{5}, \frac{2\pi}{5k}, \frac{f}{4}, \frac{1}{4k}\},$$

then X induces a unique Delaunay triangulation T, (X, T) induces a piecewise linear complex M,

1. *M is homeomorphic to S, the nearest distance map $g : M \to S$ is a homeomorphism.*
2. *The Hausdorff distance*
$$d_H(M, S) \leq 4k\varepsilon^2 \qquad (24)$$
3. *The normal distance*
$$d_n(M, S) \leq 9k\varepsilon \qquad (25)$$

4. The infinitesimal length ratio

$$1 - 4k^2\varepsilon^2 \leq \frac{dl_M}{dl} \leq \frac{1 + 4k^2\varepsilon^2}{1 - 9k\varepsilon}. \tag{26}$$

Proof. Because X is an ε-sample, ε satisfies the Delaunay triangulation condition in eqn. 1, therefore the unique Delaunay triangulation T exists according to [13]. ε is less than a quarter of the local feature size 3, then the shortest distance map is a homeomorphism.

Suppose C is a circumscribe circle of a triangle in T, then there is no interior point belonging to X. If the radius of C is greater than 2ε, then C contains at least disks with radii ε, therefore, it contains at least one point of X as its interior. Thus the radius of C is no greater than 2ε.

From theorem 1, the Hausdorff distance is no greater than $4k\varepsilon^2$. From theorem 2, the normal distance is no greater than $9k\varepsilon$.

In the inverse shortest distance map eqn. 4, ϕ is less or equal to the Hausdorff distance. From formula 5 we can derive the 26. □

Although the sample density ε is a constant here, it can be generalized to be a function on the surface, such that

$$\varepsilon(p) \leq \min\{\frac{2\tau(p)}{5}, \frac{2\pi}{5k(p)}, \frac{f(p)}{4}, \frac{1}{4k(p)}\},$$

then we can estimate the Hausdorff distance, normal distance and metric distortion at point p using the formula similar to 24,25,26 with ε replaced by $\varepsilon(p)$.

4 Experimental Results

In order to verify our theorems, we tessellate several smooth surfaces with different resolutions, and measure the Hausdorff distance and normal deviation.

The smooth surfaces are represented as NURBS surfaces, therefore the computation of the bound of principle curvature is straight forward. The Hausdorff distance is calculated by minimizing the following functional, suppose $p \in M$, S is $S(u,v)$ then $f(u,v) = < S(u,v) - p, S(u,v) - p >$. For any point p on M, first we find the closest vertex p_0 on M, p_0 is also on S with parameter (u_0, v_0). Then we use (u_0, v_0) as the initial point, then use Newton's method to find the global minimum of $f(u,v)$. We densely sample M, and find the maximum distance between the sample points to S.

Table 1 illustrates the comparison between numerical results and the theoretic estimation. From the table, it is clear that the numerical results are always no greater than the theoretic upper bound, and the real values are close to their theoretic predictions.

5 Conclusion and Future Work

This work gives explicit formula of approximation error bounds for both Hausdorff distance and normal distance in terms of sampling density. For a set of sample points

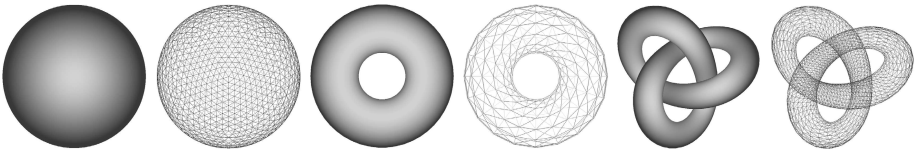


Fig. 3. Smooth surfaces are approximated by meshes. The Hausdorff distances really measured are consistent with the theoretic formula.

Table 1. The Hausdorff distance really measured and the theoretic estimations $\frac{1}{4}kd^2$. κ maximal principal curvature. d maximal length of edges. $1/4\kappa d^2$ is the estimated distance, D is the real Hausdorff distance.

Shape	Vertices	Faces	D	κ	d	$1/4\kappa d^2$
sphere	1002	2000	0.010025	1.758418	0.314097	0.014678
torus	240	480	0.049931	4.525904	0.616819	0.065040
knot	2000	4000	0.032552	9.280646	0.391005	0.053959

on a surface with sufficient density, it induces a unique Delaunay triangulation, and a discrete mesh. With the increase of sampling density, the Delaunay meshes converge to the original surface under both Hausdorff distance and normal distance, therefore, the area, the Riemannian metrics and the Laplace-Beltrami operators are also convergent.

In the future, we will apply these error estimation formula to prove the convergence of other advanced algorithms in geometric modeling and processing, such as the conformal parameterizations, Poisson editing etc.

Acknowledgements

This work was supported in part by the NSF CAREER Award CCF-0448339 to X. Gu.

References

1. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J.A., Stuetzle, W.: Surface reconstruction from unorganized points. In: SIGGRAPH. (1992) 71–78
2. Eck, M., Hoppe, H.: Automatic reconstruction of b-spline surfaces of arbitrary topological type. In: SIGGRAPH. (1996) 325–334
3. Amenta, N., Bern, M.W., Kamvysselis, M.: A new voronoi-based surface reconstruction algorithm. In: SIGGRAPH. (1998) 415–421
4. Amenta, N., Bern, M.W.: Surface reconstruction by voronoi filtering. Discrete & Computational Geometry **22** (1999) 481–504
5. Amenta, N., Choi, S., Dey, T.K., Leekha, N.: A simple algorithm for homeomorphic surface reconstruction. Int. J. Comput. Geometry Appl. **12** (2002) 125–141
6. Bajaj, C.L., Bernardini, F., Xu, G.: Automatic reconstruction of surfaces and scalar fields from 3d scans. In: SIGGRAPH. (1995) 109–118
7. Bernardini, F., Bajaj, C.L.: Sampling and reconstructing manifolds using alpha-shapes. In: CCCG. (1997)

8. Ju, T., Losasso, F., Schaefer, S., Warren, J.D.: Dual contouring of hermite data. In: SIGGRAPH. (2002) 339–346
9. Floater, M.S., Reimers, M.: Meshless parameterization and surface reconstruction. Computer Aided Geometric Design **18** (2001) 77–92
10. Benkö, P., Martin, R.R., Várady, T.: Algorithms for reverse engineering boundary representation models. Computer-Aided Design **33** (2001) 839–851
11. He, Y., Qin, H.: Surface reconstruction with triangular b-splines. In: GMP. (2004) 279–290
12. Hoppe, H.: Progressive meshes. In: SIGGRAPH. (1996) 99–108
13. Leibon, G., Letscher, D.: Delaunay triangulations and voronoi diagrams for riemannian manifolds. In: Symposium on Computational Geometry. (2000) 341–349
14. Elber, G.: Error bounded piecewise linear approximation of freeform surfaces. Computer Aided Design **28** (1996) 51–57
15. Morvan, J., Thibert, B.: On the approximation of a smooth surface with a triangulated mesh. Computational Geometry Theory and Application **23** (2002) 337–352
16. Morvan, J., Thibert, B.: Approximation of the normal vector field and the area of a smooth surface. Discrete and Computational Geometr **32** (2004) 383–400
17. Surazhsky, V., Surazhsky, T., Kirsanov, D., Gortler, S.J., Hoppe, H.: Fast exact and approximate geodesics on meshes. ACM Trans. Graph. **24** (2005) 553–560
18. Floater, M.S., Hormann, K.: Surface parameterization: a tutorial and survey. In Dodgson, N.A., Floater, M.S., Sabin, M.A., eds.: Advances in multiresolution for geometric modelling. Springer Verlag (2005) 157–186
19. Ben-Chen, M., Gotsman, C.: On the optimality of spectral compression of mesh data. ACM Trans. Graph. **24** (2005) 60–80
20. Hildebrandt, K., Polthier, K., Wardetzky, M.: On the convergence of metric and geometric properties of polyhedral surfaces. (2005) submitted.
21. Federer, H.: Curvature measures. Transactions of the American Mathematical Society **93** (1959) 418–491
22. Schoen, R., Yau, S.T.: Lectures on Differential Geometry. International Press, Cambridge, MA (1994)

Quadric Surface Extraction by Variational Shape Approximation

Dong-Ming Yan, Yang Liu, and Wenping Wang

The University of Hong Kong, Pokfulam Road, Hong Kong, China
{dmyan, yliu, wenping}@cs.hku.hk

Abstract. Based on Lloyd iteration, we present a variational method for extracting general quadric surfaces from a 3D mesh surface. This work extends the previous variational methods that extract only planes or special types of quadrics, i.e., spheres and circular cylinders. Instead of using the exact L^2 error metric, we use a new approximate L^2 error metric to make our method more efficient for computing with general quadrics. Furthermore, a method based on graph cut is proposed to smooth irregular boundary curves between segmented regions, which greatly improves the final results.

Keywords: variational surface approximation, quadric surface fitting, graph cut, segmentation.

1 Introduction

Polygonal mesh surfaces are an important shape representation of complex 3D models, now readily acquired with 3D digital scanners or derived from CT/MRI volume data. But a high level concise and faithful geometric representation of mesh data is always desirable for geometry processing or rendering in graphics and CAD/CAM.

The Lloyd method for data clustering is employed in [1] to generate piecewise planar approximation of mesh surfaces. Each planar facet is called a *proxy* representing the part of the mesh surface approximated by the facet. This approach is extended by Wu and Kobbelt [2] to include spheres, circular cylinders and rolling-ball surfaces as additional types of proxies to achieve a more compact approximation. The confinement to these special surfaces is largely due to the relative ease of computing the exact Euclidean distance from a point to such surfaces.

There are two contributions in the present paper. Firstly, motivated by wide application and superior approximation power of quadrics, within the same clustering framework, we further extend the surface types of proxies to include general quadric surfaces, or *quadrics* for short, plus planes. We show how Euclidean distance from a triangle to a quadric can be computed in an approximate but efficient manner, while delivering robust segmentation results.

Secondly, we propose a new method for smoothing irregular boundary curves between adjacent segmented regions through energy minimization using a graph cut approach. This step produces more regular boundary curves, resulting in significantly improved segmentation results compared against previous results (e.g., [2]).

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 73–86, 2006.
© Springer-Verlag Berlin Heidelberg 2006

1.1 Related Work

There are two areas of research that are closely related to our work: *shape approximation* and *mesh segmentation.* The main purpose of shape approximation is to compute a simple and compact surface representation of a complex geometric shape, based on different surface types or different computational approaches. We will mainly review those methods that employ the clustering approach.

Shape Approximation. Cohen-Steiner et. al [1] propose a shape approximation algorithm based on clustering approach to optimally approximate a mesh surface by a specified number of planar faces. This optimization problem is solved as a discrete partition problem using the Lloyd algorithm [3], which is commonly used for solving the k-mean problem in data clustering. There are two iterative steps in this method: *mesh partition* and *fitting a plane face*, called a *proxy*, to each partitioned region. This method proves effective especially for extracting features and planar regions, but tends to produce an overly large number of planar proxies for a good approximation of a free-form surface. Because of its optimization nature, the method is often referred to as a *variational method*.

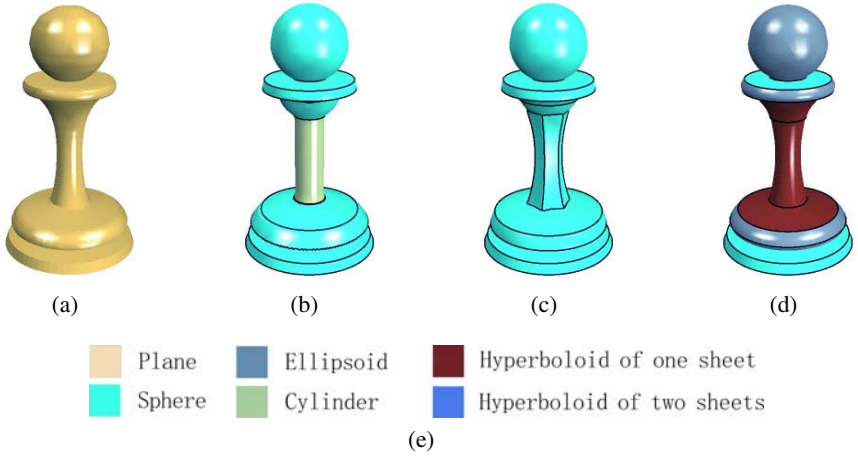


Fig. 1. (a) Original Chess piece (12K triangles); (b) Approximated by 12 hybrid proxies by the method in [2]: 1 plane, 1 cylinder and 10 spheres; (c) Approximated by 18 hybrid proxies by the method in [2]: 1 planes and 17 spheres; (d) Approximated by 12 quadric proxies by our method: 1 plane, 4 spheres, 4 ellipsoids, and 3 hyperboloids of one sheet; (e) Colors for different types of quadric surfaces used in this paper

Wu and Kobbelt [2] extend the work in [1] by introducing spheres, cylinders and rolling ball patches as additional basic proxy types, so that a complex shape can be approximated to the same accuracy by a much fewer number of proxies, leading to a more compact representation. However, these newly added surface types mentioned are still rather restricted, even for CAD models and other man-made objects. For example, the middle part of the Chess piece in Fig. 1 cannot be well approximated either by a circular cylinder or a collection of spherical surface strips.

Simari et al. [4] use ellipsoids as the only type of proxies for approximating mesh surfaces, again using the Lloyd method with the error metric being a combination of Euclidean distance, angular distance and curvature distance. The segmentation boundaries are smoothed by a constrained relaxation of the boundary vertices. They also approximate the volume bounded by a mesh surface using a union of ellipsoids, where whole ellipsoids, rather than ellipsoidal surface patches, are used. Julius et. al [5] segment mesh surfaces into developable surface charts for texture mapping and pattern design. Open segmentation boundaries are straightened by a shortest path algorithm and interior segmentation boundaries are smoothed by a graph cut method similar to that described in [6].

Attene et al. [7] propose a fast algorithm using automatic face clustering to segment a mesh hierarchically. A binary cluster tree is created from bottom to top. At each iteration, every pair of adjacent clusters are fitted by plane, sphere and cylinder, the pair with the minimal fitting error is merged into one cluster. Smoothing of segmentation boundaries is not considered.

Implicit surfaces have long been used for shape approximation and segmentation. Based on *region growing*, Besl et al. [8] segment range image data by fitting implicit surfaces of variable orders. Their algorithm works well on objects with sharp features or curvature discontinuities, but cannot handle free-from shapes. Fitzgibbon et al. [9] improve this work, also using region growing, to fit general quadric surfaces and planes to the range images, and compute surface intersections to extract a B-rep from the segmented image.

Since region growing relies mainly on local consideration, such as mean curvature and Gaussian curvature estimation, the segmentation result can be poor when there is no obvious curvature discontinuity, e.g., when two quadric patches join with near G^1 continuity. In this regard the iterative variational method has a distinct advantage that the local error in partition can be corrected by the fitting process, and the improved partition in turn provides a more reliable basis for better fitting.

Mesh Segmentation. Besides being used as a preparatory step for surface approximation, mesh segmentation is also used to partition a surface model into meaningful parts for various other purposes [10, 11, 12, 13, 6, 14, 15, 7, 16]. A detailed discussion of mesh segmentation methods is out of the scope of this paper. We refer the reader to the survey in [17].

Two recent methods for mesh segmentation are worth mentioning. Katz et al. [6] use fuzzy clustering and graph cut to segment a mesh. The mesh is first clustered by the geodesic distance. A fuzzy region is created between every two adjacent components. Finally the fuzzy region is segmented by a graph cut method to yield a smooth boundary. Lavoué et al. [16] present a mesh segmentation algorithm based on curvature tensor analysis. The mesh is first decomposed into several patches, each patch with nearly constant curvature. Then the segmentation boundary is rectified based on the curvature tensor directions.

Smoothing boundaries between adjacent segmented regions is usually considered as a post-processing step after mesh segmentation [6, 15, 16, 5]. We propose a new graph cut based strategy to smooth the segmentation boundary, which considers both the approximation error and the smoothness of the boundary between neighbor regions, and delivers better results in smooth regions of a surface (e.g., see Fig. 2(d)).

2 Preliminary

In this section we describe the variational shape approximation framework and introduce a new error function for measuring the distance between a mesh surface and a quadric surface.

2.1 Variational Framework

Let $\mathcal{M}$ denote an input mesh surface, and $\mathcal{T}$ denote the set of triangles of $\mathcal{M}$. Suppose that $\mathcal{M}$ is partitioned into n non-overlapping regions, denoted as $\mathcal{R} = \{\mathcal{R}_i\}_{i=1}^n$, each region $\mathcal{R}_i$ containing a set of triangles $\mathcal{T}_i = \{t_k^i\}_{k=1}^{n_i}$ such that $\bigcup_{i=1}^n \mathcal{T}_i = \mathcal{T}$. Each region is approximated by a quadric proxy (including the plane as a special case). A quadric proxy, denoted as $\mathcal{P}_i$, is represented by the coefficients of its associated quadratic form. A seed face, denoted as $\mathcal{S}_i$, is a triangle face in $\mathcal{T}_i$ that has the smallest error to the quadric proxy $\mathcal{P}_i$.

In a variational framework the optimal partition $\mathcal{R} = \{\mathcal{R}_i\}_{i=1}^n$ is found by minimizing the following objective function [1, 2]:

$$E(\mathcal{R}, \mathcal{P}) = \sum_{i=1}^{n} E'(\mathcal{R}_i, \mathcal{P}_i) = \sum_{i=1}^{n} \sum_{k=1}^{n_i} d(t_k^i, \mathcal{P}_i), \tag{1}$$

where $d(t_k^i, \mathcal{P}_i)$ measure the error between the triangle t_k^i and the proxy $\mathcal{P}_i$. Therefore, $E'(\mathcal{R}_i, \mathcal{P}_i)$ is the error between the partitioned region $\mathcal{R}_i$ and its approximating proxy $\mathcal{P}_i$. The error terms used in our method will be defined in Section 2.2.

Lloyd's algorithm minimizes the above objective function through iterative partition and fitting. Given a specified number n of proxies, the surface mesh $\mathcal{M}$ is first partitioned into n non-overlapping regions. Then the two alternative steps of quadric surface fitting and region partitioning are performed iteratively to reduce the value of the objective function until convergency or a specified number of iterations is reached. More details about this framework can be found [1].

For initialization, we randomly choose n initial seed faces. Then each seed face determines a planar proxy which is the plane containing the seed face. Then a distortion-minimizing flooding is performed, as described in [1], to give an initial partitioned mesh consisting of n regions $\mathcal{R} = \{\mathcal{R}_i\}_{i=1}^n$.

2.2 Error Metric for Proxies

The objective function in the variational shape approximation framework is defined in terms of error terms. Both $L^{2,1}$ and L^2 metrics have been tested in [1, 2]. For planar proxies, it is possible to derive closed formulas of L^2 and $L^{2,1}$ error terms, and $L^{2,1}$ proves to produce better results. Wu and Kobbelt [2] use an approximate L^2 error term to measure the distance from a triangular face to a *hybrid proxy*, which is a sphere, a circular cylinder or a rolling ball blending surface, expressed in terms of the exact L^2 distances from the three vertices of the triangle to the proxy.

While it is easy to compute the Euclidean distance from a point to a sphere or a circular cylinder, it is not desirable to use the same error term as in [2] when extending proxy

types to general quadric surface, because computing the exact distance from a point to a quadric involves solving for the roots of a degree six univariate polynomial. This is a very time consuming task, because the distance computation needs to be performed for many vertices/triangles of a large mesh in many iterations – according to our test on an implementation based on computation of the exact Euclidean distance, each iteration takes about 20 seconds for a mesh of 10K vertices on a PC with a Xeon(TM) 2.66 GHz CPU. This would render our method too inefficient. We have also tested both algebraic distance $|f|$ and first-order approximation $\frac{|f|}{\|\nabla f\|}$ in our algorithm. But they turned out not to work well due to relatively large approximation errors.

Balancing efficiency and accuracy, we choose to use Taubin's second order approximation of the Euclidean distance $\delta_d(\mathbf{p}, \mathbf{Z}(f))$ [18] from the point $\mathbf{p}$ to the surface $\mathbf{Z}(f)$, which is the zero set of the function f. The function $f(x, y, z)$ is given as:

$$f(x,y,z) = C_0 + C_1 x + C_2 y + C_3 z + C_4 x^2 + C_5 xy + C_6 xz + C_7 y^2 + C_8 yz + C_9 z^2. \quad (2)$$

This approximate distance $\delta_d(\mathbf{p}, \mathbf{Z}(f))$ has the favorable property that it is bounded between 0 and $d(\mathbf{p}, \mathbf{Z}(f))$. For quadric surfaces, $\delta_d(\mathbf{p}, \mathbf{Z}(f))$ is given by the only non-negative root of a quadratic polynomial $g(t) = at^2 + bt + c$, where

$$\begin{aligned} a &= -\left[\|C_1 + < 2C_4, C_5, C_6 > \cdot p\,\|^2 + \|C_2 + < C_5, 2C_7, C_8 > \cdot p\,\|^2 + \right. \\ &\quad \left. \|C_3 + < C_6, C_8, 2C_9 > \cdot p\,\|^2\right]^{1/2}, \\ b &= -\left[(C_5^2 + C_6^2 + C_8^2)/2 + C_4^2 + C_7^2 + C_9^2\right]^{1/2}, \\ c &= |f(p)|. \end{aligned}$$

Based on this approximate distance, the approximated L^2 distance for a triangle t_j with respect to a quadric surface $\mathcal{P}_i$ is defined as:

$$d(t_j, \mathcal{P}_i) = \frac{1}{m} \sum_{k=1}^{m} \delta_d(\mathbf{p}_k, \mathbf{Z}_i(f))^2 \cdot A_j, \quad (3)$$

where $\{\mathbf{p}_k\}_{k=1}^m$ are uniformly sampled points on the triangle t_j, and A_j, the area of t_j, serves as a weighting factor to account for triangles of different sizes. In implementation we set $m = 4$, i.e., use the vertices and the barycenter point of t_i, and have obtained satisfactory results.

The approximated L^2 distance between $\mathcal{R}_i$ and $\mathcal{P}_i$ is then defined as:

$$E'(\mathcal{R}_i, \mathcal{P}_i) = \sum_{t_j \in \mathcal{R}_i} d(t_j, \mathcal{P}_i) / \sum_{t_j \in \mathcal{R}_i} A_j. \quad (4)$$

We use $\widetilde{L}^2$ to denote this approximation to L^2 distance. To have a uniform comparison, all mesh surfaces are scaled uniformly to fit in a rectangular box with their longest side being 1.

2.3 Quadric Surface Fitting

Given a region $\mathcal{R}_i$, we need to fit a quadric surface to $\mathcal{R}_i$ in L^2 metric. Two common ways of fitting implicit surfaces are the algebraic distance-based fitting and orthogonal

distance-based fitting [19]; in general, the latter produces better fitting results than the former but is computationally much more costly. Since surface fitting is performed repeatedly in our present application, it is not necessary to take much time to compute the best fitting result in L^2 metric in each single intermediate iteration. Hence, we use Taubin's method [20] based on a first-order approximation of L^2 metric for quadric surface fitting.

Let $f(x, y, z) = 0$ be a quadric surface (see Eqn. 2). The squared distance from a point $\mathbf{p}$ to the implicit surface $Z(f) = \{(x, y, z)|f(x, y, z) = 0, x, y, z \in \mathbb{R}\}$ is approximated as $d(\mathbf{p}, \mathbf{Z}(f))^2 \approx \dfrac{f(\mathbf{p})^2}{\|\nabla f(\mathbf{p})\|^2}$. The original method proposed by Taubin is applied to a set of data points. Given a set of points $\{\mathbf{p}_i, i = 1, \cdots, n\}$, the sum of approximated squared distance is, following [20]

$$\frac{1}{n}\sum_{i=1}^{n} d(\mathbf{p}_i, \mathbf{Z}(f))^2 \approx \frac{\dfrac{1}{n}\sum\limits_{i=1}^{n} f(\mathbf{p}_i)^2}{\dfrac{1}{n}\sum\limits_{i=1}^{n} \|\nabla f(\mathbf{p}_i)\|^2} = \frac{\mathbf{s}^T M \mathbf{s}}{\mathbf{s}^T N \mathbf{s}},$$

where M, N are coefficient matrices and $\mathbf{s} =< C_0, C_1, \ldots, C_9 >^T$.

For our application, we need to treat the data points as the continuum of surface points distributed uniformly over the mesh surface. Therefore, we adapt Taubin's method by defining the sum of approximated squared distance as follows:

$$\frac{1}{A}\sum_{k=1}^{n_i}\int_{t_k} d(\mathbf{p}, \mathbf{Z}(f))^2 \mathrm{d}\mathbf{p} \approx \frac{\dfrac{1}{A}\sum\limits_{k=1}^{n_i}\int_{t_k} f(\mathbf{p})^2 \mathrm{d}\,\mathbf{p}}{\dfrac{1}{A}\sum\limits_{k=1}^{n_i}\int_{t_k} \|\nabla f(\mathbf{p})\|^2 \mathrm{d}\,\mathbf{p}} = \frac{\mathbf{s}^T M_t \mathbf{s}}{\mathbf{s}^T N_t \mathbf{s}}, \tag{5}$$

where M_t, N_t are coefficient matrices, and A is the sum of the areas of all the triangles in $\mathcal{R}_i$. Hence, the fitting problem is reduced to computing the eigenvector of $M_t - \lambda N_t$ associated with the minimum eigenvalue [20].

Although, due to efficiency consideration, only approximations to the L^2 error metric (i.e., the true squared Euclidean distance) are used in our fitting and partitioning steps, we find that this treatment works robustly and efficiently in practice with the variational shape approximation framework.

3 Variational Quadric Approximation

There are two stages in our method: *global optimization* and *post-processing*. In first stage, the surface $\mathcal{M}$ is partitioned into regions $\mathcal{R}_i$ iteratively according to the error metric defined in the preceding section. Then the boundary curves between neighboring regions are smoothed using a graph cut method and an approximate surface is created by projecting mesh vertices onto their proxies. Fig. 2 illustrates some intermediate steps of our algorithm.

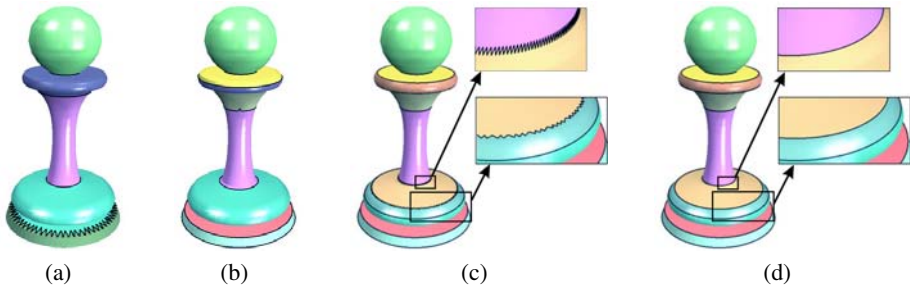


Fig. 2. Intermediate results of our method. New proxies are inserted progressively and the final projected result is shown in Fig. 1(d). (a) Result when the proxy number is 5; (b) Result when the proxy number is 9; (c) Lloyd iteration finishes when proxy number is 12; (d) The boundaries in (c) are smoothed.

3.1 Global Optimization

The main idea of Lloyd iterations and the initialization have been discussed in Section 2. Typically we choose the number of regions $n = 1$ at the beginning and the new proxies are inserted progressively. The algorithm terminates when an error threshold is met and the Lloyd iteration has converged. Proxies are inserted or merged to achieve the optimal approximation, as described below. Other proxy operations, such as proxy deletion or teleportation, have also been implemented, as in [1].

Proxy Insertion. When the Lloyd iteration has converged, we need to check the validity of each quadric proxy. If the quadric surface is a pair of planes or a *hyperboloid of two sheets* and the projected data points are contained in both sheets, then the proxy is considered as *invalid*, because it is not an appropriate representation. If such a case occurs, new proxies will be inserted in such a region. Fig. 3 gives an example of a region fitted by one degenerate quadric consisting of two intersecting planes, which needs to be split into two separate planar proxies.

If every proxy is valid but the fitting error is still larger than a pre-specified threshold, we use the farthest-point criterion to add a new proxy $\mathcal{P}_{new}$; that is, we first find the region with maximal fitting error and a face which has the largest $\widetilde{L}^2$ error belonging to this region as the seed face $\mathcal{S}_{new}$. The new proxy $\mathcal{P}_{new}$ is then set to be the plane containing the seed face $\mathcal{S}_{new}$. Then the Lloyd iteration is continued.

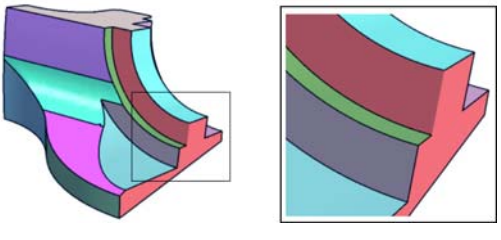


Fig. 3. Left: The red region is fitted by a degenerate quadric consisting of two intersecting planes; Right: Close-up view

Proxy Merging. When the Lloyd iteration converges, we also check whether there are redundant proxies by considering merging each pair of adjacent regions $\mathcal{R}_i$ and $\mathcal{R}_j$. We use a quadric surface $\mathcal{P}_t$ to fit the union of $\mathcal{R}_i$ and $\mathcal{R}_j$. Let E_t be the fitting error. If $|E_t - (E'(\mathcal{R}_i, \mathcal{P}_i) + E'(\mathcal{R}_j, \mathcal{P}_j))| < \epsilon$ (we set $\epsilon = 0.5 * \max_i E'(\mathcal{R}_i, \mathcal{P}_i)$, as in [1]), and $\mathcal{P}_t$ is a valid proxy, then $\mathcal{R}_i$ and $\mathcal{R}_j$ are merged into one region. If there are several pairs can be merged at the same time, the pair with the smallest fitting error is chosen to be merged first.

3.2 Post-processing

Boundary Smoothing. After the global optimization stage, the surface mesh $\mathcal{M}$ has been partitioned into non-overlapping regions $\mathcal{R}_i$, each being fitted by a quadric proxy $\mathcal{P}_i$. Triangle faces next to region boundary curves always have nearly equal $\widetilde{L}^2$ errors, often leading to zigzag boundary curves. The graph cut method has already been used in [6, 15, 5] to segment mesh in the fuzzy region and boundary regularization, but only dihedral angle and edge length are used in their approach, so it works well mainly in regions with salient features or curvature discontinuity. We propose a new graph cut based strategy which is particularly effective for smoothing boundary curves in a smooth region of the mesh.

Consider the dual graph of the original mesh, each triangle face is corresponding to a dual vertex. Given two neighboring regions $\mathcal{R}_0$ and $\mathcal{R}_1$, the faces belonging to the neighbor of their common boundary are marked as belonging to the fuzzy region (Fig. 4(a) illustrates the fuzzy region. The size of neighbor can be set by the user.). Let $\mathcal{V}_f$ denote the set of the dual vertices of the fuzzy region. Suppose that the faces in the fuzzy region are removed from $\mathcal{R}_0$ and $\mathcal{R}_1$. Then the dual vertices of faces in the regions $\mathcal{R}_0$ and $\mathcal{R}_1$ that are adjacent to $\mathcal{V}_f$ are denoted as $\mathcal{V}_0$ and $\mathcal{V}_1$, respectively.

The goal of boundary smoothing is to label the vertices in $\mathcal{V}_f$ with 0 or 1 by minimizing a cost function $E(X)$, this is similar to the binary labeling problem for edge detection widely used in image segmentation. The solution X is a binary vector $X = (x_0, x_1, \ldots)$, $x_i \in \{0, 1\}$. If $v_i \in V_f$ is labeled with 0, i.e., set $x_i = 0$, then its corresponding face is assigned to the region $\mathcal{R}_0$; otherwise, if $x_i = 1$, the face is assigned to the region $\mathcal{R}_1$.

Let $G = \{\mathcal{V}, \mathcal{E}\}$ be an undirected sub-graph of the dual graph of the mesh $\mathcal{M}$, where $\mathcal{V} = \mathcal{V}_f \cup \mathcal{V}_0 \cup \mathcal{V}_1$ is the set of nodes. Here $\mathcal{E}$ is the set of undirected edges, with each dual edge $e = (v_i, v_j), (v_i, v_j \in \mathcal{V}, i \neq j)$ corresponding to an edge shared by two adjacent faces in $\mathcal{V}$. In Fig. 4 the background is composed of two regions $\mathcal{R}_0$ and $\mathcal{R}_1$. The set $\mathcal{V}_0$ consists of the green triangles in $\mathcal{R}_0$, the set $\mathcal{V}_1$ consists of the red triangles

(a) (b)

Fig. 4. Boundary smoothing. (a) Un-smoothed boundary; (b) Smoothed boundary.

in $\mathcal{R}_1$, and the set $\mathcal{V}_f$ consists of those triangles between $\mathcal{V}_0$ and $\mathcal{V}_1$. Here $\mathcal{V}_0$ and $\mathcal{V}_1$ are hard constraints to $\mathcal{R}_0$ and $\mathcal{R}_1$ in the sense that the triangles in both sets will keep their labels; only the triangles in $\mathcal{V}_f$ may be re-labeled.

The energy function $E(X)$ is defined in a similar way to [21] used for image segmentation:

$$E(X) = E_1(X) + \lambda E_2(X) = \sum_{v_i \in \mathcal{V}} \hat{E}_1(x_i) + \lambda \sum_{(v_i,v_j)\in\mathcal{E}} \hat{E}_2(x_i, x_j).$$

In order to keep the triangle faces in the fuzzy region from deviating too much from their quadric proxies and improve boundary smoothness, we consider both the distance from the boundary faces to their proxies and the edge length along the boundary. The region energy term E_1 is determined by how the nodes v_i in $\mathcal{V}_f$ are labeled. Let $d_i^0 = d(v_i, P_0)$ and $d_i^1 = d(v_i, P_1)$ be the $\widetilde{L}^2$ distance of v_i to proxies $\mathcal{P}_0$ and $\mathcal{P}_1$. Then we define

$$\hat{E}_1(x_i = 0) = \begin{cases} 0, & v_i \in \mathcal{V}_0 \\ \infty, & v_i \in \mathcal{V}_1 \\ \dfrac{d_i^0}{d_i^0 + d_i^1}, & v_i \in \mathcal{V}_f \end{cases}; \quad \hat{E}_1(x_i = 1) = \begin{cases} \infty, & v_i \in \mathcal{V}_0 \\ 0, & v_i \in \mathcal{V}_1 \\ \dfrac{d_i^1}{d_i^0 + d_i^1}, & v_i \in \mathcal{V}_f \end{cases}$$

The term $\hat{E}_2$ is the cost of a dual edge connecting two adjacent face nodes $\{v_i, v_j\}$, and defined by

$$\hat{E}_2(x_i, x_j) = \frac{length(i,j)}{length(i,j) + \text{ave_length}} |x_i - x_j|,$$

where $length(i, j)$ is the length of the common edge shared by v_i and v_j, and ave_length is the average edge length of the mesh $\mathcal{M}$. Clearly, $E_2(X)$ becomes larger when the edge length of the cut boundary resulting from re-labeling is longer. The cost function $E(X)$ is minimized using the max-flow/min-cut algorithm described in [22]. Fig. 4(b) shows the result after running the max-flow algorithm.

For different mesh models, two parameters in the above smoothing algorithm can be set by the user in order to obtain satisfactory results. For the CAD models with clear structure, such as the Fandisk and the Chess piece, the one ring neighborhood of the boundary usually suffices; and we use $\lambda = 1.0$ for this type of models in all the examples presented in this paper. For the free-form objects like the Bunny, Homer, the two- or three-ring neighbor of the boundary works well in our experiments; and the value of λ is selected by the user in the interval $[1, 10]$.

Quadric Surface Classification. To simplify the final representation, we would like to identify some commonly used types of special quadrics, such as spheres and circular cylinders, which have occurred as approximating proxies. Given the coefficients of proxy $\mathcal{P}_i$, we detect whether the quadric is nearly a cylinder or a sphere by analyzing the eigenvalues of the corresponding quadratic form [9]. After type identification, the region is fitted by a quadric of the special type that has been identified. Only circular cylinders and spheres are considered as special types in our current implementation.

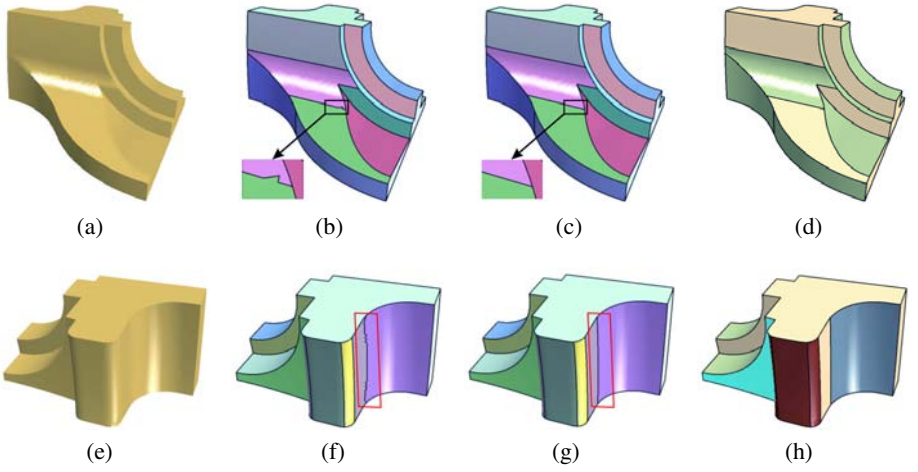


Fig. 5. Fandisk (13K triangle faces): Two views of observation. (a)&(e) The original model; (b)&(f) Partitioned by 22 quadric proxies; (c)&(g) Result after boundary smoothing ($\lambda = 1.0$); (d)&(h) Vertices projected onto proxies.

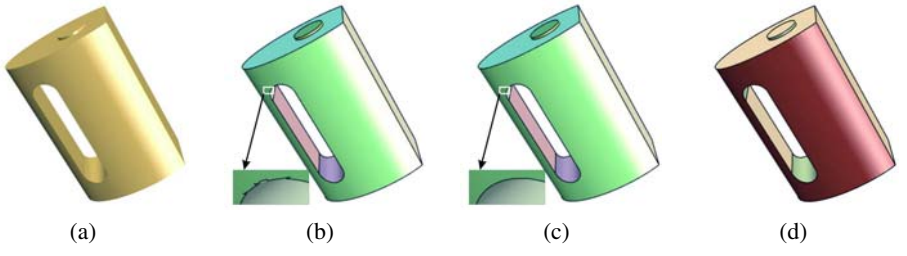


Fig. 6. Tesa [23] (22K triangle faces): (a) The original model; (b) Partitioned by 12 quadric proxies; (c) Result after boundary smoothing ($\lambda = 1.0$); (d) Vertices projected onto proxies

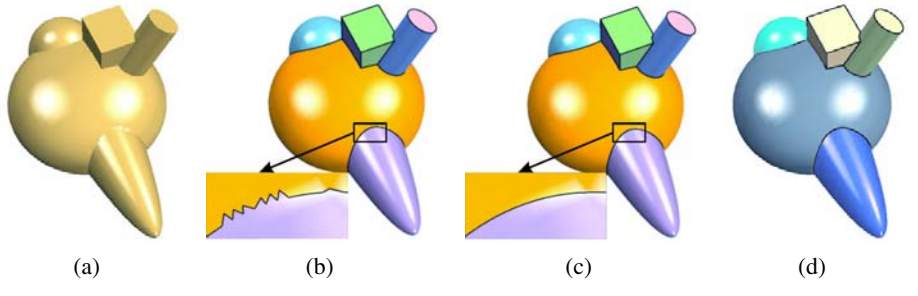


Fig. 7. CSG model (21K triangle faces): (a) The original model; (b) Partitioned by 10 quadric proxies; (c) Result after boundary smoothing ($\lambda = 1.0$); (d) Mesh vertices projected onto proxies

Proxy Projection. As the final step of post-processing, the vertices of each region $\mathcal{R}_i$ of the partitioned mesh $\mathcal{M}$ are projected onto the corresponding proxy $\mathcal{P}_i$ of $\mathcal{R}_i$. The computation of foot points on a plane, sphere or cylinder is straightforward. If the quadric surface belongs to some other types, we compute the exact foot point by solving a degree six univariate equation [9]. For an interior vertex of a region $\mathcal{R}_i$, its projected position is the foot point on the proxy of $\mathcal{R}_i$; if a mesh vertex is shared by two or more regions, the final position is the average of its foot points on all the proxies the vertex belongs to.

4 Results

In this section we present some test examples to show the effectiveness of our method and to compare it with some previous methods that are also based on variational shape approximation. These examples are shown in Fig. 5, 6, 7 and 9. The meaning of colors

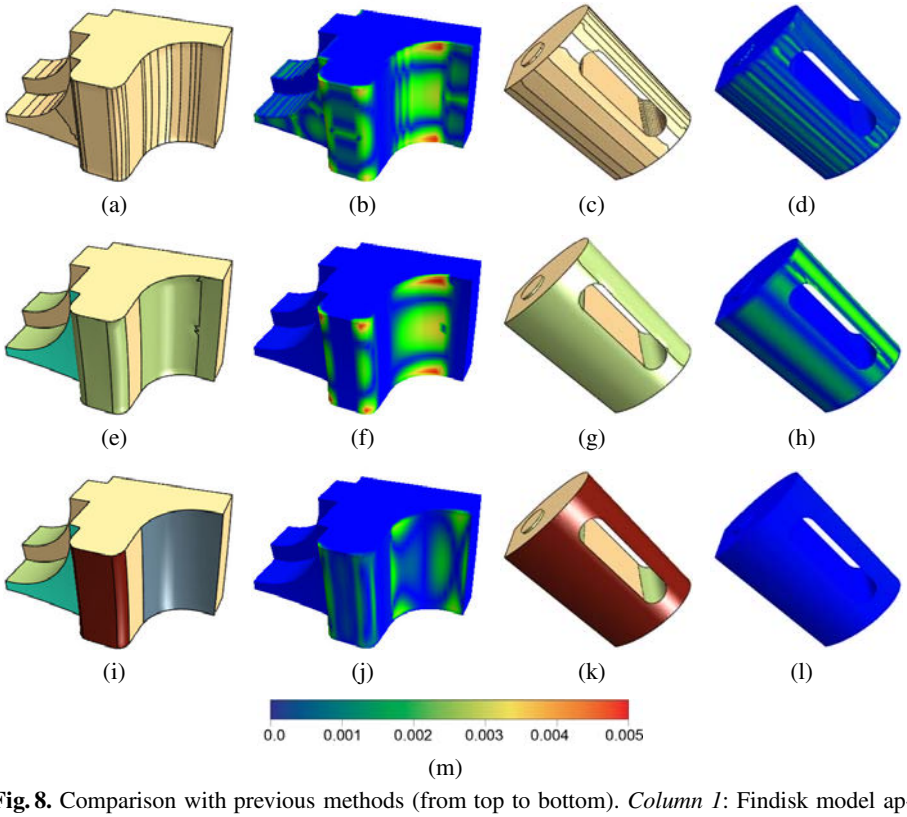


Fig. 8. Comparison with previous methods (from top to bottom). *Column 1*: Findisk model approximated by 80 planar proxies [1], 24 hybrid proxies [2] and 22 quadric proxies by our method; *Column 2*: Color coding of local errors between the approximated model and the original model. The RMS Hausdorff errors are 4.2×10^{-2}, 3.9×10^{-2} and 2.1×10^{-2}, respectively; *Column 3*: Tesa model approximated by 100 planar proxies [1], 14 hybrid proxies [2] and 12 quadric proxies by our method; *Column 4*: Color coding of the local errors. The RMS Hausdorff errors are 3.4×10^{-2}, 4.8×10^{-2} and 4.0×10^{-3}, respectively; (m) Color error bar.

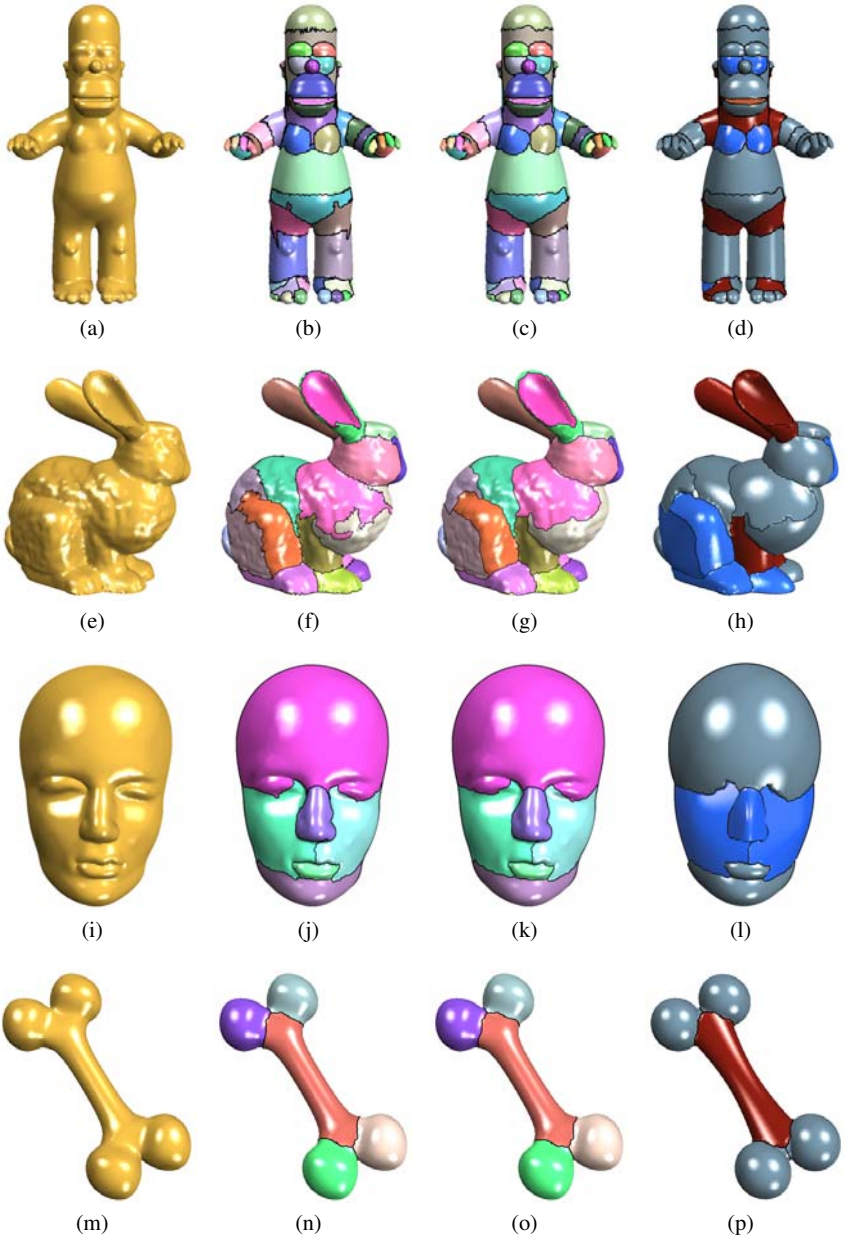


Fig. 9. Four free-form models. From left to right, the four figures on each row are the original model, final partitioned result, result after boundary smoothing, and result after vertex projection. (a)-(d) Homer (40K triangle faces): Approximated by 61 quadric proxies ($\lambda = 6.92$); (e)-(h) Bunny (40K triangle faces): Approximated by 28 quadric proxies ($\lambda = 5.49$); (i)-(l) Mask (62K triangle faces): Approximated by 6 quadric proxies ($\lambda = 4.06$); (m)-(p) Bone (30K triangle faces): Approximated by 5 quadric proxies ($\lambda = 6.2$).

Table 1. Timing Statistics

Mesh	No. of faces	No. of proxies	Lloyd iteration (sec.)	Post-processing (sec.)
Fandisk	13K	22	11	0.017
Tesa	22K	12	18	0.042
CSG	21K	10	16	0.031
Chess	24K	12	15	0.100
Bunny	40K	28	103	0.352
Homer	40K	61	148	0.393
Mask	62K	6	203	0.115
Bone	30K	5	14	0.067

of the projected results is given in Fig. 1(e). All the examples were run on a PC with a Xeon (TM) 2.66GHz CPU. Table 1 gives the running time and other statistics of all the examples.

It can be seen that our method works well for free-form geometry (cf. Fig. 9) as well as for CAD models or CSG objects with salient features (Fig. 5, 6, 7). The RMS Hausdorff errors (divided by the bounding box diagonal) by our method and the methods in [1] and [2] are presented in Fig. 8. It can be seen that the new method gives a more compact or more accurate approximation.

Acknowledgments

We thank Leif Kobbelt of RWTH for discussions and motivation leading to this work. The Bunny model is the courtesy of the Stanford graphics group; the Fandisk, Homer and Mask models are obtained from the aim@shape repository.

References

1. D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *ACM Transactions on Graphics*, 23(3):905–914, 2004.
2. J.H. Wu and L. Kobbelt. Structure recovery via hybrid variational surface approximation. *Computer Graphics Forum*, 24(3):277–284, 2005.
3. S. Lloyd. Least square quantization in PCM. *IEEE Trans. Inform Theory*, 28:129–137, 1982.
4. P. D. Simari and K. Singh. Extraction and remeshing of ellipsoidal representations from mesh data. In *Graphics Interface*, pages 161–168, 2005.
5. D. Julius, V. Kraevoy, and A. Sheffer. D-charts: Quasi-developable mesh segmentation. *Computer Graphics Forum*, 24(3):981–990, 2005.
6. S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics*, 22(3):954–961, 2003.
7. M. Attene, B. Falcidieno, and M. Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer*, 22(3):181–193, 2006.
8. P.J. Besl and R.C. Jain. Segmentation through variable-order surface fitting. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 10(2):167–192, March 1988.
9. A.F. Fitzgibbon, D.W. Eggert, and R.B. Fisher. High-level CAD model acquisition from range images. *Computer-Aided Design*, 29(4):321–330, 1997.

10. A.P. Mangan and R.T. Whitaker. Partitioning 3D surface meshes using watershed segmentation. *IEEE Trans. on Visualization and Computer Graphics*, 5(4):308–321, 1999.
11. P.V. Sander, Z.J. Wood, S.J. Gortler, J. Snyder, and H. Hoppe. Multi-chart geometry images. In *Proceedings of Eurographics Symposium on Geometry 2003*, pages 146–155, 2003.
12. Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H.-P. Seidel. Mesh scissoring with minima rule and part salience. *Computer Aided Geometric Design*, 22(5):444–465, 2005.
13. M. Marinov and L. Kobbelt. Automatic generation of structure preserving multiresolution models. *Computer Graphics Forum*, 24(3):479–486, 2005.
14. R. Liu and H. Zhang. Segmentation of 3D meshes through spectral clustering. In *Proceedings of Pacific Graphics 2004*, pages 298–305, 2004.
15. S. Katz, G. Leifman, and A. Tal. Mesh segmentation using feature point and core extraction. *The Visual Computer*, 21(8-10):649–658, October 2005.
16. G. Lavoué, F. Dupont, and A. Baskurt. A new CAD mesh segmentation method, based on curvature tensor analysis. *Computer-Aided Design*, 37(10):975–987, 2005.
17. A. Shamir. A formulation of boundary mesh segmentation. In *Proceedings of the second International Symposium on 3DPVT*, pages 82–89, 2004.
18. G. Taubin. An improved algorithm for algebraic curve and surface fitting. In *International Conference on Computer Vision*, pages 658–665, 1993.
19. S.J. Ahn, W. Rauh, H.S. Cho, and H.J. Warnecke. Orthogonal distance fitting of implicit curves and surfaces. *IEEE Tran. on Pattern Analysis and Machine Interlligence*, 24(5):620–638, 2002.
20. G. Taubin. Estimation of planar curves, surfaces and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13:1115–1138, 1991.
21. Y. Li, J. Sun, C.K. Tang, and H.Y. Shum. Lazy snapping. *ACM Transaction on Graphics*, 23(3):303–308, Aug 2004.
22. Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Tran. on Pattern Analysis and Machine Interlligence*, 26(9):1124–1137, 2004.
23. Y. Liu, H. Pottmann, and W. Wang. Constrained 3D shape reconstruction using a combination of surface fitting and registration. *Computer-Aided Design (to appear)*, 2006.

Tracking Point-Curve Critical Distances*

Xianming Chen, Elaine Cohen, and Richard F. Riesenfeld

School of Computing, University of Utah, Salt Lake City, UT 84112, USA

Abstract. This paper presents a novel approach to continuously and robustly tracking critical (geometrically, perpendicular and/or extremal) distances from a moving plane point $p \in \mathbb{R}^2$ to a static parametrized piecewise rational curve $\gamma(s)$ $(s \in \mathbb{R})$. The approach is a combination of local marching, and the detection and computation of global topological change, both based on the differential properties of a constructed implicit surface. Unlike many techniques, *it does not use any global search strategy* except at initialization.

Implementing the mathematical idea from singularity community, we encode the critical distance surface as an implicit surface $\mathcal{I}$ in the augmented parameter space. A point $p_s = (p, s)$ is in the augmented parametric space $\mathbb{R}^3 = \mathbb{R}^2 \times \mathbb{R}$, where p varies over $\mathbb{R}^2$. In most situations, when p is perturbed, its corresponding critical distances can be evolved without structural change by marching along a sectional curve on $\mathcal{I}$. However, occasionally, when the perturbation crosses the evolute of γ, there is a transition event at which a pair of p's current critical distances is annihilated, or a new pair is created and added to the set of p's critical distances. To *safely* eliminate any global search for critical distances, we develop *robust* and efficient algorithm to perform the detection and computation of transition events.

Additional transition events caused by various curve discontinuities are also investigated. Our implementation assumes a B-spline representation for the curve and has interactive speed even on a lower end laptop computer.

1 Introduction

Given a plane point p and a closed plane curve $\gamma(s) : \mathbb{R} \to \mathbb{R}^2$, the **squared distance function** $f : \mathbb{R} \to \mathbb{R}$ is defined as,

$$f(p, s) = (\gamma(s) - p)^2. \tag{1}$$

Following the convention of [5], we call the distance from the plane point $\boldsymbol{p}$ to the curve **foot point** $\boldsymbol{\gamma}(s)$ an $\boldsymbol{\mathcal{A}_n}$ **distance**, if $f^{(n+1)}(s)$ is the first non-vanishing derivative at s. An $\mathcal{A}_n$ distance is called a **critical distance** if $n > 0$, and it is further classified as **regular** if $n = 1$, or **degenerate with multiplicity** $\boldsymbol{n}$ if $n > 1$. We generalize the idea of critical distances in Section 8, to include

* This work is supported in part by NSF CCR-0310705 and NSF IIS0218809. All opinions, findings, conclusions or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 87–100, 2006.
© Springer-Verlag Berlin Heidelberg 2006

local extremal distances from p to C^0 points on a piece-wise smooth curve. In this paper, we encode a critical distance from p to the curve point $\gamma(s)$ as (p, s), and consequently regard it as a point in $\mathbb{R}^3 = \mathbb{R}^2 \times \mathbb{R}$ (cf. Fig. 1b).

Critical distances, especially when generalized to those between a space point and a surface, are of interest in many geometric computations including minimal distance computation (e.g., [11, 12]), collision detection (e.g. [18, 19]), and applications in motion planning and haptic rendering. It also has close relation to medial axis transformation [2] and Voronoi diagrams (e.g., see [6] for the closed smooth curve case). In this paper, we consider the issues of computing all *topological changes to the critical distances*as p moves in the plane, and evolving critical distances where there is no topological change. By *topology of critical distances*, we mean the total number of critical distances, and the type of each one (i.e., whether it is local minimal, local maximal, or degenerate with certain multiplicity). To track the critical distances, it is necessary to start with an initial point position and *all* the corresponding critical distances for that location. Typically, this is done by solving Eq. (2) (see Section 3) using some constraint solver as discussed in [26, 8]. We will not go into more details on this, and simply assume that the global initialization, including all critical distances, is given.

In this paper, we are especially interested in a *topological change* of critical distance, called a **transition event**. Mathematically, this is related to singularity and catastrophe theories [1, 23, 14, 22]. Most recently, [5] defines the extended curve evolute to serve as the transition set of critical distances on *piecewise smooth* curves, especially deriving *algebraically* all the unfolding formulas for degenerate critical distances. This paper deals with practical implementation issues, including especially the robust and efficient detection of transition events. The set of all critical distances for all possible planar points is formulated as an implicit surface $\mathcal{I}$ in the augmented parametric space $\mathbb{R}^3 = \mathbb{R}^2 \times \mathbb{R}$, and subsequently, the evolution and transition of critical distances are performed using first and second order shape computation on $\mathcal{I}$.

2 Motivation

We present a set of algorithms to track point-curve critical distances exactly, continuously, and robustly. In this implementation,the critical distances from the point to the static curve are updated interactively as the user moves the point (by mouse) on the plane without restriction. The critical distance tracking does not approximate the curve with a polyline, as is done frequently for distance and collision queries. While point-curve distance is important on its own, we are mostly motivated by its future extension to the surface case, that is, tracking the point-surface or surface-surface critical distances, and our ultimate goal is the continuous distance tracking between two trimmed NURBS models both under either rigid motion or more general deformation. Many techniques in both haptic rendering and motion planning discretely approximate the curves [9, 20, 13]. While [10, 21] works directly on the NURBS models, a global minimal distance search still must be conducted periodically.

Transition events of critical distances are important to be able to ensure for algorithm robustness and efficiency. If every transition event is detected and the corresponding topological change of critical distances is computed, then one can guarantee that no any critical distances are missed, and be assured of *robustly* reporting the global minimum or maximum. Furthermore, this also eliminates typically expensive (compared to local updating) global search, which gives an efficiency benefit.

For the point-curve case, the transition set is identified either with the evolute of the curve if the considered curve is piecewise smooth with at least C^2 continuity at its break points [14], or with the extended evolute if the piecewise smooth curve is C^0 continuous [5]. For the point-surface case, the transition set is the two focal surfaces [15, 22]. More complex situations arise for the point-model case when the model consists of multiple trimmed NURBS surfaces with C^0 continuity between surfaces and for the surface-surface and model-model cases. On the other hand, the basic idea of replacing the global extremal distance search with robust transition detection and computation, has a straightforward extension to higher dimension, and we present the current work as a first step toward that more ambitious goal.

The rest of the paper is organized as follows. Section 3 presents the problem formulation in the augmented parametric space. Section 4 performs the evolution of the critical distance by marching locally on a sectional curve $\mathcal{I}_\delta$ on $\mathcal{I}$. An osculating circle based correction algorithm is presented in Section 4.1. Section 5 computes the newly created pair of critical distances by contouring the local osculating parabola to $\mathcal{I}_\delta$. By exploiting the rational B-spline representation for the evolute, robust and efficient detection of transition events is investigated in Section 6 using bounding volume tree of the curve evolute. Section 7 presents a way to classify the transition event by using at the sign of $\kappa\kappa'$. To apply our approach to realistic curve models, Section 8 develops algorithms for the additional transition events caused by various curve discontinuities that occur in real models. After examples in Section 9, conclusions are presented in Section 10.

3 Implicit Surface Formulation in the Augmented Parametric Space

The condition for critical distances between point $p \in \mathbb{R}^2$ and plane curve $\gamma(s) \in \mathbb{R}^2$ is,

$$f' = (\gamma(s) - p) \cdot \gamma(s)' = 0. \tag{2}$$

Regarding the LHS of Eq. (2) as a function of $g = f' : \mathbb{R}^3 \to \mathbb{R}$, the locus of all critical distances, as points (p, s) in $\mathbb{R}^3$, is the zero set of g. The Jacobian of g,

$$\mathcal{J} = (\nabla g) = \begin{pmatrix} -\gamma_0' \\ -\gamma_1' \\ (\gamma - p) \cdot \gamma'' + \|\gamma'\|^2 \end{pmatrix}, \tag{3}$$

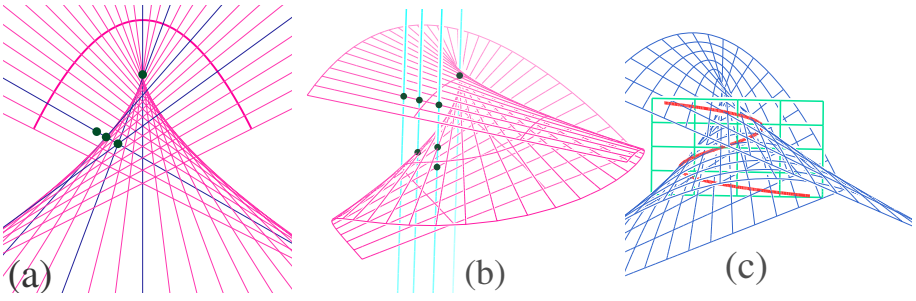


Fig. 1. Implicit Surface Formulation of Point-Curve Critical Distances. (a) shows the normal bundle to a parabola curve; also shown are 4 plane points with their corresponding critical distances (shown as perpendicular lines to the curve). From left to right, they have one regular critical distance (CD), one regular plus another degenerate (with multiplicity 2) CD, 3 regular CD, and a degenerate (with multiplicity 3) CD to the curve, respectively. Lifting into 3-space, in (b), the corresponding vertical lines pierce $\mathcal{I}$ once, pierce once and touch once (on the fold), pierce three times, and pierce once (at the cusp of the fold), respectively. Finally a sectional curve of $\mathcal{I}$ is shown in (c).

always has rank 1 when γ is regular; so the 0-set of g is a 2-manifold in $\mathbb{R}^3$, denoted hereafter as $\mathcal{I}$. Geometrically, $\mathcal{I}$ is the lifted normal bundle to the curve (Fig. 1 (a)&(b)), and is called the catastrophe surface in [14].

Notice that the normal to $\mathcal{I}$ actually has the same expression as the RHS of Eq. (3), and can be succinctly written as

$$N_{\mathcal{I}} = -\gamma' + \mathfrak{D} e_s. \tag{4}$$

where e_s is the unit vector along the vertical s-axis in $\mathbb{R}^3$, $\gamma' \in \mathbb{R}^2$ is regarded as $\gamma' \in \mathbb{R}^3$ in a natural way (i.e., the last component is 0), and $\mathfrak{D} = (\gamma - p) \cdot \gamma'' + \|\gamma'\|^2$.

Finally, we recall several identities which are used in this paper.

$$e_s \times a = a_{\mathfrak{r}}, \qquad a \times b = (a_{\mathfrak{r}} \cdot b) e_s, \qquad (a_{\mathfrak{r}})_{\mathfrak{r}} = -a, \qquad a_{\mathfrak{r}} \cdot b_{\mathfrak{r}} = a \cdot b \tag{5}$$

where a and b are vectors in $\mathbb{R}^2$, and the subscript $\mathfrak{r}$ denotes a 90 degree rotation around the positive s-axis.

4 Evolution

Given any perturbation of p, δ_p, the evolution problem is to transform all critical distances corresponding to p to those corresponding to $\hat{p} = p + \delta_p$. Geometrically, the set of critical distances $\{(p, s)\}$, for a fixed p, is the set of intersection points of $\mathcal{I}$ with the vertical line passing through p. There may be several critical distances for a fixed p corresponding to different values of s. Analogously, the set of critical distances $\{(p + \delta_p, s + \delta_s)\}$ is the set of intersection points of $\mathcal{I}$

with the vertical line passing through $\hat{p} = p + \delta_p$ (cf. Fig. 1b). In the following, we consider a particular critical distance (p, s) only *locally* .

First, construct the sectional curve, $\mathcal{I}_\delta$, on $\mathcal{I}$, that is, the intersection of $\mathcal{I}$ with a vertical plane $\mathcal{P}$ passing through both plane points p and $p+\delta_p$ (Fig. 1c).

By Eq. (4), and since $\mathcal{P}$ has normal $N_{\mathcal{P}} = (\delta_p)_{\mathfrak{r}}$, the tangent to $\mathcal{I}_\delta$ is,

$$T = N_{\mathcal{P}} \times N_{\mathcal{I}} = (\delta_p)_{\mathfrak{r}} \times (-\gamma' + \mathfrak{D} e_s) = -(\delta_p)_{\mathfrak{r}} \times \gamma' + \mathfrak{D}(\delta_p)_{\mathfrak{r}} \times e_s.$$

By Eq. (5),

$$T = (\delta_p \cdot \gamma')\ e_s + \mathfrak{D}\ \delta_p. \tag{6}$$

Therefore, the critical distance (p, s) is evolved to $(p + \delta_p, s + \frac{\delta_p \cdot \gamma'}{\mathfrak{D}})$, assuming that $\mathfrak{D}$ does not vanish (otherwise, there has to be a transition event, which is discussed later in Section 5).

4.1 Correction

The local evolution of critical distances, as just described, essentially uses a tangent plane approximation to the implicit surface $\mathcal{I}$, and there is error accumulated over time. We develop a curvature-based correction algorithm in this section.

The basic idea is to approximate the local curve with its osculating circle (Fig. 2(a)). Suppose that for a plane point p, one of its approximate critical distances has a foot point F. However, p is not really on the normal line to F; instead, it deviates from the normal line by an angle $d\alpha = \angle FCp$, where C is the curvature center corresponding to F. Recall that, for a plane curve, the signed curvature is the rate of change α with respect to arc length, i.e., $\kappa = \frac{1}{\|\gamma'\|}\frac{d\alpha}{ds}$; so,

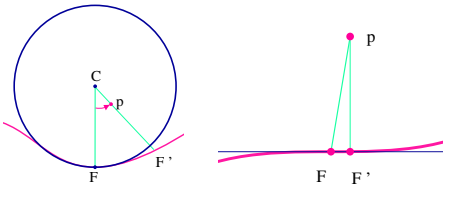


Fig. 2. Circle/Tangent Approx

$$ds = \frac{d\alpha}{\kappa \|\gamma'\|}. \tag{7}$$

If F is near a point with $\kappa = 0$, we replace the osculating circle based correction algorithm by a tangent based one (Fig. 2(b)),

$$ds = \frac{\|F' - F\|}{|\gamma'\|} = \frac{(p - F) \cdot \gamma'}{\|\gamma'\|^2}. \tag{8}$$

5 Transition

If $\mathfrak{D} = 0$, $\mathcal{I}_\delta$ is locally vertical at the considered $(p, s) \in \mathbb{R}^3$ (cf. Eq. (6)), then there is no way of evolving (p, s) to the next approximate critical distance by

following $\mathcal{I}_\delta$ tangentially. Mathematically, the locus of such points (p, s) forms the fold singularity of projection of $\mathcal{I}$ on s-axis. For C^2 curve case, the projection of the fold is actually the evolute of γ [14].

When the plane point moves across a point on the evolute, there will be a transition event, i.e., a structural change to the form of the critical distances. Therefore, all points on the evolute (or the extended evolute, for piecewise smooth curves later in Section 8) are called **transition points**. At a transition point, the corresponding critical distance, is degenerate with multiplicity 2. Away from the transition point, the critical distance disappears completely in one direction, and unfolds into *a pair* of critical distances in the other direction. They are called an **annihilation event** and a **creation event**, respectively. Notice that the created pair of critical distances have to be of opposite types (one minimum, and the other maximum). Detailed algebraic derivations related to transition events are given in [5].

In this section, we initialize the created pair of critical distances by second order differential computation on $\mathcal{I}_\delta$, i.e, more specifically, by contouring the local osculating parabola to $\mathcal{I}_\delta$.

Eq. (6) gives the tangent vector field on the curve $\mathcal{I}_\delta$, and it allows us to compute the covariant derivative with respect to itself. At a singular point where $\mathfrak{D} = 0$ and $T = (\delta_p \cdot \gamma')\ e_s + \mathfrak{D}\ \delta_p = (\delta_p \cdot \gamma')\ e_s$ (cf. Eq. (6)), the curvature of $\mathcal{I}_\delta$ is (we denote, generally, $\boldsymbol{\kappa}$ as the unit normal of the curve under consideration, scaled by the signed curvature κ),

$$\begin{aligned}\boldsymbol{\kappa}_{\mathcal{I}_\delta} &= \frac{(T \times \nabla_T T) \times T}{T^4} = \frac{\left((\delta_p \cdot \gamma')\ e_s \times (\delta_p \cdot \gamma') \frac{\partial T}{\partial s}\right) \times (\delta_p \cdot \gamma')\ e_s}{(\delta_p \cdot \gamma')^4} \\ &= \frac{\left(e_s \times \frac{\partial \left((\delta_p \cdot \gamma')\ e_s + \mathfrak{D}\ \delta_p\right)}{\partial s}\right) \times e_s}{\delta_p \cdot \gamma'} = \frac{\partial \mathfrak{D}}{\partial s}\ \frac{(e_s \times \delta_p) \times e_s}{\delta_p \cdot \gamma'} = \frac{\mathfrak{D}'}{\delta_p \cdot \gamma'}\ \delta_p \qquad (9)\end{aligned}$$

where (see detail in [3]),

$$\mathfrak{D}' = \frac{\partial \mathfrak{D}}{\partial s} = \left((\gamma - p) \cdot \gamma'' + \|\gamma'\|^2\right)' = -\|\gamma'\|^2 \frac{\kappa'}{\kappa}. \qquad (10)$$

Assume that p is originally at a transition point with a degenerate critical distance (p, s) of multiplicity 2. Further,assume that p is perturbed by δ_p. If $\frac{\mathfrak{D}'}{(\delta_p \cdot \gamma')} > 0$, then $\boldsymbol{\kappa}_{\mathcal{I}_\delta}$ has the same direction as δ_p by Eq. (9), or the perturbation direction is toward the curved side of $\mathcal{I}_\delta$. Thus, a pair of new critical distances are created by this perturbation (cf. Fig 1c). Approximating the local curve of $\mathcal{I}_\delta$ by its osculating parabola,

$$\delta_p = \frac{1}{2}\boldsymbol{\kappa}_{\mathcal{I}_\delta}\delta_s^2 = \frac{1}{2}\frac{\mathfrak{D}'}{(\delta_p \cdot \gamma')}\delta_s^2\ \delta_p,$$

so,

$$\frac{1}{2}\frac{\mathfrak{D}'}{(\delta_p \cdot \gamma')}\delta_s^2 = 1.$$

Therefore, the pair of critical distances are $(p + \delta_p, s + \delta_s)$ and $(p + \delta_p, s - \delta_s)$, where δ_s is,

$$\delta_s = \sqrt{\frac{2(\delta_p \cdot \gamma')}{\mathfrak{D}'}}. \tag{11}$$

On the other hand, a perturbation away from the curved side of $\mathcal{I}_\delta$, causes the original critical distance to disappear. It is clear that which pair of critical distances must be annihilated, given the fact since there is no other critical distance between (p, s_0) and (p, s_1), that is, for with $s \in (s_0, s_1)$.

In addition to $\mathcal{A}_2$ critical distances, there are also $\mathcal{A}_3$ or even higher degenerate ones[5]. $\mathcal{A}_3$ critical distances correspond to *isolated* ordinary cusps on the evolute. and therefore, due to numerical error and/or intentional numerical perturbation, the creation events can be safely assumed to be of only $\mathcal{A}_2$ type. However, for a robust tracking algorithm, special implementation is required for the situation when the plane point is close to a cusp point (other type of cusps may also arise on an extended evolute, cf. Fig. 4). See [3] for details.

6 Detecting Transition Events

In principle, detecting a transition event is not more complicated than a special curve-curve intersection problem [17, 16, 25, 24]. We use the interval subdivision method [16] to do the subdivision, and to construct a bounding volume tree[1](BVT) from the axis aligned bounding boxes resulting from the interval subdivision. For most situations, the BVT allows the intersection algorithm to stop at a very early stage. In this section, the word "intersection" will mean *line-diagonal* intersection, while the word "hit" will refer to the intersection of the line with the box *edges*.

Construct Evolute BVT by Interval Subdivision. Interval subdivision requires a pre-processing of breaking the initial curve at any point where a component of the curve derivative vanishes. However, the evolute is not a regular curve, and the pre-precessing also needs to split the evolute at both its asymptote and cusp points (see Fig. 4). See [3] for more implementation details.

Line Hits Axis Aligned Box. The first stage of transition detection checks if a parametrized line $\mathcal{L}(t)$, with $\mathcal{L}(0) = p$ and $\mathcal{L}(1) = \hat{p}$, hits an axis aligned box. While this is essentially the typical ray tracing algorithm [27], more hit information is required for the next stage algorithm. Specifically, if there is a hit, the algorithm should compute the following three pieces of information for both near and far hit points.

1. the hit edge.
2. the ratio of hit point with respect to $p\hat{p}$, i.e., parameter t of hit point.
3. the ratio of hit point with respect to the hit edge.

[1] Notice that the volume here is the 2-dimensional area on the plane.

If both hit points are outside the line segment $p\hat{p}$, $p\hat{p}$ cannot intersect the box diagonal. Therefore there is no transition event, and the algorithm stops.

Line Segment Intersects Box Diagonal. A leaf node axis aligned box of the BVT, is constructed simply from the 2 end points of the control polygon resulting from interval subdivision. Throughout this section, "diagonal"is used only to mean the diagonal that connects these 2 end points. Notice that the diagonal segment is supposed to approximate the local curve, and has parameters for both of its ends that are *on* the evolute.

The second stage of the transition detection checks if the line segment intersects the diagonal of the hit box, and, if so, computes the ratio of the intersected point with respect to the two end points of the diagonal. The intersection point is an approximation to the real intersection point of $p\hat{p}$ with the curve evolute, and its parameter is *interpolated* from those of the two diagonal ends instead of the simple midpoint approximation.

Remark 1. An interpolation based approach gives more accurate result than the midpoint approximation, which is highly desirable because, at a transition point, perturbing the point p by δ_p causes the corresponding critical distance foot points to perturb by $\sqrt{\|\delta_p\|}$. For details see Lemma 1 in [5].

Three generic situations of line-diagonal intersection are illustrated in Fig. 3. It is either a through-intersection, when the near and far hit points are on the opposite edges, or a corner-intersection, when they are on the neighboring edges. On the other hand, considering the relative orientation of the diagonal segment qq' with respect to the line segment pp', either $qq'[i]$ has the same sign as $pp'[i]$ for *both* $i = 0$ and $i = 1$, or the sign relations are opposite for $i = 0$ and $i = 1$ (the boolean array $diag[2]$ in Algorithm 1 keeps this information).

Based on this classification, and by constructing the auxiliary similar triangles (shaded in Fig. 3), the ratio of intersection point with respect to the diagonal is computed. See details in Algorithm 1.

7 Classification of Transition Types

If Algorithm 1 returns *true*, then the line segment $p\hat{p}$ (i.e., the perturbation) intersects the evolute so a transition event occurs. If the perturbation is toward curved side of $\mathcal{I}_\delta$, the transition event is of creation type: otherwise it is of annihilation type(cf. Fig. 1c). By Eq. (9), the perturbation direction is toward the curved side of the sectional curve $\mathcal{I}_\delta$, or $\boldsymbol{\kappa}_{\mathcal{I}_\delta}$ has the same direction as δ_p, if and only if $\mathfrak{D}'(\gamma' \cdot \delta_p) > 0$. Hence, evaluation of $\mathfrak{D}'$ and $\gamma' \cdot \delta_p$ at the transition point will determine the exact transition type. However, by Eq. (10), $\mathfrak{D}'$ only changes sign at isolated curve points where $\kappa\kappa'$ changes sign; therefore, the evaluation of $\mathfrak{D}'$, which involves the third order derivative, is not necessary at run time, provided that all the sign flipping points of $\kappa\kappa'$ are already pre-computed.

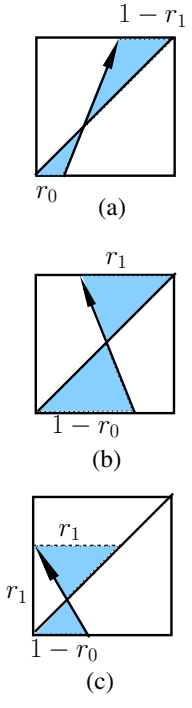


Fig. 3. Generic Line Box-Diagonal Intersection

Input:
diag[**2**] box diagonal direction w.r.t. $\mathbf{p\hat{p}}$
$(\mathbf{e_0}, \mathbf{t_0}, \mathbf{r_0})$ near hit point
$(\mathbf{e_1}, \mathbf{t_1}, \mathbf{r_1})$ far hit point
Output: on *true* return,
t ratio of intersected point w.r.t. $\mathbf{p\hat{p}}$
r ratio of intersected point w.r.t. diagonal
Return:
true if intersected (i.e. $t \in [0, 1]$), *false* otherwise
Begin
If **diag**[**0**] $\neq$ **diag**[**1**]
$\mathbf{t} \Leftarrow (\mathbf{1} - \mathbf{r_0})/(\mathbf{1} - \mathbf{r_0} + \mathbf{r_1})$
$\mathbf{i} \Leftarrow$ *is_horizontal*$(\mathbf{e_0})$? **0** : **1**
diag[**i**] $\Leftarrow$! **diag**[**i**]
Else
If corner cut situation **Return** *false*
$\mathbf{t} \Leftarrow \mathbf{r_0}/(\mathbf{r_0} + \mathbf{1} - \mathbf{r_1})$
$\mathbf{r} \Leftarrow \mathbf{t} \Leftarrow (\mathbf{1}-\mathbf{r})\mathbf{t_0}+\mathbf{r}\mathbf{t_1}$
If $\mathbf{t} \notin [\mathbf{0}, \mathbf{1}]$ **Return** *false*
If case (c) in Fig. 3 $\mathbf{r} \Leftarrow \mathbf{r} * \mathbf{r_1}$
If(! **diag**[**0**]) $\mathbf{r} \Leftarrow \mathbf{1} - \mathbf{r}$
Return *true*
End

Algorithm 1. Line Segment Intersects Box Diagonal

The following algorithm determines the sign of $\kappa\kappa'$, without evaluating κ'.

Algorithm 2. Pre-computing signs of $\kappa\kappa'$

1. *Split the curve at all locations where $\kappa = 0$, at critical curvature points, and at all points with continuity $C^{(<3)}$,*
2. *Evaluate the curvature at all split points. The evaluation may be not necessary, if the point is 0-curvature point or it may need to be performed twice for left and right limit evaluations at a $C^{(<2)}$ point.*
3. *For each segment, tag the sign of κ to be the same as that of either of its non zero curvature end points, and tag the sign of κ' to be the same as the difference of κ at its two end points.*

The algorithm requires as input the 0 and critical curvature points, which are computed using a NURBS combined symbolic-numeric computation [7], with degree reduction strategy as presented in [4]. Fig. 4 shows all the flipping points of $\kappa\kappa'$ for a quadratic B-spline curve.

$\boldsymbol{\kappa' = 0}$: 1, 3, 5, 7, 9, 11
vertices on the curve, ordinary cusps on the extended evolute.

$\boldsymbol{\kappa_l' \kappa_r' < 0}$ **and** $\boldsymbol{\kappa_- \kappa_+ > 0}$: 2, 8, 10, 12
C^1 breaks on the curve, cusps on the extended evolute.

$\boldsymbol{\kappa_l' \kappa_r' > 0}$ **and** $\boldsymbol{\kappa_- \kappa_+ < 0}$: 4, 6 C^1 breaks on the curve, asymptotes on the extended evolute.

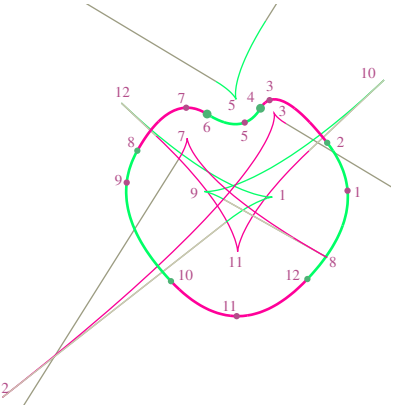


Fig. 4. Flipping Points of $\kappa\kappa'$ of a quadratic B-spline curve

8 Additional Transition Events at Curve Break Points

In this section, we make extension to our algorithms so that critical distances can be tracked across curve break points of at least C^0 continuity. Notice that the corresponding $C^{(-1)}$ situation is not any more difficult, and is omitted here under the assumption that a curve boundary to a 2D shape must be closed.

C^2 Break Points. First, observe that, by Eq. (6), Eq. (7) and Eq. (11), as long as the curve is C^3, the algorithms are valid. The requirement, however, can be relaxed to C^2. A C^2 point of the curve corresponds to a C^1 point on the implicit surface $\mathcal{I}$, and so does not affect the evolution algorithm. On the other hand, it does affect the transition computation based on $\boldsymbol{\kappa}_{\mathcal{I}_\delta}$, because $\boldsymbol{\kappa}_{\mathcal{I}_\delta}$ is $C^{(-1)}$ by Eq. (9). A simple solution is to evaluate the left and right limits of Eq. (11). However, observing that there are only isolated transition points on the evolute that correspond to C^2 break points on the curve this could rarely happen due to numerical error.

C^1 Break Points. Usually one point of any lifted normal line is on the fold of $\mathcal{I}$, and the projection of that point to $\mathbb{R}^2$ is on the evolute, or it is the curvature center (cf. Fig. 1b). However, if the considered normal line corresponds to a C^1 curve point, there is a segment on a fold of $\mathcal{I}$. There is a creation or an annihilation of a pair of critical distances when the perturbation crosses any point of that segment. The normal line segment, serving as extra transition points, is either the line segment connecting the two (left limit and right limit) curvature centers, or the complement of it with respect to the whole normal line (see [5] for more details). The important thing to note, though, is that the apparent transition event occurs because of 2 *evolution* events, one performed on the left segment and the other performed on the right segment. The following algorithm computes the transition event or evolution event at a point (p, s) where γ is C^1 at s, given a perturbation of δ_p. In the rest of the paper, a subscript of l (r) denotes the left (right) limit evaluation.

Algorithm 3. Transition/Evolution at C^1 Break

1. *If* $(\delta_s)_l = \frac{\delta_p \cdot \gamma'}{\mathfrak{D}_l} < 0$, $(p + \delta_p, s + (\delta_s)_l)$ *is a perturbed critical distance.*
2. *If* $(\delta_s)_r = \frac{\delta_p \cdot \gamma'}{\mathfrak{D}_r} > 0$, $(p + \delta_p, s + (\delta_s)_r)$ *is a perturbed critical distance.*

A creation/evolution/annihilation event happens if two/one/none perturbed distances are returned from the algorithm.

C^0 Break Points. At a C^0 curve break point, there are 2 normal lines, and each of the lifted ones has some segment on the fold of the implicit surface $\mathcal{I}$. We can do the transition computation directly for C^0 break points, but, instead use the strategy in [5] to convert C^0 break point into two (collapsed) C^1 break points by inserting an arc with 0-radius between the two unfolded break points. The arc has tangent continuity at both its ends, and has positive (negative) infinite curvature if the two tangents at the left and right ends form a right (left) hand rotation. Notice that this essentially assigns a whole span of normal lines to a C^0 point, generated by right (left) rotating the left limit normal to the right limit normal; consequently, there will be an extra critical point if the plane point is on any of these normal lines.

9 Examples

The two examples in this section are snapshots taken from dynamically tracking critical distances from a moving (user interactive with a mouse) point to a static curve. Demo videos are accessible by following the link
http://www.cs.utah.edu/∼xchen/papers/more.html

Fig. 5 gives an example of continuously tracking critical distances on a cubic B-spline curve, with 6 snapshots taken from the animation. The plane point is shown in dark square box, and foot points are shown in filled circles, while the corresponding points on the evolute (light gray) are shown in unfilled circles. At some point between each pair of neighboring snapshots, 5 transition events have occurred. The first transition annihilates a pair of critical distances, while the last one is actually two transition events, each of which annihilates a pair of critical distances. Each of the rest of the transitions creates a pair of critical distances. The pair that becomes annihilated is shown in boxes, while the created pair, in larger filled circles.

Fig. 6 shows extremal distance tracking on a C^0 B-spline curve. Only part of the extended evolute curve is shown in light color (details in [5, 3]).

10 Conclusion

In this paper, we have formulated the set of critical distances from a plane point to a curve, as an implicit surface $\mathcal{I}$ in the augmented parametric space. Evolution and transition of critical distances are performed using first and second order differential computation on $\mathcal{I}$, respectively. Detection of transitions is robustly and

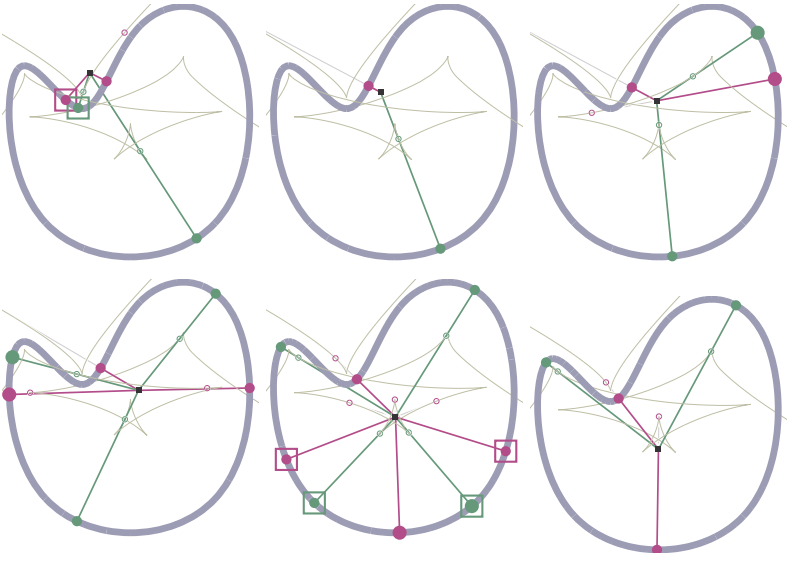


Fig. 5. Example 1: Tracking Critical Distances on a C^2 B-spline Curve

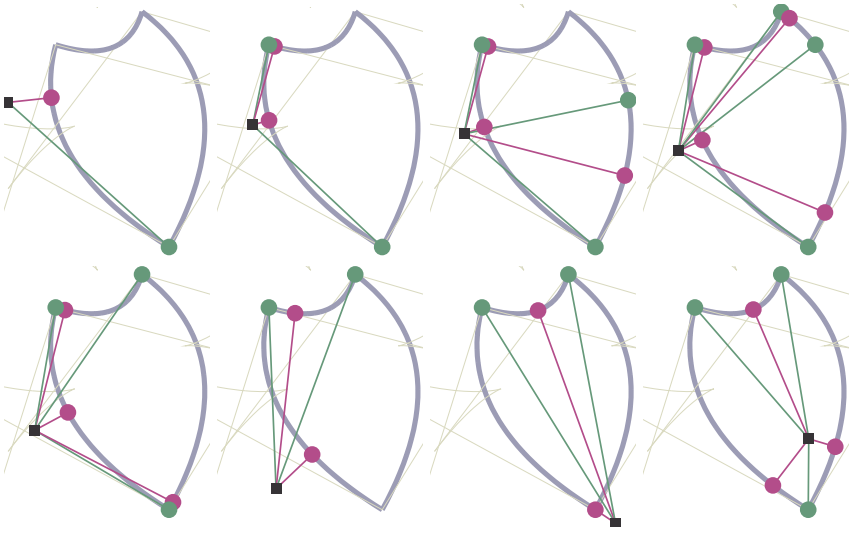


Fig. 6. Example 2: Tracking Critical Distances on a C^0 B-spline Curve

efficiently implemented using the BVT of the curve evolute. Additional transition events, corresponding to breaks in the curve smoothness, are also computed.

We have not yet mentioned global minimal/maximal distance tracking because, to robustly track *global minimal/maximal* distance, all the *local critical* distances must be tracked. Because critical distances always occur with alternat-

ing types[2] (cf. Section 8), the global minimal/maximal distance can be computed at run time by comparing all current critical distances.

With the techniques in this paper, point-curve critically distances can be continuously and efficiently tracked, without resorting to global searches. Our future research will include extending the approach to design robust algorithms for the point-model and the model-model cases.

References

[1] V. Arnold, *Catastrophe Theory*, 3 edition, Springer-Verlag, 1992.

[2] H. Blum, "A transformation for extracting new descriptors of shape," *Models for the perception of speech and visual forms*. 1967, pp. 362–380, MIT Press.

[3] X. Chen, "Dynamic Geometric Computation by Singularity Detection and Shape Analysis," *Ph.D. Thesis Manuscript*, 2006.

[4] X. Chen, R. Riesenfeld, and E. Cohen, "Degree Reduction Strategies for NURBS Symbolic Computation," *To Appear SMI'06*, 2006.

[5] X. Chen, R. Riesenfeld, and E. Cohen, "Extended Curve Evolute as the Transition Set of Distance Functions," *Under submission*, 2006.

[6] J. J. Chou, "Voronoi diagrams for planar shapes," *IEEE Computer Graphics and Applications*, vol. 15, no. 2, 1995, pp. 52–59.

[7] G. Elber, "Free Form Surface Analysis using a Hybrid of Symbolic and Numeric Computation," *Ph.D. thesis, University of Utah, Computer Science Department*, 1992.

[8] G. Elber and M.-S. Kim, "Geometric constraint solver using multivariate rational spline functions," *Symposium on Solid Modeling and Applications*, 2001, pp. 1–10.

[9] A. Gregory, M. C. Lin, S. Gottschalk, and R. Taylor, "A Framework for Fast and Accurate Collision Detection for Haptic Interaction," *IEEE VR 1999*, 1999.

[10] T. V. T. II and E. Cohen, "Direct Haptic Rendering Of Complex Trimmed NURBS Models," *ASME Proc. 8th Annual Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, Nov. 1999.

[11] D. Johnson and E. Cohen, "A framework for efficient minimum distance computations," *Proc. IEEE Intl. Conf. Robotics and Automation*, May 1998, pp. 3678–3684.

[12] D. Johnson and E. Cohen, "Bound Coherence for Minimum Distance Computations," *IEEE Proc. International Conference on Robotics and Automation*, 1999.

[13] D. E. Johnson, P. Willemsen, and E. Cohen, "A Haptic System for Virtual Prototyping of Polygonal Models," *DETC 2004*, 2004.

[14] J.W.Bruce and P.J.Giblin, *Curves And Singularities*, 2 edition, Cambridge University Press, 1992.

[15] J. J. Koenderink, *Solid Shape*, MIT press, 1990.

[16] P. Koparlar and S. P. Mudur, "A new class of algorithms for the processing of parametric curves," *Computer-Aided Design*, vol. 15, 1983, pp. 41–45.

[17] J. M. Lane and R. F. Riesenfeld, "A theorectical development for the computer generation and display of piecewise polynomial surfaces," *IEEE Trans. PAMI*, vol. 2, 1980, pp. 35–46.

[2] A degenerate critical distance of neither minimum nor maximum could never happen in practical implementation due to numerical error or by intentional ϵ-perturbation.

[18] M. C. Lin, "Efficient Collision Detection for Animation and Robotics," *Ph.D. thesis,* University of California, Berkeley, 1993.
[19] M. C. Lin and D. Manocha, "Collision and proximity queries," *Handbook of discrete and computational geometry*, 2004, pp. 787–807.
[20] W. A. McNeely, K. D. Puterbaugh, and J. J. Troy, "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling," *SIGGRAPH 1999*, 1999, pp. 401–408.
[21] D. D. Nelson, D. Johnson, and E. Cohen, "Haptic Rendering of Surface-to-Surface Sculpted Model Interaction," *ASME Proc. 8th Annual Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, Nov. 1999.
[22] I. R. Porteous, *Geometric Differentiation: For the Intelligence of Curves and Surfaces*, 2 edition, Cambridge University Press, 2001.
[23] P. T. Saunders, *An Introduction to Catastrophe Theory*, 2 edition, Cambridge University Press, 1980.
[24] T. W. Sederberg and T. Nishita, "Curve intersection using Bézier clipping," *Computer-Aided Design*, vol. 22, 1990, pp. 538–549.
[25] T. W. Sederberg and S. R. Parry, "A comparison of curve-curve intersection algorithms," *Computer-Aided Design*, vol. 18, 1986, pp. 58–63.
[26] E. C. Sherbrooke and N. M. Patrikalakis, "Computation of the solutions of non-linear polynomial systems," *Computer Aided Geometric Design*, vol. 10, no. 5, 1993, pp. 379–405.
[27] P. Shirley and R. K. Morley, *Realistic Ray Tracing*, 2 edition, A K Peters Ltd., 2003.

Theoretically Based Robust Algorithms for Tracking Intersection Curves of Two Deforming Parametric Surfaces*

Xianming Chen[1], Richard F. Riesenfeld[1], Elaine Cohen[1], and James Damon[2]

[1] School of Computing, University of Utah, Salt Lake City, UT 84112
[2] Department of Mathematics, University of North Carolina, Chapel Hill, NC 27599

Abstract. This paper applies singularity theory of mappings of surfaces to 3-space and the generic transitions occurring in their deformations to develop algorithms for continuously and robustly tracking the intersection curves of two deforming parametric spline surfaces, when the deformation is represented as a family of generalized offset surfaces. The set of intersection curves of 2 deforming surfaces over all time is formulated as an implicit 2-manifold $\mathcal{I}$ in the augmented (by time domain) parametric space $\mathbb{R}^5$. Hyper-planes corresponding to some fixed time instants may *touch* $\mathcal{I}$ at some isolated transition points, which delineate transition events, i.e., the topological changes to the intersection curves. These transition points are the 0-dimensional solution to a rational system of 5 constraints in 5 variables, and can be computed efficiently and *robustly* with a rational constraint solver using subdivision and hyper-tangent bounding cones. The actual transition events are computed by contouring the local osculating paraboloids. Away from any transition points, the intersection curves do not change topology and evolve according to a simple evolution vector field that is constructed in the *euclidean space* in which the surfaces are embedded.

1 Introduction and Related Work

In this paper, we consider the dynamic intersection of two deforming parametric surfaces. The surface deformation is represented by a family of generalized offset surfaces, which is an example of a "radial flow"of a generalized offset vector field introduced in [6, 7] (also see [8] for a mathematically less technical discussion). This extends the standard unit normal offset surfaces. Specifically, let $\varsigma(s)$, $s \in \mathbb{R}^2$, be a parameterization of a regular initial surface; and let $U(s)$ denote an offset vector field (parameterized again by s). Such a U need be neither unit-length nor orthogonal to the tangent plane, but does not lie in the tangent plane. The generalized offset surface flow is defined by,

$$\sigma(s;t) = \varsigma(s) + tU(s); \tag{1}$$

* This work is supported in part by NSF CCR-0310705, NSF CCR-0310546, and NSF DMS-0405947. All opinions, findings, conclusions or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 101–114, 2006.
© Springer-Verlag Berlin Heidelberg 2006

where $0 \leq t \leq 1$ is the offset time. Each of the two deforming surfaces is assumed to remain regular and be free of self-intersections throughout the deformation process. Conditions ensuring such regularity are given in [6] and [8].

Research into finding surface-surface intersections has mostly focused on the static problem [4, 16, 28, 34, 39], and the case of the unit normal offset surfaces [9, 11, 12, 18, 21, 23, 29, 37], We emphasize the topological robustness of surface intersection, which has been an important and extensively researched topic for static surface-surface intersection([2, 15, 20, 24, 25, 31, 32]). In [17], Jun et al. worked on surface slicing, i.e., the intersection of a surface with a series of parallel planes, exploring the relation between the transition points and the topology of contour curves. The transition points, though, are used only to efficiently and robustly find the starting point of the contour curves for a marching algorithm [3, 5] to trace out the entire curve. Ouyang et al. [27] applied a similar approach to the intersection of two unit normal vector offset surfaces.

Applied to mappings of surfaces to $\mathbb{R}^3$, singularity theory [1] provides a theoretical classification of both the local stable properties of mappings of surfaces and of the generic transitions they undergo under deformation. Our assumptions on the regularity of the surfaces characterizes the transition of the intersection curves of the deforming surfaces to one of a list of standard generic transitions. Between transitions, the intersection curves evolve in a smooth way without undergoing topological transitions.

This paper is organized to deal with these two cases. In Section 2, we construct an evolution vector field which allows us to follow the evolution of intersection curves (ICs) by discretely solving a differential equation in the parametric space. In Section 3, we represent the locus of intersection curves of the two deforming surfaces as a 2–manifold $\mathcal{I}$ in a 5–dimensional *augmented parameter space*. In Section 4 we turn to the second problem of computing the transition events, and tracking the topological changes of the intersection curves occurring at transition points. In Section 4.1, we enumerate the generic transition points classified by singularity theory and provide an alternative characterization as critical points of a function on the implicit surface $\mathcal{I}$. This provides the theoretical basis to our algorithm that detects transition points as the simultaneous 0-set of a rational system of 5 constraints in 5 variables. Then, in Section 4.2 we compute the transitions in the intersection curves using contours on the local osculating quadric of the surface $\mathcal{I}$ at the critical points. A concluding discussion of the issues ensues in Section 5.

2 Evolution of Intersection Curves

Consider two deforming surfaces, σ and $\hat{\sigma}$, represented as generalized offset surfaces,

$$\sigma(s,t) = \varsigma(s) + t\ U(s),$$
$$\hat{\sigma}(\hat{s},t) = \hat{\varsigma}(\hat{s}) + t\ \hat{U}(\hat{s}),$$

where $s = (s_1, s_2) \in \mathbb{R}^2$ and $\hat{s} = (\hat{s}_1, \hat{s}_2) \in \mathbb{R}^2$ are the parameters of $\varsigma(s)$ and $\hat{\varsigma}(\hat{s})$, and their corresponding offset vector fields $U(s)$ and $(\hat{U}(\hat{s})$, respectively. We write the coordinate representation of the deforming surfaces by

$$\sigma(s,t) = (x(s),t), y(s,y), z(s,t)) \text{ and } \hat{\sigma}(\hat{s},t) = (\hat{x}(\hat{s},t), \hat{y}(\hat{s},t), \hat{z}(\hat{s},t)).$$

Define $\mathcal{L}_0$ to be the set of all points in $\mathbb{R}^3$ on a local intersection curves of the two deforming local surfaces over all times t. Consider a point P on an intersection curve of the two deforming surfaces at some time t. We first assume that the tangent planes to the two offset surfaces at P are different; otherwise, we are in the singular case corresponding to a transition event, which we will discuss in section 4. We use the notation $\hat{\sigma}_i = (\hat{x}_i, \hat{y}_i, \hat{z}_i)$ to denote the partial derivative $\frac{\partial \hat{\sigma}}{\partial \hat{s}_i} = (\frac{\partial \hat{x}}{\partial \hat{s}_i}, \frac{\partial \hat{y}}{\partial \hat{s}_i}, \frac{\partial \hat{z}}{\partial \hat{s}_i})$ $(i = 1, 2)$, and analogously for σ_i. Define

$$N = \sigma_1 \times \sigma_2, \quad \hat{N} = \hat{\sigma}_1 \times \hat{\sigma}_2$$

to be the 2 non-unit length normals to each of the two surfaces, respectively. Further let

$$\bar{N} = (N \times \hat{N}) \times \hat{N}.$$

to be the tangent vector of $\hat{\sigma}$ at P that is perpendicular to the intersection curve.

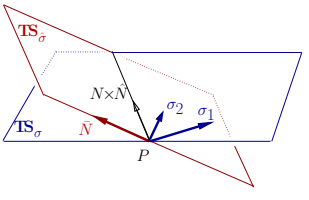


Fig. 1. Local Basis $\{\sigma_1, \sigma_2, \bar{N}\}$

Because the two tangent planes to the two surfaces at P are different, $\{\sigma_1, \sigma_2, \bar{N}\}$ is a basis of $\mathbb{R}^3$ (Fig. 1). Decomposing $\delta U = \hat{U} - U$ in this basis gives,

$$\delta U = \hat{U} - U = a\sigma_1 + b\sigma_2 + c\bar{N}$$

Because the last term $c\bar{N}$ lives entirely in the tangent plane to the surface $\hat{\sigma}$ at P, it has a decomposition relative to the basis $\{\hat{\sigma}_1, \hat{\sigma}_2\}$,

$$c\bar{N} = \hat{a}\hat{\sigma}_1 + \hat{b}\hat{\sigma}_2,$$

Thus, we have

$$\delta U = \hat{U} - U = a\sigma_1 + b\sigma_2 + (-\hat{a}\hat{\sigma}_1 - \hat{b}\hat{\sigma}_2),$$

or,

$$\hat{U} + (\hat{a}\hat{\sigma}_1 + \hat{b}\hat{\sigma}_2) = U + a\sigma_1 + b\sigma_2$$

Consequently, we have the **evolution vector field** with two equivalent representations (over two different basis of $\mathbb{R}^3$),

$$\eta = U + a\sigma_1 + b\sigma_2 \tag{2}$$

$$\hat{\eta} = \hat{U} + \hat{a}\hat{\sigma}_1 + \hat{b}\hat{\sigma}_2 \tag{3}$$

This vector field is defined on a neighborhood of the point P in $\mathbb{R}^3$, rather than just on the surfaces.

Next, for any point P which lies on a curve of intersection for the deforming surfaces, we can define a scalar field ϕ in a neighborhood of P (in $\mathbb{R}^3$). By the inverse function theorem, there is a neighborhood of P which is entirely covered by each deforming family. For a point P' in this neighborhood, we define $\phi(P') = \hat{t} - t$, where $\hat{t}$ (resp. t) is the time when the surface $\hat{\sigma}(\hat{s}, t)$ (resp. $\sigma(s, t)$) reaches P'. Although ϕ is not defined everywhere on $\mathbb{R}^3$, it is defined on a neighborhood of $\mathcal{L}_0$.

The following properties involving ϕ, η, and $\mathcal{L}_0$ can be shown to hold:

1. The directional derivative $\frac{\partial \phi}{\partial \eta} = 0$ identically wherever ϕ is defined.
2. The zero level set of ϕ is exactly $\mathcal{L}_0$.
3. Hence, η is tangent to $\mathcal{L}_0$ at all points.

Now suppose point P is on $\mathcal{L}_0$, and lies on an intersection curve at time t. The condition that η is tangent to $\mathcal{L}_0$ allows us to follow the evolution of P on future intersection curves by solving the differential equation

$$\frac{dx}{dt} = \eta(x) \quad \text{with initial condition } x(0) = P$$

for $x(t) \in \mathbb{R}^3$. The evolution vector field η is the image of the vector field $\xi = \frac{\partial}{\partial t} + a\frac{\partial}{\partial s_1} + b\frac{\partial}{\partial s_2}$ under the parametrization map σ. Thus, the evolution could likewise be followed on the parameter space using instead the vector field ξ, and analogously for $\hat{\sigma}$.

Then, we can use a discrete algorithm for solving the differential equations to follow the evolution of the intersection curves over a time interval void of transitions. Specifically, for small time dt P moves to $Q = P + dt\ (a\sigma_1 + b\sigma_2)$ on the physical surface and if $p = s \in \mathbb{R}^2$ corresponds to P then $q = s + (a\ dt, b\ dt)$ will correspond to Q in the parameter space, and analogously for $\hat{\sigma}$. The first order marching algorithm accumulates error over time, so point correction can be used to increase the quality. Various point correction algorithms are discussed in [5] in the context of static surface-surface intersection. We have adopted the middle point algorithm as presented in [4] to relax the points onto the actual intersection curve.

Also notice that we are not tracing out an entire intersection loop from some starting point; instead, we represent intersection curves by an ordered list of sample points. Sample points are adaptively inserted or deleted so that the spacing of two consecutive sample points is neither too far away nor too close, and so that the angle deviation of 3 consecutive sample points stays small.

3 Formulation in the Augmented Parametric Space

Define a vector distance mapping

$$d(s, \hat{s}, t) = \hat{\sigma} - \sigma : \mathbb{R}^5_{\{s,\hat{s},t\}} \longrightarrow \mathbb{R}^3 \tag{4}$$

where $\mathbb{R}^5_{\{s,\hat{s},t\}}$ [1] is the combined parametric space of the 2 surfaces and the time domain, and is thus called **augmented parametric space**. The canonical orthonormal basis $\mathbb{R}^5_{\{s,\hat{s},t\}}$ is denoted as $\{e_{s_1}, e_{s_2}, e_{\hat{s}_1}, e_{\hat{s}_2}, e_t\}$. The 0-set of this mapping, denoted $\mathcal{I}$ hereafter in this paper, gives the set of all intersection points in $\mathbb{R}^5_{\{s,\hat{s},t\}}$. Note that $d(s,\hat{s},t)$ concisely represents related equations for three separate coordinate functions. Considering the x-component $d_x(s,\hat{s},t)$. $d_x(s,\hat{s},t) = 0$ defines a hyper-surface in $\mathbb{R}^5_{\{s,\hat{s},t\}}$, with corresponding normal

$$N_x = \nabla d_x = (-x_1, \ -x_2, \ \hat{x}_1, \ \hat{x}_2, \ \delta U_x) \tag{5}$$

The component functions y and z define another two hyper-surfaces with analogous expressions for their normals N_y and N_z. Geometrically, $\mathcal{I}$ is the locus of intersection points of these three hyper-surfaces in $\mathbb{R}^5_{\{s,\hat{s},t\}}$. The Jacobian [22] of the mapping $d(s,\hat{s},t) : \mathbb{R}^5_{\{s,\hat{s},t\}} \longrightarrow \mathbb{R}^3$ is,

$$\mathcal{J} = (N_x \ N_y \ N_z)^t = \begin{pmatrix} -x_1 & -x_2 & \hat{x}_1 & \hat{x}_2 & \delta U_x \\ -y_1 & -y_2 & \hat{y}_1 & \hat{y}_2 & \delta U_y \\ -z_1 & -z_2 & \hat{z}_1 & \hat{z}_2 & \delta U_z \end{pmatrix} = (-\sigma_1 \quad -\sigma_2 \quad \hat{\sigma}_1 \quad \hat{\sigma}_2 \quad \delta U). \tag{6}$$

Remark 1. If the 2 tangent planes to the two deforming surfaces at the intersection point are not the same, then both of the triple scalar products (determinants) $[\sigma_1 \sigma_2 \hat{\sigma}_i]$'s $(i = 1, 2)$ can not simultaneously vanish, and so $\mathcal{J}$ has the full rank of 3. Otherwise, the two tangent planes must be the same. Assuming, at such a touching point, δU is not on the common tangent plane, i.e., $[\hat{\sigma}_1 \hat{\sigma}_2 \delta U] \neq 0$ and $[\sigma_1 \sigma_2 \delta U] \neq 0$, $\mathcal{J}$ again has the full rank. Therefore, the 0-set of the distance mapping $d(s,\hat{s},t) = \hat{\sigma} - \sigma : \mathbb{R}^5_{\{s,\hat{s},t\}} \longrightarrow \mathbb{R}^3$, is a well defined implicit 2-manifold in the augmented parametric space.

4 Transition of Intersection Loops

In singularity theory, the situation we consider is considered generic. That is, except for a finite set of times, the two closed surfaces intersect transversely, that is, at each intersection point the tangent planes of the surfaces are different. Thus, the method presented in Section 2 can be applied to track the evolution of the curves. Over such time intervals topological changes are guaranteed not to occur.

At the remaining finite number of times, there will be intersection points at which the tangent planes coincide (non-transverse points). Again for *generic deformations*, singularity theory describes exactly the transitions in intersection curves that can occur as the evolution passes such times. These transitions can always be given (up to a change of coordinates) by standard model equations, so there is essentially a unique way for each transition to occur. We shall refer to points (and times) at which transitions occur as *transition points*. These transitions are classified as,

[1] $\mathbb{R}^5_{\{s,\hat{s},t\}}$ denotes $\mathbb{R}^5$ with the five coordinates being $s_1, s_2, \hat{s}_1, \hat{s}_2, t$ and analogously, for $\mathbb{R}^3_{\{s,t\}}$, etc.

1. *a creation event*, when a new intersection loop is created (Fig. 2),
2. *an annihilation event*, when one of the current loops collapses and disappears (Fig. 2 in the reverse direction),
3. *an exchange event*, when two branches of intersection curves meet and exchange branches (Fig. 3).

The exchange event can have two different global consequences. If the two branches are part of the same curve, an intersection loop is split into 2 loops and we refer to this as a *splitting event* (Fig. 6). If the branches are from distinct intersection loops, a single loop is formed in a *merge event* (Fig. 6 in reverse order).

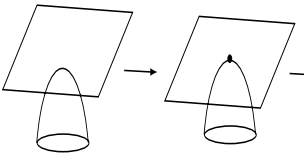


Fig. 2. Creation of IC Component

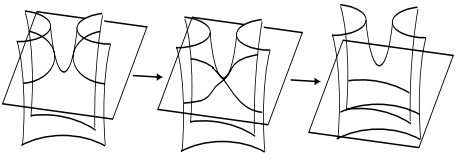


Fig. 3. Exchange of IC Components

4.1 Detection of Transition Events

In this sub-section, we formulate the topological transition points as the 0-set of a rational system of 5 nonlinear constraints in 5 variables. The 0-set has dimension 0, i.e., it is a discrete collection of points. It can be *robustly and efficiently* computed using a rational constraint solver [10, 33]. The robustness is achieved by bounding the subdivided implicit surface $\mathcal{I}$ with the corresponding hyper-tangent cone [10], an extension of the bounding tangent cones for explicit plane curves and explicit surfaces [31, 32].

Let us recall that the implicit 2-manifold $\mathcal{I}$ in $\mathbb{R}^5_{\{s,\hat{s},t\}}$ is the locus of intersection points of the two deforming surfaces, over the whole time period. Geometrically, the intersection curves, at some time point, are the corresponding height contour of $\mathcal{I}$ when the t is regarded as the vertical axis. Therefore, it is obvious that there will be one of the three transition events listed earlier, if the tangent space to $\mathcal{I}$ at a point $(s, \hat{s}, t)$ is orthogonal to the t-axis. Since $\mathcal{I}$ and its tangent space have the same dimension, namely, 2, the orthogonality condition is tantamount to satisfying two equations,

$$T_1 \cdot e_t = 0, \qquad T_2 \cdot e_t = 0,$$

where T_1 and T_2 are any two vectors spanning the tangent space. A simple and natural way to construct such a pair of tangent vectors is to let T_1 be the tangent to an s_2-iso-curve on $\mathcal{I}$ with the extra constraint $s_2 = c_2$ for some constant c_2, and let T_2 be the tangent to an s_1-iso-curve on $\mathcal{I}$ with the extra constant $s_1 = c_1$ for some constant c_1. Noticing that an s_2-iso-curve is the intersection of 4 hyper surfaces in $\mathbb{R}^5_{\{s,\hat{s},t\}}$, defined by $s_2 = c_2$, $d_x = 0$, $d_y = 0$, and $d_z = 0$,

$$T_1 = \begin{vmatrix} e_{s_1} & e_{s_2} & e_{\hat{s}_1} & e_{\hat{s}_2} & e_t \\ 0 & 1 & 0 & 0 & 0 \\ -x_1 & -x_2 & \hat{x}_1 & \hat{x}_2 & \delta U_x \\ -y_1 & -y_2 & \hat{y}_1 & \hat{y}_2 & \delta U_y \\ -z_1 & -z_2 & \hat{z}_1 & \hat{z}_2 & \delta U_z \end{vmatrix} = (\mathscr{T}^{\hat{1}\hat{2}\delta},\ 0,\ \ \mathscr{T}^{1\hat{2}\delta},\ \ -\mathscr{T}^{1\hat{1}\delta},\ \ \mathscr{T}^{1\hat{1}\hat{2}}),$$

where $\mathscr{T}$' denotes the triple scalar product of its 3 corresponding vectors indicated by the superscripts. Superscripts i and $\hat{i}$ represent σ_i and $\hat{\sigma}_i$, respectively, while a superscript δ represents δU (e.g., $\mathscr{T}^{\hat{1}\hat{2}\delta} = [\hat{\sigma}_1\ \hat{\sigma}_2\ \delta U]$). A similar derivation exists for $\mathscr{T}_2$, and in general, we have,

$$T_i = \mathscr{T}^{\hat{1}\hat{2}\delta}\ e_{s_i} + \mathscr{T}^{i\hat{2}\delta}\ e_{\hat{s}_1} - \mathscr{T}^{i\hat{1}\delta}\ e_{\hat{s}_2} + \mathscr{T}^{i\hat{1}\hat{2}}\ e_t, \quad i = 1, 2. \tag{7}$$

At transition points, the last component of T_1 and T_2 vanishes, i.e.

$$T_1 \cdot e_t = \mathscr{T}^{1\hat{1}\hat{2}} = [\sigma_1\ \hat{\sigma}_1\ \hat{\sigma}_2] = 0, \quad T_2 \cdot e_t = \mathscr{T}^{2\hat{1}\hat{2}} = [\sigma_2\ \hat{\sigma}_1\ \hat{\sigma}_2] = 0, \tag{8}$$

Remark 2. By Remark 1, $\mathscr{T}^{\hat{1}\hat{2}\delta} \neq 0$ at any transition point. Therefore, at a transition point, T_1 and T_2 are guaranteed to be independent of each other. It is also easily seen that Eq. (8) simply require the two tangents σ_1 and σ_2 to the first offset surface to be perpendicular to the normal of the second offset surface, i.e., the two tangent planes to the two deforming surfaces in the euclidean space are coincident.

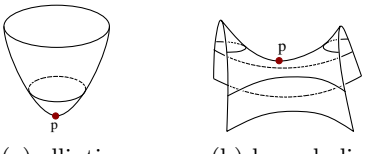
Finally, together with $\hat{\sigma} - \sigma = 0$, Eq. (8) gives a rational system with 5 constraints in 5 variables, whose 0-dimensional solution set contains all the transition points we are seek.

4.2 Compute the Structural Change at Transition Events

In this section, we perform the shape computation of the 2-manifold $\mathcal{I}$ at a transition point, and subsequently compute the corresponding transition event by contouring the osculating paraboloid [19, 26] to the local shape (Fig. 4).

The implicit surface $\mathcal{I}$ is a 2-manifold in a 5-space $\mathbb{R}^5_{\{s,\hat{s},t\}}$. Shape computation is difficult because it is an implicit surface, and also because its codimension $\neq 1$.

Most recently, a comprehensive set of formulas for curvature computation on implicit curves/surfaces with further references were presented in [13]. However, it is limited to curves/surfaces embedded in $2D$ or $3D$ spaces. There exists some literature from the visualization community, e.g., [14, 36] and references therein, that develops second order derivative computation on iso-surfaces extracted from trivariate functions. Most of these approaches use discrete approximations.



(a) elliptic (b) hyperbolic

Fig. 4. Contour Osculating Paraboloid

Recently, [35] developed B-spline representations for the Gaussian curvature and squared mean curvature of the iso-surfaces extracted from volumetric data defined as a trivariate B-spline function, and subsequently presented an exact curvature computation for every possible point of the 3D domain. While we seek an exact differential computation, the task here significantly differs from that in [35] and [14, 36] since the implicit 2-manifold $\mathcal{I}$ has codimension 3. In [38], a set of formulas for computing Riemannian curvature, mean curvature vector, and principal curvatures, specifically for a 2-manifold, and with *arbitrary* codimension, is presented. The specific 2nd order problem we seek to solve, namely initializing the newly created intersection loop, or switching the two pairs of hyperbolic-like segments, is based on *shape approximation*. For a surface in 3-space, the local shape approximation is simply the osculating quadric, expressed in the second fundamental form [26] as $z = II(a, a)$ where a is any tangent vector, and z is the vertical distance from the local surface point to the tangent plane. Observing that the second order shape approximation is best done if the codimension is 1, we do not compute the second fundamental form directly on the 2-manifold in 5-space. Instead, we project the 2-manifold to a 3-space of either $\mathbb{R}^3_{\{s,t\}}$ (our choice in this paper)or $\mathbb{R}^3_{\{\hat{s},t\}}$. The second fundamental form is then computed for this projected 2-manifold, and shape approximation is achieved subsequently. Notice that the shape approximation in the projected 3-space gives only a partial answer to the transition event; the full solution is achieved by the tangential mapping between the projected 2-manifold and the original one (cf. Observation 1 below).

Projection of $\mathcal{I}$ to $\mathbb{R}^3_{\{s,t\}}$. Near a critical point, T_1 and T_2 (cf. Eq. (7)) give two vectors spanning the tangent space to $\mathcal{I}$. By projecting $\mathcal{I}$ onto $R^3_{\{s_1,s_2,t\}}$ and ignoring the $\hat{s}_1$ and $\hat{s}_2$ components, we transform $\mathcal{I}$, a 2-manifold in $\mathbb{R}^5_{\{s,\hat{s},t\}}$, into a surface in $R^3_{\{s,t\}}$, denoted as $\mathcal{I}^s$. Furthermore, the projection, denoted as π hereafter, is a diffeomorphism. and the projected tangent plane (the tangent plane to $\mathcal{I}^s$) is spanned by,

$$T_1^s = \mathscr{T}^{\hat{1}\hat{2}\delta}\ e_{s_1} + \mathscr{T}^{1\hat{1}\hat{2}}\ e_t, \quad T_2^s = \mathscr{T}^{\hat{1}\hat{2}\delta}\ e_{s_2} + \mathscr{T}^{2\hat{1}\hat{2}}\ e_t, \tag{9}$$

where we have used the superscript s to distinguish the tangents from their counterparts of $\mathcal{I}$ in the original augmented parametric space $\mathbb{R}^5_{\{s,\hat{s},t\}}$.

Exactly at the transition point where $\mathscr{T}^{1\hat{1}\hat{2}} = \mathscr{T}^{2\hat{1}\hat{2}} = 0$ (cf. Eq. (8)), we have,

$$T_1^s = \mathscr{T}^{\hat{1}\hat{2}\delta}\ e_{s_1}, \quad T_2^s = \mathscr{T}^{\hat{1}\hat{2}\delta}\ e_{s_2}.$$

Hereafter, a point in the tangent space $\mathbf{TS}_{\mathcal{I}^s}$ is typically specified by its 2 coordinates, say, a_1 and a_2, with respect to the basis $\{T_1^s, T_2^s\}$, the canonical frame $\{e_{s_1}, e_{s_2}\}$ scaled by $\mathscr{T}^{\hat{1}\hat{2}\delta}$.

Observation 1. *At a transition point, the inverse of the tangent map of π is (cf. Eqs.* (7)*),*

$$(1,0) \mapsto (\mathscr{T}^{\hat{1}\hat{2}\delta}, 0, \mathscr{T}^{1\hat{2}\delta}, -\mathscr{T}^{1\hat{1}\delta}, 0), \qquad (0,1) \mapsto (0, \mathscr{T}^{\hat{1}\hat{2}\delta}, \mathscr{T}^{2\hat{2}\delta}, -\mathscr{T}^{2\hat{1}\delta}, 0),$$

where $(1,0)$ *and* $(0,1)$ *are the coordinates of two points in the local tangent space* $\mathbf{TS}_{\mathcal{I}^s}$ *with basis* $\{T_1^s, T_2^s\}$.

The Shape Computation. The local shape of $\mathcal{I}^s$ in $\mathbb{R}^3_{\{s,t\}}$ is determined from the second fundamental form II. At *a transition point*,

$$II = \left(N^s \cdot \nabla_{T_i^s} T_j^s\right) = \left(\nabla_{T_i^s} T_j^s \cdot e_t\right), \tag{10}$$

and the local shape is approximated by the osculating quadric [26, 19],

$$\delta t = II(a,a) = \begin{pmatrix} a_1 & a_2 \end{pmatrix} II \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, \tag{11}$$

where $a = (a_1, a_2) \in \mathbf{TS}_{\mathcal{I}^s}$. Notice that we wrote the left hand side as δt, because, at a transition point, the tangent plane $\mathbf{TS}_{\mathcal{I}^s}$ is horizontal, and thus the local vertical height is exactly the time deviation from the considered transition point.

The covariant derivatives are best computed in the original 5-space, i.e.,

$$\nabla_{T_i^s} T_j^s \cdot e_t = \nabla_{T_i} T_j \cdot e_t.$$

By Eq. (7),

$$\begin{aligned} \nabla_{T_i^s} T_j^s \cdot e_t = \nabla_{T_i} T_j \cdot e_t = \nabla_{T_i}(T_j \cdot e_t) = \nabla_{T_i} \mathscr{T}^{j\hat{1}\hat{2}} \\ = \mathscr{T}^{\hat{1}\hat{2}\delta} \frac{\partial \mathscr{T}^{j\hat{1}\hat{2}}}{\partial s_i} + \mathscr{T}^{i\hat{2}\delta} \frac{\partial \mathscr{T}^{j\hat{1}\hat{2}}}{\partial \hat{s}_1} - \mathscr{T}^{i\hat{1}\delta} \frac{\partial \mathscr{T}^{j\hat{1}\hat{2}}}{\partial \hat{s}_2}. \end{aligned}$$

Introducing the following notations $(i, j, k \in \{1, 2\})$,

$$\mathscr{T}^{j_i\hat{1}\hat{2}} = [\frac{\partial \sigma_j}{\partial s_i} \quad \hat{\sigma}_1 \quad \hat{\sigma}_2], \quad \mathscr{T}^{i\hat{1}_k\hat{2}} = [\sigma_i \quad \frac{\partial \hat{\sigma}_1}{\partial \hat{s}_k} \quad \hat{\sigma}_2], \quad \mathscr{T}^{i\hat{1}\hat{2}_k} = [\sigma_i \quad \hat{\sigma}_1 \quad \frac{\hat{\sigma}_2}{\partial \hat{s}_k}]$$

yields,

$$\nabla_{T_i^s} T_j^s \cdot e_t = \mathscr{T}^{\hat{1}\hat{2}\delta} \mathscr{T}^{j_i\hat{1}\hat{2}} + \mathscr{T}^{i\hat{2}\delta} (\mathscr{T}^{j\hat{1}_1\hat{2}} + \mathscr{T}^{j\hat{1}\hat{2}_1}) - \mathscr{T}^{i\hat{1}\delta} (\mathscr{T}^{j\hat{1}_2\hat{2}} + \mathscr{T}^{j\hat{1}\hat{2}_2}).$$

Throughout this paper, we make the generic assumption that the transition point is non-degenerate, i.e., $det(II) \neq 0$.

Heuristically Uniform Sampling of Local Height Contours. To compute various transition events, the height contour curves of the local osculating quadric needs to be uniformly sampled in the *euclidean space* $\mathbb{R}^3$.

Suppose we are sampling the height contour with the time deviation δt. By Eq. (11), the sample point $p_v \in \mathbf{TS}_{\mathcal{I}^s}$ along a direction $v \in \mathbf{TS}_{\mathcal{I}^s}$, is $p_v = \sqrt{\frac{2\delta t}{II(v,v)}}\, v$. Therefore, given an initial list of sample *directions*, the following algorithm generates a list of heuristically uniform sample points.

Algorithm 1. Heuristically Uniform Sampling

1. *Turn the given list of sample* directions *into a list of sample* points *by scaling each element* p *by* $\sqrt{\frac{2\delta t}{II(p,p)}}$.
2. *In the current list, find a neighboring sample pair* $p, q \in \mathbf{TS}_{\mathcal{I}^s}$ *with maximal distance.*
3. *Let* $m = \frac{p+q}{2}$, *scale* m *by* $\sqrt{\frac{2\delta t}{II(m,m)}}$, *and insert it into the list in between* p *and* q.
4. *If not enough sample points, or the distances are not approximately uniform, goto* **Step 2**.

Compute Transition Events

Compute Creation Events: If $det(II) > 0$ (or equivalently, the Gaussian curvature of $\mathcal{I}^s$ is positive), the osculating quadric (Eq. (11)) is an elliptic paraboloid, and the transition point has elliptic type. See Fig. 4(a).

For an upward elliptic type and offset surfaces deforming forward, or for a downward elliptic type and offset surfaces deforming backward, a creation event is occurring, i.e., an entirely new intersection loop is created from nothing. The following algorithm computes the intersection loop at the time deviation δt from the transition point.

Algorithm 2. Compute Ellipse Contour for a Creation Event

1. *Put directions* $(1,0), (0,1), (-1,0)$ *into the ordered list of directions* $\mathcal{V}$.
2. *Apply Algo. 1 to transform* $\mathcal{V}$ *to a ordered list of uniform samples in* $\mathbf{TS}_{\mathcal{I}^s}$.
3. *Except the first and the last ones, copy and negate in order all elements, in* $\mathcal{V}$, *and append to itself.*
4. *Map* $\mathcal{V}$ *to a ordered list of samples in* $\mathbb{R}^4_{\{s,\hat{s},t\}}$ *(cf. Observation 1).*

Compute Annihilation Events: At an upward elliptic transition point when offset surfaces deform backward (in time), or at a downward elliptic transition point when offset surfaces deforming forward, there is an *annihilation event* happening, i.e. an intersection loop collapses and disappears. See Fig. 4(a). The key issue here is to choose the right current intersection loop to annihilate. If there is currently only one intersection loop, annihilate it. Otherwise, we use an "evolve-to-annihilate" strategy as illustrated in Fig. 5. First, evolve all intersection loops at time t_1 to the time t' (i.e., the contour position used for the pre-computation of the corresponding creation event). Then, using the inclusion test [30], find the one that the critical point p identifies to annihilate.

Compute Switch Events: If $det(II) < 0$ (or equivalently, the Gaussian curvature of $\mathcal{I}^s$ is negative), the osculating quadric (Eq. (11)) is a hyperbolic paraboloid, and the transition point has hyperbolic type. See Fig. 4(b).

Deforming across a hyperbolic transition point is a quite different situation from an elliptic point inasmuch as there is a switch of two pairs of hyperbolic-like segments (cf. Fig. 3 in the euclidean space, and Fig. 4 of $\mathcal{I}^s$): 2 local segments

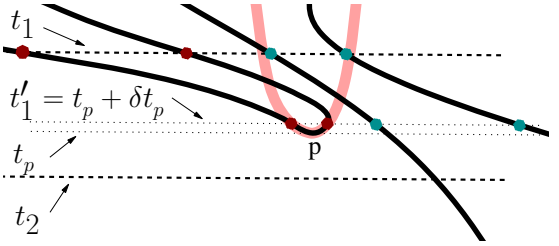


Fig. 5. Evolve to Annihilate

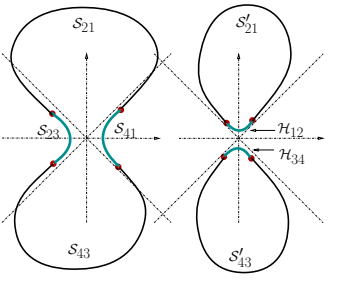


Fig. 6. Split/Merge

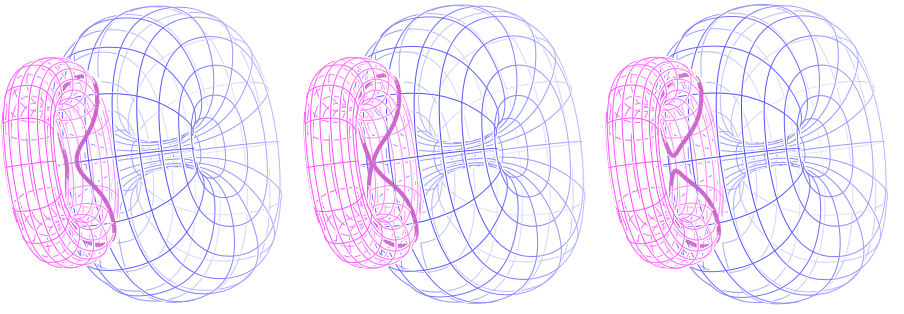


Fig. 7. Transition: A split event of 2 deforming torus-like surfaces

approach each other, say, from *above* the transition point p , touch *at* p, and then swap and depart into another two local segments *below* p. If the approaching pair of segments is from one intersection loop (Fig. 6), we have a split event; and if it is from 2 intersection loops (Fig. 6 in reverse order), we have a merge event. Each segment is a height contour of the local shape approximated by the osculating hyperbolic paraboloid. The following algorithm computes one pair of such height contours.

Algorithm 3. Compute Hyperbolic Contours for a Switch Event

1. *Put directions* $u_1 + (u_2 - u_1) * \lambda, u_1 + u_2, u_2 + (u_1 - u_2) * \lambda$ *into the ordered list of directions* $\mathcal{V}$.
2. *Invoke Algo. 1 to transform* $\mathcal{V}$ *to an ordered list of uniform samples in* $\mathbf{TS}_{\mathcal{I}^s}$.
3. *In order, copy and negate all elements into another list* $\mathcal{V}'$.
4. *Map* $\mathcal{V}$ *to an ordered list of samples in* $\mathbb{R}^4_{\{s,\hat{s},t\}}$ *(cf. Observation 1). Do the same for* $\mathcal{V}'$.

In the algorithm, u_1 and u_2 are the two asymptotic directions, which can be solved (for u) from the equation $II(u, u) = 0$ using the second fundamental form in Eq. (10). The other pair of contours, with the opposite height value, can be sampled similarly, with one of the asymptotic directions reversed.

Based on the deforming direction we can determine which of the 2 principal curvature directions is the approaching direction, and which is the departing

direction. Then the approaching pair of segments of current intersection curves is the one that is closest to the considered transition point along the approaching direction. Using Algorithm 3, the switch event can be computed by cutting the two approach segments, evolving the rest across, say upward, p, and then pasting the other pair of contours to the departing pair of segments (Illustrated Fig. 6). Finally, Fig. 7 gives an example of split event of two deforming torus-like surfaces. For demo videos, see http://www.cs.utah.edu/∼xchen/papers/more.html

5 Conclusion

In this paper, we have applied a mathematical framework provided by singularity theory to develop algorithms for continuously and robustly tracking the intersection curves of two generically deforming surfaces, on the assumption that both the base surfaces and the deforming vectors have rational parametrization. The core idea is to divide the process into two steps depending on when transition points occur. Away from any transition points, the intersection curves evolve without any structural change. We found a simple and robust method which constructs an evolution vector field directly in the euclidean space R^3 and evolves the intersection curves accordingly.

We further developed a method for identifying transition points and following topological changes in the intersection curves through the introduction of an implicit 2-manifold $\mathcal{I}$, which consists of the union of intersection curves in the augmented (by time domain t) joint parameter space. The transition points are identified as the points on $\mathcal{I}$ where the tangent spaces are orthogonal to t-axis, and the topological change of the intersection curves is subsequently computed by 2nd order differential geometric computations on $\mathcal{I}$.

There are further transitions which can occur for deforming surfaces, including the surface developing singularities, self-intersections, and triple intersection points.. We are now developing a similar formulation for tracking the intersection curve end points that correspond to surface boundaries, and for tracking triple intersection points.

References

[1] J. Mather *Stability of C^∞–mappings, I: The Division Theorem* Ann. of Math **89** (1969) 89–104; *II. Infinitesimal stability implies stability*, Ann. of Math. **89** (1969) 254–291; *III. Finitely determined map germs* , Inst. Hautes Etudes Sci. Publ. Math. **36** (1968) 127–156; *IV. Classification of stable germs by $\mathbb{R}$–algebras* , Inst. Hautes Etudes Sci. Publ. Math. **37** (1969) 223–248; *V. Transversality* , Adv. in Math. **37** (1970) 301–336; *VI. The nice dimensions* , Liverpool Singularities Symposium I , Springer Lecture Notes in Math. **192** (1970) 207–253

[2] K. Abdel-Malek and H. Yeh, "Determining intersection curves between surfaces of two solids," *Computer-Aided Design*, vol. 28, no. 6-7, June-July 1996, pp. 539–549.

[3] C. L. Bajaj, C. M. Hoffmann, R. E. Lynch, and J. E. H. Hopcroft, "Tracing surface intersections," *Computer Aided Geometric Design*, vol. 5, no. 4, November 1988, pp. 285–307.

[4] R. E. Barnhill, G. Farin, M. Jordan, and B. R. Piper, "Surface/surface intersection," *Computer Aided Geometric Design*, vol. 4, no. 1-2, July 1987, pp. 3–16.
[5] R. E. Barnhill and S. N. Kersey, "A marching method for parametric surface/surface intersection," *Computer Aided Geometric Design*, vol. 7, no. 1-4, June 1990, pp. 257–280.
[6] J. Damon, "On the Smoothness and Geometry of Boundaries Associated to Skeletal Structures I: Sufficient Conditions for Smoothness," *Annales Inst. Fourier*, vol. 53, 2003, pp. 1941–1985.
[7] J. Damon, "On the Smoothness and Geometry of Boundaries Associated to Skeletal Structures II: Geometry in the Blum Case," *Compositio Mathematica*, vol. 140, no. 6, 2004, pp. 1657–1674.
[8] J. Damon, "Determining the Geometry of Boundaries of Objects from Medial Data," *Int. Jour. Comp. Vision*, vol. 63, no. 1, 2005, pp. 45–64.
[9] G. Elber and E. Cohen, "Error bounded variable distance offset operator for free form curves and surfaces," *Int. J. Comput. Geometry Appl*, vol. 1, no. 1, 1991, pp. 67–78.
[10] G. Elber and M.-S. Kim, "Geometric constraint solver using multivariate rational spline functions," *Symposium on Solid Modeling and Applications*, 2001, pp. 1–10.
[11] G. Elber, I.-K. Lee, and M.-S. Kim, "Comparing offset curve approximation methods," *IEEE Computer Graphics and Applications*, vol. 17, 1997, pp. 62–71.
[12] R. T. Farouki and C. A. Neff, "Analytic properties of plane offset curves," *Computer Aided Geometric Design*, vol. 7, no. 1-4, 1990, pp. 83–99.
[13] R. Goldman, "Curvature formulas for implicit curves and surfaces," *cagd*, vol. 22, no. 7, 2005, pp. 632–658.
[14] B. Hamann, "Visualization and Modeling Contours of Trivariate Functions, Ph.D. thesis," *Arizona State Univeristy*, 1991.
[15] M. E. Hohmeyer, "A surface intersection algorithm based on loop detection," *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*. May 1991, pp. 197–207, ACM Press.
[16] C. Y. Hu, T. Maekawa, N. M. Patrikalakis, and X. Ye, "Robust Interval Algorithm for Surface Intersections," *Computer Aided Design*, vol. 29, no. 9, September 1997, pp. 617–627.
[17] C.-S. Jun, D.-S. Kim, D.-S. Kim, H.-C. Lee, J. Hwang, and Tien-ChienChang, "Surface slicing algorithm based on topology transition," *Computer-Aided Design*, vol. 33, no. 11, September 2001, pp. 825–838.
[18] R. Kimmel and A. M. Bruckstein, "Shape offsets via level sets," *Computer-Aided Design*, vol. 25, no. 3, March 1993, pp. 154–162.
[19] J. J. Koenderink, *Solid Shape*, MIT press, 1990.
[20] G. A. Kriezis, N. M. Patrikalakis, and F. E. Wolter, "Topological and differential-equation methods for surface intersections," *Computer-Aided Design*, vol. 24, no. 1, January 1992, pp. 41–55.
[21] G. V. V. R. Kumar, K. G. Shastry, and B. G. Prakash, "Computing offsets of trimmed NURBS surfaces," *Computer-Aided Design*, vol. 35, no. 5, April 2003, pp. 411–420.
[22] S. Lang, *Undergraduate Analysis*, 2 edition, Springer, 1997.
[23] T. Maekawa, "An overview of offset curves and surfaces," *Computer-Aided Design*, vol. 31, no. 3, March 1999, pp. 165–173.
[24] T. Maekawa and N. M. Patrikalakis, "Computation of singularities and intersections of offsets of planar curves.," *Computer Aided Geometric Design*, vol. 10, no. 5, 1993, pp. 407–429.

[25] R. P. Markot and R. L. Magedson, "Solutions of tangential surface and curve intersections," *Computer-Aided Design*, vol. 21, no. 7, September 1989, pp. 421–427.
[26] B. O'Neill, *Elementary Differential Geometry*, 2 edition, Academic Press, 1997.
[27] Y. Ouyang, M. Tang, J. Lin, and J. Dong, "Intersection of two offset parametric surfaces based on topology analysis," *Journal of Zhejiang Univ SCI*, vol. 5, no. 3, 2004, pp. 259–268.
[28] N. M. Patrikalakis, T. Maekawa, K. H. Ko, and H. Mukundan, "Surface to Surface Intersections," *Computer-Aided Design and Applications*, vol. 1, no. 1-4, 2004, pp. 449–458.
[29] B. Pham, "Offset curves and surfaces: a brief survey," *Computer-Aided Design*, vol. 24, no. 4, April 1992, pp. 223–229.
[30] F. P. Preparata and M. I. Shamos, *Computational geometry: an introduction*, Springer-Verlag, 1985.
[31] T. W. Sederberg, H. N. Christiansen, and S. Katz, "Improved test for closed loops in surface intersections," *Computer-Aided Design*, vol. 21, no. 8, October 1989, pp. 505–508.
[32] T. W. Sederberg and R. J. Meyers, "Loop detection in surface patch intersections," *Computer Aided Geometric Design*, vol. 5, no. 2, July 1988, pp. 161–171.
[33] E. C. Sherbrooke and N. M. Patrikalakis, "Computation of the solutions of non-linear polynomial systems," *Computer Aided Geometric Design*, vol. 10, no. 5, 1993, pp. 379–405.
[34] T. S. Smith, R. T. Farouki, M. al Kandari, and H. Pottmann, "Optimal slicing of free-form surfaces," *Computer Aided Geometric Design*, vol. 19, no. 1, Jan. 2002, pp. 43–64.
[35] O. Soldea, G. Elber, and E. Rivlin, "Global Curvature Analysis and Segmentation of Volumetric Data Sets using Trivariate B-spline Functions," *Geometric Modeling and Processing 2004*, April 2004, pp. 217–226.
[36] J.-P. Thirion and A. Gourdon, "Computing the Differential Characteristics of Isointensity Surfaces," *Journal of Computer Vision and Image Understanding*, vol. 61, no. 2, March 1995, pp. 190–202.
[37] J. Wallner, T. Sakkalis, T. Maekawa, H. Pottmann, and G. Yu, "Self-Intersections of Offset Curves and Surfaces," *International Journal of Shape Modelling*, vol. 7, no. 1, June 2001, pp. 1–21.
[38] G. Xu and C. L. Bajaj, "Curvature Computations of 2-Manifolds in $\mathbb{R}^k$," .
[39] X. Ye and T. Maekawa, "Differential Geometry of Intersection Curves of Two Surfaces," *Computer Aided Geometric Design*, vol. 16, no. 8, September 1999, pp. 767–788.

Subdivision Termination Criteria in Subdivision Multivariate Solvers

Iddo Hanniel and Gershon Elber

Department of Computer Science,
Technion, Israel Institute of Technology,
Haifa 32000, Israel
{iddoh, gershon}@cs.technion.ac.il

Abstract. The need for robust solutions for sets of non-linear multivariate constraints or equations needs no motivation. Subdivision-based multivariate constraint solvers [1,2,3] typically employ the convex hull and subdivision/domain clipping properties of the Bézier/B-spline representation to detect all regions that may contain a feasible solution. Once such a region has been identified, a numerical improvement method is usually applied, which quickly converges to the root. Termination criteria for this subdivision/domain clipping approach are necessary so that, for example, no two roots reside in the same sub-domain (root isolation).

This work presents two such termination criteria. The first theoretical criterion identifies sub-domains with at most a single solution. This criterion is based on the analysis of the normal cones of the multiviarates and has been known for some time [1]. Yet, a computationally tractable algorithm to examine this criterion has never been proposed. In this paper, we present such an algorithm for identifying sub-domains with at most a single solution that is based on a dual representation of the normal cones as parallel hyper-planes over the unit hyper-sphere. Further, we also offer a second termination criterion, based on the representation of bounding parallel hyper-plane pairs, to identify and reject sub-domains that contain no solution.

We implemented both algorithms in the multivariate solver of the IRIT [4] solid modeling system and present examples using our implementation.

1 Introduction

The robust solution of a univariate non-linear equation is considered a difficult problem. The simultaneous solution of sets of multivariate non-linear constraints is typically far more difficult. In recent years, solvers that draw from geometric design developments and are based on recursive subdivision of the constraints represented in the Bézier and/or B-spline forms were developed [1,2,3]. Coming from the geometric design field, these solvers are very much suited to solving non linear constraints in geometric design. One additional motivation for the use of subdivision based solvers in geometric design may be found in the fact that the domain is typically limited to the domain of curves or surfaces. In other words, the search for simultaneous zeros is conducted in a limited domain only.

Geometric constraint solvers play a crucial role in geometric modeling environments. The basic problem such solvers encounter is computing the simultaneous roots of a set

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 115–128, 2006.
© Springer-Verlag Berlin Heidelberg 2006

of non-linear equations. Following the work of Lane and Riesenfeld [5], who used a Bernstein-Bézier subdivision scheme for univariate root-finding, many efficient methods have been developed for a large variety of root-finding problems in geometric design and modeling. Examples include ray-surface, curve-curve and surface-surface intersections. Sherbrook and Patrikalakis [3] extended this subdivision approach to solving a set of multivariate polynomial equations given in the Bernstein-Bézier form. A similar approach can also be applied to the more general B-spline representation [1].

These solvers employ subdivision or domain clipping to reduce the size of the domain that is suspected to contain roots [1,2,3]. This process is combined with the convex hull property of the Bézier/B-spline form to discard sub-domains in which no solution exists. In other words, a constraint whose coefficients are all positive (or all negative) cannot have zeros. This is a simple yet first example of a termination criterion for this type of subdivision solver.

In every subdivision step, the domain that may contain a feasible solution becomes smaller. Nonetheless, the whole-positivity criterion cannot differentiate between sub-domains that hold one-solution or more. Typically, the process is terminated at a predefined subdivision depth, by a-priori fixing the minimal sub-domain to be considered. Furthermore, the subdivision approach is relatively slow, and therefore, reducing the number of subdivision steps, whenever possible, is highly desirable. For example, in our solver we are able to eliminate unnecessary subdivisions in the final stage once we can guarantee that there is at most one root in each sub-domain. If a sub-domain is known to contain at most one solution, we can apply a numerical improvement step (e.g., Newton-Raphson iterations) that quickly converges to the root, if it exists. Furthermore, such a condition will guarantee that we have not prematurely terminated the subdivision process before all roots have been isolated correctly, i.e., two "close" roots will not be considered as a single root.

To the best of our knowledge, this is the first subdivision-based multivariate solver that can guarantee all roots have been isolated, based on geometric conditions, and not terminate at a predefined subdivision depth.

The condition that a sub-domain contains at most one root is thus an important component in a subdivision based multivariate solver. In [6,7], Sederberg et al. introduced the concepts of a normal cone and a surface bounding cone for curve-curve intersections and loop detection in surface intersections. Elber and Kim [1] generalized these concepts to higher dimensions and used them to define a condition for isolating single solutions. However, their condition was based on the intersection of d cones in $\mathbb{R}^d$, d being the dimension of the problem, which is a computationally difficult problem for which no efficient algorithm has so far been presented.

In this paper, an efficient algorithm is presented for cone intersections in $\mathbb{R}^d$ and hence for identifying sub-domains with at most a single root, following [1]. This algorithm is based on a dual representation of the normal cones using hyper-planes and the unit hyper-sphere in $\mathbb{R}^d$. This representation leads to a simple and efficient algorithm for solving the problem.

We also present an additional sufficient criterion that ensures that a sub-domain contains no roots. The additional condition, and the algorithm for testing it, exploit pairs of parallel bounding hyper-planes on the constraints.

The rest of this paper is organized as follows. In Section 2, we define the problem and summarize the main results obtained in [1]. In Section 3, we present our first criterion and algorithm for identifying intersections of bounding cones at a single point, which correspond to isolations of sub-domains that contain at most a single solution. In Section 4, we present the second criterion and algorithm for purging away sub-domains with a zero-solution. In Section 5, we show examples from our implementation of the algorithm, and finally, we conclude the paper in Section 6.

2 Background

We consider the problem of identifying sub-domains with no or at most a single solution for a set of implicit non-linear multivariate functions in the Bézier/B-spline representation, assuming the dimension of the solution space to be zero (thus, forming a set of discrete isolated roots, in the general, non-degenerate, case). Given d implicit algebraic equations in d variables,

$$\mathcal{F}_i(u_1, u_2, \cdots, u_d) = 0,\ i = 1, \cdots, d, \tag{1}$$

we seek all $\boldsymbol{u} = (u_1, u_2, \cdots, u_d)$ that simultaneously satisfy Equation (1). We will assume that $\mathcal{F}_i$, $i = 1, \cdots, d$, are represented as B-spline or Bézier multivariate scalar functions, i.e.,

$$\mathcal{F}_i = \sum_{i_1} \cdots \sum_{i_d} P_{i_1,\cdots,i_d} B_{i_1,k_{i_1}}(u_1) \cdots B_{i_d,k_{i_d}}(u_d), \tag{2}$$

where $B_{i_j,k_{i_j}}$ are the i'th k_{i_j}-degree Bézier/B-spline basis functions. We repeat, for completeness, the result from Elber and Kim [1] of the condition for the uniqueness of a solution in a sub-domain.

Sederberg and Meyers [6] used the Hodograph as the basis of a termination condition in the intersection of two planar Bézier curves. Two planar curves intersect at most once if their Hodographs share no common direction vector. Sederberg et al. [6,7] also developed a similar condition for the intersection of two parametric surfaces, where the surface bounding (or tangent) cone plays an important role in loop detection. Elber and Kim [1] generalized this approach to the higher-dimensional problem of intersecting d implicit hyper-surfaces $\mathcal{F}_i(\boldsymbol{u}) = 0$, for $i = 1, \cdots, d$, in $\mathbb{R}^d$ (i.e., Equation (1)). Being the gradient, the normal space of an implicit hyper-surface is considerably easier to compute than that of a general parametric hyper-surface. This fact greatly simplifies the computation of the normal bounding cones.

Let $\mathcal{C}(\overline{\boldsymbol{v}}, \alpha)$ denote the cone with the axis in the direction of a unit-length vector $\overline{\boldsymbol{v}}$ and an opening angle α:

$$\mathcal{C}(\overline{\boldsymbol{v}}, \alpha) = \{\boldsymbol{u} \mid \langle \boldsymbol{u}, \overline{\boldsymbol{v}} \rangle^2 = \|\boldsymbol{u}\|^2 \cos^2 \alpha\}.$$

Furthermore, let $\mathcal{C}^{in}$ denote the set of vectors on or in the cone $\mathcal{C}$, and let $\mathcal{C}^{out}$ denote the set of vectors on or out of the cone $\mathcal{C}$. That is:

$$\mathcal{C}^{in}(\overline{\boldsymbol{v}}, \alpha) = \{\boldsymbol{u} \mid \langle \boldsymbol{u}, \overline{\boldsymbol{v}} \rangle^2 \geq \|\boldsymbol{u}\|^2 \cos^2 \alpha\}.$$

$$\mathcal{C}^{out}(\overline{\boldsymbol{v}}, \alpha) = \{\boldsymbol{u} \mid \langle \boldsymbol{u}, \overline{\boldsymbol{v}} \rangle^2 \leq \|\boldsymbol{u}\|^2 \cos^2 \alpha\}.$$

Note that if $\boldsymbol{u}$ is a vector on $\mathcal{C}$ (and similarly for $\mathcal{C}^{in}$ and $\mathcal{C}^{out}$), then so is the vector $c\boldsymbol{u}$, where $c \in \boldsymbol{R}^+$. This is a direct consequence of the cone definition as the linearity of the inner product $\langle \boldsymbol{u}, \overline{\boldsymbol{v}} \rangle^2 = \|\boldsymbol{u}\|^2 \cos^2 \alpha$ implies that $\langle c\boldsymbol{u}, \overline{\boldsymbol{v}} \rangle^2 = \|c\boldsymbol{u}\|^2 \cos^2 \alpha$.

For an implicit hyper-surface $\mathcal{F}_i(\boldsymbol{u}) = 0$, we define its normal cone $\mathcal{C}_i^n = \mathcal{C}^n(\overline{\boldsymbol{v}}_i^n, \alpha_i^n)$ in $\boldsymbol{R}^d$ (see Figure 1 (a)) as the set of all possible normal vectors, $\nabla \mathcal{F}_i(\boldsymbol{u})$, and their scalar multiples. Clearly, we have,

$$\begin{aligned}
\nabla \mathcal{F}_i(\boldsymbol{u}) &= \left(\frac{\partial \mathcal{F}_i}{\partial u_1}(\boldsymbol{u}), \frac{\partial \mathcal{F}_i}{\partial u_2}(\boldsymbol{u}), \cdots, \frac{\partial \mathcal{F}_i}{\partial u_d}(\boldsymbol{u}) \right) \\
&= \sum_{i_1} \sum_{i_2} \cdots \sum_{i_d} \left(P^{u_1}_{i_1,i_2,\cdots,i_d}, P^{u_2}_{i_1,i_2,\cdots,i_d}, \cdots, P^{u_d}_{i_1,i_2,\cdots,i_d} \right) \\
&\quad B_{i_1,k_{i_1}}(u_1) B_{i_2,k_{i_2}}(u_2) \cdots B_{i_d,k_{i_d}}(u_d),
\end{aligned} \tag{3}$$

where the terms $P^{u_j}_{i_1,i_2,\cdots,i_d}$ denote the coefficients of the partial derivative of $\mathcal{F}_i$ with respect to u_j, elevated to a common subspace. Then, the normal cone to $\mathcal{F}_i(\boldsymbol{u})$, $\mathcal{C}_i^n = \mathcal{C}(\overline{\boldsymbol{v}}_i^n, \alpha_i^n)$ can be derived, for example, by letting $\overline{\boldsymbol{v}}_i^n$ be the average (normalized to unit-length) of the vectors $\left(P^{u_1}_{i_1,i_2,\cdots,i_d}, P^{u_2}_{i_1,i_2,\cdots,i_d}, \cdots, P^{u_d}_{i_1,i_2,\cdots,i_d} \right), \forall i_1, i_2, \cdots, i_d$, and α_i^n be the maximum angle between $\overline{\boldsymbol{v}}_i^n$ and these vectors.

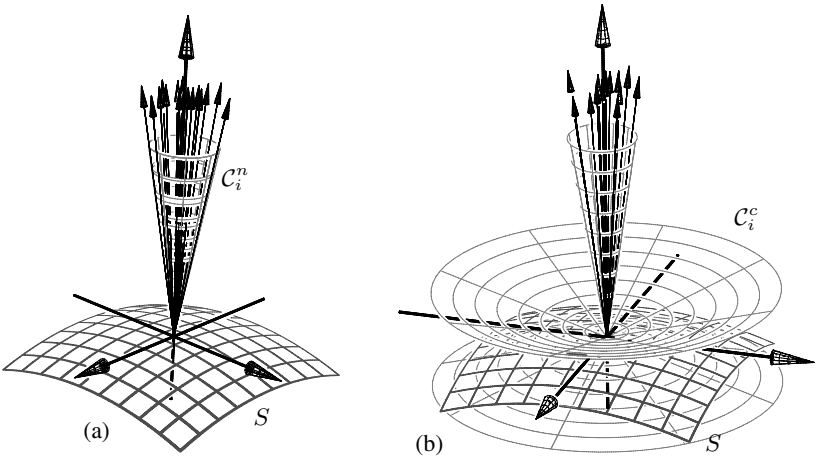


Fig. 1. (a) The normal cone, $\mathcal{C}_i^n$ (in gray) and (b) the complementary tangent cone, $\mathcal{C}_i^c$ (in gray) of a freeform surface, S

The bounding normal cone obtained by the procedure described above is, in general, not optimal. In order to get an optimal cone, Barequet and Elber [8] proposed an algorithm based on the expected linear time algorithm for minimal spanning spheres [9].

Given the normal cone, $\mathcal{C}_i^n$, we can also define its complementary cone, $\mathcal{C}_i^c$ (see Figure 1 (b)), which contains all vectors that are orthogonal to vectors in $\mathcal{C}_i^n$:

$$\mathcal{C}_i^c = \left\{ \boldsymbol{w} \in \boldsymbol{R}^d \mid \exists \boldsymbol{u} \in \mathcal{C}_i^n \text{ such that } \langle \boldsymbol{u}, \boldsymbol{w} \rangle = 0 \right\}. \tag{4}$$

$\mathcal{C}_i^c$ is also called the tangent cone [6] and it contains the tangent space or all possible tangent directions of the implicit hyper-surface $\mathcal{F}_i(\boldsymbol{u}) = 0$. $\mathcal{C}_i^c$ can easily be derived from $\mathcal{C}_i^n$ as follows (see Figure 1):

$$\mathcal{C}_i^c = \mathcal{C}^{out}(\overline{\boldsymbol{v}}_i^c, \alpha_i^c) = \mathcal{C}^{out}(\overline{\boldsymbol{v}}_i^n, 90 - \alpha_i^n), \tag{5}$$

where $\overline{\boldsymbol{v}}_i^c = \overline{\boldsymbol{v}}_i^n$. In other words, $\mathcal{C}_i^c$ and $\mathcal{C}_i^n$ share the same axis, but have complementary angles. Finally, let $\mathcal{C}_i^c[\boldsymbol{u}^0] \subset \boldsymbol{R}^d$ denote the translation of $\mathcal{C}_i^c$ by $\boldsymbol{u}^0$, and from [1], we have $\{\boldsymbol{u} \mid \mathcal{F}_i(\boldsymbol{u}) = 0\} \subset \mathcal{C}_i^c[\boldsymbol{u}^0], \forall \boldsymbol{u}^0$ such that $\mathcal{F}_i(\boldsymbol{u}^0) = 0$.

The main result obtained in this section, following [1], is:

Theorem 1. *Given d implicit hyper-surfaces $\mathcal{F}_i(\boldsymbol{u}) = 0,\ i = 1, \cdots, d$, in $\boldsymbol{R}^d$, there exist at most one common solution to $\mathcal{F}_i(\boldsymbol{u})$ if*

$$\bigcap_{i=1}^{d} \mathcal{C}_i^c = \{\mathbf{0}\},$$

where $\mathbf{0}$ is the origin of the coordinate system, and $\mathcal{C}_i^c$ is the complementary tangent cone of $\mathcal{F}_i$.

Proof. Assume that $\boldsymbol{u}^0 \in \boldsymbol{R}^d$ is a common solution of the d equations $\mathcal{F}_i(\boldsymbol{u}) = 0,\ i = 1, \cdots, d$, and consider $\mathcal{C}_i^c[\boldsymbol{u}^0]$. From the relation $\{\boldsymbol{u} \mid \mathcal{F}_i(\boldsymbol{u}) = 0\} \subset \mathcal{C}_i^c[\boldsymbol{u}^0]$, we have

$$\bigcap_{i=1}^{d} \{\boldsymbol{u} \mid \mathcal{F}_i(\boldsymbol{u}) = 0\} \subset \bigcap_{i=1}^{d} \mathcal{C}_i^c[\boldsymbol{u}^0] = \{\boldsymbol{u}^0\}.$$

Thus, there can be no other common solution except $\boldsymbol{u}^0$. ■

Direct attempts to compute $\bigcap_{i=1}^{d} \mathcal{C}_i^c$ are bound to be highly inefficient. We are now ready to examine a tractable alternative.

3 Identifying Intersections of Bounding Cones

In this section, we present the first result of this paper — an algorithm, based on a dual representation, for identifying intersections of bounding cones in $\boldsymbol{R}^d$, following Theorem 1. Section 3.1 describes our dual representation of the bounding cones in $\boldsymbol{R}^d$, and Section 3.2 presents the algorithm that is based on this representation.

3.1 The Dual Representation of the Bounding Cones

The condition derived in Theorem 1 enables us, in theory, to detect when a sub-domain has at most a single solution. However, in order to verify this condition we have to check the intersections of the complementary tangent cones — a difficult problem even in $\boldsymbol{R}^3$ (see Figure 2(a)). In this section, we consider a dual representation of the tangent cone that enables one to reduce the query of whether the tangent cones intersect at a single point to the much simpler query of intersections of hyper-planes.

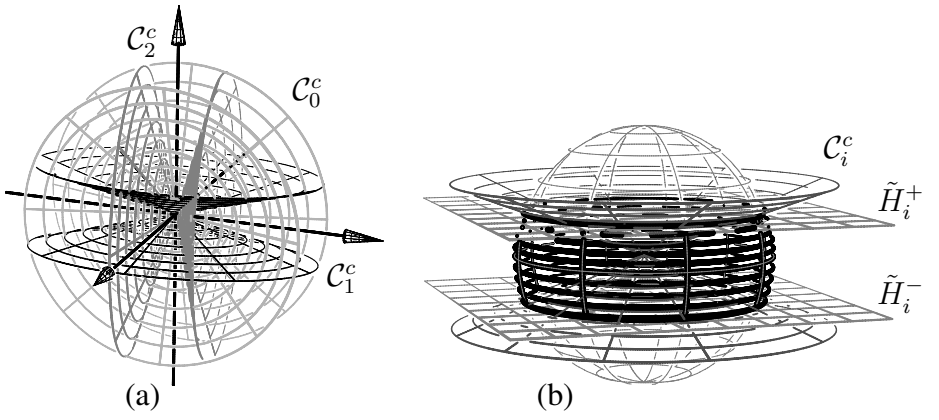


Fig. 2. (a) The intersection of three complementary cones in $\boldsymbol{R}^3$, $\mathcal{C}_i^c,\ i = 0, 1, 2$. (b) The (thick) strip on the unit sphere is the intersection of the unit sphere and the complementary cone, $\mathcal{C}_i^c$. It is also the intersection of the unit sphere and the region between the parallel delimiting planes, $\tilde{H}_i^+$ and $\tilde{H}_i^-$.

Let $\mathcal{C}_i^c(\overline{\boldsymbol{v}}_i, \alpha_i)$ be a given complementary tangent cone and let S^{d-1} denote the unit hyper-sphere in $\boldsymbol{R}^d$, i.e., $S^{d-1} = \{\boldsymbol{u} \in \boldsymbol{R}^d \mid \|\boldsymbol{u}\|^2 = 1\}$. Assigning S^{d-1} into the tangent cone definition, we get the intersection of the tangent cone and S^{d-1}:

$$\langle \boldsymbol{u}, \overline{\boldsymbol{v}}_i \rangle^2 = \cos^2 \alpha_i,$$

or

$$\langle \boldsymbol{u}, \overline{\boldsymbol{v}}_i \rangle = \pm |\cos \alpha_i|.$$

In other words, the intersection of the tangent cone $\mathcal{C}_i^c$ and the unit hyper-sphere is delimited by the symmetric parallel hyper-planes $\tilde{H}_i^+ : \langle \boldsymbol{u}, \overline{\boldsymbol{v}}_i \rangle = |\cos \alpha_i|$ and $\tilde{H}_i^- : \langle \boldsymbol{u}, \overline{\boldsymbol{v}}_i \rangle = -|\cos \alpha_i|$ (see Figure 2(b)). In explicit form, these hyper-planes will be written as

$$\tilde{H}_i^{\pm}(u_1, \cdots, u_d) : \quad v_1^i u_1 + v_2^i u_2 + \cdots + v_d^i u_d = \pm |\cos \alpha_i|,$$

where $\overline{\boldsymbol{v}}_i = (v_1^i, \cdots, v_d^i)$ is the normalized cone axis, α_i is the complementary cone's angle, and $\boldsymbol{u} = (u_1, \cdots, u_d)$ are the unknowns.

Thus, given a complementary tangent cone, its delimiting hyper-planes can be computed as described above. The converse is also true, given a delimiting hyper-plane

$$v_1^i u_1 + v_2^i u_2 + \cdots + v_d^i u_d \pm c = 0,$$

where the coefficient vector $\overline{\boldsymbol{v}}_i$ is normalized and $\pm c \in [-1, 1]$, the complementary tangent cone is simply the cone $\mathcal{C}_i^c(\overline{\boldsymbol{v}}_i, \arccos(c))$. Therefore, there is a one-to-one mapping between the complementary tangent cones and their dual delimiting hyper-planes, over S^{d-1}.

The duality between the tangent cone and its delimiting hyper-planes over S^{d-1} enables us to represent the complementary cones by their delimiting hyper-planes. More

precisely, the complementary cone is represented as the strip on the unit hyper-sphere, which is the intersection of the unit hyper-sphere and the region bounded between the delimiting hyper-planes (note the thick lines' strip of S^2 in Figure 2(b)).

Let H_i^+ be the half-space bounded by $\tilde{H}_i^+$ and containing the origin (i.e., the half-space defined by the equation $\langle \boldsymbol{u}, \overline{\boldsymbol{v}}_i \rangle \leq |\cos \alpha_i|$), and similarly let H_i^- be the half-space bounded by $\tilde{H}_i^-$ and containing the origin. The strip on the unit hyper-sphere is, therefore, the intersection $H_i^+ \bigcap H_i^- \bigcap S^{d-1}$. The intersection of the d cones can, consequently, be represented as the intersection of the unit hyper-sphere with the regions bounded by the delimiting hyper-planes:

$$\bigcap_{i=1}^{d} \{H_i^+ \bigcap H_i^- \bigcap S^{d-1}\} = S^{d-1} \bigcap_{i=1}^{d} \{H_i^+ \bigcap H_i^-\}.$$

3.2 Algorithm for Identifying the Uniqueness of a Solution

In Section 3.1, we introduced a dual representation of the complementary tangent cones by the intersection of their delimiting hyper-planes and the unit hyper-sphere. This representation enables one to represent the intersection of a set of complementary cones as the intersection of the regions bounded by their delimiting hyper-planes (i.e., an intersection of half-spaces) and the unit hyper-sphere. If the axes of the d complementary cones are linearly independent, then the intersection between the d infinite regions bounded by the delimiting hyper-planes is a bounded convex polytope (see Figure 3 for examples in $\boldsymbol{R}^3$), where the vertices of the polytope are the d-dimensional points of intersection between subsets of d hyper-planes.[1] The following gives the correspondence between the intersection of complementary cones and the intersection of the convex polytope and S^{d-1}.

Lemma 1. *$\bigcap_{i=1}^{d} \mathcal{C}_i^c$ contains a vector other than $\mathbf{0}$, if and only if the intersection $S^{d-1} \bigcap_{i=1}^{d} \{H_i^+ \bigcap H_i^-\}$ is not empty, where $\mathcal{C}_i^c$ are the complementary cones and H_i^+ and H_i^- are the bounding half-spaces.*

Proof. If there is a non-trivial intersection between the complementary cones, then there exists a vector $\boldsymbol{u} \neq \mathbf{0}$ such that $\boldsymbol{u} \in \mathcal{C}_i^c$ for all i. Therefore, for all $c \in R$, $c\boldsymbol{u} \in \mathcal{C}_i^c$ for all i. In particular, there is a unit-length vector $\overline{\boldsymbol{u}}$ (i.e., $\overline{\boldsymbol{u}} \in S^{d-1}$), such that $\overline{\boldsymbol{u}} \in \mathcal{C}_i^c$ for all i. However, by duality, $S^{d-1} \bigcap \mathcal{C}_i^c = S^{d-1} \bigcap \{H_i^+ \bigcap H_i^-\}$, and so we have $\overline{\boldsymbol{u}} \in S^{d-1} \bigcap \{H_i^+ \bigcap H_i^-\}$ for all i. Thus, $S^{d-1} \bigcap_{i=1}^{d} \{H_i^+ \bigcap H_i^-\}$ is not empty.

On the other hand, if $S^{d-1} \bigcap_{i=1}^{d} \{H_i^+ \bigcap H_i^-\}$ is not the empty set, then there exists a unit vector $\overline{\boldsymbol{u}} \in S^{d-1} \bigcap_{i=1}^{d} \{H_i^+ \bigcap H_i^-\}$. This means that $\overline{\boldsymbol{u}} \in S^{d-1} \bigcap \mathcal{C}_i^c$ for all i, and therefore $\bigcap_{i=1}^{d} \mathcal{C}_i^c$ contains a vector other than $\mathbf{0}$. ■

Since $\bigcap_{i=1}^{d} \{H_i^+ \bigcap H_i^-\}$ is the bounded convex polytope, then from Lemma 1 it follows that there is an intersection of the complementary cones at a location other than the

[1] In the degenerate case where the axes are not linearly independent, the polytope will be unbounded and thus $S^{d-1} \bigcap_{i=1}^{d} \{H_i^+ \bigcap H_i^-\}$ will not be empty, therefore, by Lemma 1, $\bigcap_{i=1}^{d} \mathcal{C}_i^c$ can contain more than a single point and the termination condition will fail.

origin if and only if there is an intersection of the convex polytope and S^{d-1} in $\boldsymbol{R}^d$. While computing the intersection of a convex polytope with S^{d-1} in $\boldsymbol{R}^d$ can be a non-trivial task, for our needs a smaller task is actually required. We are only interested in knowing whether the intersection of the strips on the hyper-sphere is an empty set. If the intersection is an empty set, then the complementary cones intersect only at the origin. But, because of convexity, an empty set can result only if all vertices of the convex polytope are inside the unit hyper-sphere (see Figure 3(b)). Thus, herein, we simply need to compute all the vertices of the polytope and check whether any of them is outside the unit hyper-sphere (i.e., if their distance from the origin is larger than one).

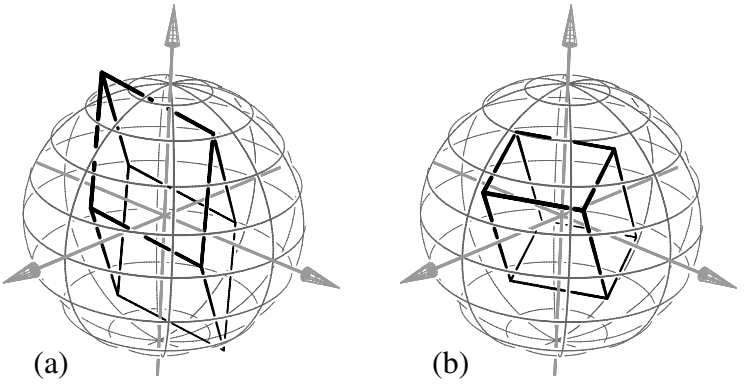


Fig. 3. (a) A non-empty intersection corresponding to a vertex of the polytope outside the unit sphere. (b) All vertices of the polytope are inside the unit sphere and hence the complementary cones intersect only at the origin.

The number of vertices of the convex polytope is the number of possible combinations of d intersecting hyper-planes. In the general case of $2d$ hyper-planes, the number of vertices is $\binom{2d}{d}$. However, since in our case the hyper-planes are arranged in parallel pairs, only 2^d intersections can occur.[2] Furthermore, because of symmetry, if a vertex $\boldsymbol{u}$ is inside (outside) the unit hyper-sphere, then so will its antipodal vertex, $-\boldsymbol{u}$. Thus, it is sufficient to check at most 2^{d-1} combinations of hyper-plane intersections.

Computing the intersection of d hyper-planes amounts to solving the set of d linear equations:

$$v_1^i u_1 + v_2^i u_2 + \cdots + v_d^i u_d = \pm|\cos\alpha_i|, \quad i = 1 \cdots d \tag{6}$$

where the signs of the terms on the right change according to the combination of hyper-planes we are checking.

Solving Equation (6) can be done using any standard method such as LU or QR factorization [10]. Note that the matrix inversion/factorization itself needs to be performed only once, and then for each permutation only backtracking is performed.

If we assume factorization of a $d \times d$ matrix takes $O(d^c)$ operations, where $2 < c < 3$ (and for practical purposes $c = 3$), then computing all the 2^{d-1} solutions takes

[2] This can be viewed as a d-digit binary number, each digit representing a cone, where one can choose either a positive or a negative delimiting hyper-plane but not both.

$O(d^c + 2^{d-1}d^2)$ operations.[3] This can be improved, however, using the symmetry of the solution vector. Let $b_i = |\cos\alpha_i|$, for $i = 1, \cdots, d$. Then, the system of equations $V\boldsymbol{u} = \{\pm b_i\}$, presented in Equation (6), can be rewritten as:

$$V\boldsymbol{u} = \Sigma_{i=1}^{d} \pm b_i \boldsymbol{e}_i,$$

where $\boldsymbol{e}_i$ are the standard base vectors. The solution is thus:

$$\boldsymbol{u} = \Sigma_{i=1}^{d} \pm b_i V^{-1} \boldsymbol{e}_i.$$

Therefore, if we solve for $\boldsymbol{c}_i = V^{-1}\boldsymbol{e}_i$ once, we can encode all 2^{d-1} solutions as the combinations:

$$+b_1\boldsymbol{c}_1 \cdots \pm b_i\boldsymbol{c}_i \cdots \pm b_d\boldsymbol{c}_d.$$

We can code the 2^{d-1} solutions by their sequences of pluses and minuses. Assigning 1 for a $+$ and a 0 for a $-$, we get a d-digit binary number. For example, the solution, $+b_1\boldsymbol{c}_1 \cdots - b_i\boldsymbol{c}_i \cdots - b_d\boldsymbol{c}_d$, corresponds to the d-digit number $100...00$.

Now with the solution $\boldsymbol{u}$, computing a solution with an identical code except for one bit takes only $O(d)$ operations. For example, for $d = 3$ computing $\boldsymbol{u}_{101}$, after we have already computed $\boldsymbol{u}_{100}$, amounts to computing $\boldsymbol{u}_{100} + 2b_3\boldsymbol{c}_3$, i.e., a single vector addition, taking $O(d)$ operations. Thus, if we order the solutions according to Gray coding [11], where adjacent binary numbers differ by a single bit, the overall time for computing all the solutions will be reduced to $O(d^c + 2^{d-1}d)$.

We can now summarize the full algorithm for identifying whether a sub-domain contains at most a single solution:

1. Compute the normal cones, $\mathcal{C}_i^n$, $i = 1, \cdots, d$, for each hyper-surface as described in Section 2.
2. For each cone, extract the pair of delimiting hyper-planes as described in Section 3.1.
3. For each of the 2^{d-1} combinations in Equation (6):
   (a) Solve the set of equations. Let $\boldsymbol{u}$ be the d-dimensional solution vector.
   (b) If $\|\boldsymbol{u}\| \geq 1$ return False, i.e., the sub-domain is not guaranteed to have at most a single solution.
4. Return True. The sub-domain is guaranteed to have at most a single solution.

4 Purging Away Zero-Solution Domains

The algorithm presented in Section 3 identifies whether a sub-domain has at most a single solution. However, it does not guarantee that a solution exists in the sub-domain. Thus, we might terminate the subdivision and start the numerical iterations only to find that no root exists in this sub-domain. Such a numerical step, starting at a sub-domain with no zeros, can require a relatively large number of iterations, since the initial point might be far from a root. Therefore, we seek to purge away, as much as possible,

[3] 2^{d-1} times doing back substitution of the solution, where each back substitution takes $O(d^2)$ operations.

sub-domains that contain no solution. In this section, we present a second criterion for identifying sub-domains with no solution, in addition to the whole-positivity criterion presented in the introduction.

Recall the scalar Bézier/B-spline form of $\mathcal{F}_i$, in Equation (2). We consider them as hyper-surfaces in $\boldsymbol{R}^{d+1}$, and use the Bézier/B-spline representation to bound the hyper-surface with two parallel hyper-planes. We denote by *promotion* the process of converting the *scalar* function $\mathcal{F}_i(u_1, u_2, \cdots, u_d)$ to its *vector* function counterpart $\hat{\mathcal{F}}_i$: $\boldsymbol{R}^d \rightarrow \boldsymbol{R}^{d+1}$, $\hat{\mathcal{F}}_i = (u_1, u_2, \cdots, u_d, \mathcal{F}_i)$. Promotion is performed by employing the nodal points, also known as the Greville abssica [12], for the first d dimensions. For the Bézier case, the nodal points are simply $(i_1/k_{i_1}, i_2/k_{i_2}, \cdots, i_d/k_{i_d})$.

As the subdivision process progresses, the smaller and smaller sub-domains are likely to become almost (hyper-) planar. Hence, we can bound $\hat{\mathcal{F}}_i$ by two parallel hyper-planes.[4] These two bounding hyper-planes are constructed as follows. Compute the unit normal, $\overline{\boldsymbol{n}}_i = (n_1, \cdots, n_{d+1}) \in \boldsymbol{R}^{d+1}$, of $\hat{\mathcal{F}}_i$ at the midpoint of the sub-domain. Then, project all control points of $\hat{\mathcal{F}}_i$ onto $\overline{\boldsymbol{n}}_i$.

Denote by $\boldsymbol{u}_{max} \in \boldsymbol{R}^{d+1}$ (resp. $\boldsymbol{u}_{min} \in \boldsymbol{R}^{d+1}$) the point on $\overline{\boldsymbol{n}}_i$ that is the maximal (resp. minimal) projection of the control points onto $\overline{\boldsymbol{v}}_i$. Then, the $(d+1)$-dimensional parallel bounding hyper-planes of $\mathcal{F}_i$ are:

$$\langle \boldsymbol{u} - \boldsymbol{u}_{min}, \overline{\boldsymbol{n}}_i \rangle = 0, \qquad \langle \boldsymbol{u} - \boldsymbol{u}_{max}, \overline{\boldsymbol{n}}_i \rangle = 0,$$

where $\boldsymbol{u} \in \boldsymbol{R}^{d+1}$.

Since we are only interested in bounding $\mathcal{F}_i = 0$, we only need the intersection of these two $(d+1)$-dimensional hyper-planes with the $u_{d+1} = 0$ hyper-plane. Eliminating the u_{d+1} coordinate, we remain with the d-dimensional hyper-planes bounding $\mathcal{F}_i = 0$:

$$\tilde{K}_i^{min} : \quad \langle \boldsymbol{u}, \overline{\boldsymbol{n}}_i \rangle \big|_{u_{d+1}=0} = u_1 n_1^i + u_2 n_2^i + \cdots + u_d n_d^i = b_i^{min},$$

and

$$\tilde{K}_i^{max} : \quad \langle \boldsymbol{u}, \overline{\boldsymbol{n}}_i \rangle \big|_{u_{d+1}=0} = u_1 n_1^i + u_2 n_2^i + \cdots + u_d n_d^i = b_i^{max},$$

where $b_i^{min} = \langle \boldsymbol{u}_{min}, \overline{\boldsymbol{n}}_i \rangle$ and $b_i^{max} = \langle \boldsymbol{u}_{max}, \overline{\boldsymbol{n}}_i \rangle$.

As in Section 3.1, we denote by K_i^{min} and K_i^{max} the half-spaces bounded by $\tilde{K}_i^{min}$ and $\tilde{K}_i^{max}$, oriented so that $\mathcal{F}_i = 0$ is on their positive side. $\mathcal{F}_i = 0$ is thus bounded in the region $K_i^{min} \bigcap K_i^{max}$. Given a pair of bounding hyper-planes $\tilde{K}_i^{min}$ and $\tilde{K}_i^{max}$ for each constraint $\mathcal{F}_i = 0$, we have to determine whether the d-dimensional polytope defined by $\bigcap_{i=1}^{d} \{K_i^{min} \bigcap K_i^{max}\}$ is entirely outside of the sub-domain. Being outside the sub-domain, no zeros can exist in this sub-domain.

This intersection problem can be solved using linear programming methods (see, for example, [13][Chapter 29] or [9][Chapter 4]), by adding the $2d$ half-spaces $\tilde{K}_i^{min}$ and $\tilde{K}_i^{max}$, for $i = 1, \cdots, d$, to the $2d$ hyper-planes of the sub-domain itself. If there is no feasible solution to the linear programming problem defined by these $4d$ hyper-planes, the solution is guaranteed to be entirely outside of the domain.

While the problem can indeed be solved using general linear programming methods, in our implementation and for reasons of efficiency, we chose to solve it in a different way. The configuration of the problem resembles the configuration in Section 3.2,

[4] These two parallel hyper-planes are unrelated to the pairs of hyper-planes used in Section 3.

namely identifying whether an intersection of parallel-hyper-plane pairs is inside or outside some region. Therefore, we computed the 2^d vertices of the polytope using the same procedure that was presented in Section 3.2. Here instead of switching between $+b_i$ and $-b_i$, we now switch between b_i^{min} and b_i^{max}. If all the vertices of the polytope are outside of one half-space bounding the sub-domain, then the solution is verified to be entirely outside of the sub-domain.

From our experience, solving the problem in the same manner as in Section 3.2 creates little overhead, since most of the algorithm's time is spent on constructing the pairs of hyper-planes from the $\mathcal{F}_i$ constraints. On the other hand, computing all the vertices has the advantage that now we can use all these intersection locations for better clipping of the sub-domain (see possible future extensions in Section 6).

5 Examples

Figures 4 and 5 present a simple example of the solution of two bivariate constraints. The two constraints are expressed as bicubic Béziers:

$$\mathcal{F}_1(u,v)=\sum_{i=0}^{3}\sum_{j=0}^{3} P_{ij}B_i(u)B_j(v), \qquad P_{ij} = \begin{bmatrix} 0 & 2.2 & 1.1 & 1.1 \\ -1 & -1 & -1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & -1 & -2 & 0 \end{bmatrix}, \qquad u,v \in [0,1], \tag{7}$$

and $\mathcal{F}_2(u,v) = \mathcal{F}_1(v,u)$ (reflected along the diagonal). The simultaneous solution of $\mathcal{F}_1(u,v) = \mathcal{F}_2(u,v) = 0$, for $u,v \in [-1,2]$ yields seven roots. Figure 4 (a) presents the entire subdivision tree for a subdivision tolerance of 10^{-3}. In (b), a zoom-in of the thick square area near the center of Figure 4 (a)) is presented. The whole region between these two (close) roots is finely subdivided, only to determine that no other root exists in between.

Figure 5 (a) presents the same subdivision tree, but this time with the introduced single solution cone test, presented in Section 3. The subdivision tree is now (compared to Figure 4 (a)) much smaller. A zoom-in on the region between the two closest roots in Figure 5 (b) reveals how efficiently the roots were isolated, compared to Figure 4. Only in the middle, between the two roots in Figure 5 (a), were some excessive subdivisions applied. This is due to the fact that the two zero sets of $\mathcal{F}_1$ and $\mathcal{F}_2$ are almost parallel, rendering the cone test invalid.

In Figure 6, the same subdivision tree with cells that passed both the single solution test and the parallel hyper-plane test (Section 4) are presented. The numerical step will be performed only on the marked cells. We can see that the parallel hyper-plane test not only significantly reduced the total number of cells in the subdivision tree but also reduced the number of cells on which a numerical step is performed. Moreover, the sub-domains where the two zero sets of $\mathcal{F}_1$ and $\mathcal{F}_2$ are almost parallel, rendering the cone test invalid, were successfully detected and purged away by the parallel hyper-planes test.

All examples presented in this section used an implementation that incorporated the presented algorithms into the multivariate solver of the IRIT [4] solid modeling system.

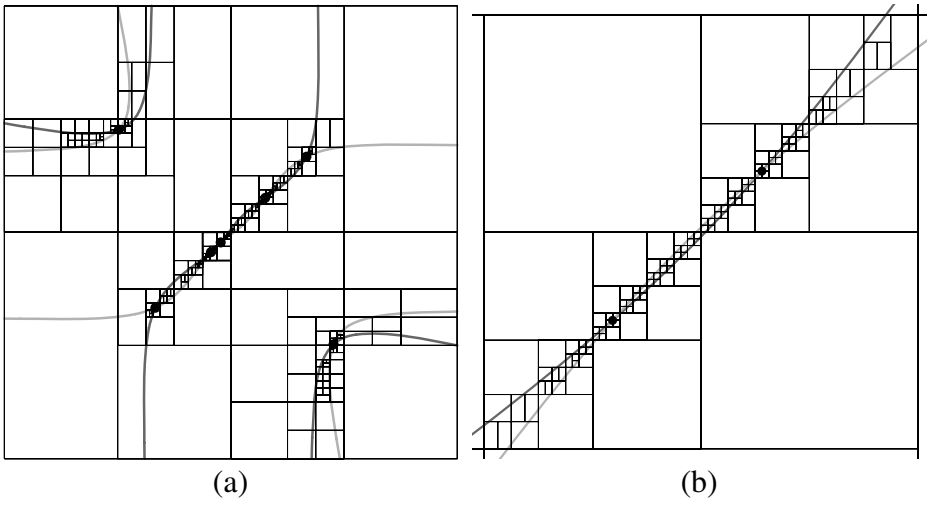


Fig. 4. (a) The zeros of two implicit bi-cubics, $\mathcal{F}_1$ and $\mathcal{F}_2$ (in two shades of gray) along with their simultaneous zero points (see Equation (7)), without the single solution cone test. In (a), the seven zero points are presented along with the subdivision tree. (b) shows a zoom-in over the thick square region near the center of (a). See also Figure 5.

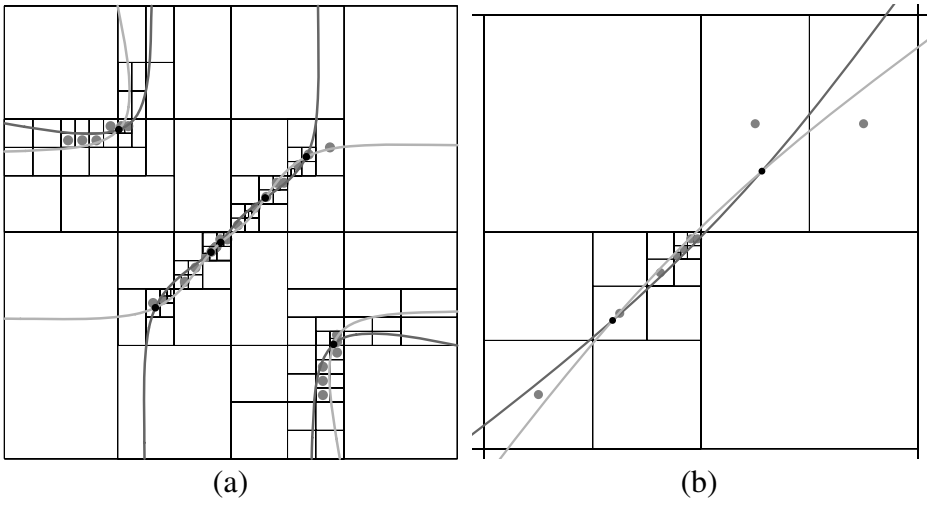


Fig. 5. (a) The zeros of two implicit bi-cubics, $\mathcal{F}_1$ and $\mathcal{F}_2$ along with their simultaneous zero points (see Equation (7)), with the introduced single solution test. The zero points are marked by black dots. In (a), the seven zero points are presented along with the subdivision tree. (b) shows a zoom-in over the thick square region near the center of (a). Compare with Figure 4. The cells where the subdivision process terminated because of the successful single solution test are marked by gray dots. Compare with Figure 6.

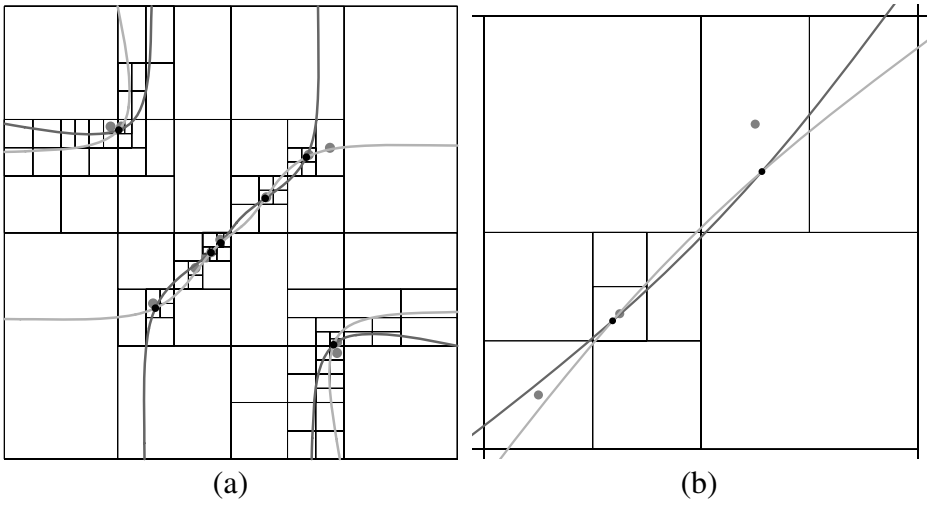


Fig. 6. (a) Cells where the termination of the subdivision process was due to the success of both the single solution test and the parallel hyper-plane test. Again, cells where the single solution test was successful are marked by gray dots. (b) shows a zoom-in over the thick square region near the center of (a). Compare with Figure 5.

6 Conclusions

In this paper, we have presented efficient and simple algorithms for identifying no and single solution sub-domains in a set of d implicit constraints in d unknowns. The algorithms are based on a dual representation of the normal cones as hyper-planes over the unit hyper-sphere in $\mathbb{R}^d$ and on pairs of bounding hyper-planes. These algorithms offer ways to improve the performance of subdivision based solvers such as the ones presented in [1,2,3].

While the single solution test provides a guarantee that all separable roots in the solution set have been isolated correctly, the computational costs of computing these termination criteria are not trivial and effort should be made to optimize these costs. We plan to compare our single solution test to other root isolation approaches such as multivariate Sturm sequences [14] and the Kantorovich theorem [15]. The ideas presented in this paper can be extended for further improvement of subdivision-based solvers. The solution is guaranteed to be within the polytope bounded by the hyper-plane pairs. Thus, the vertices of the bounding polytope, which were computed in Section 4, can be further used for clipping the domain, and not just for purging away zero-solution domains, as was presented.

The proper handling of under- (and over-) constrained systems, when the number of degrees of freedom is larger (smaller) than the number of constraints and the zero-set is a k-manifold, $k > 0$, is also of major interest in geometric applications. Surface-surface intersection is just one simple instance. While the extension of Theorem 1 to under-constrained systems is not trivial, the bounding hyper-planes criterion (presented

in Section 4) can also be employed in these systems. Given an under-constrained system, the polytope that is the intersection of the parallel hyper-plane pairs is unbounded. Still, by using linear programming methods, we can purge away sub-domains with no solutions. If the sub-domain potentially contains roots, we can intersect the unbounded polytope and the hyper-planes bounding the sub-domain itself to receive a smaller sub-domain. It remains to be seen whether this scheme is beneficial. We leave these extensions for future work.

Acknowledgment

The authors are grateful to Avraham Sidi and Moshe Israeli for their help in reducing the complexity of solving Equation (6). This research was supported in part by the Israel Science Foundation (grant No. 857/04) and in part by European FP6 NoE grant 506766 (AIM@SHAPE).

References

1. Elber, G., Kim, M.S.: Geometric constraint solver using multivariate rational spline functions. In: Proceedings of the Symposium on Solid Modeling and Applications 2001, Ann Arbor, Michigan (2001) 1–10
2. Mourrain, B., Pavone, J.P.: Subdivision methods for solving polynomial equations. Technical report, INRIA Sophia-Antipolis (2005)
3. Sherbrooke, E.C., Patrikalakis, N.M.: Computation of the solutions of nonlinear polynomial systems. Computer Aided Geometric Design **10**(5) (1993) 279–405
4. Elber, G.: The IRIT 9.5 User Manual. (2005) http://www.cs.technion.ac.il/ irit.
5. Lane, J., Riesenfeld, R.: Bounds on a polynomial. BIT **21** (1981) 112–117
6. Sederberg, T.W., Meyers, R.J.: Loop detection in surface patch intersections. Computer Aided Geometric Design **5**(2) (1988) 161–171
7. Sederberg, T.W., Zundel, A.K.: Pyramids that bound surface patches. Graphical Models and Image Processing **58**(1) (1996) 75–81
8. Barequet, G., Elber, G.: Optimal bounding cones of vectors in three and higher dimensions. Information Processing Letters **93** (2005) 83–89
9. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry, Algorithms and Applications. 2nd edn. Springer, New York (1998)
10. Golub, G.H., Loan, C.F.V.: Matrix Computation. 3rd edn. The Johns Hopkins University Press, Baltimore and London (1996)
11. Weisstein, E.W.: (Gray code) From MathWorld — A Wolfram Web Resource, http://mathworld.wolfram.com/GrayCode.html.
12. Farin, G.E.: Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide. 4th edn. Academic Press (1996)
13. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. 2nd edn. MIT Press and McGraw-Hill (2001)
14. Milne, P.S.: On the solutions of a set of polynomial equations. In: Symbolic and Numerical Computation for Artificial Intelligence. Academic Press (1992) 89–102
15. Dennis, J.E., Schnabel, R.B.: Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice Hall Series in Computational Mathematics. Prentice Hall Inc. (1983)

Towards Unsupervised Segmentation of Semi-rigid Low-Resolution Molecular Surfaces

Yusu Wang[1] and Leonidas J. Guibas[2]

[1]Department of Computer Science and Engineering, the Ohio State University, Columbus, OH 43210
yusu@cse.ohio-state.edu
[2]Department of Computer Science, Stanford University, Stanford, CA 94305
guibas@graphics.stanford.edu

Abstract. In this paper, we study a particular type of surface segmentation problem motivated by molecular biology applications. In particular, two input surfaces are given, coarsely modeling two different conformations of a molecule undergoing a semi-rigid deformation. The molecule consists of two subunits that move in a roughly rigid manner. The goal is to segment the input surfaces into these semi-rigid subcomponents. The problem is closely related to non-rigid surface registration problems, although considering only a special type of deformation that exists commonly in macromolecular movements (such as the popular *hinge motion*). We present and implement an efficient paradigm for this problem, which combines several existing and new ideas. We demonstrate the performance of our new algorithm by some preliminary experimental results in segmenting low-resolution molecular surfaces.

1 Introduction

Registration of shapes is an important problem arising in many research areas, such as computer graphics, vision, pattern recognition, and structural biology. In general, given two structures represented as, say, surfaces, one wishes to identify for every point from one surface the *corresponding point* from the other. This registration problem is closely related to the problem of measuring shape similarities.

Much of previous work has been focused on the so-called *rigid-body registration*, where given two structures A and B, the goal is to find the best rigid transformation for B so that the distance between A and B is minimized (and thus the similarity between A and B is maximized). In this paper, we consider the case where the input object consists of a small number of components. Each component moves roughly in a rigid manner. Given two conformations of this object, represented by surfaces S_A and S_B, we wish to segment S_A (and/or S_B) into these components without any prior knowledge on their correspondences, and call the resulting problem the *semi-rigid segmentation* problem. Note that the solution of this problem, namely the segmentation as well as the correspondences between the respective subunits, can then be used to find a full registration between the two input surfaces.

Motivation. The semi-rigid segmentation problem has many applications. For example, in human body tracking, the major body parts undergo hinge-type motions. Our main

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 129–142, 2006.
© Springer-Verlag Berlin Heidelberg 2006

motivation, however, comes from molecular structural biology. In particular, a macromolecule (e.g, a protein) may change its conformation significantly during important biological processes. With current structure determination technologies, it is possible to obtain some "snapshots" of this deformation process, such as the beginning and ending conformations. In order to understand the entire deformation, it is necessary to find correspondences among these few obtained conformations. On the other hand, high-resolution structures, which include the types and positions of atoms in a molecule, are very hard and time-consuming to obtain. More and more research turn to low-resolution structural data, like cryo-EM data, where the atomic information of the molecule is not available. As such, effectively, we are only given two surfaces in arbitrary orientation with no correspondences information, and we wish to register them. One major type of macro-molecular deformation is the so-called "hinge-motion", where components of the molecule, usually protein domains, rotate around some "hinge pivot". The registration between conformations under hinge-type motion is essentially a semi-rigid segmentation problem.

Related work. Registration has been widely studied and used in a broad range of applications, where input objects can be image data, curves, surfaces, and so on. We refer the readers to [1,2] for some surveys on this topic. Below we give a brief review for surface registration methods, with a focus on non-rigid cases, as well as related work in the field of molecular structural biology.

One basic challenge in the registration problem is that there are two subproblems needed to be optimized simultaneously, and the solutions of them interact with each other. In particular, one needs to find both the right *transformation* to align the two surfaces, and the *correspondences* between points from input surfaces. To make things worse, the input data are usually noisy and possibly incomplete (thus may require partial matching). Hence the problem is complex even for rigid-body registrations. To make the problem computationally more manageable, many approaches first extract *features* from input objects and check only those transformations that align compatible features. In other words, they only sample the transformational space at a few potential positions. The problem then becomes how to capture representative and discriminative features [3,4,5,6,7,8]. There are also several popular paradigms for the rigid-body registration that can be combined with the use of features, such as the Iterative closest point (ICP) algorithm [9] and Geometric hashing technique [10]. Nevertheless, despite a great amount of research devoted to the rigid-body registration problem, the field still contains many unanswered questions.

The non-rigid registration problem is much harder. Much research on this topic is motivated by applications from computer vision to track patterns in video sequences [11] or in medical image processing for describing deformations of organs such as the brain and heart [12,13]. Different methods have been developed for different types of objects. Given the temporal coherence that exists in video sequences, the deformation between two consecutive conformations is usually not too great. Some (physics-based) deformable model is usually built for the input object to help to establish correspondences between two consecutive conformations, as well as the deformation between them [2,11,13,14]. Most works assume either (some of) the correspondences are known, or that the objects are already roughly aligned. For example, a few correspondences can

be provided a priori, obtained either manually [15,16] or by attaching markers to the object [17,18,19]. There is also a lot of research on the articulated motion, such as tracking human bodies or hand gestures [20,21], which is similar to the type of semi-rigid deformations we consider. In general, the problem of non-rigid registration remains largely open.

Registration also plays an important role in computational biology. The widely investigated problem of protein structure classification is essentially curve matching, where proteins are represented by their backbone curves [22,23,24]; and surface registration is crucial to the study of protein-protein recognition, which can be modeled as a partial surface matching problem under the constraint that the two molecular surfaces do not penetrate each other much [25,26,27]. Most previous work has focused on rigid-body registration. For non-rigid cases, the normal modes analysis (NMA), including the so called elastic network theory, has been one of the main tools for modeling deformations (other than molecular dynamics). NMA is a powerful tool and has obtained success in several cases [28,29,30,31]. Such approaches usually start with a high-resolution structure where both the type and the position of each atom is known, although recently there are some work on constructing elastic network by discretizing a low-resolution model [32,33,34]. The deformable model (i.e, the normal modes) are constructed based purely on one single conformation. Hence it does not take advantage of the other conformations available, neither can it produce the registration between two conformations directly. Recently, in [30], in order to fit a high-resolution X-ray structure into a low-resolution cryo-EM map (which can be considered as the registration between the two structures), they first represent the X-ray structure in a virtual coordinate system using the low-order normal modes as bases. They then optimize the structure in this coordinate system to best match the EM map (roughly a density map). Initial alignments between the two input structures are required, and the involved optimization problem is time-consuming to solve (the time is usually measured in hours). Finally, for hinge-type motion, there is research aiming at segmenting the input structure into semi-rigid subunits, as well as identifying the hinge pivot, when both the high-resolution structures and correspondences between input conformations are given [35,36].

Our contribution. We focus on a specific type of deformation of molecular structures, where several subunits (domains) of a molecule deform in a roughly rigid manner. This includes the most popular types of macromolecular movements, such as the hinge motion, as well as many instances of the so-called "shear" motion. Our goal is to develop a simple geometric approach that can identify large deforming subunits reliably and automatically. Compared to previous more general approaches, our method is efficient and robust, and works when no high-resolution structures and prior correspondences are available at all. The resulting segmentation of input structure into semi-rigid components can then serve as inputs for more refined registration procedures, or for visualization tools so that biologists can inspect them visually to obtain insights. Furthermore, since the registration between two surfaces becomes relatively easy once the segmentation and correspondences are determined, our approach can facilitate the search of a given structure (say, some protein) in the low-resolution structure data of a large and complex system (such as a ribosome), in which case efficiency is a crucial factor.

More specifically, we consider the aforementioned semi-rigid segmentation problem for two given molecular surfaces S_A and S_B. We design and develop a segmentation framework that combines several ideas, some existed already, in a novel way: The new approach uses landmark-based virtual coordinates instead of the usual Cartesian coordinates to handle deformations. In order to identify the landmarks automatically, we exploit a voting idea, taking advantages of a set of potentially good rigid registrations. In particular, by exploiting the feature pairs computed from the so-called elevation function [37], we develop a scheme to vote for *landmarks* using *landmark-based* coordinates. This scheme is facilitated by the one-to-one correspondence between rigid transformations and pairs of feature pairs existed in our framework. Finally, the extracted landmarks also give rise to a natural procedure to segment input molecular surfaces. The entire framework is easy to implement and we present preliminary experimental results at the end to demonstrate its performances.

2 Coarse Rigid Registrations

First, we compute a set of coarse rigid registrations between S_A and S_B. Our approach is based on the *elevation function* $\mathrm{Elev} : S \to \mathbb{R}$ over a surface S introduced in [37]. Roughly speaking, every point $x \in S$ has a canonical pairing partner y that shares the same normal direction n_x with x: the pair (x, y) describes a feature in direction n_x, and $\mathrm{Elev}(x)$, defined as the height difference between x and y in this direction, indicates the size of this feature (See Figure 1 (a) for an illustration in the plane). Furthermore, the set of maxima of the elevation function, together with their canonical pairing partners, form a set of *feature pairs*, capturing important protrusions and cavities from the given surface. Each feature pair $s = (x, y)$ consists of a pair of points x and y, together with their common normal and the elevation value. We sometimes refer to s as a *segment* and x, y as its endpoints.

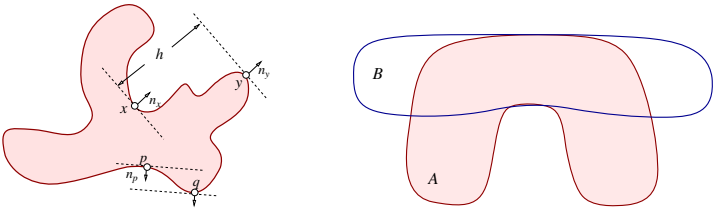


Fig. 1. (a) In 2D: x is paired with y with common normal. $\mathrm{Elev}(x) = \mathrm{Elev}(y) = h$. The maxima of elevation return a set of feature pairs like (x, y) and (p, q). (b) The top-scored registration between A and B may not align any pair of corresponding components.

Pairs of feature-pairs and rigid transformations. In three dimensions, a *pair of feature-pairs (*PFP*)* (s_1, s_2), with s_1 from S_A and s_2 from S_B, is sufficient to determine a rigid transformation $\mu(s_1, s_2)$. This can be achieved by aligning the corresponding endpoints of s_1 and s_2, as well as the normal directions. Given surfaces S_A and S_B, we first

compute a set $\mathcal{F}(A)$ (resp. $\mathcal{F}(B)$) of feature pairs from S_A (resp. S_B) using the maxima of elevation function. The set of points involved in feature pairs from $\mathcal{F}(A)$ and $\mathcal{F}(B)$ are denoted by $\mathcal{P}(A)$ and $\mathcal{P}(B)$, respectively. We then consider only transformations produced by aligning a pair of feature-pairs, one from $\mathcal{F}(A)$ and one from $\mathcal{F}(B)$. The resulting set of transformations is denoted by Π. For each rigid transformation $\mu \in \Pi$, we compute its *score* based on some scoring function $\sigma(S_A, \mu(S_B))$ to measure how good μ is. We sort Π in decreasing order of their scores. At this state, we simply take $\sigma(S_A, \mu(S_B))$ as the number of pair of points $p \in \mathcal{P}(A)$ and $q \in \mu(\mathcal{P}(B))$ such that $d(p, q)$, the Euclidean distance between p and q, is smaller than some given threshold. We use the standard geometric hashing technique [10] to compute Π.

Let $n = |S_A|$ be the number of vertices in surface S_A; and $m = |S_B|$. The size of $\mathcal{F}(A)$ and $\mathcal{F}(B)$, thus $\mathcal{P}(A)$ and $\mathcal{P}(B)$, are upper bounded by $O(n)$ and $O(m)$ respectively. In practice, they are much smaller: around 100 in our experiments, as we only keep those pairs with large elevation value (thus more significant features). The size of Π is bounded by $|\mathcal{F}(A)| \cdot |\mathcal{F}(B)| = O(nm)$.

Finally, we remark that in this work, all transformations we consider are produced by aligning PFPs. Hence we sometimes do not distinguish between a transformation and a PFP. For example, the set of transformations Π obtained above can also be viewed as a set of PFPs.

3 Segmentation Algorithm

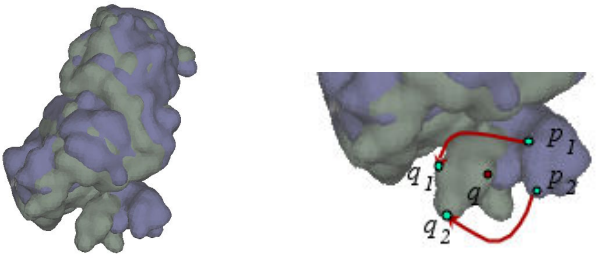
The method described above performs well for rigid transformations [27]. However, in our case, there are certain non-rigid deformation between input surfaces. Hence a rigid transformation is unable to identify all matching components at once. In order to segment the input surface, say S_A, into semi-rigid components, a natural approach consists of the following steps: (i) start with a coarse registration μ_1 that align one pair of components well, (ii) eliminate all points of S_A that are close to their correspondences on S_B under transformation μ_1, (iii) extract a good registration from the remaining points to identify the second pair of matching components, and (iv) repeat step (ii) and (iii) to identify more semi-rigid components.

There are, however, a few problems with this approach: (P1.) How to choose the first registration μ_1? The top-ranked transformation from Π is a natural choice, but it may in fact align none of the components well (see Figure 1 (b)). (P2.) How to define *correspondences* between points from S_A and S_B w.r.t. μ_1? If we use Euclidean distance to measure closeness, a point on a deformed component may find a completely wrong correspondence, which will induce wrong registration at step (iii). (P3.) How to extract the next matching components? (P4.) How to eventually segment S_A into semi-rigid components?

Below we first address problem P2, after which we describe our new algorithm.

3.1 Landmark-Based Coordinates

For rigid alignments, one common way to define the correspondence of $p \in S_A$ is by finding its nearest neighbor in S_B under Euclidean distance metric. This approach unfortunately fails for the non-rigid scenario. For example, in Figure 2 we are given two



(a) (b)

Fig. 2. (a) Two surfaces aligned by their main component. Under this registration, in (b), points p_1 and p_2 from S_A should correspond to q_1 and q_2 from S_B respectively. But they will most likely be matched to q under Euclidean distances.

surfaces where their main components are well-aligned, but not the small components. As such, for a point p in this small component, its nearest neighbor under the Euclidean distance can be quite far from its real correspondence q (Figure 2 (b)). This remains to be the case even if we augment the Euclidean distance with normal information. To find more reliable correspondences, we would like to use a different distance measure that is hopefully invariant w.r.t. the type of semi-rigid deformation we consider. Intuitively, given a few points $p_1, ..., p_k$ from S_A, as S_A undergoes hinge-type motion , the geodesic distance from any point $p \in S_A$ to p_i's does not vary significantly. This suggests the following landmark-based virtual coordinates for points from each surface.

Given surfaces S_A, a set of *landmarks* of S_A is simply a set of points $\mathrm{L}=\{\mathrm{l}_1, \ldots, \mathrm{l}_k\}$, $\mathrm{l}_i \in S_A$. Let $\mathrm{gd}(p, q)$ denote the geodesic distance from $p \in S_A$ to $q \in S_A$. We define the *landmark-based coordinate* of $p = (p_x, p_y, p_z)$ w.r.t. L as the following (k+3)-tuple

$$\mathbf{p} = \xi_{\mathrm{L}}(p) = \langle p_x, p_y, p_z, \mathrm{gd}(p, \mathrm{l}_1), \ldots, \mathrm{gd}(p, \mathrm{l}_k)\rangle.$$

The distance between two points p and q represented in landmark-based coordinates (i.e, for $\mathbf{p}$ and $\mathbf{q}$) is defined as

$$\delta_L(p, q) = \sqrt{\lambda \sum_{i=1}^{3} (\mathbf{p}[i] - \mathbf{q}[i])^2 + (1-\lambda) \sum_{i=4}^{k+3} (\mathbf{p}[i] - \mathbf{q}[i])^2}, \tag{1}$$

where λ specifies the weight of the Euclidean distance between p and q. We sometimes omit the subscript L from $\xi_{\mathrm{L}}()$ and $\delta_{\mathrm{L}}()$ when its choice is clear from the context.

Given surfaces S_A and S_B, a set of landmarks $\mathrm{L} = \{\mathrm{l}_1, \ldots, \mathrm{l}_k\}$ for S_A corresponding to a set of landmarks $\mathrm{M} = \{\mathrm{m}_1, \ldots, \mathrm{m}_k\}$ for S_B means that l_i corresponds to m_i for $1 \leq i \leq k$. We sometimes refer to set L and M as *matching sets of landmarks* for S_A and S_B; $\delta_L(p, q)$ can be extended for $p \in S_A$ and $q \in S_B$ once L and M are given.

3.2 Overview of the Algorithm

Ideally, to find the right correspondences for all points from S_A, we wish to have at least one landmark from each semi-rigid component of S_A, as well as one from its

corresponding component from S_B. Our algorithm aims at identifying each pair of matching components with one PFP, (s_1, s_2), with $s_1 \in \mathcal{F}(A)$ and $s_2 \in \mathcal{F}(B)$. The overview of our segmentation algorithm is shown in Figure 3. PREPROCESSING and STEP 1 were described in previous section. Next we explain steps 2 – 4 in detail. We remark how the four aforementioned problems (P1 to P4) are addressed in this new approach at the end of this section.

SegRigidComponents (S_A, S_B)

PREPROCESSING) Compute sets of feature pairs $\mathcal{F}(A)$ and $\mathcal{F}(B)$ for S_A and S_B
STEP 1) Construct and sort Π by aligning PFPs from $\mathcal{F}(A)$ and $\mathcal{F}(B)$
STEP 2) Compute a set of *reliable* PFPs
STEP 3) Compute matching sets of landmarks L and M
STEP 4) Segment S_A based on L and M

Fig. 3. Overview of our algorithm for the semi-rigid segmentation problem.

3.3 Landmark Selection

The goal here is to compute the matching sets of landmarks L and M for S_A and S_B. Note that a pair of matching landmarks, $(\mathrm{l}_i, \mathrm{m}_i)$, is nothing but two points corresponding to each other reliably. A natural approach is to first identify some feature points from each surface, and then establish reliable correspondences among them based on some shape descriptor around each point [5]. However, molecular surfaces are quite homogeneous in the sense that many points look alike in their local neighborhood. In order to be more discriminative, it is then desirable to use more complex features, such as pairs of points, or triple-points, instead of simply points. Unfortunately, increasing the complexity of the definition of features also increases the number of potential features: for example, there are $O(n^3)$ triple-points for a surface with n vertices. In our approach, we choose to use the feature-pairs ($\mathcal{F}(A)$ and $\mathcal{F}(B)$) computed from the elevation function as our basic features. Note that although the number of pairs of points is $\Theta(n^2)$, there are only linear ($O(n)$) number of feature pairs.

More specifically, in STEP 2, we first compute a set Ω of *reliable* PFPs, by which we mean that the two feature-pairs involved in each PFP potentially correspond to each other. We then select from Ω a small number of *consistent* ones to construct the final sets of matching landmarks (STEP 3). The details are described below.

Voting for reliable PFPs. We now describe how to compute Ω. Recall that we have sorted the set of registrations between S_A and S_B, Π, by their scores. Consider the top N transformations from Π, $\Pi(N)$. They roughly provide a set of "good" registrations between S_A and S_B. Intuitively, if a feature-pair $s_1 \in \mathcal{F}(A)$ corresponds to $s_2 \in \mathcal{F}(B)$, then s_1 should be close to s_2 w.r.t many *good* registrations from $\Pi(N)$. Here, we measure the *closeness* using the landmark-based distance using the following *temporary* landmarks: For $\mu \in \Pi(N)$, recall that it is associated with a pairs of feature-pairs, say $(a_0, a_1) \in \mathcal{F}(A)$ and $(b_0, b_1) \in \mathcal{F}(B)$. We use a_i's and b_i's, $1 \leq i \leq 2$, as landmarks for S_A and S_B respectively, under current registration μ. So each point in

$p \in S_A$ is represented by a 5-tuple $\mathbf{p} = \langle p_x, p_y, p_z, \mathrm{gd}(p, a_0), \mathrm{gd}(p, a_1)\rangle$, and similarly for a point $q \in S_B$. Next, for any PFP (s_1, s_2), we compute the distance between the two feature-pairs s_1 and $\mu(s_2)$using this landmark coordinates, and increase a *vote count* for (s_1, s_2) by one if the distance is smaller than some threshold λ_1. More precisely, let $s_1 = (p_1, p_2)$ and $s_2 = (q_1, q_2)$. We say that s_1 and s_2 are compatible if (1) difference between the length of p_1p_2 and q_1q_2 is small and (2) difference between $\mathrm{Elev}(p_1)$ and $\mathrm{Elev}(q_1)$ is also small. For a compatible PFP (s_1, s_2), the distance between s_1 and s_2 is defined as $\delta(s_1, s_2) = \delta(p_1, \mu(q_1)) + \delta(p_2, \mu(q_2))$, where $\delta()$ is the landmark-based distance as introduced earlier in Eqn (1). For every $\mu \in \mathit{\Pi}(N)$, we scan through all PFPs between $\mathcal{F}(A)$ and $\mathcal{F}(B)$. After checking all registrations from $\mathit{\Pi}(N)$, we rank the set of PFPs by their vote counts. Obviously, higher votes indicate more reliable PFPs. In our implementation, we compute the votes for PFPs by hashing idea to further improve its efficiency (details omitted), where different from the traditional geometric hashing algorithms which vote for transformations, our approach is a somewhat dual version where we use a set of transformations to vote for feature pairs.

We take Ω as the set of PFPs with a vote greater than some threshold λ_2. We next aim at selecting from Ω one reliable PFP from each semi-rigid component as landmarks. So if there are k semi-rigid components, ideally, we choose k feature pairs; denote by LP the resulting set of k PFPs.

Selecting landmarks from Ω. To find LP[0], we scan through Ω in order of decreasing votes, and return the first PFP whose corresponding transformation produces a registration with score greater than some threshold λ_3. In our experiments, the first pair from Ω is usually returned as LP[0].

Suppose we have one reliable PFP, say (s_1, s_2), that does not lie on the component identified by LP[0]. Intuitively, it should be *consistent* w.r.t. LP[0], namely, the landmark-based distance between s_1 and $\mu(s_2)$ is small (i.e, smaller than some threshold λ_4), where $\mu = \mu[0]$; while at the same time, the Euclidean distances between corresponding points from s_1 and $\mu(s_2)$ are large (i.e, greater than some threshold λ_5). Hence our algorithm first eliminates from Ω those PFPs that are *not* consistent with LP[0]. It then choose from the remaining PFPs the first one (thus with highest vote) that has a large euclidean distance between corresponding points w.r.t. $\mu[0]$, and set it as LP[1].

One can then repeat this procedure to identify more components till no PFPs from Ω are left. Once LP is computed, we collect the set of endpoints from s_1's (resp. s_2's) for all $(s_1, s_2) \in$ LP as the landmark set L for S_A (resp. M for S_B).

3.4 Segmentation

Given LP, L and M as constructed above, let $k = |\mathrm{LP}|$; $|\mathrm{L}| = |\mathrm{M}| = 2k$. We compute the landmark coordinates (a $(2k + 3)$-tuple) for every point on S_A and S_B. To segment the input surface, say S_A, into the k corresponding components, we construct a function $f_i : S_A \to \mathbb{R}$, for each $1 \le i \le k$, as follows. Align S_A and S_B based on $\mu[i]$, the transformation corresponding to LP$[i]$. For every point $p \in S_A$, find its nearest neighbor $\mathrm{NN}(p)$ from S_B based on the $(2k+3)$-tuple landmark coordinates. Set $f_i(p) = \delta(p, \mathrm{NN}(p))$. We say that a point p belongs to *c*omponent-i if $f_i(p) = \min_{j=1}^{k} f_j(p)$. To obtain the i'th segment, we simply collect all points from *c*omponent-i.

Remarks. We now come back to the four problems raised at the beginning of this section. Our algorithm uses the top N transformations from Π to vote for reliable PFPs, thus reduces the probability of having false positives (P1). We use landmark-based distance instead of euclidean distance, which is more stable under semi-rigid deformations (P2). In particular, the set of feature-pairs obtained from the elevation function produces a meaningful set of point-pairs, and they provide landmarks both in the voting process and in the final segmentation. Finally, landmark-based distance helps us to distinguish PFPs from different components (P3) and induce the final segmentation (P4).

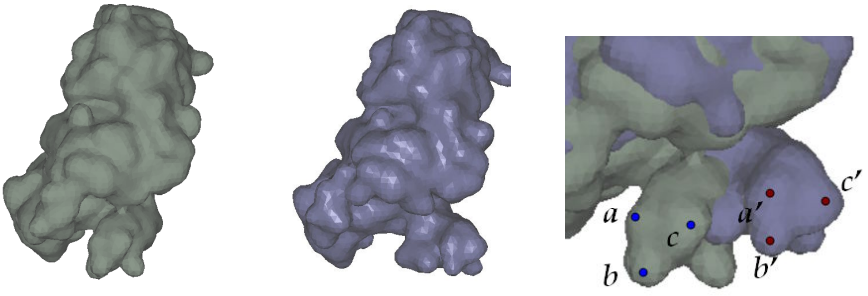
4 Experimental Results

Time complexity. Given two input surfaces S_A and S_B, of n and m vertices respectively, the most time consuming step is the PREPROCESSING step: compute the set of feature pairs of S_A and S_B using the elevation function [37]. The worst case complexity for computing elevation maxima is $O(n^4)$, although in practice, it is much faster. We consider this step as the preprocessing step, as once the set of feature pairs of a surface is computed, one can use it for multiple registration tasks.

It is hard to give an exact time complexity for the remaining algorithm, as geometric hashing technique is involved, and it also depends on choices of different parameters. More specifically, STEP 1 computes the set of coarse rigid transformations using geometric hashing technique with running time $O(t_1(|\mathcal{F}(A)|^2 + r_1|\mathcal{F}(B)|^2))$, where t_1 is the time to access a particular entry in the hash table given an index, and r_1 is the maximum size of a bin associated with any index in the hash table; both t_1 and r_1 are usually considered to be constants. In STEP 2, we approximate geodesic distance by the graph distance (i.e., using only edges from the input mesh). Thus it takes $O(N(n+m)\log nm)$ to approximate geodesic distance from all vertices to all points contributing to the top N transformations of Π using Dijkstra's algorithm. The voting algorithm runs in roughly $O(N(|\mathcal{F}(A)|+|\mathcal{F}(B)|))$ time. A straightforward implementation of STEP 3 takes $O(|\Omega|^2 + |\mathcal{F}(A)||\mathcal{F}(B)|)$ time, and STEP 4 runs in $O(nm)$ time (for every $p \in S_A$, we find its nearest neighbor under Landmark-based distance in $O(m)$ time). In practice $N, |\mathcal{F}(A)|, |\mathcal{F}(B)|$, and $|\Omega|$ are usually around 100.

Inputs setup. Our targeted application is to segment low resolution data, such as cryo-EM data, in order to facilitate the analysis of their deformation when high resolution atomic structure of an input molecule is not available. In this section, we use the so-called "pseudo-maps" as in [38] to test our algorithm so that the correct answer is known. In particular, we take a pair of different conformations of the same molecule that undergoes some large hinge-type deformations. For each conformation, to obtain its pseudo-EM-map, we take its high-resolution structure and use the EMAN software package [39] to introduce certain amount of Gaussian noise to its original density map. We then compute the iso-surface w.r.t some prefixed value in the coarsened density map (pseudo-EM-map) to produce input surface for our algorithm.

We applied our algorithm to two sets of test data. The first set includes the prehydrolyzing state (PDB code 1FMW, [40]) and the ATP hydrolyzing state (PDB code 1VOM [41]) of myosin. The data we use are after pre-procession at Pande's group at Stanford University for normal modes analysis applications, where some extra residues



(a) (b) (c)

Fig. 4. (a) and (b) show the two input surfaces (isosurfaces from the pseudo-EM-map computed for molecules with PDB codes 1FMW and 1VOW respectively. In (c), we see that the points from the small leg are locally similar, thus hard to decide how to align abc with $a'b'c'$ just by local information.

are removed so that the two conformations have the same length. The second set is obtained from the Database of Macromolecular Movements [29]. They are two conformations of DNA Polymerase beta (PDB codes 1BPD and 2BPG respectively). As we will see later, these two sets present different properties in their motion. For both sets of data, we read in their PDB file, generated their pseudo-EM-map using EMAN software [39], and extract two sets of surfaces. The input surfaces are in random relative orientations. We then compute the set of feature pairs for all input surfaces using algorithm from [27].

Segmentation results. The Myosin data set has a relatively small deformation among the two input sets. Roughly speaking, this molecular motor has a main body and two small legs, one of them moves outwards during the deformation (Figure 4 (a) and (b)). Although the moving component and the deformation is relative small, we note that the small leg is rather homogeneous. It does not have any locally very distinguishable features to help to establish correspondences for this small leg. Thus locally there are ambiguity how to align triangle abc from S_A to $a'b'c'$ in S_B. This problem is alleviated in our approach by also considering their geodesics to landmarks on the main components (thus incorporating more global information). The two input surfaces, S_A and S_B, have 8314 and 8226 vertices respectively. After preprocessing, we have $|\mathcal{F}(A)| = 132$ and $|\mathcal{F}(B)| = 72$, thus representing S_A and S_B in a much more concise way. Our algorithm identifies two components for this data sets, and the corresponding landmarks are shown in Figure 5 (a) and (b). The resulting two components produced are shown in 5 (c). The entire algorithm finishes within seconds, much more efficient than the optimization approach in [30] that takes at least several hours. Its fast speed enables users to play with different parameters and compute more than one possible segmentations as seeds for later more refined registration or other types of processes.

The DNA Polymerase beta data presents large scaled deformation in its two conformations. The motion includes hinge-type bending as well as some twisting. Furthermore, the topology of the two input surfaces are also different: the genus of the ending surface is one (see Figure 6 (a) and (b)). So there can be reasonable amount of distortion

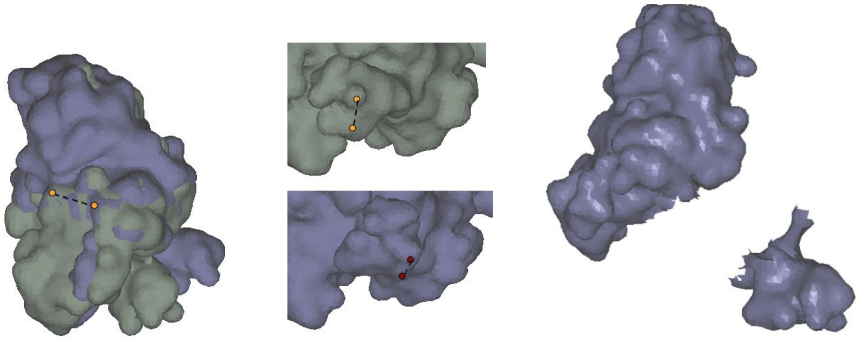


Fig. 5. In (a), we align the two surfaces by their first pair of feature pairs computed by our algorithm (identifying the main component); one of them is roughly marked. In (b), we mark the pair of corresponding feature pairs identifying the smaller components from the two surfaces respectively. The resulting segmentation is shown in (c).

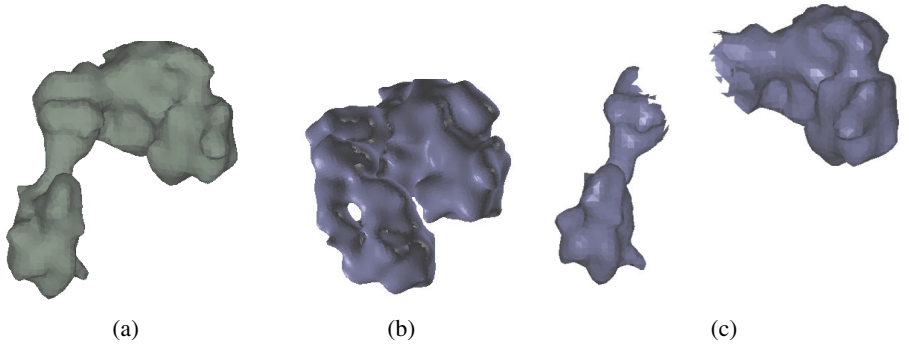


Fig. 6. (a) and (b) show the input surfaces extracted from the pseudo-EM-map for molecules with PDB codes 1BPD and 2BPG respectively. The resulting segmentation is shown in (c).

in geodesics as well. Nevertheless, our algorithm is able to identify two components as shown in Figure 6 (b), demonstrating its robustness. The running time is of the same order as the previous one.

Parameters. There are several parameters involved in our algorithm. Ideally, we hope that the values of these parameters can be decided automatically. Currently, however, the user need to input these thresholds. Part of the reason is because that these parameters are case-dependent. For example, geodesic distances are much less preserved in the second test data compared to the first set. To increase the tolerance in the geodesic distance, it is necessary to increase the threshold related to geodesic distance. Since the deformation is also large in this case, we are still able to identify semi-rigid subcomponents reliably.

5 Discussion

We have proposed in this paper a new method to extract a few reliable landmarks for surfaces undergoing hinge-type deformation, which further help define a segmentation of the input surface into semi-rigid subunits. Our preliminary experimental results show that the algorithm is efficient and effective, identifying semi-rigid subunits automatically, once input parameters are given.

Although our targeted application is to segment low-resolution molecular data to facilitate analysis of their deformation when no atomic structure is available, the proposed method can be applied to other fields where semi-rigid deformations are involved. The landmarks computed by our algorithm are of independent interest, and can be inputs for other segmentation or tracking algorithms. For example, these landmarks can help to produce a deformation between two conformations using approaches such as 'as-rigid-as possible' shape interpolation [42].

Our current experiments with the pseudo-EM data serves as a proof-of-principle test for our new algorithm. In the next step, we plan to apply the algorithm to real EM data to provide an efficient tool to, for example, facilitate biologists detecting the presence of a certain structure (e.g, a particular protein) in the low resolution structure of a complex system (e.g, in the ribosome), as well as the deformation involved. One main challenge involved is the reliability of geodesic distances for isosurfaces extracted from real EM data, which can be very noisy: for example, the surfaces may connect to each other by small bridges, creating short cuts and changing geodesics greatly. It is an interesting and important problem to characterize major types of topological features created this way, and to develop methods to remove them.

Finally, currently, our algorithm focuses on a very specific type of deformation where geodesic distances are relatively well-preserved. We will investigate other types of invariants (other than geodesics) which can embrace more general types of non-rigid deformations. We leave this as one important future direction. Note that both the landmark based distance and the voting scheme in our framework are general, and can be modified to accommodate other types of invariants.

Acknowledgment. The work was supported by NSF grants FRG 0354543 and ITR 0205671, and NIH grant GM072970. The authors would like to thank Vijay Pande for motivating the study of the semi-rigid segmentation problem, Natasha Gelfand and Niloy Mitra for helpful discussions, and the anonymous reviewers for useful comments.

References

1. Girod, B., Greiner, G., Niemann, H., eds.: Principles of 3D image analysis and synthesis. Kluwer Academic Publishers (2000)
2. Yoo, T., ed.: Insight into images: Principles and practices for segmentation, registration, and image analysis. A. K. Peters (2004)
3. Belongie, S., Malik, J., Puzicha, J.: Shape matching and object recognition using shape contexts. IEEE Transactions on Pattern Analysis an Machine Intelligence **24**(4) (2002) 509–522

4. Barequet, G., Sharir, M.: Partial surface and volume matching in three dimensions. IEEE Trans. Pattern Anal. Mach. Intell. **19**(9) (1997) 929–948
5. Gelfand, N., Mitra, N.J., Guibas, L.J., Pottmann, H.: Robust global registration. In: Proc. Symp. Geom. Processing. (2005) 197–206
6. Johnson, A.E., Hebert, M.: Using spin images for efficient object recognition in cluttered 3D scences. IEEE Transactions on Pattern Analysis and Machine Intelligence **21**(5) (1999) 433–449
7. Koenderink, J.J.: Solid shape. MIT Press, Cambridge, MA, USA (1990)
8. Manay, S., Hong, B., Yezzi, A.J., Soatto, S.: Integral invariant signatures. In: European Conference on Computer Vision. (2004) 87–99
9. Besl, P.J., McKay, N.D.: A method for registration of 3-D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence **14**(2) (1992) 239–256
10. Wolfson, H.J., Rigoutsos, I.: Geometric hashing: An overview. IEEE Computational Science and Engineering **4**(4) (1997) 10 – 21
11. Metaxas, D.N.: Physics-Based Deformable Models. Kluwer Academic (1997)
12. Ruechert, D., Hawkes, D.J.: Registration of biomedical images. In Baldock, R., Graham, J., eds.: Image Processing and Analysis - A Practical Approach. Oxford University Press (1999)
13. Rueckert, D.: Non-rigid registration: Techniques and applications. In Hajnal, J.V., Hill, D.L.G., Hawkes, D.J., eds.: Medical Image Registration. CRC Press (2001)
14. Sclaroff, S., Pentland, A.P.: Modal matching for correspondence and recognition. IEEE Trans. Pattern Anal. Mach. Intell. **17**(6) (1995) 545–561
15. Noh, J.Y., Neumann, U.: Expression cloning. In: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. (2001) 277–288
16. Pauly, M., Mitra, N.J., Giesen, J., Gross, M., Guibas, L.: Example-based 3d scan completion. In: Symposium on Geometry Processing. (2005) 23–32
17. Allen, B., Curless, B., Popović, Z.: The space of human body shapes. ACM Transactions on Graphics **22**(3) (2003) 587–594
18. Guenter, B., Grimm, C., Wood, D., Wmlvar, H., Pighin, F.: Making faces. In: SIGGRAPH '98: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. (1998) 55–66
19. Kalberer, G.A., Gool, L.V.: Face animation based on observed 3d speech dynamics. In: IEEE Conference on Computer Animation. (2001) 20–27
20. Kakadiaris, I.A., Metaxas, D., Bajcsy, R.: Active part-decomposition, shape and motion estimation of articulated objects: a physics-based approach. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (1994) 980–984
21. Zelnik-Manor, L., Machline, M., Irani, M.: Multi-body factorization with uncertainty: Revisiting motion consistency. To appear in IJCV special issue on Vision and Modeling of Dynamic Scenes (2006)
22. Holm, L., Sander, C.: Protein structure comparison by alignment of distance matrices. J. Mol. Biol. **233** (1993) 123–138
23. Shindyalov, I.N., Bourne, P.E.: Protein structure alignment by incremental combinatorial extension (CE) of optimal path. Protein Engineering **11**(9) (1998) 739–747
24. Pearl, F.M.G., Lee, D., Bray, J.E., Sillitoe, I., Todd, A.E., Harrison, A.P., Thornton, J.M., Orengo, C.A.: Assigning genomic sequences to CATH. Nucleic Acids Research **28**(1) (2000) 277 – 282
25. Chen, R., Li, L., Weng, Z.: ZDOCK: An initial-stage protein docking algorithm. Proteins **52**(1) (2003) 80–87
26. Smith, G.R., Sternberg, M.J.E.: Prediction of protein-protein interactions by docking methods. Current Opinion in Structural Biology **12** (2002) 29–35

27. Wang, Y., Agarwal, P.K., Brown, P., Edelsbrunner, H., Rudolph, J.: Coarse and reliable geometric alignment for protein docking. In: Pac Symp Biocomput. (2005) 66–77
28. A, H.K.T., Field, M.J., Perahia, D.: Tertiary and quaternary conformational changes in aspartate transcarbamylase: a normal mode study. Proteins **34** (1999) 96–112
29. Alexandrov, V., Lehnert, U., Echols, N., Milburn, D., Engelman, D., Gerstein, M.: Normal modes for predicting protein motions: a comprehensive database assesement and associated web tool. Protein Science **14**(3) (2005) 633–643
30. Tama, F., Miyashita, O., III, C.L.B.: Normal mode based flexible fitting of high-resolution structure into low-resolution experimental data from cryo-em. Journal of Structural Biology **147** (2004) 315–326
31. Tama, F., Sanejouand, Y.H.: Conformational change of proteins arising from normal mode calculations. Protein Engineering **14** (2001) 1–6
32. Delarue, M., Dumas, P.: On the use of low-frequency normal modes to enforce collective movements in refining macromolecular structural models. Proceedings of National Academy of Science **101**(18) (2004) 6957–6962
33. Ming, D., Kong, Y., Wakil, S.J., Brink, J., Ma, J.: Domain movements in human fatty acid synthase by quantized elastic deformational model. Proceedins of National Academy of Science **99**(12) (2002) 7895–7899
34. Tama, F., Wriggers, W., III, C.L.B.: Exploring global distortions of biological macromolecules and assemblies from low-resolution structural information and elastic network theory. Journal of Molecular Biology **321** (2002) 297–305
35. Wriggers, W., Schulten, K.: Protein domain movements: detetion of rigid domains and visualization of hinges in comparisons of atomic coordinates. Proteins **29** (1997) 1–14
36. Krebs, W.G., Gerstein, M.: The morph server: a standardized system for analyzing and visualizing macromolecular motions in a database framework. Nucleic Acids Research **28**(8) (2000) 1665–1675
37. Agarwal, P.K., Edelsbrunner, H., Harer, J., Wang, Y.: Extreme elevation on a 2-manifold. In: Proc. 20th Sympos. Comput. Geom. (2004) 357–365
38. Ceulemans, H., Russell, R.B.: Fast fitting of atomic structures to low-resolution electron density maps by surface overlap maximization. Journal of Molecular Biology **338** (2004) 783–793
39. Ludtke, S., Jiang, W., Peng, L., Tang, G., Baldwin, P., Fang, S., Khant, H., Nason, L.: EMAN software package. http://ncmi.bcm.tmc.edu/homes/stevel/EMAN/ (2006)
40. Bauer, C.B., Holden, H.W., Thoden, J.B., Smith, R., Rayment, I.: X-ray structures of the apo and MgATP-bound states of *dictyostelium discoideum* myosin motor domain. J. Biol. Chem. **275** (2000) 38494–38499
41. Smith, C.A., Rayment, I.: X-ray structure of the magnesium(II).ADP.vanadate complex of the *dictyostelium discoideum* myosin motor domain to 1.9Å resolution. Biochemistry **35** (1996) 5405–5417
42. Alexa, M., Cohen-Or, D., Levin, D.: As-rigid-as-possible shape interpolation. In: SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press/Addison-Wesley Publishing Co. (2000) 157–164

Piecewise Developable Surface Approximation of General NURBS Surfaces, with Global Error Bounds

Jacob Subag and Gershon Elber

Technion - Israel Institute of Technology, Haifa 32000, Israel
jsubag@cs.technion.ac.il

Abstract. Developable surfaces possess qualities that are desirable in the manufacturing processes of CAD/CAM models. Specifically, models formed out of developable surfaces can be manufactured from planar sheets of material without distortion. This quality proves most useful when dealing with materials such as paper, leather or sheet metal, which cannot be easily stretched or deformed during production.

In this work, we present a semi-automatic algorithm to form a piecewise developable surface approximation of a general NURBS surface. These developable surfaces are constructed as envelopes of the tangent planes along a set of curves on the input surface. Furthermore, the Hausdorff distance between the given surface and the approximating set of developables is globally bounded by a user-provided threshold.

1 Introduction and Related Work

Developable surfaces are surfaces that can be unfolded to the plane (flattened) with no distortion and are divided into three families of surfaces: cylinders, cones[1] and envelopes of the tangent planes along curves on surfaces [3]. In many manufacturing processes, specifically when dealing with sheet materials such as paper, leather or sheet metal, developable surfaces are used to determine the actual regions to be cut from the material in order to construct the final product. However, most freeform surfaces created by CAD/CAM applications are not developable by design nor can they be divided in such a way to produce parts that are all developable.

This modeling/manufacturing problem has been addressed by several approaches in the past. One approach has been to model with developable surfaces to begin with. Aumann [1] provided necessary and sufficient conditions for Bézier surfaces, interpolating two curves, to be developable and free of singular points. Pottmann and Farin [19] presented ways to represent and model developable Bézier and B-spline surfaces and Park et al. [16] described an interpolative optimal control problem, generating a developable surface from two points and a curve of tangent directions connecting them.

Another approach was to assume the existence of a developable surface and reconstruct or approximate it. Peternell [17] [18] presented algorithms for reconstructing

[1] By cylinders we refer to surfaces which can be represented as a one parameter family of parallel lines and by cones we refer to surfaces which can be represented as a one parameter family of lines that share an intersection point.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 143–156, 2006.
© Springer-Verlag Berlin Heidelberg 2006

developable surfaces from point clouds. Hoschek and Pottmann [13] used samples of tangent planes, on a developable surface, to find a developable B-spline surface which interpolates/approximates them. Leopoldseder and Pottmann [14] approximated pre-existing developable surfaces with (parts of) cones. Lastly, Pottmann and Wallner [20] approximated a set of tangent planes with a developable surface, measuring the quality of the approximation with a distance metric for tangent planes in a small region of interest. While these two approaches circumvent some design problems, there are objects, in everyday products, which cannot be modeled as a union of developable surfaces, as they are inherently non-developable, i.e. they are doubly curved.

The third approach was the approximation of a freeform surface, or samples taken from one, by a set of developable surfaces. Hoschek [12] presented algorithms to approximate surfaces of revolution with developable polygonal strips or cones while bounding the Hausdorff distance with a predetermined threshold, by reducing the problem to bounding the distance between a curve and a polyline, both in the plane. However, surfaces of revolution are not expressive enough to be used exclusively when modeling, in real life applications. Elber [8] approximated a general freeform surface by generating developable surfaces that connect isolines on the input surface and bound the Hausdorff distance with a predetermined threshold, by bounding the magnitude of the surface-developables difference. However, this restriction of the developable surfaces to be between isolines of the input surface, results in a limited set of possible approximating developables for each approximated surface. Over-segmentation of the original surface with the approximating set of developables may result, in many cases. Finally, Chen et al. [4] approximated a set of sampled points and accompanying normals with a set of smoothly joined cones and cylinders. This algorithm performs well for samples on a developable (or nearly developable) surface, but would require heavy segmentation of the samples to handle complex non-developable surfaces. Furthermore, Chen et al. bound the approximation error only for the sampled points.

Our proposed method subscribes to the latter approach. We approximate general freeform surfaces unlike [12], which approximates only surfaces of revolution, and use envelopes of the tangent planes along a set of curves on the input surface, which generalize cones and cylinders, rather than limit the output surfaces to cones and cylinders such as in [14,4] or ruled surfaces between two isocurves on the input surface such as in[8]. Furthermore, the approximation error is globally bounded with a user supplied threshold, unlike in [17,18,13,20].

The approximating developable surfaces are constructed by sampling specific lines, on the envelopes, which are restricted to the minimal length needed for approximating a calculated region on the input surface. By interpolating in-between these sampled lines, a finite developable surface is formed, which is contained in the infinite envelope of tangents. The computed approximation error is used to refine our developable surfaces until the user threshold is achieved.

The rest of this article is divided as follows: Section 2 details our main algorithm. Section 3 presents experimental results and in Section 4, we summarize and discuss future work.

2 Approximation Technique

In this section, the piecewise developable approximation algorithm is presented. Our approximation errors are measured using the Hausdorff distance metric, which is defined for two objects, O_1 and O_2, as:

$$D_H(O_1, O_2) = \max\{\max_{p_1 \in O_1}\{\min_{p_2 \in O_2}\{\|p_1 - p_2\|\}\}, \max_{p_2 \in O_2}\{\min_{p_1 \in O_1}\{\|p_1 - p_2\|\}\}\},$$

where p_i is a point on O_i.

The piecewise developable approximation algorithm comprises of several stages. The specifics of each stage are given below. We now present a short overview of the algorithm. Given a parametric curve, $c = (u(t), v(t))$, supplied by the user, generate an infinite developable surface, E, as the envelope of tangent planes along $S(c)$. Then, calculate the region of S, which is approximated up to an ε by an individual line along E. Using a sampled set of such lines, a new finite developable surface, $\widetilde{E} \subset E$, is interpolated and the Hausdorff distance between $\widetilde{E}$ and a relevant region in S, denoted $\widetilde{S}$, is bounded. Then, the developable surface is subsequently refined, until the user supplied threshold of approximation is met and its boundaries are smoothed. The user is interactively prompted to continue and add curves over S, for which the above process is repeated, until the union of the resulting developable surfaces forms a complete approximation of S. Possible ways of automating the process of adding the curves are discussed in Section 4.

2.1 Envelope Surfaces

Given a C^2 continuous surface, $S(u, v)$, and a C^2 continuous curve, $c(t) = (u(t), v(t))$ in the parametric domain of S. An envelope surface along $C(t) = S(c(t))$ is defined as:

$$E(t, r) = C(t) + r\hat{W}(t), \quad r \in \mathbb{R}, \tag{1}$$

where $W(t) = N(t) \wedge N'(t)$, $N(t)$ is defined as the normal of S at point $S(c(t))$, $\wedge$ denotes the cross product, and $\hat{A} = \frac{A}{\|A\|}$.

$E(t, r)$ is developable [3]. The definition presented by Equation (1) necessitates the normalization factor of W, which is non-rational. We use normalized vectors for some of the pointwise evaluations (denoted by $\hat{W}$) and employ the unnormalized W, whenever possible. When $C(t)$ passes through planar regions of S, $N'(t)$ vanishes and $\hat{W}$ is undefined. This problem can be solved by trimming planar regions from S, being developable, before proceeding with our algorithm[2]. In the ensuing discussion, we assume W is always defined.

Another problem that stems from Equation (1) is the order of curve $W(t)$. Recalling that $N = \frac{\partial S}{\partial u} \wedge \frac{\partial S}{\partial v}$, composed over $c(t)$, we know that $W(t)$ can frequently attain extremely high orders. High order curves may produce numerical errors when evaluated. In our implementation, we fit low order spline approximation [6] to $W(t)$, while bounding the error, a lower order representation used hereafter. See [5] for more details on reducing degrees of symbolically computed curves.

[2] This segmentation could be performed by trimming away from S all regions for which $|K| < \varepsilon$, K being the Gaussian curvature, is computed symbolically [9].

2.2 Approximation by Ruling

A *single ruling* at parameter value t_0 is an infinite line, $L_{t_0}(r) = E(t_0, r)$, parallel to $W(t_0)$ and incident on $C(t_0)$ (see Figure 1 (b)). The region of S that is approximated by L_{t_0} is, therefore, all the points on S that are within some prescribed distance, ε, (in the Euclidean sense) from L_{t_0}. In other words, all the points on S for which the following equation holds:

$$\|S(u,v) - C(t_0) - \langle S(u,v) - C(t_0), \hat{W}(t_0)\rangle \hat{W}(t_0)\| \leq \varepsilon. \tag{2}$$

Equation (2) can be reformulated as:

$$F_{t_0}(u,v) = \|S(u,v) - C(t_0) - \langle S(u,v) - C(t_0), \hat{W}(t_0)\rangle \hat{W}(t_0)\|^2 - \varepsilon^2 \leq 0, \tag{3}$$

in order to deal with a rational function. Furthermore, $F_{t_0}(u,v) = 0$ defines an implicit curve in the parametric domain of S, which bounds the region(s) of S that are within ε from L_{t_0}. Geometrically speaking, $F_{t_0} = 0$ defines the intersection curve of an ε-radius cylinder centered around L_{t_0} and the input surface S.

The topology of implicit Equation (3) is analyzed by employing a modified version of the algorithm presented by Hass et al. in [11], which provides the accurate topology of an implicit curve. The result of Hass' algorithm is a set of polygons, $\{P^i_{t_0}\}$, whose vertices lie on $F_{t_0} = 0$ and are topologically equivalent to the implicit curve defined by $F_{t_0} = 0$ (see Figure1 (c)). This result is insufficient for our purpose, as the edges of $\{P^i_{t_0}\}$ can be arbitrarily far from $F_{t_0} = 0$. More importantly, $\{P^i_{t_0}\}$ may contain points not approximated by the ruling to within ε, i.e., points for which $F_{t_0} > 0$. In order to guarantee that $\forall i, F_{t_0}(P^i_{t_0}) \leq 0$, we enhanced Hass' algorithm by sampling $F_{t_0} = 0$ up to a threshold, adding the inflection points of $F_{t_0} = 0$ (see [2] for an analysis of inflection points of implicit curves) as vertices of the resulting polygons, and adding a post processing refinement step, ensuring the result, $\{P^i_{t_0}\} \subset (F_{t_0} \leq 0)$. Due to space limitations, the complete account of these modifications is omitted.

Only one polygon in $\{P^i_{t_0}\}$ contains $c(t_0)$. Denote it as $P^0_{t_0}$ (see Figure 1 (a)) and denote its individual edges as $\{e_j\}$. Since $F_{t_0}(c(t_0)) = -\varepsilon^2$ and S (and therefore, F) is continuous, there is an environment surrounding $c(t_0)$, for which Equation (3) holds and which will be contained in $\{P^i_{t_0}\}$. This last step eliminates disjoint approximated regions as they may result in undesirable disjoint developable surfaces.

Only $P^0_{t_0}$ will be employed in the coming steps. The region $S(P^0_{t_0})$ is approximated to within ε by the infinite ruling L_{t_0}. However, there is a unique and minimal interval along L_{t_0} that approximates $P^0_{t_0}$ to within ε. We seek to derive this minimal interval by calculating the minimal and maximal coordinate values of this interval, with respect to Equation (1), parameterized by r.

Consider point $(u_0, v_0) \in P^0_{t_0}$. The r-coordinate value of (u,v) prescribes the point on $L_{t_0}(r)$ closest to $S(u,v)$ and equals,

$$r_{t_0}(u,v) = \frac{\langle S(u,v) - C(t_0), W(t_0)\rangle}{W(t_0)^2}, \tag{4}$$

where the unnormalized $W(t_0)$ is used, resulting in r-coordinate values that counter its magnitude's effect in the upcoming construction of the developable surface.

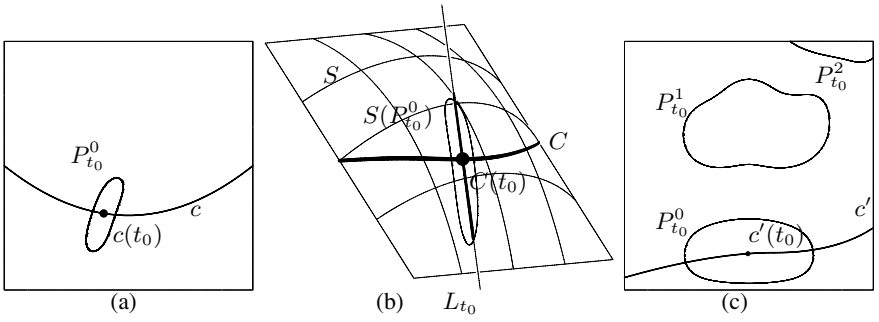


Fig. 1. (a) The parametric domain of S with the parametric curve $c(t)$ and the parametric region approximated by the ruling L_{t_0}, i.e. $P^0_{t_0}$. (b) A single ruling along $C(t) = S(c(t))$. The thick section of the ruling is the minimal interval needed to approximate $P^0_{t_0}$ to within ε, i.e. $L_{t_0}(r)$. Also shown, in (b), is the composition of S over $P^0_{t_0}$. (c) An example of several approximated regions, $\{P^i_{t_0}\}$, for different surface and parametric curve.

Equation (4) can be evaluated for all $(u, v) \in P^0_{t_0}$. Since r_{t_0} is continuous and $P^0_{t_0}$ is a one connected region, in the parametric domain of S, the locus of r-coordinate values over $P^0_{t_0}$ and its interior, is a continuous interval. As such, we need only find the minimal and maximal values of r. We do so by first solving for:

$$\nabla r_{t_0}(u, v) = 0, \quad (u, v) \in P^0_{t_0}, \tag{5}$$

finding interior local extrema. Then, the solutions of

$$\frac{\partial r_{t_0}(e_j(s))}{\partial s} = 0, \quad \forall e_j \in P^0_{t_0}, \tag{6}$$

detect extrema along edges of $\in P^0_{t_0}$. Finally, endpoints of every edge e_j (the vertices of $P^0_{t_0}$), are examined. This analysis is possible when S is piecewise C^3 which results in $r_{t_0}(u, v)$ being differentiable. When this is not the case, we need to also examine Equation (4) along the isolines corresponding to each knot. Denote these computed minimal and maximal values of r by $r^{t_0}_{min}$ and $r^{t_0}_{max}$. Since the r-coordinate value for $c(t_0)$ is zero and $P^0_{t_0}$ was chosen to include $c(t_0)$, $r^{t_0}_{min}$ is negative and $r^{t_0}_{max}$ is positive.

We now consider the line segment:

$$L_{t_0}(r) = C(t_0) + rW(t_0), \quad r \in [r^{t_0}_{min}, r^{t_0}_{max}], \tag{7}$$

as the ruling approximating $P^0_{t_0}$, see the thick segment along L_{t_0} in Figure 1 (b).

2.3 Inter-ruling Interpolation

Consider a set of rulings $\{L_{t_i}\}$, sampled along $C(t)$ at $\{t_i\}_{i=0}^n, n \geq 1$. Without loss of generality, assume $t_i < t_{i+1}$. Analyzing each ruling's approximated region, $P^0_{t_i}$ (see Figure 2 (a)), we seek to interpolate a complete developable surface along $C(t)$. As

a first step, interpolate two scalar curves, $r_{min}(t)$ and $r_{max}(t)$, such that: $r_{min}(t_i) = r^{t_i}_{min}$ and $r_{max}(t_i) = r^{t_i}_{max}, \forall i$. Then, we proceed by creating two new curves:

$$C_1(t) = C(t) + r_{min}(t)W(t), \quad C_2(t) = C(t) + r_{max}(t)W(t). \tag{8}$$

Finally, construct the local developable surface, as a ruled surface between $C_1(t)$ and $C_2(t)$:

$$\widetilde{E}(t, r) = (1 - r)C_1(t) + rC_2(t), \quad r \in [0, 1], t \in [t_0, t_n], \tag{9}$$

(see Figure 2 (b)). $\widetilde{E} \subset E$ (recall Equation (1)) and is hence developable.

So far, this construction guarantees nothing with regard to the approximating qualities and quantities of $\widetilde{E}$ at $t \notin \{t_i\}_{i=0}^n$. The next section provides such bounds.

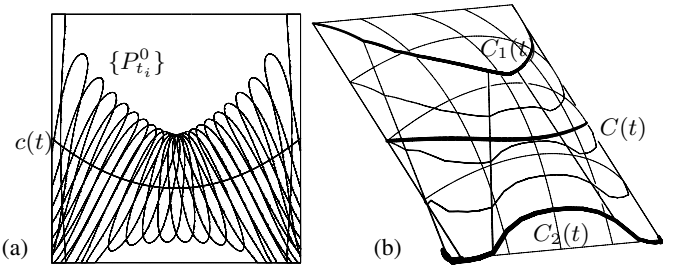


Fig. 2. (a) The individual regions approximated by a set of sampled rulings in the paramteric domain of S, $\{P^0_{t_i}\}_{i=0}^n$. (b) The developable ruled surface $\widetilde{E}$ and curves $C_1(t)$ and $C_2(t)$, which are used to generate it.

2.4 Bounding the Hausdorff Distance Between the Developable and Input Surface

In this section, we bound the Hausdorff distance between $\widetilde{E}$ (recall Equation (9)) and the relevant region of S, which is approximated by $\widetilde{E}$ and denoted as $\widetilde{S}$. Construct a "matching" function $T(t, r) \rightarrow (u, v)$ that assigns to each point in the parametric domain of $\widetilde{E}$, a point in the parametric domain of S. Then, globally bound $\|S(T(t, r)) - \widetilde{E}(t, r)\|$. If this bound is found to be smaller than or equal to ε, clearly $D_H(\widetilde{S}, \widetilde{E}) = D_H(S(T(t, r)), \widetilde{E}(t, r))$ will also be bounded by ε.

Thus, we seek a matching function $T(t, r) = (t_u(t, r), t_v(t, r))$, which reparameterizes S so as to diminish:

$$\max_{t,r} \|S(T(t, r)) - \widetilde{E}(t, r)\|. \tag{10}$$

Therefore, an ideal T would assign to each point on $\widetilde{E}$, the parametric coordinates of the closest point on S. We compromise by constructing T as a piecewise bilinear mapping, which interpolates solutions of Equation (10) for a set of (initially) three points for each ruling, corresponding to the ruling's start, incidence and end points. Specifically,

$$T(t,r) = \sum_{i=0}^{n} \sum_{j=0}^{2} Q_{i,j} B_{i,1}(t) B_{j,1}(r), \tag{11}$$

where $B_{i,1}$ is the linear i'th B-spline basis function and where

$$\{Q_{i,j}\}_{i=0,j=0}^{n,2} = \begin{cases} \underset{u,v}{argmin} \|S(u,v) - \widetilde{E}(t_i,0)\|, & j = 0, \\ c(t_i), & j = 1, \\ \underset{u,v}{argmin} \|S(u,v) - \widetilde{E}(t_i,1)\|, & j = 2. \end{cases} \tag{12}$$

Having constructed T, we are ready to bound $\|S(T(t,r)) - \widetilde{E}(t,r)\|$ with the user supplied ε. As before and in order to deal with rational functions, we bound the following against ε^2:

$$\|S(T(t,r)) - \widetilde{E}(t,r)\|^2 \triangleq \|\widetilde{S}(t,r) - \widetilde{E}(t,r)\|^2. \tag{13}$$

Consider the original surface $S(u,v) = \sum_{i=0}^{N} \sum_{j=0}^{M} P_{i,j} B_{i,n}(u) B_{j,m}(v)$, where $P_{i,j}$ are the control points of S and $B_{i,n}$ is the i'th B-spline basis function of degree n. In order to calculate $S(t_u(t,r), t_v(t,r))$, divide S at all its internal knots into Bézier patches, $\{S_k\}$, $S_k(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j}^k \theta_{i,n}(u) \theta_{j,m}(v)$, where $P_{i,j}^k$ are the control points of S_k and $\theta_{i,n}$ is the i'th Bézier basis function of degree n. Now composing S_k over T, one gets,

$$S_k(t_u^k(t,r), t_v^k(t,r)) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j}^k \theta_{i,n}(t_u^k(t,r)) \theta_{j,m}(t_v^k(t,r)),$$

where $t_u^k(t,r) = \frac{t_u(t,r) - u_{min}^k}{u_{max}^k - u_{min}^k}$ and $t_v^k(t,r) = \frac{t_v(t,r) - v_{min}^k}{v_{max}^k - v_{min}^k}$ and where u_{min}^k, u_{max}^k, v_{min}^k, v_{max}^k are the knot values that bound the region in the parametric domain of S, from which S_k was extracted.

Recalling that, $\theta_{i,n}(t) = \binom{n}{i}(1-t)^{n-i} t^i$, we end up with

$$S_k(t_u^k(t,r), t_v^k(t,r)) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j}^k \binom{n}{i} \binom{m}{j} (1-t_u^k)^{n-i} {t_u^k}^i (1-t_v^k)^{m-j} {t_v^k}^j. \tag{14}$$

In the last expression, t_u^k and t_v^k may exceed the domain of S_k. While this limitation prevents us from computing the complete composition of S over T, it is sufficient for the error bound analysis we require. Interested in the maximal value of Equation (13), instead of trimming t_u^k and t_v^k to fit each of the S_k's domains, we leave them "as is" and evaluate each S_k composition formula (14). The resulting set of surfaces $S_k \circ T$, denoted as $\widetilde{S}_k$, are each identical to $S \circ T$ for their shared domain. Therefore, only extremums inside the domain of $\{S_k\}$, are considered in:

$$(k_{max}, t_{max}, r_{max}) = \underset{k,t,r}{argmax} \|\widetilde{S}_k(t,r) - \widetilde{E}(t,r)\|^2.$$

Now, the evaluation of $\|\widetilde{S}_{k_{max}}(t_{max}, r_{max}) - \widetilde{E}(t_{max}, r_{max})\|^2$ provides a bound on Expression (13) and, hence, a bound for the squared Hausdorff distance between $\widetilde{E}$ and $\widetilde{S}$.

2.5 Refinement of the Approximated Region

If the bound calculated in the previous subsection, is smaller than or equal to ε^2, then our approximation is sufficiently accurate. Otherwise, we must refine T and $\widetilde{E}$ until the bound is smaller than or equal to ε^2. Refinement can be achieved by three complementary methods, along the t and r parametric directions of $\widetilde{E}$ (see Figure 3).

The first method of refinement adds more rulings. This method improves T and $\widetilde{E}$, when the extremum (t_{max}, r_{max}) lies in-between two consecutive rulings corresponding to t_i and t_{i+1}, which means that either T performs poorly when interpolating the bilinear region between said rulings or $\widetilde{E}$ is too far from S at that point. In such cases, we add a new ruling at $\frac{t_i+t_{i+1}}{2}$, update $\widetilde{E}$ and T and re-analyze expression (13), see Figure 3 (b) for an example of the first method of refinement.

The second method of refinement adds control points to T, for each ruling. This method is meant to handle cases when T interpolates poorly on or near a specific ruling. In such cases, $t_{max} \approx t_i$ and the former refinement method would require many iterations to reduce the bound or even fail. Thus, when $t_{max} \approx t_i$, we apply both the first and the second refinement methods. Recall Equation (12), in which we constructed T with three control per ruling. In this second refinement method, we generalize Equation (12) with:

$$\{Q_{i,j}\}_{i=0,j=0}^{n,2^m} = \begin{cases} \underset{u,v}{argmin} \|S(u,v) - (1-\alpha_j)\widetilde{E}(t_i,0) - \alpha_j C(t_i)\|, & j < 2^{m-1}, \\ c(t_i), & j = 2^{m-1}, \\ \underset{u,v}{argmin} \|S(u,v) - (\alpha_j - 1)\widetilde{E}(t_i,1) - (2-\alpha_j)C(t_i)\|, & j > 2^{m-1}, \end{cases} \tag{15}$$

where $\alpha_j = \frac{j}{2^{m-1}}$.

Then, the meaning of the second refinement method is to increment m in Equation (15) by one, (almost) doubling the amount of control points corresponding to each ruling. After applying this method of refinement we, again, need to update T, re-analyze expression (13) and apply further refinements as needed. See Figure 3 (c) for an example of the second method of refinement.

The third method of refinement shrinks $\widetilde{E}$ towards $C(t)$ and T towards $c(t)$ and is applied when a pre-determined number of refinements, $max_refinements$, fails to reduce the bound for the Hausdorff distance. This shrinkage is attained by halving the values of $r_{min}^{t_i}$ and $r_{max}^{t_i}$, for every sampled ruling and reconstructing $\widetilde{E}$ and T. This step is taken to ensure convergence of the refinement algorithm, when

$$\forall (t,r), \exists (u,v) \in Image(T) \text{ such that } \|S(u,v) - \widetilde{E}(t,r)\| \leq \varepsilon, \tag{16}$$

while

$$\exists (u,v) \in Image(T) \text{ such that } \forall (t,r), \|S(u,v) - \widetilde{E}(t,r)\| > \varepsilon. \tag{17}$$

Meaning that $\widetilde{E}$ is approximated well by $\widetilde{S}$ (Equation (16)) while some points on $\widetilde{S}$ are too far from $\widetilde{E}$ (Equation (17)). See Figure 3 (d) for an example of the third method of refinement.

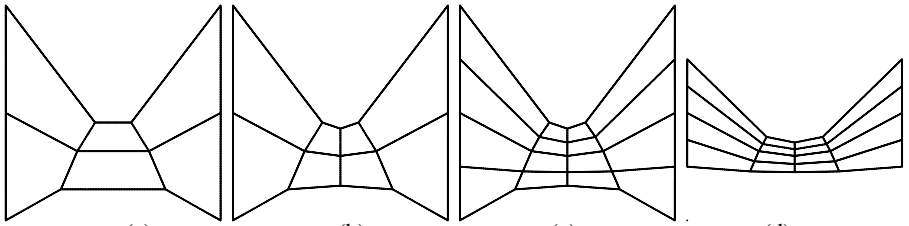


Fig. 3. Different refinements of the image of the bilinear mapping, T. (a) Initial T (4 rulings, 3 samples per ruling). (b) After applying the first refinement method to (a) (having now, 5 rulings, 3 samples per ruling). (c) After applying the second refinement method to (b) (having now, 5 rulings, 5 samples per ruling). (d) After applying the third refinement method to (c).

In summary, the algorithm used to refine T and $\widetilde{E}$ is:

Refine($S, \widetilde{E}, \{t_i\}$)

$m \leftarrow 1$;
$num_refinements \leftarrow 0$;
Construct T, using the current value of m;
Find the point of maximal difference, (t_{max}, r_{max});
while $\|\widetilde{S}(t_{max}, r_{max}) - \widetilde{E}(t_{max}, r_{max})\|^2 > \varepsilon^2$ **do**
  $num_refinements \leftarrow num_refinements + 1$;
  Find i such that $t_i \leq t_{max} \leq t_{i+1}$;
  Add ruling at $\frac{t_i + t_{i+1}}{2}$; {refinement of the first kind}
  if $t_{max} \simeq t_i$ or $t_{max} \simeq t_{i+1}$ **then**
    $m \leftarrow m + 1$; {refinement of the second kind}
  if $num_refinements = max_refinements$ **then**
    $num_refinements \leftarrow 0$;
    for all ruling t_i **do**
      $r^{t_i}_{min} \leftarrow \frac{r^{t_i}_{min}}{2}, \quad r^{t_i}_{max} \leftarrow \frac{r^{t_i}_{max}}{2}$; {refinement of the third kind}
  Reconstruct $\widetilde{E}$;
  Reconstruct T, using the current value of m;
  Find the point of maximal difference, (t_{max}, r_{max});
End

Using the combined application of the first and second methods of refinement, T converges to a mapping function, which solves Equation (10) for every point (t, r) in the domain of $\widetilde{E}$. This process is usually sufficient. However, in cases identified by Equations (16) and (17), the third method of refinement ensures convergence, as it shrinks $\widetilde{E}$ to $C(t)$ and T to $c(t)$. In summary, the Hausdorff distance between the limit developable surface and the composition of S over the limit matching function is bounded by ε and the refinement algorithm stops.

The parametric region of S we approximated is $Image(T)$. Notice that as T is piecewise bilinear, $Image(T)$ is a polygon. Specifically, it is the polygon $\{Q_{0,j}\}_{j=0}^{2^m}$, $\{Q_{i,2^m}\}_{i=0}^{n}$, $\{Q_{n,j}\}_{j=2^m}^{0}$, $\{Q_{i,0}\}_{i=n}^{0}$, where $\{Q_{i,j}\}_{i=0,j=0}^{n,2^m}$ is the set of control points of T.

2.6 Smoothing the Approximated Region

As the boundary of $Image(T)$ is piecewise linear, depending on the amount of rulings sampled, these boundaries may be jagged. In order to avoid these jagged boundaries which, in turn, lead to jagged developables, we smooth the boundaries of $Image(T)$.

$Image(T)$ is comprised of the control points $\{Q_{i,j}\}_{i=0,j=0}^{n,2^m}$, of T. As $n+1$, the number of rulings, is typically significantly larger than 2^m+1, the number of sampled control points along a ruling, we opt to smooth the boundary polylines: $\{Q_{i,0}\}_{i=0}^{n}$ and $\{Q_{i,2^m}\}_{i=0}^{n}$, which are, therefore, more prone to jagged features.

This smoothing problem is defined as follows: Let $c(t)$ be a curve in the parametric domain of S, as described in Section 2.1 and $\{Q_{i,j}\}_{i=0,j=0}^{n,2^m}$ be the set of control points of the piecewise bilinear surface defined in Section 2.4 as T. Then, we wish to find two new sets of control points $\{\widetilde{Q}_{i,0}\}_{i=0}^{n}$ and $\{\widetilde{Q}_{i,2^m}\}_{i=0}^{n}$, which satisfy the following expressions:

$$\widetilde{Q}_{i,j} = (1-\beta_{i,j})c(t_i) + \beta_{i,j}Q_{i,j}, \quad 0 \le \beta_{i,j} \le 1, \tag{18}$$

for $0 \le i \le n$, $j \in \{0, 2^m\}$, such that the following expression is minimized:

$$\gamma \sum_{i=1}^{n-1} \sum_{j\in\{0,2^m\}} \|\widetilde{Q}_{i-1,j} - 2\widetilde{Q}_{i,j} + \widetilde{Q}_{i+1,j}\|^2 + \sum_{i=0}^{n} \sum_{j\in\{0,2^m\}} \|\widetilde{Q}_{i,j} - Q_{i,j}\|^2, \tag{19}$$

where γ is a user selected smoothness weight. The left-hand side of Expression (19) penalizes (curved) jagged edges and the right-hand side penalizes large changes.

This problem is linear with regards to $\beta_{i,j}$ and we solve it as an over-determined, linearly constrained, least squares problem. See Figures 4 and 5 for an example of the smoothing algorithm.

After smoothing the edges of $Image(T)$, we recalculate for every ruling, $r_{min}^{t_i}$ and $r_{max}^{t_i}$ over the smoothed approximated region, and then reconstruct $\widetilde{E}$ and T (with the same values of n and m as in the last version) and re-analyze the Hausdorff distance. If the Hausdorff distance is bounded by ε without additional refinement, we stop. Otherwise, we refine T and $\widetilde{E}$, as needed, and repeat the smoothing algorithm and so on.

Note that as each smoothing only diminishes the magnitudes of $r_{min}^{t_i}$ and $r_{max}^{t_i}$, i.e. shrinks $\widetilde{E}$ towards $C(t)$, bounding the Hausdorff distance would require a less refined T. In all our experiments, we never needed to apply further refinement after applying the smoothing algorithm.

2.7 Adding Developables

So far, we handled one user supplied curve, c over S, which we used to create the first developable surface. In order to generate a complete piecewise developable approximation of S, we need to add more developable surfaces along additional curves on S. Our algorithm provides the user with the boundary of the already approximated region as a candidate curve, which he can accept or modify using the GUI. Alternatively, the user can provide an entirely new curve by drawing it on S, which we project to the parametric domain of S for our algorithm to use. Alternatively, a greedy scheme that uses the boundary of an approximated region to construct one developable surface after another can be used, until all the domain of S is covered.

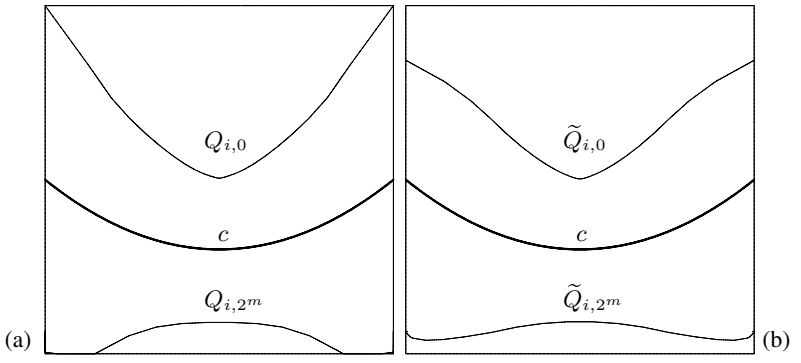


Fig. 4. (a) The polylines $\{Q_{i,0}\}_{i=0}^{n}$, and $\{Q_{i,2^m}\}_{i=0}^{n}$ in the parametric domain of S prior to and (b) after smoothing, see also Figure 5

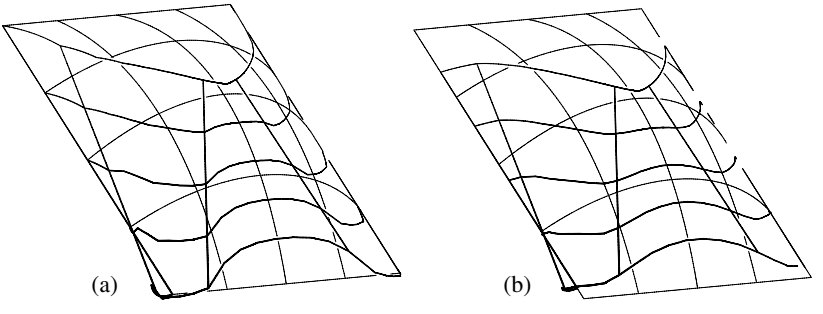


Fig. 5. (a) The developable ruled surface $\widetilde{E}$ before smoothing and (b) after smoothing the parametric edges, see also Figure 4

An obvious difficulty to overcome, when more than one developable is present, is the problem of overlaps between adjacent envelope surfaces, i.e. having the coverage of S being mutually exclusive. We've used three approaches to solving this problem. Due to lack of space they are only briefly described. The first approach limits the r-values generated in the analysis of Expression (4). This is achieved by subtracting the already approximated regions from the region approximated by each sampled ruling, $P_{t_i}^0$. The second approach constructs each developable surface independently, then projects boundaries of the already approximated regions as trimming curves of the new developable surface. The third approach uses a triangulation of the developables, created by either the first or the second approach, followed by a merge process of vertices on adjacent developables' boundaries.

3 Results

Figures 6 and 7 show experimental results of our algorithm. In Figure 6 we show an approximation of a bi-cubic B-spline surface (not a surface of revolution), bounded by the unit cube and modeling half of a fruit bowl shown in Figure 6 (a). The approximation

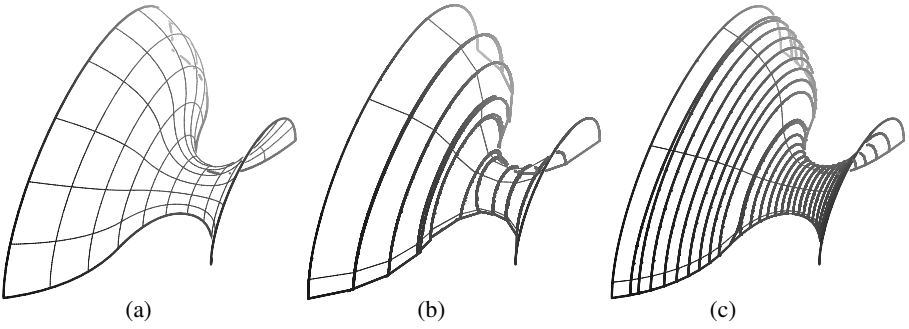


Fig. 6. Approximation of a bi-cubic surface bounded by the unit cube. (a) The input surface. (b) Approximation with tolerance of $\varepsilon = 10^{-2}$ (11 developable surfaces needed). (c) Approximation with tolerance of $\varepsilon = 10^{-3}$ (27 developable surfaces needed).

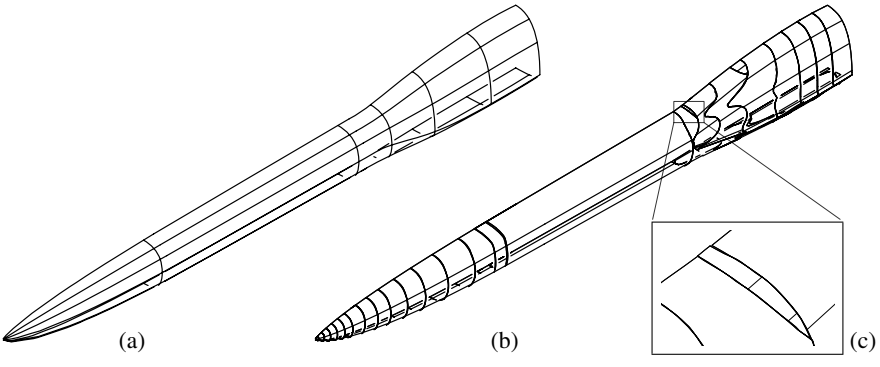


Fig. 7. Approximation of a bi-quadratic surface bounded by a $10 \times 1 \times 1$ box. (a) The input surface. (b) Approximation with tolerance of $\varepsilon = 10^{-3}$ (33 developable surfaces needed).

shown in Figure 6(b) is up to $\varepsilon = 10^{-2}$ and required 11 developables. The approximation shown in Figure 6 (c) is up to $\varepsilon = 10^{-3}$ and required 27 developables. In Figure 7 we show an approximation of a bi-quadratic B-spline surface, bounded by a $(10 \times 1 \times 1)$ box and modeling part of a jet fighter's fuselage shown in Figure 7 (a). The approximation shown in Figure 7 (b) is up to $\varepsilon = 10^{-3}$ and required 33 developables, some of which are too small to notice in the figure (see Figure 7 (c)). All of these examples were executed on a P-IV 2.8Ghz with 512mb of RAM and required, on average, 20 seconds were needed to generate each developable surface (aside from the time it took the user to add each curve).

In our implementation, we used MATLAB [15] to solve the constrained linear least squares problem defined in Section 2.5 and the IRIT solid modeler [7] as the GUI and as our symbolic computational environment (for more details on symbolic multivariate computations see [10]).

4 Conclusions and Future Work

In this paper, we presented an algorithm capable of approximating a general NURBS surface with a set of developable surfaces, constructed as envelopes of tangents of curves along the input surface and whose Hausdorff distance to the input surface is bounded by a user defined threshold. Currently, the algorithm suggests boundary curves to the user, who can accept, modify or ignore these suggestions and draw completely new curves. The presented procedure is time consuming and requires a certain expertise from the user. However, we feel this investment is justified, as this semi-automatic development process will be performed only once, and with typically better results than a completely automatic process, which can then be manufactured, in any quantity. The selection of curves greatly affects the number of developables needed to approximate the input surface as well as the size of developables. Clearly, the question of which parametric curves would yield the best results needs to be further explored.

Furthermore, we experimented with a completely automatic method of curve generation. This method uses one of the input surface's (trimmed) boundaries as the initial curve and creates an initial developable. Then, the boundary of the region approximated by the generated developable is used as the next curve and so on. This greedy "advancing front" process, when repeated, can generate a complete approximation with no user intervention. However, the resulting developable surfaces are not visually pleasing and are certainly not optimal, in the number of developables. We feel that a completely automatic process is the next logical step and intend to improve or extend this method to generate better results.

We also intend to further investigate the stitching problem of adjacent developables. Intersecting adjacent developables can be stitched, and sometimes trimmed, along the intersection curve. However, adjacent developables do not always intersect and simply connecting boundary sections of adjacent developables with new surfaces results in many small additional surfaces. Regardless, in real life, the developable surfaces, cut from the manufacturing material, have non-negligible thickness and therefore by selecting an error threshold smaller than the manufacturing tolerance, the user can ensure proper contact between adjacent developables.

Finally, some materials, such as cloth, latex etc., can be deformed (stretched) during the manufacturing process. These qualities result in relaxed demands on the developability of the approximating surfaces. An interesting subset of these materials can even handle anisotropic deformation, such as fabrics which can be stretched to different degrees in different directions. We hope to enhance our algorithm in order to exploit these qualities, thereby, generating a better approximation.

Acknowledgments

This work was partially supported by European FP6 NoE grant 506766 (AIM@SHAPE) and in part by the Minerva Schlesinger Laboratory for Life Cycle Engineering.

References

1. G. Aumann. Interpolation with developable Bezier patches. *Computer Aided Geometric Design*, 8(5):409–420, 1991.
2. J. Bloomenthal and B. Wyvill, editors. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. Section 2.6.4.
3. M. P. D. Carmo. *Differential Geometry of Curves and Surfaces*, pages 195–197. Prentice-Hall, 1976.
4. H.-Y. Chen, I.-K. Lee, S. Leopoldseder, H. Pottmann, T. Randrup, and J. Wallner. On surface approximation using developable surfaces. *Graphical Models and Image Processing*, 61(2):110–124, 1999.
5. X. Chen, R. F. Riesenfeld, and E. Cohen. Degree reduction for NURBS symbolic computation on curves. In preparation.
6. E. Cohen, R. F. Riesenfeld, and G. Elber. *Geometric Modeling with Splines*, chapter 9. A K Peters, 2001.
7. G. Elber. Irit solid modeler. `http://www.cs.technion.ac.il/~irit`
8. G. Elber. Model fabrication using surface layout projection. *Computer-aided Design*, 27(4):283–291, April 1995.
9. G. Elber and E. Cohen. Second-order Surface Analysis Using Hybrid Symbolic and Numeric Operator. *ACM Trans. Graph*, 12(2):160–178, 1993.
10. G. Elber and M.-S. Kim. Geometric constraint solver using multivariate rational spline functions. In *The Sixth ACM/IEEE Symposium on Solid Modeling and Applications, Ann Arbor, Michigan*, pages 1–10, June 2001.
11. J. Hass, R. T. Farouki, C. Y. Han, X. Song, and T. W. Sederberg. Guaranteed consistency of surface intersections and trimmed surfaces using a coupled topology resolution and domain decomposition scheme. To appear in *Advances in Computational Mathematics*, 2005. `http://mae.ucdavis.edu/~farouki/index.html`
12. J. Hoschek. Approximation of surfaces of revolution by developable surfaces. *Computer-Aided Design*, 30(10):757–763, 1998.
13. J. Hoschek and H. Pottmann. Interpolation and approximation with developable B-spline surfaces. In M. Dæhlen, T. Lyche, and L. L. Schumaker, editors, *Proceedings of the first Conference on Mathematical Methods for Curves and Surfaces (MMCS-94)*, pages 255–264, Nashville, USA, June 16–21 1995. Vanderbilt University Press.
14. S. Leopoldseder and H. Pottmann. Approximation of developable surfaces with cone spline surfaces. *Computer-Aided Design*, 30(7):571–582, 1998.
15. Matlab ©, copyright 1984-2002, The Mathworks, Inc.
See also http://www.mathworks.com/.
16. F. Park, J. Yu, C. Chun, and B. Ravani. Design of developable surfaces using optimal control. *Journal of Mechanical Design*, 124(4):602–608, December 2002.
17. M. Peternell. Developable surface fitting to point clouds. In *Computer Aided Geometric Design*, pages 785–803, 2004.
18. M. Peternell. Recognition and reconstruction of developable surfaces from point clouds. In *GMP*, pages 301–310, 2004.
19. H. Pottmann and G. E. Farin. Developable rational Bézier and B-spline surfaces. *Computer Aided Geometric Design*, 12(5):513–531, 1995.
20. H. Pottmann and J. Wallner. Approximation algorithms for developable surfaces. *Computer Aided Geometric Design*, 16(6):539–556, 1999.

Efficient Piecewise Linear Approximation of Bézier Curves with Improved Sharp Error Bound

Weiyin Ma[1] and Renjiang Zhang[1,2]

[1] City University of Hong Kong,
Department of Manufacturing Engineering and Engineering Management,
83 Tat Chee Avenue, Kowloon, Hong Kong (SAR), China
{mewma, rzhang2}@cityu.edu.hk
http://www.cityu.edu.hk/meem/
[2] On leave from China Jiliang University, College of Science,
Hangzhou 310018, China
zrj@cjlu.edu.cn

Abstract. This paper presents an efficient algorithm for piecewise linear approximation of Bézier curves with improved sharp error bound. Given a Bézier curve of arbitrary degree, an approximation polygon having the same number of vertices as that of the control polygon is obtained through efficient local refinement of the initial control vertices. The approximation produces improved error bound compared with several existing solutions. With the explicit sharp error bound, it is also possible for prior estimation of necessary subdivisions to meet a pre-defined tolerance. The approximation can also be locally and adaptively refined for reducing the number of vertices of the piecewise linear approximation while meeting the required tolerance.

1 Introduction

Given a geometric model, it is often necessary to find a piecewise linear approximation of the underlying curve or surface with a given tolerance. One typical application is for the tessellation of a CAD (computer aided design) model in design and manufacturing. Another application is for efficient visualization of geometric models using standard graphics systems through the use of faceted linear approximations of the original surfaces. One may also find applications in subdivision based model trimming, intersection and manipulation. In literature, one may find a large number of publications on linear approximation in general of various geometric models.

For some applications, such as graphics visualization, the piecewise linear approximation is often realized through the use of refined control meshes/polygons [1]. This approach is quite efficient. However it may result in a large error bound and hence requires a fine mesh to meet the tolerance requirement. In some other applications, such as in design and manufacturing, one often uses a tessellated polygon/mesh defined by curve or surface points as a piecewise linear approximation [2, 3]. While this approach involves extra computation, it produces a small error bound for accurate approximation. These tessellation algorithms can be broadly classified into two classes, i.e. uniform tessellation and adaptive tessellation. Comparatively, uniform tessellation

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 157 – 174, 2006.
© Springer-Verlag Berlin Heidelberg 2006

produces fast results, but it may end up with a global dense mesh in order to meet the tolerance requirement of locally curved surface regions. Adaptive tessellation produces adaptively refined meshes through repeated error evaluation and subdivision based on a given tolerance. A typical example for adaptive linear approximation of Bézier curves can be found in [3].

In connection with the proposed algorithm on Bézier curves presented in this paper, Nairn et al. (1999) obtained sharp, quantitative bounds on the distance between a polynomial curve and its Bézier control polygon [4]. This pioneering work received a lot of attention and was generalized to surfaces by Reif (1999) [5]. The later also provided an elegant proof of Nairn et al's result. Lutterkort and Peters (2001) also developed a framework for efficiently computing enclosures for multivariate polynomials [6]. Karavelas et al. (2004) also generalized the result of Nairn et al. to two dimensions [7]. Recently, Zhang and Wang (2006) used quasi control polygon to approximate Bézier curves and obtained two closer sharp and quantitative bounds [8]. The advantage of the method of [8] is that with little increase of computation, it produces much better results than that of using refined control polygons for approximation. All the above-mentioned methods are based on the following considerations. The line segments for approximation should be easily constructed and the number of line segments should be as less as possible.

This paper presents an efficient algorithm for piecewise linear approximation of Bézier curves with further improved sharp error bound compared with several existing solutions. We construct an approximation polygon having the same number of edges as that of the initial control polygon. It is realized through one step of the de Casteljau central subdivision followed by a further step of local averaging. With the proposed method, it is much more accurate to approximate a Bézier curve than that using either its control polygon [4-5], two refined control polygons produced by the de Casteljau central subdivision algorithm, or the quasi-control polygon [8]. As discussed in Section 5, it also produces better results than piecewise linear approximation using interpolation points at node positions with the same number of segments. While the proposed algorithms works for arbitrary degrees, explicit sharp bounds of approximation are also derived for curves with degrees not larger than six. The estimation is bounded in terms of the maximal absolute difference of the sequence of control points and a constant that only depends on the degrees of the curves. Thus, it improves the approximation error bounds and, at the same time, also maintains less computing time compared with other similar approaches. In literature, one may also find related work for piecewise linear approximation of B-spline with explicit error bound using (refined) control polygon/meshes (see [9, 10] and references therein).

The rest of the paper is organized as follows. In Section 2 we present the algorithm for the construction of the new *approximation polygon* of Bézier curve for arbitrary degree using the de Casteljau subdivision and a further step of local averaging. In Section 3, we derive the corresponding sharp bounds of Bezier curves for degrees up to six. In Section 4, we illustrate how to compute the number of global refinement with given tolerance in advance. In Section 5, we compare the computation efficiency of the proposed algorithm with several existing methods and some further examples. A brief conclusion with possible future work can be found in Section 6.

2 An Algorithm for Bézier Curve Linear Approximation

A Bézier curve of degree n is defined by the following equation

$$\mathbf{p}(t) := \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t), \quad \text{for } t \in [0,1].$$

where $B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i$, for $i = 0,1,\cdots,n$, are the Bernstein basis functions and $\mathbf{b}_i$, for $i = 0,1,\cdots,n$, are the so-called control points. The piecewise linear curve connecting all the control points, $\mathbf{b}_0, \mathbf{b}_1, \cdots, \mathbf{b}_n$, defines the control polygon of the Bézier curve with n line segments.

Given a Bézier curve defined by the above equation, this paper constructs a piecewise linear approximation of the original curve with the same number of line segments as that of the control polygon. As to be discussed in Section 4, the linear approximation produces improved sharp error bound compared with several other methods using either the control polygon, the quasi-control polygon, or with the same number of curve points at node positions. The algorithm is given as follows:

Algorithm 1. Let $\mathbf{b} = \{\mathbf{b}_0, \mathbf{b}_1, \cdots, \mathbf{b}_n\}$ be the set of control points for a Bézier curve of degree n, a piecewise linear *approximation polygon* $\tilde{\mathbf{b}} = \{\tilde{\mathbf{b}}_0, \tilde{\mathbf{b}}_1, \cdots, \tilde{\mathbf{b}}_n\}$ of the original Bézier curve can be constructed in two steps as follows (see Figs. 1-2):

Step 1. Construct a set of refined control vertices using the de Casteljau central subdivision algorithm [1] as shown in Fig. 1 and Fig. 2 for odd and even degrees, respectively, and write the refined control vertices as $\mathbf{b}_0^1, \mathbf{b}_1^1, \mathbf{b}_2^1, \cdots, \mathbf{b}_{2n-1}^1, \mathbf{b}_{2n}^1$.

Step 2. Denote $\tilde{\mathbf{b}}_0 = \mathbf{b}_0$ and $\tilde{\mathbf{b}}_n = \mathbf{b}_n$. Construct a set of new vertices $\tilde{\mathbf{b}}_1, \tilde{\mathbf{b}}_2, \cdots, \tilde{\mathbf{b}}_{n-1}$ through two steps of local averaging. We distinguish two cases based on the degree n, the procedure of construction is as follows:

- *Case 1. For odd degree n with $n \geq 3$*: Denote the mid points of the line segments $\mathbf{b}_1^1\mathbf{b}_2^1$ and $\mathbf{b}_2^1\mathbf{b}_3^1$ as $\mathbf{b}_1^2$ and $\mathbf{b}_2^2$, respectively, and further denote the mid point of line segment $\mathbf{b}_1^2\mathbf{b}_2^2$ as $\tilde{\mathbf{b}}_1$. This leads to $\tilde{\mathbf{b}}_1 = \frac{1}{4}\left(\mathbf{b}_1^1 + 2\mathbf{b}_2^1 + \mathbf{b}_3^1\right)$. Similarly, denote the mid points of the line segments $\mathbf{b}_3^1\mathbf{b}_4^1$ and $\mathbf{b}_4^1\mathbf{b}_5^1$ as $\mathbf{b}_3^2$ and $\mathbf{b}_4^2$, respectively, and further denote the mid point of the line segment $\mathbf{b}_3^2\mathbf{b}_4^2$ as $\tilde{\mathbf{b}}_2$.

This leads to $\tilde{\mathbf{b}}_2 = \frac{1}{4}\left(\mathbf{b}_3^1 + 2\mathbf{b}_4^1 + \mathbf{b}_5^1\right)$. In general, denote the mid points of the line segments $\mathbf{b}_{2i-1}^1\mathbf{b}_{2i}^1$ and $\mathbf{b}_{2i}^1\mathbf{b}_{2i+1}^1$ as $\mathbf{b}_{2i-1}^2$ and $\mathbf{b}_{2i}^2$, respectively, and further denote the mid point of the line segment $\mathbf{b}_{2i-1}^2\mathbf{b}_{2i}^2$ as $\tilde{\mathbf{b}}_i$. It leads to $\tilde{\mathbf{b}}_i = \frac{1}{4}\left(\mathbf{b}_{2i-1}^1 + 2\mathbf{b}_{2i}^1 + \mathbf{b}_{2i+1}^1\right)$. Repeating the procedure till $i = n-1$, one obtains the last point as $\tilde{\mathbf{b}}_{n-1} = \frac{1}{4}\left(\mathbf{b}_{2n-3}^1 + 2\mathbf{b}_{2i-2}^1 + \mathbf{b}_{2i-1}^1\right)$. Fig. 1 illustrates the constructed linear approximations for odd degrees of $n = 3$ and $n = 5$, respectively.

- *Case 2. For even degree n with $n \geq 4$*: Following a similar method for *Case 1*, we have the following equations for evaluating the approximation points:

(1) $\tilde{\mathbf{b}}_1 = \frac{1}{4}\left(\mathbf{b}_1^1 + 2\mathbf{b}_2^1 + \mathbf{b}_3^1\right)$, $\tilde{\mathbf{b}}_2 = \frac{1}{4}\left(\mathbf{b}_3^1 + 2\mathbf{b}_4^1 + \mathbf{b}_5^1\right)$, …,

$\tilde{\mathbf{b}}_i = \frac{1}{4}\left(\mathbf{b}_{2i-1}^1 + 2\mathbf{b}_{2i}^1 + \mathbf{b}_{2i+1}^1\right)$, …, $\tilde{\mathbf{b}}_{\frac{n}{2}-1} = \frac{1}{4}\left(\mathbf{b}_{n-3}^1 + 2\mathbf{b}_{n-2}^1 + \mathbf{b}_{n-1}^1\right)$;

(2) $\tilde{\mathbf{b}}_{\frac{n}{2}} = \mathbf{b}_n^1$;

(3) $\tilde{\mathbf{b}}_{\frac{n}{2}+1} = \frac{1}{4}\left(\mathbf{b}_{n+1}^1 + 2\mathbf{b}_{n+2}^1 + \mathbf{b}_{n+3}^1\right)$, …, $\tilde{\mathbf{b}}_i = \frac{1}{4}\left(\mathbf{b}_{2i-1}^1 + 2\mathbf{b}_{2i}^1 + \mathbf{b}_{2i+1}^1\right)$, …,

$\tilde{\mathbf{b}}_{n-1} = \frac{1}{4}\left(\mathbf{b}_{2n-3}^1 + 2\mathbf{b}_{2i-2}^1 + \mathbf{b}_{2i-1}^1\right)$.

Fig. 2 illustrates the constructed linear approximations for even degrees of $n = 4$ and $n = 6$, respectively.

Following the above procedure, we obtain a new polygon $\tilde{\mathbf{b}}_0\tilde{\mathbf{b}}_1\cdots\tilde{\mathbf{b}}_n$ as a piecewise linear approximation to the original Bézier curve. Each of the vertices $\tilde{\mathbf{b}}_i$, for $i = 0, 1, \cdots, n$, corresponds to a control vertex $\mathbf{b}_i$ of the corresponding control polygon and a Greville abscissae $t_i = i/n$ of the original Bézier curve [1,4].

Various examples show that the effect of using the above *approximation polygon* to approximate the original Bézier curve is very good. While it is a difficult task for evaluating the approximation error bound for general degree n, we derive the error bounds for lower degrees up to $n = 6$ in the following section.

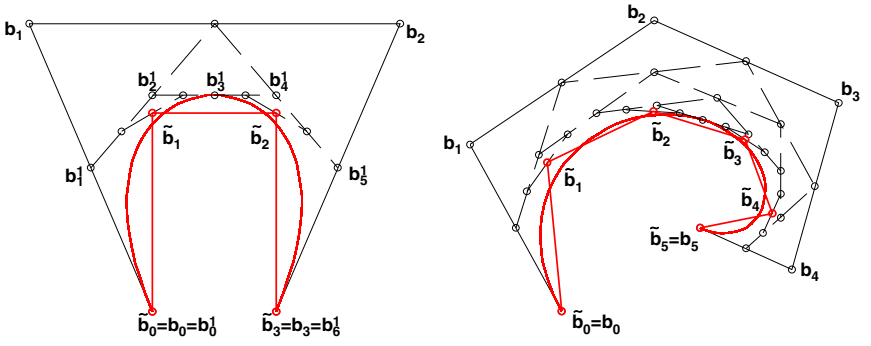


Fig. 1. Construction of approximation polygons for Bézier curves of odd degrees with $n = 3$ (left) and $n = 5$ (right), respectively

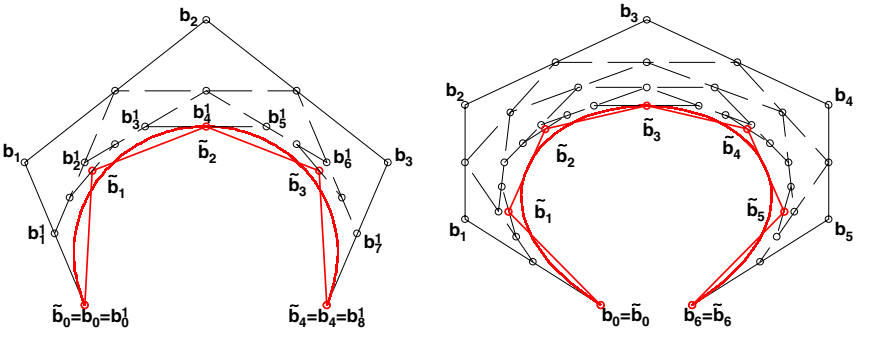


Fig. 2. Construction of approximation polygons for Bézier curves of even degrees with $n = 4$ (left) and $n = 6$ (right), respectively

3 Evaluation of the Approximation Bounds

Based on the method introduced in Section 2, we can evaluate the vertices of the approximation polygon of a Bézier curve in terms of the original control vertices. Table 1 summarizes the derived vertices of the approximation polygon for degrees $n = 3,4,5,6$.

In the following, we first derive the error bounds of the approximation polygons for Bernstein-Bézier polynomial curves followed by an extension to general Bézier curves in three-dimensional space. To further proceed, we introduce several symbols. The i-th second order difference of the control vertices and the infinitive norm of the vector formed by the second order differences are respectively abbreviated as

$$\Delta_2 b_i := b_{i-1} - 2b_i + b_{i+1}, \quad \left\|\Delta_2 b\right\|_\infty := \max_{0<i<n} \left|\Delta_2 b_i\right|.$$

Table 1. Vertices of the approximation polygon for Bézier curves of degrees n=3,4,5,6

Degree n	Vertices of the approximation polygon $\tilde{\mathbf{b}} = \{\tilde{\mathbf{b}}_0, \tilde{\mathbf{b}}_1, \cdots, \tilde{\mathbf{b}}_n\}$
$n=3$	$\{\mathbf{b}_0,\ \frac{9\mathbf{b}_0+15\mathbf{b}_1+7\mathbf{b}_2+\mathbf{b}_3}{32},\ \frac{\mathbf{b}_0+7\mathbf{b}_1+15\mathbf{b}_2+9\mathbf{b}_3}{32},\ \mathbf{b}_3\}$
$n=4$	$\{\mathbf{b}_0,\ \frac{9\mathbf{b}_0+15\mathbf{b}_1+7\mathbf{b}_2+\mathbf{b}_3}{32},\ \frac{\mathbf{b}_0+4\mathbf{b}_1+6\mathbf{b}_2+4\mathbf{b}_3+\mathbf{b}_4}{16},\ \frac{\mathbf{b}_1+7\mathbf{b}_2+15\mathbf{b}_3+9\mathbf{b}_4}{32},\ \mathbf{b}_4\}$
$n=5$	$\{\mathbf{b}_0,\ \frac{9\mathbf{b}_0+15\mathbf{b}_1+7\mathbf{b}_2+\mathbf{b}_3}{32},\ \frac{9\mathbf{b}_0+33\mathbf{b}_1+46\mathbf{b}_2+30\mathbf{b}_3+9\mathbf{b}_4+\mathbf{b}_5}{128},\ \frac{\mathbf{b}_0+9\mathbf{b}_1+30\mathbf{b}_2+46\mathbf{b}_3+33\mathbf{b}_4+9\mathbf{b}_5}{128},\ \frac{\mathbf{b}_2+7\mathbf{b}_3+15\mathbf{b}_4+9\mathbf{b}_5}{32},\ \mathbf{b}_5\}$
$n=6$	$\{\mathbf{b}_0,\ \frac{9\mathbf{b}_0+15\mathbf{b}_1+7\mathbf{b}_2+\mathbf{b}_3}{32},\ \frac{9\mathbf{b}_0+33\mathbf{b}_1+46\mathbf{b}_2+30\mathbf{b}_3+9\mathbf{b}_4+\mathbf{b}_5}{128},\ \frac{\mathbf{b}_0+6\mathbf{b}_1+15\mathbf{b}_2+20\mathbf{b}_3+15\mathbf{b}_4+6\mathbf{b}_5+\mathbf{b}_6}{64},\ \frac{\mathbf{b}_1+9\mathbf{b}_2+30\mathbf{b}_3+46\mathbf{b}_4+33\mathbf{b}_5+9\mathbf{b}_6}{128},\ \frac{\mathbf{b}_3+7\mathbf{b}_4+15\mathbf{b}_5+9\mathbf{b}_6}{32},\ \mathbf{b}_6\}$

We have then the following theorem regarding related error bounds of the proposed piecewise linear approximation:

Theorem 1. *Given a Bernstein-Bézier polynomial curve of degree* $n = 3, 4, 5, 6$*, and let* $\tilde{l}(t)$ *be the approximation polygon, we have then the following inequality:*

$$\left\| p_n(t) - \tilde{l}(t) \right\|_\infty \le C(n) \left\| \Delta_2 b \right\|_\infty, \text{ for } n \le 6 .$$

where $C(3) = \frac{75}{1024}$, $C(4) = \frac{25}{384}$, $C(5) = \frac{7}{80}$, $C(6) = \frac{5}{48}$, *and all the error bounds are sharp.*

Proof. The method for deriving all the above error bounds is the same for various degrees $n = 3,4,5,6$. We just give the proof for the case of degree 5 as shown in Fig. 3 for short. Following Table 1, we have the following piecewise linear approximation, i.e., the approximation polygon, for a Bernstein-Bézier polynomial curve of degree $n = 5$, where the first component for each of the vertices is the corresponding Greville abscissae $t_i = i/n$ of the original polynomial.

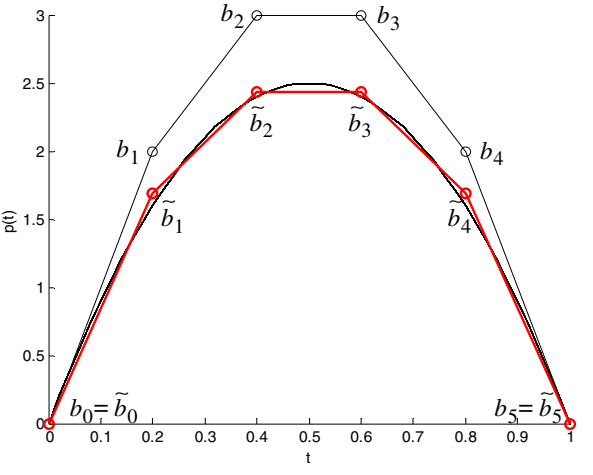


Fig. 3. Approximation of a Bernstein-Bézier polynomial curve with 5 segments

$$\left\{(0,b_0),\left(\frac{1}{5},\frac{9b_0+15b_1+7b_2+b_3}{32}\right),\left(\frac{2}{5},\frac{9b_0+33b_1+46b_2+30b_3+9b_4+b_5}{128}\right),\right.$$
$$\left.\left(\frac{3}{5},\frac{b_0+9b_1+30b_2+46b_3+33b_4+9b_5}{128}\right),\left(\frac{4}{5},\frac{b_2+7b_3+15b_4+9b_5}{32}\right),(1,b_5)\right\}.$$

The main idea of the proof is to evaluate the maximum difference for each of the segments defined by the Greville abscissa. See Appendix A for details. ■

Following Theorem 1, we can easily derive the following Corollary for a general Bézier curves in three-dimensional space.

Corollary. *Given a Bézier curve of degree* $n \le 6$ *with control points* $\mathbf{b}_i(x_i, y_i, z_i)$, *and let* $\tilde{\mathbf{I}}(t)$ *be the approximation polygon, we have then the following inequality:*

$$\left\|\mathbf{P}_n(t) - \tilde{\mathbf{I}}(t)\right\| \le C(n)\left\|\Delta_2\mathbf{b}\right\|, \text{ for } n \le 6$$

where $\left\|\Delta_2\mathbf{b}\right\| := \sqrt{\left\|\Delta_2 x\right\|_\infty^2 + \left\|\Delta_2 y\right\|_\infty^2 + \left\|\Delta_2 z\right\|_\infty^2}$, $C(3) = \frac{75}{1024}$, $C(4) = \frac{25}{384}$, $C(5) = \frac{7}{80}$, $C(6) = \frac{5}{48}$, *and all the bounds are sharp.*

4 Global Refinement and Adaptive Approximation

Given a Bézier curve, if the approximation polygon defined in the previous section does not meet the tolerance requirement, one may either perform a global refinement or apply an adaptive refinement procedure. For the former with global refinement, we

can estimate the number of subdivisions necessary for meeting the tolerance requirement in advance. For the later with adaptive refinement, we also provide an algorithm for implementation.

For a given Bézier curve, we can estimate the number of subdivisions in advance according to a given tolerance ε using the obtained sharp error bounds. Let $\mathbf{b}_0, \mathbf{b}_1, \cdots, \mathbf{b}_n$ be the set of control points of the original curve. After applying the de Casteljau's central subdivision algorithm once to the initial set of control points, we get two sub-Bézier curves. Denote the two sets of control points as $\mathbf{b}_0^{1,1}, \mathbf{b}_1^{1,1}, \cdots, \mathbf{b}_n^{1,1}$ and $\mathbf{b}_0^{1,2}, \mathbf{b}_1^{1,2}, \cdots, \mathbf{b}_n^{1,2}$, respectively, with the first superscript indicating the level of subdivision and the second superscript indicating the indices of the Bézier curves. After a further refinement using the de Casteljau algorithm, we get four sub-Bézier curves. We write the new control points as $\mathbf{b}_0^{2,1}, \mathbf{b}_1^{2,1}, \cdots, \mathbf{b}_n^{2,1}$, $\mathbf{b}_0^{2,2}, \mathbf{b}_1^{2,2}, \cdots, \mathbf{b}_n^{2,2}$, $\mathbf{b}_0^{2,3}, \mathbf{b}_1^{2,3}, \cdots, \mathbf{b}_n^{2,3}$ and $\mathbf{b}_0^{2,4}, \mathbf{b}_1^{2,4}, \cdots, \mathbf{b}_n^{2,4}$. In general after k levels of refinement, we obtain refined control points as $\mathbf{b}_0^{k,1}, \mathbf{b}_1^{k,1}, \cdots, \mathbf{b}_n^{k,1}$, $\mathbf{b}_0^{k,2}, \mathbf{b}_1^{k,2}, \cdots, \mathbf{b}_n^{k,2}$, $\cdots$, $\mathbf{b}_0^{k,2^k}, \mathbf{b}_1^{2,2^k}, \cdots, \mathbf{b}_n^{2,2^k}$. It is easy to show [4] that

$$\left\|\Delta_2 \mathbf{b}^{k,j}\right\| \le \frac{1}{4}\left\|\Delta_2 \mathbf{b}^{k-1,j}\right\| \le \cdots \le \left(\frac{1}{4}\right)^k \left\|\Delta_2 \mathbf{b}\right\|.$$

We now define the 2^k *approximation polygons* for the above 2^k refined control polygons. The error between each of the individual sub-Bézier curve and the corresponding *approximation polygon* is not larger than $C(n)\left\|\Delta_2 \mathbf{b}^{k,j}\right\|$. Let

$$\left(\frac{1}{4}\right)^k C(n)\left\|\Delta_2 \mathbf{b}\right\| < \varepsilon,$$

we finally obtain

$$k > \frac{\ln\left(C(n)\left\|\Delta_2 \mathbf{b}\right\|/\varepsilon\right)}{\ln 4}$$

In other words, to guarantee that the error between the line segments to the original Bézier curve is not larger than ε, the number of refinement that we need to apply using the de Casteljau algorithm should be at least $\left[\frac{\ln\left(C(n)\left\|\Delta_2 \mathbf{b}\right\|/\varepsilon\right)}{\ln 4}\right] + 1$, where $[x]$ denotes the integral part of x, followed by the construction of the individual *approximation polygons*.

All previous discussions are based on global refinement. To reduce the number of line segments of the approximation, one may use the following adaptive subdivision algorithm. In this algorithm, we adaptively refine the input curve into a set of k properly ordered Bézier curves S_i, for $i = 0, 1, \cdots, k-1$. In the process of adaptive refinement, each refined Bézier curve is associated with a flag $f_i = 1$ if it satisfies the

required tolerance, but otherwise with a flag $f_i = 0$. The refinement is performed starting from curve with the lowest index of the refined Bézier curves. At certain level of iteration, a pointer p indicates the position for current refinement, i.e., all curves S_i, for $i < p$, should have been properly refined with $f_i = 1$, and the remaining curves S_i, for $i \geq p$, need to be further refined starting from current position S_p.

Algorithm 2

Input: The degree of the original Bézier curve n, the control points $\mathbf{b}_0, \mathbf{b}_1, \cdots, \mathbf{b}_n$, and a given tolerance ε.

Output: Approximation line segments of Bézier curve.

Steps:

1. Denote the initial Bézier curve as $S_0(\mathbf{b}_0^0, \mathbf{b}_1^0, \cdots, \mathbf{b}_n^0)$ and set the current number of Bézier curves $k = 1$. Evaluate $C(n)\left\|\Delta_2 \mathbf{b}^0\right\|$ and mark S_0 with a tolerance flag $f_0 = 1$ and set a pointer $p = 1$ if it satisfies the inequality $C(n)\left\|\Delta_2 \mathbf{b}^0\right\| \leq \varepsilon$. In this case, there is no need to perform curve refinement and directly go to Step 6 for producing the final approximation polygon. Otherwise mark S_0 with a tolerance flag $f_0 = 0$ and set $p = 0$.
2. We have now a set of k properly ordered Bézier curves S_i, for $i = 0, 1, \cdots, k-1$. All curves S_i for $i < p$ meet the required tolerance and there is no need of further subdivision. We need to verify all other curves S_i, for $i \geq p$, for possible further refinement. If the flag f_p of S_p is 1, there is no need to subdivide the current curve and we can shift the pointer to the next curve by letting $p = p + 1$ and go to Step 5. Otherwise continue the following step for local refinement.
3. In this case, it is necessary to subdivide the current Bézier curve $S_p(\mathbf{b}_0^p, \mathbf{b}_1^p, \cdots, \mathbf{b}_n^p)$ into two sub-Bézier curves using the de Casteljau central subdivision algorithm. After subdivision, we replace the current Bézier curve by its two sub-Bézier curves, update the indices of S_i and their flags for $i = p+1, \cdots, k-1$, if any, with increment by 1, and update the total number of curves $k = k + 1$.
4. For $i = p, p+1$, compute $C(n)\left\|\Delta_2 \mathbf{b}^i\right\|$ and mark S_i with a flag $f_i = 1$ if it satisfies the inequality $C(n)\left\|\Delta_2 \mathbf{b}^i\right\| \leq \varepsilon$. Otherwise mark S_i with a flag $f_i = 0$. Go to Step 2 to shift the pointer if necessary.
5. If the current pointer $p \geq k$, i.e., all refined Bézier curves have meet the tolerance requirement. Go to Step 6 for producing the final approximation polygons. Otherwise, go to Step 2 for further verification.

6. Construct the *approximation polygons* for all S_i, for $i = 0,1,\cdots,k-1$, and output the approximation line segments of the Bézier curve.
7. The end of the algorithm.

We now give two examples. Example 1 shows that one can approximate a Bernstein-Bézier curve with less number of line segments using the approach of this paper compared with that using the refined control polygon. Example 2 shows that, one achieves further data reduction with adaptive approximation.

Example 1. Let us consider a Bernstein-Bézier curve of degree $n = 3$ defined by a set of control points $\{b_0, b_1, \cdots, b_n\} = \{0,1,1,0\}$. For a given tolerance $\varepsilon = 0.005$, we need 3 times subdivision and 24 line segments in order to approximate the original curve using refined control polygons. However, we only need 2 times subdivision and 12 line segments with the proposed approximation presented in this paper.

Example 2. Let us now consider another Bézier curve of degree $n = 5$ defined by a set of control points $\{\mathbf{b}_0, \mathbf{b}_1, \cdots, \mathbf{b}_n\} = \{(0,0), (1,4), (2,0), (4,2), (6,2), (8,0)\}$ as shown in Fig. 4. Given a tolerance $\varepsilon = 0.05$, we need 3 times subdivision and 40 line segments in order to approximate the original curve using the refined control polygons with global refinement. With adaptive refinement, we need 3 times subdivision, but only need to use 6 control polygons, namely $S_1^3, S_2^3, S_2^2, S_3^2, S_7^3, S_8^3$, and a total of

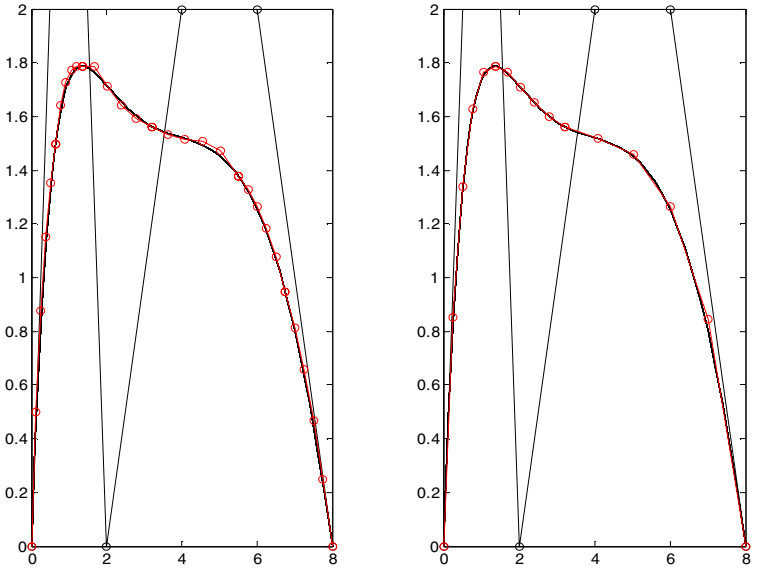


Fig. 4. A total number of 30 line segments is required to approximate the curve using adaptively refined control polygons (left); and a total number of 15 line segments is required to approximate the curve using the adaptive approximation polygons of this paper (right)

30 line segments (left) to approximate the curve using refined control polygons, where the superscript and subscript indicate the level of subdivision and indices of the control polygons at that level, respectively. However, we only need 2 times subdivision and 3 approximation polygons $S_0 = T_1^2, S_1 = T_2^2, S_2 = T_2^1$ with a total number of 15 line segments (right) to approximate the original curve with the adaptive approach (Algorithm 2) presented in this paper.

5 Computation Efficiency and Further Discussions

For a comparison, Table 2 summarizes various approximation error bounds of control polygon [4], quasi control polygon [8], interpolatory polygon at node positions [8], and the *approximation polygon* presented in this paper, respectively, and for approximating a curve of degrees $n = 2,3,4,5,6$, respectively. In this table, $\Delta_2 = \|\Delta_2 b\|_\infty$.

Following Table 2, it is obvious that the *approximation polygon* presented in this paper produces much better results than that using the original control polygon. As a matter of fact, the proposed approximation is even better than that using two refined sub-control polygons produced by the de Casteljau's algorithm with mid subdivision. The actual error bounds using two refined sub-control polygons to approximate the original Bézier curve for degrees $3,4,5,6$ are $\frac{1}{12}\|\Delta_2 b\|_\infty$, $\frac{1}{8}\|\Delta_2 b\|_\infty$, $\frac{2}{20}\|\Delta_2 b\|_\infty$ and $\frac{3}{16}\|\Delta_2 b\|_\infty$, respectively. Another obvious disadvantage in using two refined sub-control polygon is that the number of line segments is doubled compared with that of the proposed approximation of this paper.

Compared with the approximation using quasi control polygon [8], the approach of this paper also produces tighter error bounds with slight increase in computation. For a Bézier curve of degree 5, e.g., the approximation using quasi control polygon needs 12 additions and 4 multiplications, while the proposed approximation of this paper needs 27 additions and 27 multiplications. However, the error bound using the proposed approximation is reduced by a factor of $\frac{1}{4}$ compared with that of the quasi control polygon. In general for the cases of $n = 3,4,5,6$, the error bounds using the

Table 2. Comparison of various bounds for Bézier piecewise linear approximation

Degree n	2	3	4	5	6
Error bound using control polygon [4]	$\frac{1}{4}\Delta_2$	$\frac{1}{3}\Delta_2$	$\frac{1}{2}\Delta_2$	$\frac{3}{5}\Delta_2$	$\frac{3}{4}\Delta_2$
Error bound using a quasi control polygon [8].	$\frac{1}{16}\Delta_2$	$\frac{1}{12}\Delta_2$	$\frac{1}{4}\Delta_2$	$\frac{7}{20}\Delta_2$	$\frac{1}{2}\Delta_2$
Error bound using an interpolatory polygon [8].	$\frac{1}{16}\Delta_2$	$\frac{1}{12}\Delta_2$	$\frac{3}{32}\Delta_2$	$\frac{1}{10}\Delta_2$	$\frac{5}{48}\Delta_2$
Error bound using the proposed approximation of this paper.	$\frac{1}{16}\Delta_2$	$\frac{75}{1024}\Delta_2$	$\frac{25}{384}\Delta_2$	$\frac{7}{80}\Delta_2$	$\frac{5}{48}\Delta_2$

proposed approximation are reduced by a factor of $^{900}/_{1024}$, $^{100}/_{384}$, $^{1}/_{4}$ and $^{5}/_{24}$, respectively, compared with that using the quasi control polygon approximation.

Compared with the case of interpolatory polygon at node positions, the proposed approximation of this paper also produces better approximation as shown in Table 2.

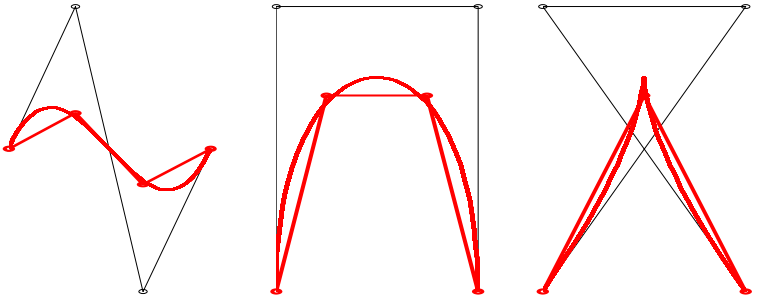


Fig. 5. Illustration of approximation effect for Bézier curves of degree 3 using control polygon and the *approximation polygon* of this paper

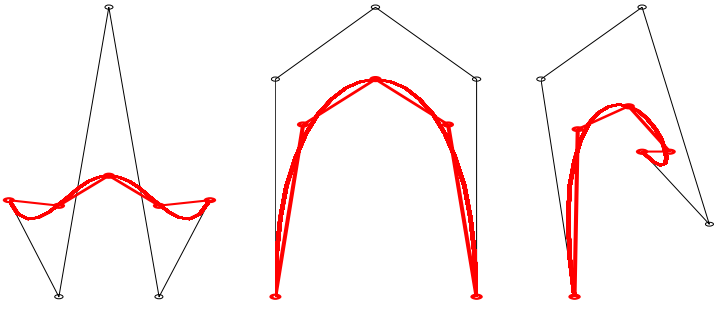


Fig. 6. Illustration of approximation effect for Bézier curves of degree 4 using control polygon and the *approximation polygon* of this paper

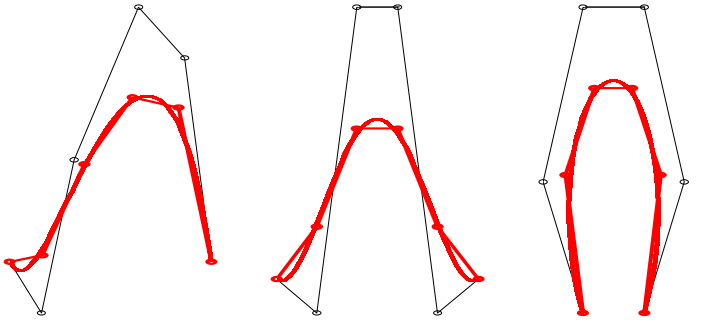


Fig. 7. Illustration of approximation effect for Bézier curves of degree 5 using control polygon and the *approximation polygon* of this paper

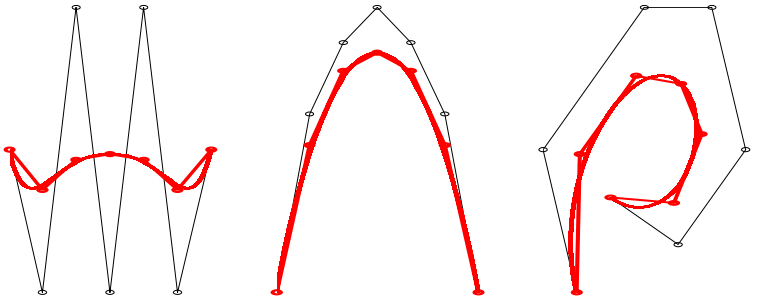


Fig. 8. Illustration of approximation effect for Bézier curves of degree 6 using control polygon and the *approximation polygon* of this paper

In addition, the computation complexity is much less with the proposed approximation than that of the approximation using interpolatory polygon at node positions.

Figs. 5-8 show the approximation effect of using control polygon and the proposed approximation for Bézier curves of degrees 3, 4, 5 and 6 respectively.

6 Conclusions and Future Work

This paper presents an efficient algorithm for linear approximation of Bézier curves with improved sharp error bounds compared with a few other existing solutions. From a given control polygon, an approximation polygon with the same number of linear segments as that of the control polygon of the original Bézier curve is constructed. It is realized through one step of the de Casteljau central subdivision followed by a further step of local averaging. The proposed approximation can be efficiently implemented with either global refinement or adaptive approximation. While the proposed algorithm can be applied to approximate a Bézier curve of arbitrary degrees, this paper only presents the sharp approximation bounds for degrees up to six and the sharp approximation bounds for Bézier curve of higher degrees need to be further derived. It is also quite interesting for practical applications to extend the proposed approach to surfaces and to other modeling schemes, such as B-splines and subdivision surfaces.

Acknowledgement

The work presented in this paper is sponsored by City University of Hong Kong through a Strategic Research Grant (#7001928) and by the Research Grants Council of Hong Kong through a CERG research grant (#CityU 1131/03E).

References

1. Farin, G., Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide, Academic Press, Boston, 2002.
2. Piegl, L., Tiller, W. The NURBS Book, Springer, New York, 1995.

3. Cho, W. J., Maekawa, T., Patrikalakis, N. M., Topologically reliable approximation of composite Bezier curves, Computer Aided Geometric Design, 1996, 13(6):497-520.
4. Nairn, D., Peter, J. and Lutterkort, D. Sharp, quantitative bounds on the distance between a polynomial piece and its Bézier control polygon. Computer Aided Geometric Design, 1999, 16(7):613-631.
5. Reif, U. Best bounds on the approximation of polynomials and splines by their control structure, Computer Aided Geometric Design, 2000, 17:579-589.
6. Lutterkort, D. and Peters, J. Optimized refinabled enclosures of multivariate ploynomial pieces, Computer Aided Geometric Design, 2001, 18:861-863.
7. Karavelas, M. I., Kaklis, P. D. and Kostas, K. V. Bounding the distance between 2D parametric Bezier curves and their control polygon. Computing, 2004, 72(1-2):117-128.
8. Zhang, R. J. and Wang, G. J. Sharp bounds on the approximation of a Bézier polynomial by its quasi control polygon, Computer Aided Geometric Design, 2006, 23(1):1-16.
9. Lutterkort, D. and Peters, J. Tight linear envelopes for splines, Numerische Mathematik, 2001, 89:735-748.
10. Peters, J., Efficient one-sided linearization of spline geometry, In M. J. Wilson and R. R. Martin (eds.), Mathematics of Surfaces 2003, LNCS 2768, Springer-Verlag, Berlin, 2003, 297-319.

Appendix A. Proof of Theorem 1

We first derive the error bounds segment by segment (case by case) and finally find the largest one as the error bound from the approximation polygon to the original Bézier curve, i.e. $\left\| p_5(t) - \tilde{l}(t) \right\|_\infty$ for $0 \le t \le 1$.

Case 1. For the first segment within the interval $\left[0, \frac{1}{5}\right]$, the expression of the approximation polygon is as follows:

$$\tilde{l}_{[0,1/5]}(t) = \frac{5}{32}(-23b_0 + 15b_1 + 7b_2 + b_3)t + b_0$$

We have then the following estimation of the error bound for this segment

$$p_5(t) - \tilde{l}_{[0,1/5]}(t)$$

$$= \left(B_0^5 + \frac{115}{32}t - 1 \right) b_0 + \left(B_1^5 - \frac{75}{32}t \right) b_1 + \left(B_2^5 - \frac{35}{32}t \right) b_2 + \left(B_3^5 - \frac{5}{32}t \right) b_3 + B_4^5 b_4 + B_5^5 b_5$$

$$:= \sum_{i=0}^{5} \beta_i(t) b_i$$

It is easy to check that in the above equation,

$$\sum_{i=0}^{5} \beta_i(t) = 0 \quad \text{and} \quad \sum_{i=0}^{5} i\beta_i(t) = 0 .$$

Following the linear precision [8], we obtain

$$p_5(t) - \tilde{l}_{[0,1/5]}(t) = \alpha_1 \Delta_2 b_1 + \alpha_2 \Delta_2 b_2 + \alpha_3 \Delta_2 b_3 + \alpha_4 \Delta_2 b_4$$

where

$$\alpha_1 = B_0^5 + \frac{115}{32}t - 1\,,\quad \alpha_2 = B_3^5 + 2B_4^5 + 3B_5^5 - \frac{5}{32}t\,,\quad \alpha_3 = 2B_5^5 + B_4^5,\ \alpha_4 = B_5^5,$$

with $\alpha_3 > 0$, $\alpha_4 > 0$. Within the interval $[0,1/5]$, it is easy to show that:

$$\left|\alpha_4 + \alpha_3 + \alpha_2 + \alpha_1\right| = \left|10t(t - \frac{5}{32})\right| \le \frac{7}{80} \tag{1}$$

Similarly, it is not difficult to compute that

$$\left|\alpha_1 - \alpha_2 + \alpha_3 + \alpha_4\right| \le 0.0454 \tag{2}$$

$$\left|-\alpha_1 - \alpha_2 + \alpha_3 + \alpha_4\right| \le 0.0728 \tag{3}$$

$$\left|-\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4\right| \le 0.0458 \tag{4}$$

Combine the above expressions (1)-(4), and note that $\alpha_3 > 0$ and $\alpha_4 > 0$, we have

$$\left|p_5(t) - \tilde{l}_{[0,1/5]}(t)\right| \le (\left|\alpha_1\right| + \left|\alpha_2\right| + \left|\alpha_3\right| + \left|\alpha_4\right|)\left\|\Delta_2 b\right\|_\infty = (\left|\alpha_1\right| + \left|\alpha_2\right| + \alpha_3 + \alpha_4)\left\|\Delta_2 b\right\|_\infty$$
$$\le \max\{\max_{0\le t\le\frac{1}{5}}\left|\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4\right|, \max_{0\le t\le\frac{1}{5}}\left|\alpha_1 - \alpha_2 + \alpha_3 + \alpha_4\right|,$$
$$\max_{0\le t\le\frac{1}{5}}\left|-\alpha_1 - \alpha_2 + \alpha_3 + \alpha_4\right|, \max_{0\le t\le\frac{1}{5}}\left|-\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4\right|\}\left\|\Delta_2 b\right\|_\infty \le \frac{7}{80}\left\|\Delta_2 b\right\|_\infty.$$

Case 2. For the interval $\left[\frac{1}{5}, \frac{2}{5}\right]$, the approximation polygon can be defined as:

$$\tilde{l}_{[1/5,2/5]}(t) = \frac{1}{128}(5t-1)(-27b_0 - 27b_1 + 18b_2 + 26b_3 + 9b_4 + b_5)$$
$$+\frac{1}{32}(9b_0 + 15b_1 + 7b_2 + b_3)$$

Applying the linear precision again, we deduce that

$$p_5(t) - \tilde{l}_{[1/5,2/5]}(t) = \left[B_0^5 + \frac{27}{128}(5t-1) - \frac{9}{32}\right]b_0 + \left[B_1^5 + \frac{27}{128}(5t-1) - \frac{15}{32}\right]b_1 +$$
$$\left[B_2^5 - \frac{18}{128}(5t-1) - \frac{7}{32}\right]b_2 + \left[B_3^5 - \frac{26}{128}(5t-1) - \frac{1}{32}\right]b_3$$
$$+\left[B_4^5 - \frac{9}{128}(5t-1)\right]b_4 + \left[B_5^5 - \frac{1}{128}(5t-1)\right]b_5$$
$$= \alpha_1\Delta_2 b_1 + \alpha_2\Delta_2 b_2 + \alpha_3\Delta_2 b_3 + \alpha_4\Delta_2 b_4\,,$$

where

$$\alpha_1 = B_0^5 + \frac{27}{128}(5t-1) - \frac{9}{32},$$

$$\alpha_2 = 2\left(B_0^5 + \frac{27}{128}(5t-1) - \frac{9}{32} \right) + B_1^5 + \frac{27}{128}(5t-1) - \frac{15}{32},$$

$$\alpha_3 = 2B_5^5 + B_4^5 - \frac{10}{128}(5t-1),$$

$$\alpha_4 = B_5^5 - \frac{1}{128}(5t-1).$$

It is easy to find out that

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 10t^2 - \frac{795}{128}t + \frac{119}{128}.$$

Thus, we get the following for the interval $[1/5, 2/5]$

$$\left|\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4\right| \le \frac{7}{80} \tag{5}$$

In a similar way, it is not difficult to compute that

$$\left|\alpha_1 + \alpha_2 + \alpha_3 - \alpha_4\right| \le 0.0869 \tag{6}$$

$$\left|\alpha_1 + \alpha_2 - \alpha_3 + \alpha_4\right| \le 0.0734 \tag{7}$$

$$\left|\alpha_1 - \alpha_2 - \alpha_3 + \alpha_4\right| \le 0.0728 \tag{8}$$

$$\left|\alpha_1 - \alpha_2 + \alpha_3 + \alpha_4\right| \le 0.0201 \tag{9}$$

$$\left|\alpha_1 - \alpha_2 + \alpha_3 - \alpha_4\right| \le 0.0194 \tag{10}$$

$$\left|\alpha_1 - \alpha_2 - \alpha_3 + \alpha_4\right| \le 0.0256 \tag{11}$$

$$\left|\alpha_1 - \alpha_2 - \alpha_3 - \alpha_4\right| \le 0.0304 \tag{12}$$

Combining Equations (5)-(12), we get that

$$\left| p_5(t) - \tilde{l}_{[1/5,2/5]}(t) \right| \le (|\alpha_1| + |\alpha_2| + |\alpha_3| + |\alpha_4|) \left\| \Delta_2 b \right\|_\infty$$

$$\le \max\{ \max_{\frac{1}{5} \le t \le \frac{2}{5}} |\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4|,$$

$$\max_{\frac{1}{5} \le t \le \frac{2}{5}} |\alpha_1 + \alpha_2 + \alpha_3 - \alpha_4|, \max_{\frac{1}{5} \le t \le \frac{2}{5}} |\alpha_1 - \alpha_2 - \alpha_3 + \alpha_4|, \max_{\frac{1}{5} \le t \le \frac{2}{5}} |\alpha_1 - \alpha_2 + \alpha_3 + \alpha_4|,$$

$$\max_{\frac{1}{5} \le t \le \frac{2}{5}} |\alpha_1 - \alpha_2 + \alpha_3 + \alpha_4|, \max_{\frac{1}{5} \le t \le \frac{2}{5}} |\alpha_1 - \alpha_2 + \alpha_3 - \alpha_4|, \max_{\frac{1}{5} \le t \le \frac{2}{5}} |\alpha_1 - \alpha_2 - \alpha_3 + \alpha_4|,$$

$$\max_{\frac{1}{5} \le t \le \frac{2}{5}} |\alpha_1 - \alpha_2 - \alpha_3 - \alpha_4| \} \left\| \Delta_2 b \right\|_\infty \le \frac{7}{80} \left\| \Delta_2 b \right\|_\infty .$$

Case 3. On the interval $\left[\frac{2}{5},\frac{3}{5}\right]$, we have

$$\tilde{l}_{[2/5,3/5]}(t)=\frac{1}{128}(5t-2)(-8b_0-24b_1-16b_2+16b_3+24b_4+8b_5)$$
$$+\frac{1}{128}(9b_0+33b_1+46b_2+30b_3+9b_4).$$

Applying the linear precision again, we obtain that

$$p_5(t)-\tilde{l}_{[2/5,3/5]}(t)=\left[B_0^5+\frac{8}{128}(5t-2)-\frac{9}{128}\right]b_0+\left[B_1^5+\frac{24}{128}(5t-2)-\frac{33}{128}\right]b_1+$$
$$\left[B_2^5+\frac{16}{128}(5t-2)-\frac{46}{128}\right]b_2+\left[B_3^5-\frac{16}{128}(5t-2)-\frac{30}{128}\right]b_3+$$
$$\left[B_4^5-\frac{24}{128}(5t-2)-\frac{9}{128}\right]b_4+\left[B_5^5-\frac{8}{128}(5t-2)\right]b_5$$
$$=\alpha_1\Delta_2 b_1+\alpha_2\Delta_2 b_2+\alpha_3\Delta_2 b_3+\alpha_4\Delta_2 b_4$$

where

$$\alpha_1=B_0^5+\frac{8}{128}(5t-2)-\frac{9}{128},$$
$$\alpha_2=2B_0^5+B_1^5+\frac{40}{128}(5t-2)-\frac{51}{128},$$
$$\alpha_3=2B_5^5+B_4^5-\frac{40}{128}(5t-2)-\frac{9}{128},$$
$$\alpha_4=B_5^5-\frac{8}{128}(5t-2).$$

A direct calculation shows that

$$|\alpha_1+\alpha_2+\alpha_3+\alpha_4|\le 0.0609 \tag{13}$$
$$|\alpha_1+\alpha_2+\alpha_3-\alpha_4|\le 0.0405 \tag{14}$$
$$|\alpha_1+\alpha_2-\alpha_3+\alpha_4|\le 0.0234 \tag{15}$$
$$|\alpha_1-\alpha_2-\alpha_3+\alpha_4|\le 0.0334 \tag{16}$$
$$|\alpha_1-\alpha_2+\alpha_3+\alpha_4|\le 0.0383 \tag{17}$$
$$|\alpha_1-\alpha_2+\alpha_3-\alpha_4|\le 0.0079 \tag{18}$$
$$|\alpha_1-\alpha_2-\alpha_3+\alpha_4|\le 0.0256 \tag{19}$$
$$|\alpha_1-\alpha_2-\alpha_3-\alpha_4|\le 0.0561 \tag{20}$$

Combining Equations (13)-(20), we thus derive that

$$\begin{aligned}
&\left|p_5(t)-\tilde{l}_{[2/5,3/5]}(t)\right| \\
&\quad\le(\alpha_1+\alpha_2+\alpha_3+\alpha_4)\left\|\Delta_2 b\right\|_\infty \le \max\{\max_{\frac{2}{5}\le t\le\frac{3}{5}}\left|\alpha_1+\alpha_2+\alpha_3+\alpha_4\right|, \\
&\quad\max_{\frac{2}{5}\le t\le\frac{3}{5}}\left|\alpha_1+\alpha_2+\alpha_3-\alpha_4\right|,\max_{\frac{2}{5}\le t\le\frac{3}{5}}\left|\alpha_1-\alpha_2-\alpha_3+\alpha_4\right|,\max_{\frac{2}{5}\le t\le\frac{3}{5}}\left|\alpha_1-\alpha_2+\alpha_3+\alpha_4\right|, \\
&\quad\max_{\frac{2}{5}\le t\le\frac{3}{5}}\left|\alpha_1-\alpha_2+\alpha_3+\alpha_4\right|\max_{\frac{2}{5}\le t\le\frac{3}{5}}\left|\alpha_1-\alpha_2+\alpha_3-\alpha_4\right|,\max_{\frac{2}{5}\le t\le\frac{3}{5}}\left|\alpha_1-\alpha_2-\alpha_3+\alpha_4\right|, \\
&\quad\max_{\frac{2}{5}\le t\le\frac{3}{5}}\left|\alpha_1-\alpha_2-\alpha_3-\alpha_4\right|\}\left\|\Delta_2 b\right\|_\infty \le 0.0609\left\|\Delta_2 b\right\|_\infty \le \frac{7}{80}\left\|\Delta_2 b\right\|_\infty .
\end{aligned}$$

For the remaining two cases of $t\in\left[\frac{3}{5},\frac{4}{5}\right]$ and $t\in\left[\frac{4}{5},1\right]$, it is easily known that the conclusion of the theorem is also valid on these intervals by applying a parameter transformation with $t=1-u$.

Finally, we show that the inequality is sharp. Considering a large class of functions satisfying conditions $\Delta_2 b_i=1$ for i=1,2,3,$\cdots$, one easily knows that the equality holds in this case following the procedure of the proof. It thus completes the proof. ■

Approximate μ-Bases of Rational Curves and Surfaces

Liyong Shen[1,3], Falai Chen[1], Bert Jüttler[2], and Jiansong Deng[1]

[1] Department of Mathematics, University of Science and Technology of China
[2] Institute of Applied Geometry, Johannes Kepler University, Linz, Austria
[3] KLMM, Institute of Systems Science, AMSS, Chinese Academy of Sciences
{chenfl, dengjs}@ustc.edu.cn, bert.juettler@jku.at

Abstract. The μ-bases of rational curves and surfaces are newly developed tools which play an important role in connecting parametric forms and implicit forms of curves and surfaces. However, exact μ-bases may have high degree with complicated rational coefficients and are often hard to compute (especially for surfaces), and sometimes they are not easy to use in geometric modeling and processing applications. In this paper, we introduce approximate μ-bases for rational curves and surfaces, and present an algorithm to compute approximate μ-bases. The algorithm amounts to solving a generalized eigenvalue problem and some quadratic programming problems with linear constraints. As applications, approximate implicitization and degree reduction of rational curves and surfaces with approximate μ-bases are discussed. Both the parametric equations and the implicit equations of the approximate curves/surfaces are easily obtained by using the approximate μ-bases. As indicated by the examples, the proposed algorithm may be a useful alternative to other methods for approximate implicitization.

Keywords: approximate μ-bases, approximate implicitization.

1 Introduction

The concept of μ-bases was first introduced in [9] to derive a compact representation for the implicit equation of a planar rational curve. The basic idea of μ-bases originates in a method called *moving curves and surfaces* to implicitize rational curves and surfaces [16]. The μ-basis of a planar rational curve of degree n consists of two polynomials $p(x, y; t)$ and $q(x, y; t)$ which are linear in x, y and degree μ and $n - \mu$ in t respectively, where $0 \leqslant \mu \leqslant \lfloor n/2 \rfloor$. The resultant of p and q with respect to t gives the implicit equation of the rational curve. In the generic case, $\mu = \lfloor n/2 \rfloor$, and thus the implicit equation of a rational curve can be expressed as a determinant of size $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$, whereas previous resultant-based methods express the implicit equation as either an $n \times n$ determinant or an $2n \times 2n$ determinant. The μ-basis can not only compute the implicit equation of a rational curve, but also recover the parametric equation of the curve conveniently. Thus μ-bases connect the and the parametric form of a curve.

The concept of μ-bases was subsequently generalized to rational ruled surfaces[2,6] and general rational surfaces [7]. Various algorithms to compute the

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 175–188, 2006.
© Springer-Verlag Berlin Heidelberg 2006

μ-bases for rational curves and rational surfaces were developed [3,10,19]. Applications of μ-bases to implicitization, singular point computation and surface reparameterization are explored as well [4,5,8]. Thus μ-bases provide a new tool to study curves and surfaces in geometric modeling.

However, the use of exact μ-bases leads to some problems in applications. First, general μ-bases may have very complicated rational coefficients and/or high degree, and they are therefore hard to use in practice. Second, it is very costly to compute μ-bases, especially for surfaces. Finally, curves and surfaces in CAD systems are usually described by floating point coefficients, and in these situations, exact μ-bases are often unnecessary. To overcome these difficulties, we introduce the concept of *approximate* μ-bases. These bases have low degree and are described by floating point coefficients. They can be found by numerical techniques.

A direct application of approximate μ-bases is the *approximate implicitization* (see [1,11,17,18]) of rational curves and surfaces. As an obvious advantage of the new approach, both a parametric and an implicit representation of the approximating curve or surface are available, and the parametric equation can be easily recovered by evaluating the exterior product of the approximate μ-bases. In addition, the new approach can also be used as a degree reduction technique for rational curves and surfaces. See [12,13,14,15] for more information on this topic.

The organization of the paper is as follows. Section 2 reviews some preliminary results about the μ-bases of rational curves and surfaces. Section 3 introduces approximate μ-bases for rational curves and presents an algorithm to compute them. Applications of approximate μ-bases to approximate implicitization and degree reduction are discussed. In Section 4, we generalize the results of Section 3 to rational surfaces. Finally we conclude this paper.

2 μ-Bases of Rational Curves and Surfaces

Consider a planar rational curve in homogenous form

$$\mathbf{P}(t) = (a(t), b(t), c(t)), \tag{1}$$

where $a(t)$, $b(t)$, $c(t)$ are relatively prime polynomials whose maximum degree equals n. A *moving line* is a family of lines with parameter t,

$$L(x, y; t) := A(t)x + B(t)y + C(t) = 0, \tag{2}$$

where $A(t), B(t), C(t)$ are polynomials. For simplicity, sometimes we write a moving line as $\mathbf{L}(t) := (A(t), B(t), C(t))$. The moving line (2) is said to *follow* the rational curve (1) if

$$\mathbf{L}(t) \cdot \mathbf{P}(t) = A(t)a(t) + B(t)b(t) + C(t)c(t) \equiv 0. \tag{3}$$

A *μ-basis* of a planar rational curve of degree n consists of two independent moving lines $p = p_1x+p_2y+p_3$ and $q = q_1x+q_2y+q_3$ that follow the curve, where

the degree in t of p and q sums up to n. Let $\mathbf{p} = (p_1, p_2, p_3)$ and $\mathbf{q} = (q_1, q_2, q_3)$. Then the μ-basis has the following properties [3]:

1. $\mathbf{p} \times \mathbf{q} = \kappa(a, b, c)$ for some non-zero constant κ.
2. For any moving line $\mathbf{l}(t)$, there exist polynomials $h_1(t)$ and $h_2(t)$ such that
$$\mathbf{l}(t) = h_1\mathbf{p} + h_2\mathbf{q}.$$
3. The resultant of p and q with respect to t gives the implicit equation of the rational curve (1).

The concept of μ-bases can be generalized to rational surfaces. Let

$$\mathbf{P}(s,t) = (a(s,t), b(s,t), c(s,t), d(s,t)), \tag{4}$$

be a rational surface in homogeneous form, where a, b, c, d are bivariate polynomials in s and t, and $\gcd(a,\ b,\ c,\ d) = 1$. We assume that the rational surface (4) is given by a proper parameterization. A *moving plane* is defined by

$$L(x, y, z; s, t) := A(s,t)x + B(s,t)y + C(s,t)z + D(s,t) = 0,$$

where A, B, C, D are polynomials in s and t. Sometimes we use $\mathbf{L}(s,t) := (A(s,t), B(s,t), C(s,t), D(s,t))$ to denote the moving plane. The moving plane $\mathbf{L}(s,t)$ is said to *follow* the rational surface (4) if and only if

$$\mathbf{L}(s,t) \cdot \mathbf{P}(s,t) = aA + bB + cC + dD \equiv 0. \tag{5}$$

A *μ-basis* of the rational surface (4) consists of three moving planes $\mathbf{p}, \mathbf{q}, \mathbf{r}$ following (4) such that

$$[\mathbf{p}, \mathbf{q}, \mathbf{r}] = \kappa\mathbf{P}(s,t) \tag{6}$$

for some nonzero constant κ. Here $[\mathbf{p}, \mathbf{q}, \mathbf{r}]$ is the *exterior product* of $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{r}$,

$$[\mathbf{p}, \mathbf{q}, \mathbf{r}] = \left(\begin{vmatrix} p_2 & p_3 & p_4 \\ q_2 & q_3 & q_4 \\ r_2 & r_3 & r_4 \end{vmatrix}, - \begin{vmatrix} p_1 & p_3 & p_4 \\ q_1 & q_3 & q_4 \\ r_1 & r_3 & r_4 \end{vmatrix}, \begin{vmatrix} p_1 & p_2 & p_4 \\ q_1 & q_2 & q_4 \\ r_1 & r_2 & r_4 \end{vmatrix}, - \begin{vmatrix} p_1 & p_2 & p_3 \\ q_1 & q_2 & q_3 \\ r_1 & r_2 & r_3 \end{vmatrix} \right). \tag{7}$$

Furthermore, $\mathbf{p}, \mathbf{q}, \mathbf{r}$ are said to form a *minimal μ-basis* of the rational surface (4) if $\mathbf{p}, \mathbf{q}, \mathbf{r}$ have minimal degree. Unlike curves, for surfaces many possible notions of minimal degree exist. One notion that works well for tensor product surfaces is the following.

1. among all triples $\mathbf{p}, \mathbf{q}, \mathbf{r}$ satisfying (6), $\deg_t(\mathbf{p}) + \deg_t(\mathbf{q}) + \deg_t(\mathbf{r})$ is minimal, and
2. among all triples $\mathbf{p}, \mathbf{q}, \mathbf{r}$ satisfying (6) and the previous condition, $\deg_s(\mathbf{p}) + \deg_s(\mathbf{q}) + \deg_s(\mathbf{r})$ is minimal.

Here, $\deg_t(\mathbf{p}) = \max_{1 \leqslant i \leqslant 4}(\deg_t(p_i))$ when $\mathbf{p} = (p_1, p_2, p_3, p_4)$, and $\deg_t(\mathbf{q})$, $\deg_t(\mathbf{r})$, $\deg_s(\mathbf{p})$, $\deg_s(\mathbf{q})$, $\deg_s(\mathbf{r})$ are defined similarly. Sometimes we refer to the three polynomials

$$p = \mathbf{p} \cdot \mathbf{X}, \quad q = \mathbf{q} \cdot \mathbf{X}, \quad r = \mathbf{r} \cdot \mathbf{X}, \qquad \mathbf{X} = (x, y, z, 1),$$

as the μ-basis of the rational surface (4). As observed in [7], a μ-basis forms a basis for the set of all the moving planes following $\mathbf{P}(s,t)$.

3 Approximate μ-Bases of Rational Curves

In this section, we introduce the novel concept of approximate μ-bases for rational curves and present an algorithm to compute them. The applications to approximate implicitization and to degree reduction of rational curves are also discussed.

3.1 Approximate μ-Bases

For the given rational curve $\mathbf{P}(t)$ defined in (1), if a moving line satisfies

$$A(t)a(t) + B(t)b(t) + C(t)c(t) \approx 0, \tag{8}$$

then we call the moving line $A(t)x + B(t)y + C(t) = 0$ an *approximate moving line* of $\mathbf{P}(t)$. Here "$\approx$" means that the left hand side of the equation (8) is approximately zero with respect to some criteria which will be specified later.

An *approximate μ-basis* of the rational curve $\mathbf{P}(t)$ consists of two approximate moving lines $\mathbf{p}(t)$ and $\mathbf{q}(t)$ such that $\mathbf{p}(t) \times \mathbf{q}(t)$ is a good approximation to $\mathbf{P}(t)$. It is obvious that a different choice for the approximation criteria will lead to a different specification for the approximate μ-basis. In the next subsection, we will provide more details of the criteria in order to facilitate the computation of approximate μ-bases.

3.2 Computation

We describe the computation of the first and of the second approximate moving line.

Computing the first line. The moving line is written in Bézier form,

$$\mathbf{p}(t) = \sum_{i=0}^{\mu} \mathbf{p}_i B_i^{\mu}(t) = (p_1(t), p_2(t), p_3(t)),$$

where $\mathbf{p}_i = (p_{i1}, p_{i2}, p_{i3})$, $p_i(t) = \sum_{j=0}^{\mu} p_{ji} B_j^{\mu}(t)$, and $0 < \mu \leqslant \lfloor n/2 \rfloor$. In order to deal with condition (8), we introduce the following optimization problem:

$$\int_0^1 (\mathbf{P}(t) \cdot \mathbf{p}(t))^2 \mathrm{d}t = \int_0^1 (a(t)p_1(t) + b(t)p_2(t) + c(t)p_3(t))^2 \mathrm{d}t \to \min. \tag{9}$$

Furthermore, we normalize the approximate moving line by imposing

$$\int_0^1 (p_1(t)^2 + p_2(t)^2) \mathrm{d}t = 1. \tag{10}$$

In order to find the first approximate moving line, we minimize (9) subject to (10). Let

$$a(t)p_1(t) + b(t)p_2(t) + c(t)p_3(t) = \mathbf{g}(t) \cdot \mathbf{x},$$

where $\mathbf{g}(t)$ is a vector of dimension $3(\mu + 1)$ with the components

$$g_i(t) = a(t)B_i^\mu(t), \quad g_{\mu+1+i}(t) = b(t)B_i^\mu(t), \quad g_{2\mu+2+i}(t) = c(t)B_i^\mu(t),$$

$i = 0, 1, \ldots, \mu$, and $\mathbf{x} = (p_{0,1}, \cdots, p_{\mu,1}, p_{0,2}, \cdots, p_{\mu,2}, p_{0,3}, \cdots, p_{\mu,3})$ is a vector which consists of all the unknown coefficients of $\mathbf{p}(t)$. The objective function (9) can be rewritten as

$$\int_0^1 (a(t)p_1(t) + b(t)p_2(t) + c(t)p_3(t))^2 \mathrm{d}t = \int_0^1 \mathbf{x} \cdot \mathbf{g}(t)^T \mathbf{g}(t) \cdot \mathbf{x}^T \mathrm{d}t = \mathbf{x}\mathbf{M}\mathbf{x}^T,$$

where $\mathbf{M}$ is a positive semi-definite $3(\mu+1) \times 3(\mu+1)$ matrix. Similarly, the normalization condition is rewritten as

$$\int_0^1 (p_1(t)^2 + p_2(t)^2)\mathrm{d}t = \mathbf{x}\mathbf{N}\mathbf{x}^T,$$

where $\mathbf{N} = \mathrm{diag}(\mathbf{D}, \mathbf{D}, 0)$ is a positive semi-definite $3(\mu+1) \times 3(\mu+1)$ matrix, and $\mathbf{D} = (d_{ij})$ is a positive definite $\mu+1 \times \mu+1$ matrix. The components of the matrices are

$$m_{ij} = \int_0^1 g_i(t)g_j(t)\mathrm{d}t \quad \text{and} \quad d_{ij} = \int_0^1 B_{i-1}^\mu(t)B_{j-1}^\mu(t)\mathrm{d}t$$

the optimization problem can be rewritten as

$$\mathbf{x}\mathbf{M}\mathbf{x}^T \to \min \quad \text{subject to} \quad \mathbf{x}\mathbf{N}\mathbf{x}^T = 1. \tag{11}$$

If $\det \mathbf{M} = 0$, then there exists $\bar{\mathbf{x}} = (\bar{p}_{0,1}, \cdots, \bar{p}_{\mu,1}, \bar{p}_{0,2}, \cdots, \bar{p}_{\mu,2}, \bar{p}_{0,3}, \cdots, \bar{p}_{\mu,3})$ such that

$$\int_0^1 (a(t)\bar{p}_1(t) + b(t)\bar{p}_2(t) + c(t)\bar{p}_3(t))^2 \mathrm{d}t = \bar{\mathbf{x}}\mathbf{M}\bar{\mathbf{x}}^T = 0.$$

In this case, $a(t)\bar{p}_1(t) + b(t)\bar{p}_2(t) + c(t)\bar{p}_3(t) \equiv 0$, which means $\mathbf{p}(t)$ is an *exact* moving line. Otherwise, if $\det \mathbf{M} \neq 0$, i.e., if $\mathbf{M}$ is positive definite, then there do not exist exact moving lines of degree μ. The solution of (11) then defines an *approximate* moving line.

The problem (11) can be solved by using Lagrangian multipliers. A short computation leads to the equations

$$\det(\mathbf{M} - \lambda\mathbf{N}) = 0, \quad (\mathbf{M} - \lambda\mathbf{N})\mathbf{x}^T = 0, \tag{12}$$

and $\mathbf{x}\mathbf{M}\mathbf{x}^T = \lambda$. Therefore computing an approximate moving line $\mathbf{p}(t)$ is equivalent to solving the generalized eigenvalue problem (12).

The determinant $\det(\mathbf{M} - \lambda\mathbf{N})$ is a polynomial of degree $\gamma = 2(\mu+1)$ in λ. Suppose the zeros of $\det(\mathbf{M} - \lambda\mathbf{N})$ are $\lambda_1 \leqslant \lambda_2 \leqslant \cdots \leqslant \lambda_\gamma$, which are the generalized eigenvalues, and the corresponding eigenvectors are $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_\gamma$. Since $\mathbf{x}\mathbf{M}\mathbf{x}^T = \lambda$, the optimal solution is given by $\mathbf{x} = \mathbf{x}_1$. Thus we get one element $\mathbf{p}(t) = (\mathbf{x}_{11} \cdot \mathbf{t}, \mathbf{x}_{12} \cdot \mathbf{t}, \mathbf{x}_{13} \cdot \mathbf{t})$ of the approximate μ-basis. Here $\mathbf{x}_i = (\mathbf{x}_{i1}, \mathbf{x}_{i2}, \mathbf{x}_{i3})$, $\mathbf{x}_{ij}$ is a vector of dimension $\mu+1$, $i = 1, 2, \ldots, \gamma$, $j = 1, 2, 3$, and $\mathbf{t} = (B_0^\mu(t), B_1^\mu(t), \ldots, B_\mu^\mu(t))$.

Computing the second line. An obvious choice for the second element $\mathbf{q}(t)$ of the approximate μ-basis is $\mathbf{q}(t) = (\mathbf{x}_{21} \cdot \mathbf{t}, \mathbf{x}_{22} \cdot \mathbf{t}, \mathbf{x}_{23} \cdot \mathbf{t})$. However, such a choice may have some limitations. First, the degree of $\mathbf{q}(t)$ must be the same as $\mathbf{p}(t)$. Second, it may happen that the curve $\mathbf{p}(t) \times \mathbf{q}(t)$ is not defined at some parameter values in $[0, 1]$, i.e., there exists $t_0 \in [0, 1]$ such that the third component of $\mathbf{p}(t_0) \times \mathbf{q}(t_0)$ is zero. Third, $\mathbf{p}(t) \times \mathbf{q}(t)$ may not be a good approximation of the given curve $\mathbf{P}(t)$. In this section, we develop other techniques to find the second element $\mathbf{q}(t)$ of the approximate μ-basis. We assume that $\deg(\mathbf{q}) = \bar{\mu} \geqslant \mu$.

Let $\mathbf{y}$ be the vector consisting of the coefficients of $\mathbf{q}(t)$. In order to define a reasonable curve from $\mathbf{p} \times \mathbf{q} := \bar{\mathbf{P}}(t) := (\bar{a}, \bar{b}, \bar{c})$, $\mathbf{q}$ must satisfy $p_1(t)q_2(t) - q_2(t)p_1(t) \neq 0$ for all $t \in [0, 1]$. On the other hand, we expect that $\bar{\mathbf{P}}(t)$ is a good approximation of $\mathbf{P}(t)$, i.e., $\bar{a}/a \approx \bar{b}/b \approx \bar{c}/c$. Hence, we minimize

$$\int_0^1 (a\bar{b} - \bar{a}b)^2 + (b\bar{c} - \bar{b}c)^2 + (c\bar{a} - \bar{c}a)^2 \mathrm{d}t = \mathbf{y}\bar{\mathbf{M}}\mathbf{y}^T.$$

Summing up, we need to solve the optimization problem

$$\mathbf{y}\bar{\mathbf{M}}\mathbf{y}^T \to \min \quad \text{subject to} \quad \mathbf{y}\mathbf{N}\mathbf{y}^T = 1 \text{ and } p_1q_2 - p_2q_1 \neq 0, \quad t \in [0, 1]. \tag{13}$$

We write $p_1q_2 - p_2q_1$ in Bernstein-Bézier form. Suppose that its Bézier coefficient vector is $\mathbf{L}\mathbf{y}^T$, where $\mathbf{L}$ is a $(\mu + \bar{\mu} + 1) \times (3\bar{\mu} + 3)$ matrix. Then the constraint $p_1q_2 - p_2q_1 \neq 0$ can be replaced by the sufficient linear conditions $\mathbf{L}\mathbf{y}^T \leqslant -\mathbf{E}$, where $\mathbf{E} = (e_1, e_2, \cdots, e_{3\bar{\mu}+3})^T$, e_i is a small positive number, $i = 1, \ldots, 3(\mu+1)$. Thus instead of solving (13), we will solve

$$\mathbf{y}\bar{\mathbf{M}}\mathbf{y}^T \to \min \quad \text{subject to} \quad \mathbf{y}\mathbf{N}\mathbf{y}^T = 1 \quad \text{and} \quad \mathbf{L}\mathbf{y}^T \leqslant -\mathbf{E}. \tag{14}$$

In order to simplify this problem, we first solve a series of simpler optimization problems,

$$\mathbf{y}\bar{\mathbf{M}}\mathbf{y}^T \to \min \quad \text{subject to} \quad y_i = 1 \quad \text{and} \quad \mathbf{L}\mathbf{y}^T \leqslant -\mathbf{1}, \tag{15}$$

where $i = 1, \ldots, 3\bar{\mu} + 3$, and $\mathbf{1} = (1, 1, \cdots, 1)$ is a column vector of dimension $3\bar{\mu} + 3$. If $\{\mathbf{y} | y_i = 1, \ \mathbf{L}\mathbf{y}^T \leqslant -\mathbf{1}\} \neq \emptyset$, then there exists a solution for the corresponding problem.

Suppose we obtain m solutions $\mathbf{y}_1, \ldots, \mathbf{y}_m$, $m \leqslant 3\bar{\mu} + 3$. Then the coefficient vector of $\mathbf{q}(t)$ is defined as an affine combination

$$\mathbf{y} = \sum_{i=1}^{m} \alpha_i \mathbf{y}_i, \quad \text{where} \quad \alpha_i \in [0, 1] \quad \text{and} \quad \sum_{i=1}^{m} \alpha_i = 1.$$

In the following, we propose a technique to determine the optimal coefficients. In order to find them, we maximize the angle between the two moving lines $p(t) = 0$ and $q(t) = 0$. Since the normals of the two lines are $(p_2, -p_1)$ and $(q_2, -q_1)$ respectively, we will minimize $\int_0^1 (p_1q_1 + p_2q_2)^2 \mathrm{d}t$. This leads to the following optimization problem:

$$\alpha\tilde{\mathbf{M}}\alpha^T \to \min \quad \text{subject to} \quad \sum_{i=1}^{m} \alpha_i = 1 \quad \text{and} \quad 0 \leqslant \alpha_i \leqslant 1, i = 1, \ldots, m. \tag{16}$$

Consequently, in order to find the second moving line, we need solve at most $3\bar{\mu}+4$ quadratic programming problems with linear constraints.

Remark 1. If $\bar{\mu}=\mu$, we can set

$$\mathbf{q}(t)=\alpha_2\mathbf{x}_2+\ldots+\alpha_l\mathbf{x}_l, \tag{17}$$

where $x_2,\ldots,\mathbf{x}_l$ are generalized eigenvectors defined in (12), and α_i, $i=2,\ldots,l$ are the coefficients. Here we choose $l\geqslant 2$ such that $\lambda_l\leqslant 2\sqrt{\lambda_1}$. The coefficients can be computed by solving a quadratic programming problem.

Example 1. Given a rational curve $\mathbf{P}(t)=(a(t),b(t),c(t))$ of degree 12:

$$\begin{aligned}
a(t) &= 654t^{12}-5904t^{11}+20592t^{10}-38720t^9+63360t^8-126720t^7+177408t^6\\
&\quad -101376t^5+24576t-4096,\\
b(t) &= -173t^{12}+4752t^{11}-50688t^{10}+264000t^9-760320t^8+1241856t^7\\
&\quad -1005312t^6+760320t^4-675840t^3+270336t^2-49152t+4096,\\
c(t) &= 189t^{12}+660t^{11}-14916t^{10}+45760t^9+47520t^8-570240t^7+1478400t^6\\
&\quad -2027520t^5+1647360t^4-788480t^3+202752t^2-24576t+8192.
\end{aligned}$$

Set $\mu=2,\bar{\mu}=3$. With the method presented in the previous sub-subsection, the approximate μ-bases are computed as

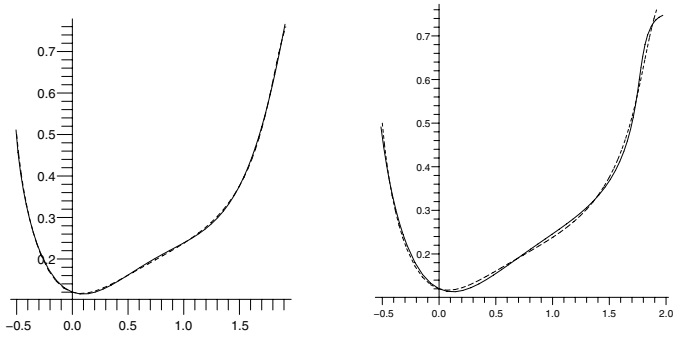
$$\begin{aligned}
\mathbf{p} &= (-0.05102032592B_0^2(t)-0.4807954605B_1^2(t)-1.038288624B_2^2(t),\\
&\quad -0.06065488069B_0^2(t)-0.1692187275B_1^2(t)-1.694077190B_2^2(t),\\
&\quad 0.005200267248B_0^2(t)-0.2832738286B_1^2(t)+3.279553214B_2^2(t)),\\
\mathbf{q} &= (-0.04159716413B_0^3(t)+0.9016040373B_1^3(t)+1.379605454B_2^3(t)\\
&\quad -1.727525622B_3^2(t),\ 0.5204348143B_0^3(t)+0.6001839987B_1^3(t)\\
&\quad +1.997824400B_2^3(t)-0.1722884058B_3^3(t),\ -0.2875610577B_0^3(t)\\
&\quad +1.146012829B_1^3(t)-4.681463727B_2^3(t)+3.431341524B_3^3(t)).
\end{aligned}$$

As a comparison, the exact μ-basis computed by the algorithm in [3] consists of two moving lines of degree six, and the coefficients in the μ-basis are integers with approximately forty digits.

3.3 Applications

We present two applications of approximate μ-bases of rational curves to degree reduction and to approximate implicitization, respectively.

Degree reduction. Based on the approximate μ-basis, a degree reduced rational curve $\bar{\mathbf{P}}(t)$ can be obtained directly from the exterior product of $\mathbf{p}(t)$ and $\mathbf{q}(t)$. Assume the error between the original curve and the degree reduced curve is measured by



a. Approximate μ-basis b. Eck's method

Fig. 1. Degree reduction without constraints

$$e(\mathbf{P}, \bar{\mathbf{P}}) := \sqrt{\int_0^1 \left(\left(\frac{a(t)}{c(t)} - \frac{\tilde{a}(t)}{\tilde{c}(t)} \right)^2 + \left(\frac{b(t)}{c(t)} - \frac{\tilde{b}(t)}{\tilde{c}(t)} \right)^2 \right) \mathrm{d}t}.$$

For the curve in Example 1, the approximation error is 0.00332. Figure 1.a illustrates the approximation result, where the original curve is dashed, and the degree reduced curve is solid.

As a comparison, if we use Eck's method [13] to reduce the same degree of the curve in Example 1, the degree reduction error is 0.0114. See Figure 1.b for an illustration.

In some cases, boundary conditions [13] are required. In order to satisfy them, we require that $\mathbf{p}(t)$ respects the conditions

$$\frac{\mathrm{d}^i}{\mathrm{d}t^i}(\mathbf{P}(t) \cdot \mathbf{p}(t))|_{t=0} = 0, \quad \frac{\mathrm{d}^i}{\mathrm{d}t^i}(\mathbf{P}(t) \cdot \mathbf{p}(t))|_{t=1} = 0, \quad i = 0, 1, \ldots, k. \tag{18}$$

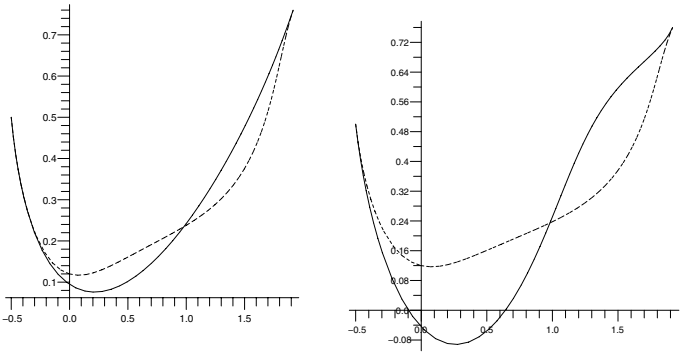
The conditions (18) can be written in matrix form $\mathbf{Q}\mathbf{x}^T = 0$, where $\mathbf{Q}$ is a matrix of order $2(k+1) \times 3(\mu+1)$. Hence $\mathbf{p}(t)$ is the solution of the following optimization problem:

$$\mathbf{x}\mathbf{M}\mathbf{x}^T \to \min \quad \text{subject to} \quad \mathbf{x}\mathbf{N}\mathbf{x}^T = 1 \quad \text{and} \quad \mathbf{Q}\mathbf{x}^T = 0. \tag{19}$$

In order to find $\mathbf{q}(t)$, we add $\mathbf{Q}\mathbf{x}^T = 0$ to (15).

Example 2. We continue the previous example. If we impose C^1 end-points interpolation conditions, and — in order to simplify the computation — set $\bar{\mu} = \mu = 2$, then

$$\begin{aligned} \mathbf{p} = (&0.7771853455(1-t)^2 + 2.355233260t(1-t) + 0.8057279534t^2, \\ &- 0.6976122684(1-t)^2 - 0.7195012836t(1-t) + 0.4112876045t^2, \\ &0.7373988081(1-t)^2 - 2.767665960t(1-t) - 1.856287881t^2), \end{aligned}$$



a. Approximate μ-bases b. Eck's method

Fig. 2. Degree reduction with C^1 constraints

$$\begin{aligned}\mathbf{q} = (&1.524480118(1-t)^2 + 1.879443954t(1-t) - 0.5076826073t^2,\\ &-0.6343217809(1-t)^2 + 1.998870979t(1-t) + 0.5295267213t^2,\\ &1.079400949(1-t)^2 - 5.200881704t(1-t) + 0.5705104408t^2).\end{aligned}$$

The error is 0.0525. If we use Eck's method to obtain an end-point C^1 interpolation reduction, the approximate error is 0.156. Figure 2.a and Figure 2.b depict the degree-reduced curves.

A more detailed comparison with other techniques for degree reduction may be a subject for further research. Unlike most existing techniques, our method can handle rational curves, and it generates a truly rational curve.

Approximate implicitization. By computing the resultant of $p = \mathbf{p}(t) \cdot (x, y, 1)$ and $q = \mathbf{q}(t) \cdot (x, y, 1)$ with respect to t, we obtain the approximate implicit equation of the original curve. Note that the curve defined by the implicit equation has — at the same time — a rational parameterization.

Example 3. An approximate implicit equation of the curve in Example 1 is

$$\begin{aligned}F(x,y) := &\,0.3790735866\,x^5 - 0.1877970243\,x^4y - 0.7592764650\,x^3y^2\\ &+ 0.038491110\,x^2y^3 + 0.2978093541\,xy^4 - 0.3360449642\,y^5\\ &- 0.0124850348\,x^4 + 1.716117584\,x^3y + 1.20600026\,x^2y^2\\ &- 0.740052063\,xy^3 + 3.030997755\,y^4 - 1.929933707\,x^3\\ &- 4.54990339\,x^2y + 0.903009110\,xy^2 - 9.592329447\,y^3\\ &+ 3.245601207\,x^2 - 0.76223425\,xy + 13.59527146\,y^2\\ &- 0.40069231\,x - 7.860226927\,y + 0.769956394 = 0.\end{aligned}$$

4 Approximate μ-Bases of Rational Surfaces

We generalize the results for approximate μ-bases of rational curves to rational surfaces. Since the discussions are similar to those for rational curves, we just outline the main results.

4.1 Definition and Computation

Consider a rational parametric surface of bi-degree (m, n) in homogeneous form,

$$\mathbf{P}(s,t) = (a(s,t), b(s,t), c(s,t), d(s,t)) := \sum_{i=0}^{m}\sum_{j=0}^{n} \mathbf{P}_{ij}\omega_{ij} B_i^m(s) B_j^n(t), \qquad (20)$$

where $\mathbf{P}_{ij} = (x_{ij}, y_{ij}, z_{ij}, 1)$ and ω_{ij}, $i = 0, 1, \ldots, m$, $j = 0, 1, \ldots, n$ are control points and their corresponding weights respectively. An *approximate moving plane* of $\mathbf{P}(s,t)$ is a moving plane $A(s,t)x + B(s,t)y + C(s,t)z + D(s,t) = 0$ which minimizes

$$\int_0^1 \int_0^1 (A(s,t)a(s,t) + B(s,t)b(s,t) + C(s,t)c(s,t) + D(s,t)d(s,t))\,\mathrm{d}s\,\mathrm{d}t \qquad (21)$$

subject to the normalization condition

$$\int_0^1 \int_0^1 (A(s,t)^2 + B(s,t)^2 + C(s,t)^2 + D(s,t)^2)\,\mathrm{d}s\,\mathrm{d}t = 1. \qquad (22)$$

An *approximate μ-basis* of $\mathbf{P}(s,t)$ consists of three approximate moving planes

$$\begin{aligned}
\mathbf{p}(s,t) &= (p_1(s,t), p_2(s,t), p_3(s,t), p_4(s,t)),\\
\mathbf{q}(s,t) &= (q_1(s,t), q_2(s,t), q_3(s,t), q_4(s,t)),\\
\mathbf{r}(s,t) &= (r_1(s,t), r_2(s,t), r_3(s,t), r_4(s,t)),
\end{aligned}$$

such that $[\mathbf{p}(s,t), \mathbf{q}(s,t), \mathbf{r}(s,t)] \neq 0$ approximates $\mathbf{P}(s,t)$ with respect to some criteria.

We represent the three moving planes in Bernstein-Bézier form,

$$\begin{pmatrix} \mathbf{p}(s,t) \\ \mathbf{q}(s,t) \\ \mathbf{r}(s,t) \end{pmatrix} = \sum_{i=0}^{m_0}\sum_{j=0}^{n_0} \begin{pmatrix} \mathbf{p}_{ij} \\ \mathbf{q}_{ij} \\ \mathbf{r}_{ij} \end{pmatrix} B_i^m(s)\, B_j^n(t) \qquad (23)$$

with control points $\mathbf{p}_{ij} = (p_{ij1}, p_{ij2}, p_{ij3}, p_{ij4})$, etc. While each of the three moving planes could have different degrees, we choose all of them to be equal to m_0, n_0.

Similar to the curve case, the approximate moving planes can be obtained by solving the generalized eigenvalue problems

$$\det(\mathbf{M} - \lambda\mathbf{N}) = 0, \quad (\mathbf{M} - \lambda\mathbf{N})\mathbf{x}^T = 0, \qquad (24)$$

where both $\mathbf{M}$ and $\mathbf{N}$ are semi-positive definitive matrices of order $4(m_0+1)(n_0+1)$. It follows that $\det(\mathbf{M} - \lambda\mathbf{N})$ is a polynomial of degree $\gamma = 3(m_0+1)(n_0+1)$

in λ. Assume the zeros of $\det(\mathbf{M}-\lambda\mathbf{N})$ are $\lambda_1 \leqslant \lambda_2 \leqslant \cdots \leqslant \lambda_\gamma$, and their corresponding generalized eigenvectors are $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_\gamma$. Then we can take $\mathbf{y}_1$, $\mathbf{y}_2$, and $\mathbf{y}_3$ to be coefficients of $\mathbf{p}(s,t)$, $\mathbf{q}(s,t)$, and $\mathbf{r}(s,t)$, respectively. But if we expect $[\mathbf{p}(s,t), \mathbf{q}(s,t), \mathbf{r}(s,t)]$ to represent a rational surface patch over $[0,1]^2$, then for any $(s,t) \in [0,1]^2$,

$$\begin{vmatrix} p_1 & p_2 & p_3 \\ q_1 & q_2 & q_3 \\ r_1 & r_2 & r_3 \end{vmatrix} \neq 0. \tag{25}$$

must hold. In order to satisfy this condition, we only select $\mathbf{y}_1$ and $\mathbf{y}_2$ as the coefficients of $\mathbf{p}$ and $\mathbf{q}$ respectively. The coefficient vector $\mathbf{z}$ of the element $\mathbf{r}$ is set to be the linear combination of $\mathbf{y}_3, \ldots, \mathbf{y}_l$ for some $l < \gamma$: $\mathbf{z} = \sum_{i=3}^{l} \alpha_i \mathbf{y}_i$. The coefficients will be determined by requiring (25) holds and the angles between $\mathbf{r}$ and $\mathbf{p}$ (and $\mathbf{q}$) are not too small. Then $\mathbf{r}$ is the solution of the following problem

$$\mathbf{z}\bar{\mathbf{M}}\mathbf{z}^T \to \min \quad \text{subject to} \quad \mathbf{z}\mathbf{N}\mathbf{z}^T = 1 \quad \text{and} \quad \mathbf{L}\mathbf{z}^T \leqslant -\mathbf{E}. \tag{26}$$

where $\mathbf{L}$ is a matrix of size $(3m_0+1)(3n_0+1) \times 4(m_0+1)(n_0+1)$. This problem can be solved in a similar way as in the curve case.

4.2 Examples and Applications

We provides two examples to illustrate some applications of approximate μ-bases of rational surfaces — approximate implicitization and degree reduction.

Example 4. Given a bicubic surface defined by:

$$\begin{aligned}
a(s,t) &= \frac{1}{4}(3s^3t^3 - 6s^2t^3 + 3st^3 - 9s^3t^2 + 18s^2t^2 - 9st^2 \\
&\quad + 9s^3t - 18s^2t + 9st - 3s^3 + 6s^2 + 9s), \\
b(s,t) &= -3s^3t^3 + 3s^3t^2 + 3s^2t^3 - 3s^2t^2 + 3t, \\
c(s,t) &= \frac{1}{2}(3s^3t^3 - 6s^3t^2 - 6s^2t^3 + 9s^2t^2 + 3st^3 + 3s^3 + 2t^3 - 6s^2 - 6\,t^2 + 4), \\
d(s,t) &= \frac{1}{5}(-s^3t^3 + 3s^3t^2 + 3s^2t^3 - 3s^3t - 9s^2t^2 - 3st^3 + s^3 + 9s^2t + 9st^2 \\
&\quad + t^3 - 3s^2 - 9st - 3t^2 + 3s + 3t + 4).
\end{aligned}$$

A linear approximate μ-basis can be computed as

$$\begin{aligned}
\mathbf{p} = (&-0.3179763603 + 0.01129769996s - 0.005706462242t, \\
&-0.05643886419 - 0.02026545412s + 0.00006577262458t, \\
&0.03369450553 - 0.01391833112s - 0.001238259243t, \\
&-0.08609427199 + 1.0s + 0.2529721651t), \\
\mathbf{q} = (&-0.08463698360 - 0.03287295861s + 0.00661036891t, \\
&0.4542005463 + 0.01115597009s - 0.02382379534t, \\
&-0.03488009368 + 0.00195597407s - 0.00519198472t, \\
&0.04412801971 + 0.3068345570s - 1.340134559t),
\end{aligned}$$

$$\begin{aligned}\mathbf{r} = (&-10.22862360 + 1.851570346s + 17.56377500t,\\ &15.86139849 - 21.35392833s + 0.2563277660t,\\ &5.474325695 + 2.113435284s - 0.650356134t,\\ &-14.90107308 + 36.65715509s - 33.69498870t).\end{aligned}$$

A cubic rational parametric surface can be obtained from $[\mathbf{p}, \mathbf{q}, \mathbf{r}]$. The approximation error between the original surface and the new surface is 0.0264.

By eliminating s, t from $\mathbf{p} \cdot (x, y, z, 1) = \mathbf{q} \cdot (x, y, z, 1) = \mathbf{r} \cdot (x, y, z, 1) = 0$, one obtains an approximate cubic implicit equation for the given surface,

$$\begin{aligned}F(x, y, z) := {} & 0.2063266181\, x^3 - 0.2187304282\, x^2 y - 0.04778543831\, x^2 z\\ & + 0.3449682259\, xy^2 + 0.1115861554\, xzy - 0.01039875210\, xz^2\\ & + 0.03791855132\, y^3 + 0.004111063445\, zy^2 - 0.005055353827\, z^2 y\\ & + 0.001142730024\, z^3 - 1.317908681\, x^2 + 1.286241307\, xy - 0.1681655\, xz\\ & - 1.252094004\, y^2 - 0.3605440520\, zy + 0.1401044281\, z^2 - 2.990107224\, x\\ & - 3.033340596\, y - 8.359734692\, z + 19.68286052 = 0.\end{aligned}$$

Note that the exact implicit degree of the surface is 18.

Example 5. We consider a given surface of bi-degree $(5, 5)$,

$$\begin{aligned}a = {} & -25/2\, s^5t^5 + 50\, s^5t^4 + 50\, s^4t^5 - 75\, s^5t^3 - 200\, s^4t^4 - 75\, s^3t^5 + 50\, s^5t^2\\ & + 300\, s^4t^3 + 300\, s^3t^4 + 50\, s^2t^5 - 25/2\, s^5t - 200\, s^4t^2 - 450\, s^3t^3 - 200\, s^2t^4\\ & - 25/2\, st^5 + 50\, s^4t + 300\, s^3t^2 + 300\, s^2t^3 + 50\, st^4 - 75\, s^3t - 200\, s^2t^2 - 75\, st^3\\ & + 50\, s^2t + 50\, st^2 - 25/2\, st + 5\, s,\\ b = {} & 25/2\, s^5t^5 - 50\, s^5t^4 - 50\, s^4t^5 + 75\, s^5t^3 + 200\, s^4t^4 + 75\, s^3t^5 - 50\, s^5t^2 - 300\, s^4t^3\\ & - 300\, s^3t^4 - 50\, s^2t^5 + 25/2\, s^5t + 200\, s^4t^2 + 450\, s^3t^3 + 200\, s^2t^4 + 25/2\, st^5\\ & - 50\, s^4t - 300\, s^3t^2 - 300\, s^2t^3 - 50\, st^4 + 75\, s^3t + 200\, s^2t^2 + 75\, st^3\\ & - 50\, s^2t - 50\, st^2 + \frac{25}{2}\, st + 5\, t,\\ c = {} & 50\, s^5t^5 - 100\, s^5t^4 - 150\, s^4t^5 + 50\, s^5t^3 + 300\, s^4t^4 + 150\, s^3t^5 - 150\, s^4t^3\\ & - 300\, s^3t^4 - 50\, s^2t^5 + 150\, s^3t^3 + 100\, s^2t^4 - 50\, s^2t^3 - 5\, s^4 - 5\, t^4 + 10\, s^3\\ & + 10\, t^3 - 10\, s^2 - 10\, t^2 + 5\, s + 5\, t,\\ d = {} & -1/6\, s^5t^5 + 5/6\, s^5t^4 + 5/6\, s^4t^5 - 5/3\, s^5t^3 - 25/6\, s^4t^4 - 5/3\, s^3t^5 + 5/3\, s^5t^2\\ & + 25/3\, s^4t^3 + 25/3\, s^3t^4 + 5/3\, s^2t^5 - 5/6\, s^5t - 25/3\, s^4t^2 - 50/3\, s^3t^3\\ & - 25/3\, s^2t^4 - 5/6\, st^5 + 1/6\, s^5 + 25/6\, s^4t + 50/3\, s^3t^2 + 50/3\, s^2t^3 + 25/6\, st^4\\ & + 1/6\, t^5 - 5/6\, s^4 - 25/3\, s^3t - 50/3\, s^2t^2 - 25/3\, st^3 - 5/6\, t^4 + 5/3\, s^3\\ & + 25/3\, s^2t + 25/3\, st^2 + 5/3\, t^3 - 5/3\, s^2 - 25/6\, st - 5/3\, t^2 + 5/6\, s + 5/6\, t + 5/6.\end{aligned}$$

An approximate μ-basis of bi-degree (1,1) is computed as

$$\begin{aligned}\mathbf{p} = (&0.2085266555 - 0.007372293503\, s - 0.1428724898\, t + 0.005537995913\, st,\\ &0.1808522216 + 0.2050867177\, s - 0.009074759098\, t + 0.007914352425\, st,\end{aligned}$$

$$
\begin{aligned}
&- 0.01878293690 + 0.01746191813\,s + 0.01597674812\,t - 0.01869048639\,st,\\
&- 0.006408507364 - 1.0\,s - 0.8531656420\,t - 0.3816705478\,st),\\
\mathbf{q} = (&- 0.08053392989 - 0.05754492900\,s + 0.005132654152\,t + 0.006652036180\,st,\\
&0.08191584666 - 0.01108549693\,s - 0.05549170597\,t + 0.003943449557\,st,\\
&- 0.1025013651 + 0.03199060344\,s + 0.02845654330\,t - 0.03187909295\,st,\\
&0.02672843678 + 0.6709819071\,s - 0.1475467029\,t - 0.002951798803\,st),\\
\mathbf{r} = (&0.1162420910 - 0.2753649567\,s + 0.09208042082\,t - 0.07226311792\,st,\\
&- 0.02104547875 + 0.2915682775\,s - 0.3705920517\,t + 0.1046918855\,st,\\
&- 0.4890160153 + 0.1394880492\,s - 0.1022402645\,t + 0.03867595945\,st,\\
&0.1307117232 + 0.6920411693\,s + 1.964792505\,t - 2.063871335\,st).
\end{aligned}
$$

The bicubic parametric surface $[\mathbf{p}, \mathbf{q}, \mathbf{r}]$ can serve as a degree-reduced surface. The approximation error is 0.0436. An approximate implicit equation of degree six can also be obtained by eliminating s, t from p, q, r. Note that the exact algebraic degree of the surface is 50.

5 Conclusion and Future Work

In this paper, approximate μ-bases of rational curves and surfaces are studied. Algorithms are provided to compute the approximate μ-bases, which amount to solve generalized eigenvalue problems and some quadratic programming problems. Applications of approximate μ-bases in degree reduction and approximate implicitization are explored. Examples seem to suggest that the techniques presented in this paper are competitive with other known methods, but this should be studied further.

In order to compute the approximate μ-bases, quadratic programming problems have to be solved. In the future, we will discuss how to define and compute approximate μ-bases in a more general and efficient approach. Other applications of approximate μ-bases will be explored as well.

Acknowledgments

The authors have been supported by the Outstanding Youth Grant of NSF of China (No.60225002), NSF of China (No. 60533060 and 60473132), a National Key Basic Research Project of China (No. 2004CB318000), and by the Special Research Programme SFB F013 "Numerical and Symbolic Scientific Computing" at Linz, Austria, which has been established by the Austrian Science Fund (FWF).

References

1. Falai Chen, Approximate implicit representation of rational curves, *Chinese J. of Computers (in chinese*), Vol. 21, 855–959, 1998.
2. Falai Chen, Jianmin Zheng, and Thomas W. Sederberg, The μ-basis of a rational ruled surface. *Aided Geom. Design*, Vol.18, 61–72, 2001.

3. Falai Chen and Wenping Wang, The μ-basis of a rational curve — properties and computation. *Graphical Models*, 64, 268–381, 2003.
4. Falai Chen and Thomas W. Sederberg, A new implicit representation of a planar rational curve with high order of singularity, Comput. *Aided Geom. Design*, vol.19, 151–167, 2002.
5. Falai Chen, Reparameterization of a rational ruled surface by μ-basis, *Aided Geom. Design*, Vol.20, 11–17, 2003.
6. Falai Chen, and Wenping Wang, Revisiting the μ-basis of a rational ruled surface, *J. Symbolic Computation*, Vol.36, No.5, 699–716, 2003.
7. Falai Chen, David Cox, and Yang Liu, The μ-basis and implicitization of a rational parametric surface, *J. Symbolic Computation*, Vol.39, 689–706, 2005.
8. Falai Chen and Wenping Wang, Computing the singular points of a planar rational curve using the μ-bases, preprint, 2006.
9. David Cox, Thomas W. Sederberg, and Falai Chen, The moving line ideal basis of planar rational curves. *Comput. Aided Geom. Des.*, Vol.15, 803–827, 1998.
10. Jiansong Deng, Falai Chen, and Liyong Shen, Computing μ-bases of rational curves and surfaces using polynomial matrix factorization, *Proceedings of the ISSAC'2005*, Manuel Kauers ed., ACM Press, USA, 132–139, 2005.7.
11. T. Dokken, and J.B. Thomassen, Overview of approximate implicitization, in *Topics in Algebraic Geometry and Geometric Modeling*, AMS Cont. Math., 169–184, 2003.
12. Matthias Eck, Degree reduction of Bézier curves, *Comput. Aided Geom. Des.*, Vol.10, 237–251,1993.
13. Matthias Eck, Least squares degree reduction of Bézier curves, *Computer-Aided Design*, Vol.27, 11, 845–851, 1995.
14. W. Woresy, Degree reduction of Bézier curves, *Computer-Aided Design*, Vol.20, 7, 398–405, 1988.
15. S. E. Weinstein and Y. Xu, Degree reduction of Bézier curves by approximation and interpolation, in: *Approximation Theory*, Anastassiou, G.A., ed., Dekker, New York, 503–512, 1992.
16. Thomas W. Sederberg and Falai Chen, Implicitization using moving curves and surfaces. *Proceedings of Siggraph*, 301–308, 1995.
17. Thomas W. Sederberg, J. Zheng, K. Klimaszewski, and T. Dokken, Approximate implicitization using monoid curves and surfaces, *Graphical Models and Images Processing*, 61(4), 177–198, 1999.
18. E. Wurm and B. Jüttler, Approximate implicitization via curve fitting, *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry Processing*, L. Kobbelt, P. Schöder, and H. Hoppe (eds.), 240–247, 2003.
19. Jianmin Zheng and Thomas W. Sederberg, A direct approach to computing the μ-basis of planar rational curves, *J. Symbolic Computation*, 31, 619–629, 2001.

Inverse Adaptation of Hex-dominant Mesh for Large Deformation Finite Element Analysis

Arbtip Dheeravongkit and Kenji Shimada

Carnegie Mellon University, Pittsburgh, PA, USA

Abstract. In the finite element analysis of metal forming processes, many mesh elements are usually deformed severely in the later stage of the analysis because of the large deformation of the geometry. Such highly distorted elements are undesirable in finite element analysis because they introduce error into the analysis results and, in the worst case, inverted elements can cause the analysis to terminate prematurely. This paper proposes an inverse adaptation method that reduces or eliminates the number of inverted mesh elements created in the later stage of finite element analysis, thereby lessening the chance of early termination and improving the accuracy of the analysis results. By this method, a simple uniform mesh is created initially, and a pre-analysis is run in order to observe the deformation behavior of the elements. Next, an input hex-dominant mesh is generated in which each element is "inversely adapted," or pre-deformed in such a way that it has approximately the opposite shape of the final shape that normal analysis would deform it into. Thus, when finite element analysis is performed, the analysis starts with an input mesh of inversely adapted elements whose shapes are not ideal. As the analysis continues, the element shape quality improves to almost ideal and then, toward the final stage of analysis, degrades again, but much less than would be the case without inverse adaptation. This method permits analysis to run to the end, or to a further stage, with few or no inverted elements. Besides pre-skewing the element shape, the proposed method is also capable of controlling the element size according to the equivalent plastic strain information collected from the pre-analysis. The method can be repeated iteratively until reaching the final stage of deformation.

1 Introduction

The simulation of metal forming processes is one of the most common applications of large deformation finite element analysis. In metal forming, the blank is stamped or punched and thus undergoes a drastic change in shape. In simulation of this process, as the blank is reshaped the mesh elements of the blank are deformed, and element shape quality decays as analysis continues. At later stages of the analysis, elements can become severely distorted or inverted causing inaccurate results, slow convergence, and premature analysis termination. An example of such large deformation analysis is illustrated in Fig.1. This is a three dimensional forming example, containing a punch that deforms a blank

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 189–206, 2006.
© Springer-Verlag Berlin Heidelberg 2006

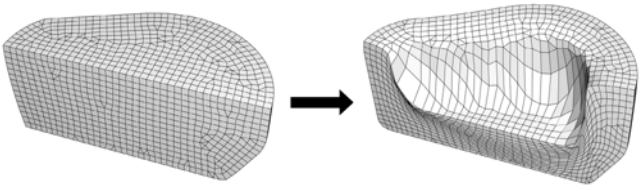


Fig. 1. An example of large deformation finite element analysis

into a geometry with high-curvature corners. In the later stage of analysis several elements become severely distorted and inverted.

In light of this problem, this paper proposes a new "inverse adaptation" approach to reduce the number of ill-shaped elements at the end of the analysis. The term "inverse adaptation" is used to illustrate the concept of this method, in which we first predict the way each mesh element will be deformed during simulation, and then create a new input mesh by pre-deforming the elements so that they have approximately opposite shapes of those predicted. Unlike conventional methods which start with a mesh of ideal elements that become severely distorted over the course of analysis, the proposed method starts with a mesh of slightly distorted elements whose quality reaches ideal as analysis continues and then degrades again in the final stage. This new method better distributes error across the life of the analysis which improves the accuracy of the results, reduces the need for remeshing and thereby improves computational cost, and lengthens the life of the analysis by reducing the risk of early termination.

Similar methods for two-dimensional quadrilateral mesh and three-dimensional tetrahedral mesh have been proposed earlier, and it has been proved that inverse adaptation can successfully extend the life of the analysis, as well as reduce the number of ill-shaped elements at the later stage [5], [6], [7]. This paper proposes a method of inverse adaptation based on a similar but extended concept to generate a hex-dominant mesh. The major difference between the adaptation method presented in this paper and the previous ones is the node mapping technique. In addition, due to the complexity of the analysis, the three-dimensional adaptation is applied iteratively in order to reach the final stage of deformation, while this is not necessary in two-dimensional pre-deformation. Details can be found in Sect.3.

Currently, there are two techniques commonly used to address the problem of severely distorted elements in large deformation finite element analysis: adaptive remeshing, and Arbitrary Lagrangian-Eulerian (ALE). Adaptive remeshing replaces a severely distorted mesh with a well-shaped mesh every certain number of steps in the analysis. In the analysis of a complicated geometry, however, frequent remeshing is necessary, which raises the computational cost significantly. The other technique, ALE, is a type of analysis reference frame developed to reduce the repetition of complete remeshing. In ALE, the mesh is not connected to the material; therefore, mesh elements do not suffer the severe deformation that the material undergoes. However, the complexity of this method, especially in solving the control equations and variable mapping processes, is a drawback in contrast to a pure Lagrangian analysis reference frame.

The purpose of the inverse adaptation method proposed here is not to be a replacement of the two existing techniques. Adaptive remeshing may still be required on complex geometries even when using the proposed inverse adaptation method, but it does not become necessary until later in the analysis; furthermore, the repetition of adaptive remeshing required during simulation is significantly reduced, improving computational cost and reducing computational errors. Moreover, because all the work is done in Lagrangian analysis, we can avoid the complications of equation solving and variable mapping of ALE analysis. If desired, the proposed inverse adaptation method can also be used for analysis performed by the ALE method.

The remainder of the paper is organized as follows: Section 2 discusses the detail of the two existing methods used in large deformation finite element analysis: adaptive remeshing and ALE. Section 3 discusses the proposed inverse adaptation method for a hex-dominant mesh. Results are shown in Sect.4, and Sect.5 is the Conclusion.

2 Previous Methods

2.1 Adaptive Remeshing

Adaptive remeshing is a technique to replace an existing mesh with a new mesh when the element quality of the existing mesh is no longer sufficient for the analysis due to severe distortion. A typical remeshing process includes the following procedures: determining the error estimator to define the remeshing criteria, generate a new mesh, and transfer the history-dependent variables from the old mesh to the new mesh [26], [12]. Most common remeshing criteria are mesh discretization error based on strain error in L_2 norm [9], [10], [12], [18], [26], [30], [31], element distortion errors based on element shape quality [9], [10], [11], [14], [15], and geometric interference errors [9], [14], [15]. Most remeshing algorithms are developed to apply automatic mesh generation techniques to completely remesh the entire domain of the workpiece [4], [14], [15]. And because the newly created mesh may not necessarily have the same topology as the original mesh, and the number of nodes and elements of the new mesh may differ from the original mesh, the state variables and history-dependent variables must also be transferred from the original to the new mesh. State variables include nodal displacements and variables of the contact algorithm. History dependent variables include the stress tensor, strain tensor, plastic strain tensor, etc.

The remeshing process is usually repeated many times during the analysis in order to replace the highly distorted mesh by a newly better quality mesh, which consequently reduces the discretization errors. Nevertheless, there are several important requirements for a good adaptive remeshing technique, namely, edge detection, contact/penetration checking, volume difference checking, and parameter adaptation. The contact/penetration checking is a very time consuming operation, and numerical methods and heuristic assumptions are usually applied; this can lead to errors in the analysis result. Volume difference checking

is also crucial for some cases, because multiple remeshings can cause a loss of volume which is not acceptable for industrial applications [13].

Additionally, when the entire domain needs to be remeshed and frequent remeshing is necessary, the computational cost increases significantly. Most remeshing algorithms are developed to apply automatic mesh generation techniques to completely remesh the entire domain of the workpiece [4], [14], [15], and typical metal forming simulations need between 20 and 100 complete remeshings during finite element analysis [13]. To reduce the computational cost, several methods are developed using error estimators and the h-adaptive process to apply remeshing locally to only a limited number of elements [26], [30], [31]. Nonetheless, the process to map the state variables from the old mesh to the new mesh in adaptive remeshing method usually induces significant loss of accuracy, especially when frequent remeshing is essential. Therefore, methods to reduce the number of complete remeshing during the simulation are worthwhile, because they would consequently bring to more accurate analysis result.

The inverse adaptation method presented in this paper delays the occurrence of ill-shaped elements and lets the analysis proceed either fully to the end of the deformation, or to a further step than without inverse adaptation where mesh quality becomes unacceptable and adaptive remeshing is required. As a result, inverse adaptation reduces the repetition of remeshing in the analysis, which saves computational time and eliminates some numerical error caused by remeshing.

2.2 Arbitrary Lagrangian Eulerian (ALE)

Arbitrary Lagrangian Eulerian is a type of analysis reference frame. It is largely used in fluid-structure simulation and in large deformation simulation in solid mechanics [2], [8], [20], [21], [22], [23]. This method combines the good features of two common types of reference frames in finite element analysis: Lagrangian and Eulerian.

Essentially, ALE is a Lagrangian analysis that takes advantage of the advection techniques of Eulerian analysis. In the ALE method, the mesh is neither connected to the material nor fixed to a spatial coordinate system. Rather, it is prescribed in an arbitrary manner. During the analysis, the mesh elements deform according to the Lagrangian method. However, instead of repositioning the nodes at their original positions and performing advection, as in the Eulerian method, the nodes are placed at other positions to obtain optimal mesh quality. The mesh nodes have velocity associated with them in order to obtain the updated mesh. Mesh velocity plays an important role in the ALE method, as it reduces the analysis error and prevents mesh distortion [23]. Another important characteristic of ALE is that it changes the locations of the nodes in the existing mesh instead of creating a completely new mesh, and it maintains the same (or similar) mesh topology throughout the analysis [2].

However, because of its complexity, the computational cost is much more expensive than using pure Lagrangian analysis. There are numerous variables that must be imputed to assure the accuracy of the controlling equations, namely, size, boundary consideration, and other material characteristics such as density,

viscosity, etc. It also involves the use of several controlling equations, all of which requires high computational time. Furthermore, in many cases the mesh suffers considerable distortion, and the same mesh topology cannot be maintained for the entire analysis. In such cases, complete adaptive remeshings are still required.

3 Inverse Adaptation of a Hex-dominant Mesh

3.1 Key Concept

An inverse adaptation method is proposed here to generate an input hex-dominant mesh for large deformation finite element analysis. Generally, in large deformation analysis the shape quality of most elements is degraded and becomes excessively distorted as analysis continues. Analysis often terminates early because elements become so severely distorted or even inverted, and multiple remeshings become necessary. To address this problem, we propose a method that predicts the way the mesh will be deformed during the analysis, and then pre-deforms the input mesh in the opposite manner; we refer to this as "inverse adaptation." The inversely adapted input mesh therefore contains elements with shapes approximately inverse of the shapes they should be deformed into in the general analysis. As the analysis is run on this new inversely adapted input, the mesh elements start out purposely distorted but still acceptable, then improve in shape until they reach optimal quality and then begin to degrade again later in the analysis. This tactic keeps mesh quality not optimal but within an acceptable range for a longer period of time than with a traditional method that starts with an optimal mesh and rapidly degrades to an unacceptable quality level. Consequently, using the proposed method allows the analysis to run longer before being terminated by poor element quality, and it reduces or eliminates the need for multiple remeshings of the deforming geometry. This idea is illustrated in Fig.2.

As shown in Fig.2, present methods of analysis start with a mesh of optimal quality ("Normal Input Mesh") but terminate, often early, when the element quality exceeds the bad quality tolerance. With inverse adaptation, the analysis starts with an inversely adapted input mesh whose quality is not optimal yet in an acceptable range. As the analysis continues, the mesh quality improves until optimal quality is reached, and then begins to degrade again. Accordingly, the analysis using an inversely adapted mesh could be ended early with better mesh quality (Case 1), or the life of the analysis could be extended, eventually reaching the bad quality tolerance but at a later point than otherwise (Case 2). This approach also incorporates the idea of averaging out the error over the course of the analysis, instead of accumulating error abruptly and forcing the termination of the analysis prematurely.

3.2 Terminology

The following terminology will be used in this paper:

Workpiece – The deformable object whose elements are severely distorted during the analysis.

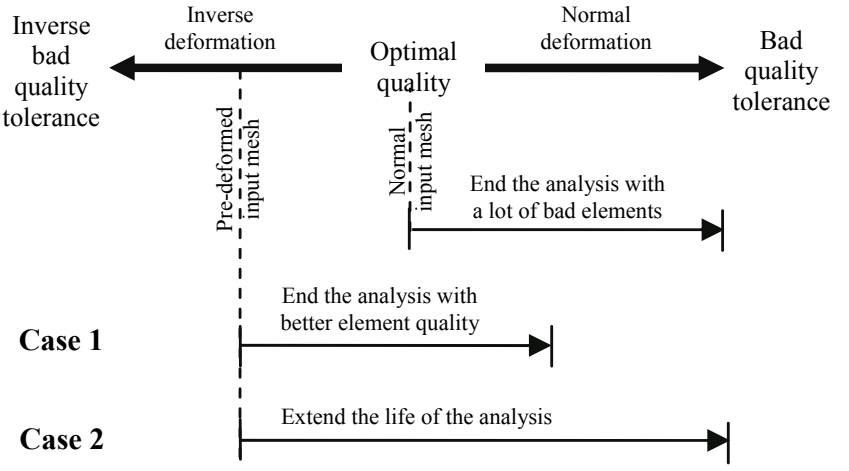


Fig. 2. Concept of the Inverse Adaptation Method

Original mesh – The simple uniform hex-dominant input mesh. It is used as the input mesh for the pre-analysis.

Inversely adapted mesh – The hex-dominant input mesh generated by the inverse adaptation method.

Deformed workpiece – The workpiece after some deformation in the pre-analysis.

Undeformed workpiece – The workpiece in the initial state before any deformation.

Deformed domain – The mesh domain of the deformed workpiece. Undeformed domain: The mesh domain of the undeformed workpiece.

3.3 Overview

The overview of the inverse adaptation method is illustrated in Fig.3. The inverse adaptation method can be summarized in four steps. In the first step, a pre-analysis is run on a simple uniform hex-dominant mesh. This input mesh is generated by the BubbleMesh method [27], [28], [29], where we first create an all-tetrahedral mesh and then convert it into a hex-dominant mesh. The purpose of pre-analysis is to predict the deformation behavior of the workpiece during the analysis. After some deformation is made and before the mesh elements become unacceptable due to severe distortion, the geometric and equivalent strain information of the deformed workpiece is collected and passed on to the BubbleMesh generator. In Step 2, an optimal mesh for the deformed workpiece is then created by the BubbleMesh generator. The new nodes are first created at optimal positions. Then in the same way as in the first step, these new nodes are connected to generate a tetrahedral mesh which is then converted into a hex-dominant mesh. In Step 3, the new node locations created in Step 2 are inversely mapped from the deformed domain to the undeformed domain using

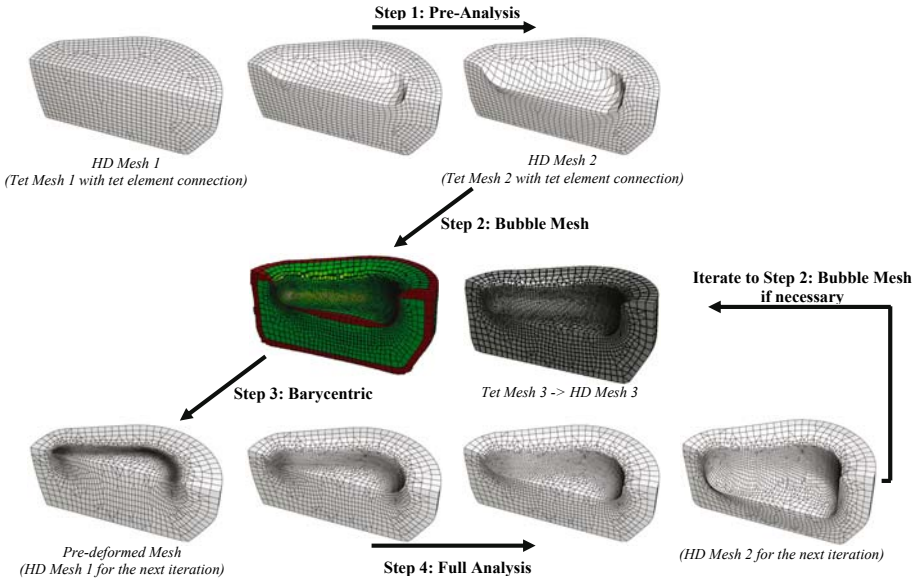


Fig. 3. Overview of the Inverse Adaptation Method

barycentric interpolation based on the all-tetrahedral meshes, and the mapped nodes are connected based on the hex-dominant mesh created in Step 2. The final mesh is the inversely adapted hex-dominant mesh. Finally, in Step 4, a full analysis is run, commencing with the inversely adapted hex-dominant mesh.

3.4 The Differences Between the Proposed Method and the Previous Versions

The framework of the proposed method is similar to the previous inverse adaptation method for two-dimensional meshes and three-dimensional tetrahedral meshes. The key difference between them is the node mapping process of Step 3. The 2-D inverse adaptation method uses inverse bilinear mapping to generate a quadrilateral mesh, and the 3-D inverse adaptation method for tetrahedral meshes uses barycentric interpolation to generate the tetrahedral mesh. Because the barycentric interpolation has proved to be a straightforward and easy-to-implement mapping algorithm for a 3-D problem, it is advantageous to use it in the current inverse adaptation method to generate a hex-dominant mesh. The barycentric interpolation has to be employed on an all-tetrahedral mesh domain however, while the hex-dominant mesh usually consists of three types of elements: tetrahedrons, hexahedrons and prisms. One solution is to subdivide the hexahedrons and prisms into tetrahedral elements, but we can get around this by taking advantage of the BubbleMesh generator to create the optimal mesh in Step 2. The BubbleMesh method must first generate an all-tetrahedral mesh before converting it into a hex-dominant mesh, and it keeps the nodes at the same

locations [27], [28]. Thus, the proposed method can easily employ the barycentric interpolation on the all-tetrahedral mesh that the BubbleMesh generator first created and run analysis on the final hex-dominant mesh.

Another point to be noted is the advantages of using a hex-dominant mesh over a tetrahedral mesh. Usually in a large deformation analysis, it is necessary to use linear hexahedral elements or quadratic tetrahedral elements. However, the disadvantage of using quadratic elements is that the number of degrees of freedom is higher which consequently results in very complex contact conditions and makes the analysis very expensive. Moreover, tetrahedral elements cause critical errors when they get distorted, while hexahedral elements still produce acceptable results even when they are distorted [13]. Therefore, it is very useful to extend the inverse adaptation method to the hex-dominant mesh.

In addition, because the deformation behavior of the three-dimensional problem is usually more complicated than the two-dimensional problem, there is a greater possibility that one iteration of inverse adaptation may not be enough for adequate results. In this paper, we show that the inverse adaptation method can be repeated iteratively to improve the analysis results progressively and extend the life of the analysis to the final stage of deformation.

3.5 Detailed Algorithm

This section explains each step of the inverse adaptation method in detail.

Step 1: Pre-analysis. The pre-analysis is carried out on a simple uniform hex-dominant input mesh. The primary goal of the pre-analysis is to predict deformation behavior and collect necessary information, e.g. geometric information of the deformed workpiece and plastic equivalent strain. These data are collected at an intermediate step before severe distortion occurs. The equivalent plastic strain is a scalar variable that represents the material's inelastic deformation, and is collected to be used to specify mesh sizes in the proposed method. Therefore, in this pre-analysis step, the equivalent plastic strain value at each input mesh node is collected.

The input mesh of this step is a uniform hex-dominant mesh which is created by the BubbleMesh generator. The BubbleMesh algorithm will be discussed briefly in step 2. In this step we use it first to create a uniform tetrahedral mesh and then convert to a hex-dominant (HD) mesh; these will be referred to as $TetMesh1$ and $HDMesh1$ respectively (see Fig.3). Pre-analysis is run on the $HDMesh1$ until it reaches the state before any element becomes inverted. The resulting mesh is a deformed mesh called $HDMesh2$. By reconnecting the nodes of $HDMesh2$ with the element connection of $TetMesh1$, we create $TetMesh2$. $TetMesh1$ and $TetMesh2$ will be used later in the node mapping step.

Step 2: BubbleMesh. In this step, the BubbleMesh method is used to pack rectangular solid bubbles in the deformed workpiece of Step 1 to create an optimal hex-dominant mesh. This optimal mesh is the desired mesh we want to achieve at the current time step, which is the time step where we collected the

information of the deformed workpiece in Step 1. Details of the BubbleMesh algorithm can be found in [27] and [28].

Element sizes of the mesh created on this step are determined using the equivalent plastic strain information collected from the pre-analysis. Smaller element sizes are specified at the regions that tend to experience more element distortion, where the equivalent plastic strain values are higher. To determine the mesh size, the first step is to specify desired minimum and maximum element sizes. These minimum and maximum sizes are user inputs and vary in different problems. The maximum element size l_{max} is then assumed for a point with minimum value of equivalent plastic strain ϵ_{min}, and the minimum element size l_{min} is assumed for a point with maximum value of equivalent plastic strain ϵ_{max}. However, because the change in element size should be more rapid than change in equivalent plastic strain the relation between the element sizes l and the value of equivalent plastic strain ϵ at a specific point is described by the following cubic function:

$$l(\epsilon) = \left[\frac{(\epsilon - \epsilon_{max})^3}{(\epsilon_{min} - \epsilon_{max})^3} \cdot (l_{max} - l_{min}) \right] + l_{min}. \tag{1}$$

In addition, to generate a more efficient mesh, the mesh directionality can also be controlled to be boundary-aligned. The mesh directionality at a specific point is obtained by determining the three principal vectors as following [27], [28]:

$u(x)$ is the unit normal vector at a point on the boundary face closest to point x.

$v(x)$ is calculated based on the boundary edge. Let x_e be a point on a boundary edge, at which the unit tangent vector $t_e(x_e)$ makes more than 45 degrees and less than 135 degrees with $u(x)$. Now let x_v be the point from all possible x_e that is closest to point x, then $v(x)$ can be calculated as:

$$v(x) = \frac{(u(x) \times (t_e(x_v)) \times u(x)}{\|(u(x) \times (t_e(x_v)) \times u(x)\|}. \tag{2}$$

$w(x)$ is simply obtained by the cross product of $u(x)$ and $v(x)$.

The mesh size and directionality of any point on the boundary and inside the geometric domain are stored on the background grid defined over the domain, where the grid size must be determined properly for each problem. Mesh size and directionality at any point on the grid can be represented by a 3×3 matrix $M = RS$, where rotational matrix R signifies directionality and scaling matrix S signifies the mesh size. For the internal point of a grid cell, the BubbleMesh generator calculates the values by linearly interpolating the values at the grid nodes. BubbleMesh generator packs rectangular solid cells on the domain and adjusting size and directionality as specified. In our case, the boundary and geometric domain are taken from the $TetMesh2$ instead of $HDMesh2$. This is because in the node mapping process with barycentric interpolation the nodes will be tested on the tetrahedral mesh. Therefore, cells should be packed on the boundary of the tetrahedral mesh instead of the hex-dominant mesh so that the boundary nodes will be mapped correctly. After cells are packed, then new nodes are placed at the centers of the cells and are connected by the advancing front method to generate a tetrahedral mesh ($TetMesh3$). Finally, the tetrahedral mesh is converted

into a hex-dominant mesh ($HDMesh3$) by merging some tetrahedrons to create hexahedrons and prisms. It should be noted that the nodal information is kept the same when converting $TetMesh3$ into $HDMesh3$.

Step 3: Barycentric interpolation. In this step, the new nodes created by BubbleMesh are mapped from the deformed domain onto the undeformed domain to generate an inversely adapted mesh using barycentric interpolation. In short, barycentric interpolation is a form of tetrahedral interpolation, and barycentric coordinates are the numbers corresponding to the weights placed at the vertices of a tetrahedron. These numbers can be used to determine the location of the center of mass of the tetrahedron corresponding to the weights put on its vertices [24].

The nodes to be inversely mapped are the nodes generated by the BubbleMesh generator from Step 2 (nodes of $HDMesh3$). We first compare the $HDMesh3$ with the deformed original mesh from Step 1 ($TetMesh2$). Note that the tetrahedral mesh is used for Mesh 2. We then locate, using the interpolation function, the position of each $HDMesh3$'s node on $TetMesh2$. Practically, for each $HDMesh3$'s node, we search on $TetMesh2$ for the element in which this node lies. Following are the barycentric interpolation equations used to locate a node inside a tetrahedron.

Let V_i $(i = 1, 2, 3, 4)$ be the vertices of tetrahedron T. Any point P in three-dimensional space can be expressed as

$$P = \theta_1 V_1 + \theta_2 V_2 + \theta_3 V_3 + \theta_4 V_4, \tag{3}$$

where θ_i's are the barycentric coordinates for point P, and

$$\theta_1 + \theta_2 + \theta_3 + \theta_4 = 1. \tag{4}$$

And point P is inside the tetrahedron if

$$\theta_i > 0 \text{ for } i = 1, 2, 3, 4. \tag{5}$$

In our problem, point P is a node of $HDMesh3$, tetrahedron T is a tetrahedral element of $TetMesh2$, and we want to solve for the barycentric coordinates θ_i's. Since Equation(3) can be decomposed into three sub-equations for x, y, and z coordinates, along with Equation(4), we have four equations to solve for four barycentric coordinates. Then by using Equation(5) we can search for an element in $TetMesh2$ that each $HDMesh3$'s node lies inside. The ID number of the found element and associated barycentric coordinates are then stored for each $HDMesh3$'s node.

Now that we know in which element of $TetMesh2$ that each $HDMesh3$'s node lies, we can map each $HDMesh3$'s node onto the original undeformed mesh ($TetMesh1$) using Equation(3). By mapping all the nodes and connecting them based on the hex-dominant mesh element connections of $HDMesh3$ the result is the inversely adapted hex-dominant mesh. Figure 4 depicts the node mapping process in two dimensions.

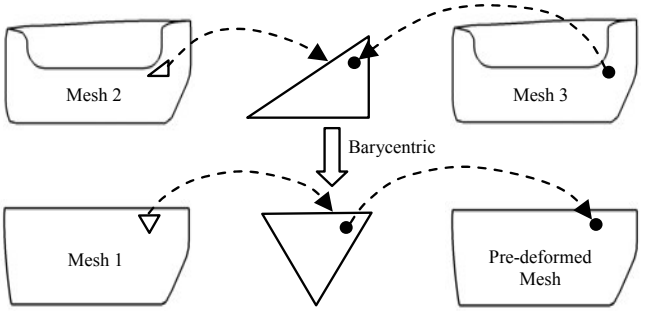


Fig. 4. Node mapping process by Barycentric Interpolation

Step 4: Full analysis. A full analysis is performed on the new inversely adapted hex-dominant mesh obtained from Step 3.

Since the inversely adapted mesh is generated based on the deformed workpiece taken from the intermediate stage in the pre-analysis, the element shape quality of the resulting mesh progressively improves until the maximum quality is reached around the stage of deformation where the initial data was taken during pre-analysis; thereafter the shape quality starts to degrade. Thus, it is possible in the case of a complex geometry that performing the inverse adaptation just once might not be enough. The inverse adaptation method can be repeated iteratively by treating the results of the analysis of the current iteration as the pre-analysis of the next iteration, then repeating Steps 2 to 4. In other words, the inversely adapted mesh of the current iteration with $HDMesh3$ and $TetMesh3$ element connections will be used as $HDMesh1$ and $TetMesh1$ respectively on the next iteration. Therefore, in brief we can use the analysis result as the starting mesh and iteratively perform inverse adaptation to progressively adapt the input mesh until analysis can proceed all the way to the final deformation stage without being terminated by ill-shaped or inverted elements.

It is important to note that even though the proposed method can be repeated iteratively, the inversely adapted mesh generated from this method should not contain unacceptably bad elements, which can cause problems in starting the analysis or cause the analysis to fail from the start. When, after iterative inverse adaptation, elements become so ill-shaped or inverted that the bad quality tolerance is reached, then adaptive remeshing must be performed. This can happen with complex geometries. Iterative inverse adaptation, however, significantly lengthens the life of the analysis before adaptive remeshing has to be applied and consequently reduces the number of remeshing operations needed.

4 Computational Experiments and Discussion

4.1 Test Problem

The model of the test problem examined in this paper is shown in Fig.5 [3]. The model consists of a punch, a die and a deformable blank. The blank is made

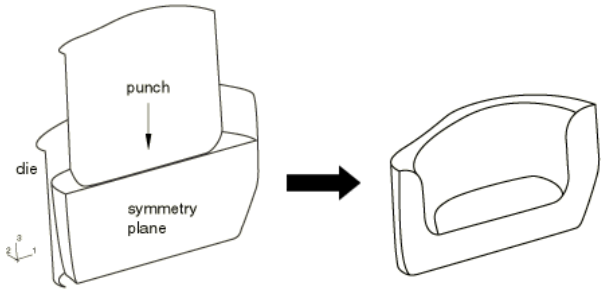


Fig. 5. Model setting of the test problem [3]

of a steel alloy with a reference stress value of 763 MPa and a work-hardening exponent of 0.245. Isotropic elasticity is assumed, with a Young's modulus of 211 GPa and a Poisson's ratio of 0.3. The punch and the die are modeled as discrete rigid bodies. The punch is moved 10.5 mm in the y-direction toward the blank at a constant velocity of 30 m/sec. The bottom face of the blank is constrained in the y-direction, and the front face of the blank has symmetric boundary conditions in the z-direction.

4.2 Inversely Adapted Mesh

We have run three iterations of inverse adaptation on this test problem. The 1st iteration uses the result in step 5 of the original mesh as the starting mesh, the 2nd iteration uses the result in step 7 of the 1st iteration as the starting mesh, and the 3rd iteration uses the result in step 12 of the 2nd iteration as the starting mesh. The first frames of Fig.6 (b) and (c) show the resultant inversely adapted meshes for the 2nd and 3rd iterations respectively. The element ratio and volume ratio information of each mesh is shown in Table1 and Table2 respectively.

4.3 Analysis Results and Discussion

In this section, the results of the analyses of the inversely adapted meshes are shown and compared with the results of the analysis of the original mesh. Fig.6 (a), (b) and (c) show the finite element analysis results of the original mesh, the 2nd iteration inversely adapted mesh and the 3rd iteration inversely adapted mesh, respectively.

It is demonstrated in Fig.6 that the analysis of the original mesh begins to produce ill-shaped elements at a very early stage (step 10), while the inversely adapted meshes can extend the life of the analysis to later stages (to step 14 for the 2nd iteration and step 18 for the 3rd iteration). This is because the thin and fine elements in the inversely adapted meshes, which were intentionally generated at locations predicted to encounter high-curvature corners during analysis, gradually unfold as the analysis continues. Consequently, the shapes of the elements improve progressively during the analysis until some later point when element shape quality begins to degrade.

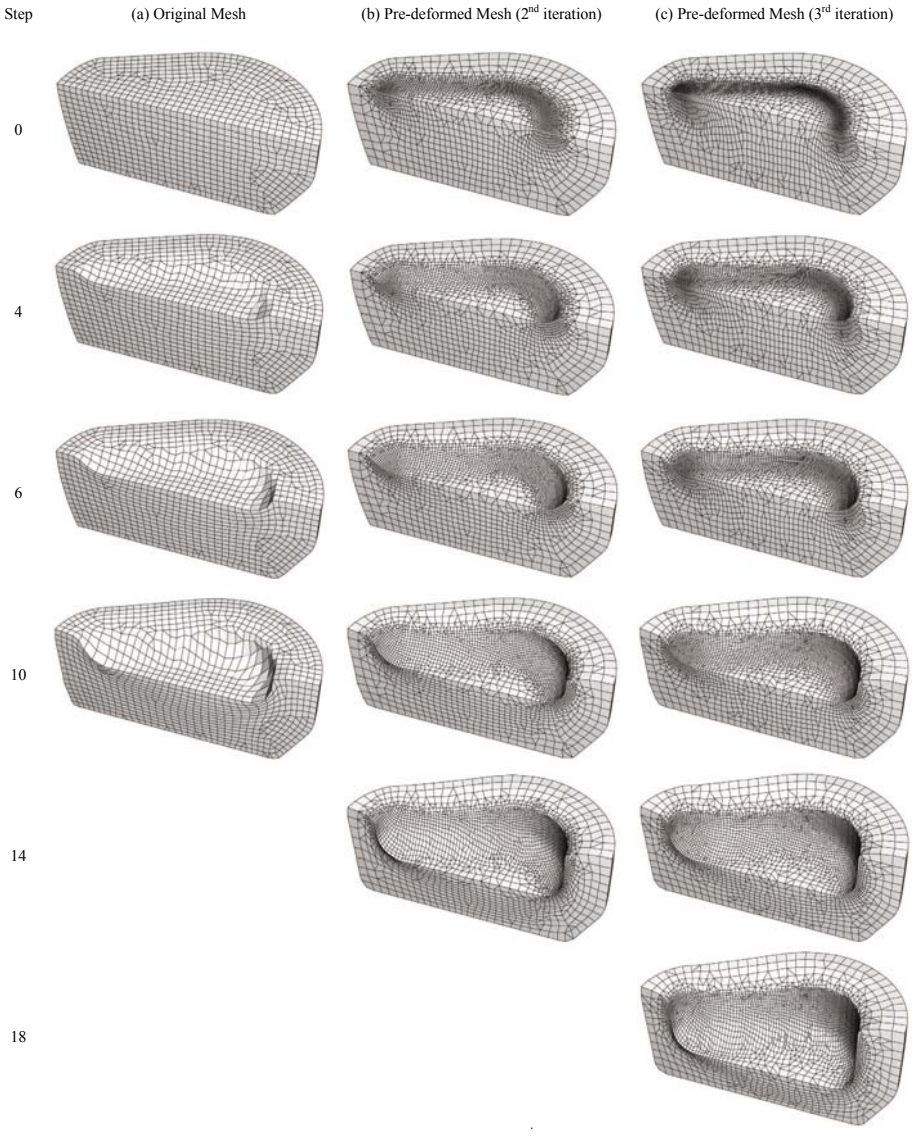


Fig. 6. Finite element analysis of the original mesh (a), inversely adapted mesh (2nd iteration) (b), and inversely adapted mesh (3rd iteration) (c)

Furthermore, it is shown that the inversely adapted meshes can capture the features of the high curvature areas and inversely deform the elements as well as refine the element sizes around these locations successfully. As a result, the geometric interference between the blank and the punch is reduced as we run more iterations of inverse adaptation. This is an important point in large deformation analysis, because usually when the geometric interference between workpiece

Table 1. Element ratio

	Hex	Tet	Prism
Original	5091 (58%)	3053 (35%)	509 (5%)
2nd Pre-deform	9439 (30%)	19507 (62%)	2410 (7%)
3rd Pre-deform	11177(30%)	23016 (61%)	2970 (7%)

Table 2. Volume ratio

	Hex	Tet	Prism
Original	6034 (88%)	571 (8%)	288 (4%)
2nd Pre-deform	4718 (69%)	1520 (22%)	621 (9%)
3rd Pre-deform	4621 (67%)	1488 (22%)	738 (11%)

and tools exceeds a specified tolerance, adaptive remeshing has to be performed to create a new mesh for the workpiece in order to achieve an accurate result. Thus, with its ability to control the size and shape of the elements around the areas expecting severe distortion, the proposed method can reduce or eliminate the need for adaptive remeshing during the analysis.

To evaluate the element shape quality results, the numbers of inverted elements for each analysis step are compared in Fig.7. Marked elements are the inverted elements existed in the resulting mesh of each particular analysis step. As clearly shown in the figure, the number of inverted elements is reduced significantly in the analysis run on the inversely adapted meshes. Because inverted elements lead to inaccurate results and difficulty in finite element analysis calculation, they must be reduced as much as possible. The results in Fig.7 demonstrate that the inverse adaptation method can successfully achieve this. Furthermore, the plots of the minimum scaled Jacobian of hexahedral and prism elements, as well as the average radius ratio of tetrahedral elements are shown in Fig.8. It is clearly seen on the plot comparing the minimum Jacobian of hexahedral elements (Fig.8(a)) that the original mesh creates elements with negative Jacobian value at an earlier stage than the inversely adapted mesh. It is also shown that the inversely adapted mesh lengthens the life of the analysis with no negative-Jacobian element.

5 Conclusion

This paper proposes the inverse adaptation method for a hex-dominant mesh, which inversely deforms and refines elements at locations where large deformation is expected in advance. Moreover, the inverse adaptation method is an iterative method. In problems with complicated die geometry, the inverse adaptation method can be run iteratively until the final deformation of the geometry is reached.

As opposed to traditional analysis that starts with an optimal mesh which becomes progressively degraded and can cause the analysis to terminate early

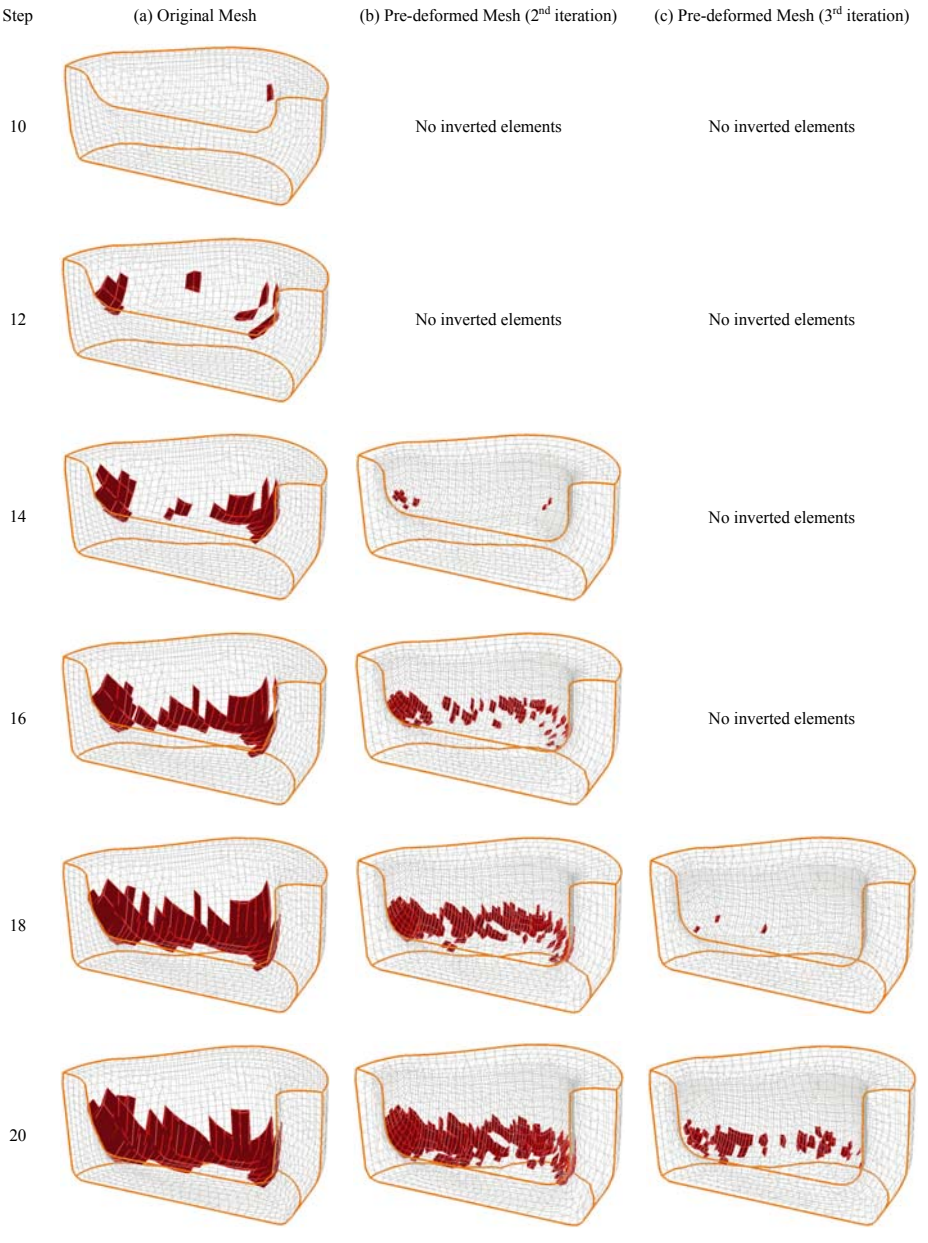


Fig. 7. Comparison of the number of inverted elements in the resulting meshes in the later analysis stages

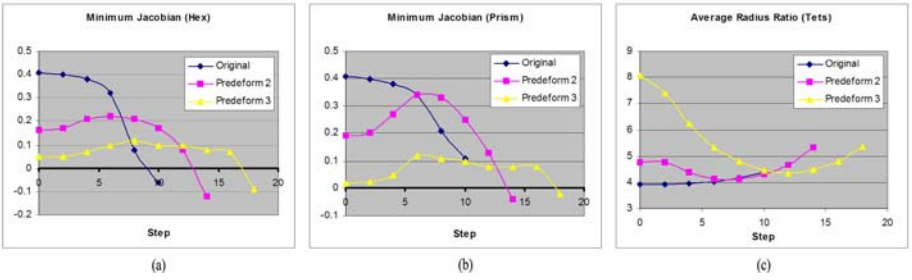


Fig. 8. Plots of minimum scaled Jacobian of hex elements (a), minimum scaled Jacobian of prism elements (b), and average radius ratio of tet elements (c)

due to excessive element distortion, the analysis of an inversely adapted mesh starts with an input mesh with slightly distorted elements whose shapes are improved as the analysis continues, and which can be carried on to the final or further stage with fewer or no inverted elements.

Furthermore, because the inverse adaptation method can successfully refine the elements around the areas expecting severe distortion, it can reduce the problem of geometric interference between workpiece and tools.

As a result, with its ability to effectively lengthen the life of the analysis before the mesh become unacceptable, as well as to reduce the problem of geometric interference, inverse adaptation can reduce the need for adaptive remeshing required during the analysis.

In addition, the users who run this type of analysis commonly have to use their own common sense to determine how the input mesh should be generated in order to run the analysis to the desired stage. This inverse adaptation method can be a tool to aid them in that purpose.

Acknowledgements

The first author acknowledges a graduate fellowship provided by the Thai Government. This material is based in part on work supported under an NSF CAREER Award (No. 9985288).

References

1. ABAQUS 6.4.: Getting Started with ABAQUS/Explicit. Chapter7 Quasi-Static Analysis. Hibbitt, Karlsson & Sorensen, Inc. (2003)
2. ABAQUS 6.4.: Analysis User's Manual. Hibbitt, Karlsson & Sorensen, Inc. (2003)
3. ABAQUS 6.4.: Example Problems Manual. Chapter 1.3.9 Forging with Sinusoidal Die. Hibbitt, Karlsson & Sorensen, Inc. (2003)
4. Coupez, T.: Automatic Remeshing in Three-dimensional Moving Mesh Finite Element Analysis of Industrial Forming. Simulation of Material Processing: Theory, Practice, Methods and Applications. (1995) 407–412

5. Dheeravongkit, A., Shimada, K.: Inverse Pre-deformation of Finite Element Mesh for Large Deformation Analysis. In Proceedings of 13th International Meshing Roundtable. (2004) 81–94
6. Dheeravongkit, A., Shimada, K.: Inverse Pre-deformation of Finite Element Mesh for Large Deformation Analysis. Journal of Computing and Information Science in Engineering. **5(4)** (2004) 338–347
7. Dheeravongkit, A., Shimada, K.: Inverse Pre-deformation of the Tetrahedral Mesh for Large Deformation Finite Element Analysis. Computer-Aided Design and Applications. /bfseries 2(6) (2005) 805–814
8. Gadala, M. S., Wang, J.: Simulation of Metal Forming Processes with Finite Element Methods. International Journal for Numerical Method in Engineering. **44** (1999) 1397–1428
9. Hattangady, N. V.: Automatic Remeshing in 3-D Analysis of Forming Processes. International Journal for Numerical Methods in Engineering. **45** (1999) 553–568
10. Hattangady, N. V.: Automated Modeling and Remeshing in Metal Forming Simulation. Ph.D. Thesis. Rensselaer Polytechnic Institute (2003)
11. Joun, M. S., Lee, M. C.: Quadrilateral Finite-Element Generation and Mesh Quality Control for Metal Forming Simulation. International Journal for Numerical Methods in Engineering. **40** (1997) 4059–4075
12. Khoei, A. R., Lewis, R. W.: Adaptive Finite Element Remeshing in a Large Deformation Analysis of Metal Powder Forming. International Journal for Numerical Methods in Engineering. **45(7)** (1999) 801–820
13. Kraft, P.: Automatic Remeshing With Hexahedral Elements: Problems, Solutions and Applications. In Proceedings of 8th International Meshing Roundtable. (1999) 357–367
14. Kwak, D. Y., Cheon J. S., Im, Y. T.: Remeshing for Metal Forming Simulations–Part I: Two-dimensional Quadrilateral Remeshing. International Journal for Numerical Methods in Engineering. **53(11)** (2002) 2463–2500
15. Kwak, D. Y., Im, Y. T.: Remeshing for metal forming simulations–Part II: Three-dimensional hexahedral mesh generation. International Journal for Numerical Methods in Engineering. **53(11)** (2002) 2501–2528
16. Lee, Y. K., Yang D. Y.: Development of a Grid-based Mesh Generation Technique and its Application to Remeshing during the Finite Element Simulation of a Metal Forming Process. Engineering Computations. **16(3)** (1999) 316–339
17. Meinders, T.: Simulation of Sheet Metal Forming Processes. Chapter 5. Adaptive Remeshing (1999)
18. Merrouche, A., Selman, A., Knoff-Lenoir, C.: 3D Adaptive Mesh Refinement. Communications in Numerical Methods in Engineering. **14** (1998) 397–407
19. Petersen, S. B., Martins, P. A. F.: Finite Element Remeshing: A Metal Forming Approach for Quadrilateral Mesh Generation and Refinement. International Journal for Numerical Methods in Engineering. **40** (1997) 1449–1464
20. Souli, M.: An Eulerian and Fluid-Structure Coupling Algorithm in LS-DYNA. 5th International LS-Dyna Users Conference. (1999)
21. Souli, M., Olovsson, L.: ALE and Fluid-Structure Interaction Capabilities in LS-DYNA. 6th International LS-Dyna Users Conference. (2000) 15-37-14-45
22. Souli, M., Olovsson, L., Do, I.: ALE and Fluid-Structure Interaction Capabilities in LS-DYNA. 7th International LS-Dyna Users Conference. (2002) 10-27-10-36
23. Stoker, H. C.: Developments of Arbitrary Lagrangian-Eulerian Method in Non-Linear Solid Mechanics. PhD thesis. University of Twente. (1999)

24. Vallinkoski, I.: The design and implementation of a color management application. Chapter 6 Gamut Mapping Algorithm. MS Thesis. Helsinki University of Technology. (1998)
25. Wagoner, R.H., Chenot, J.-L.: Metal Forming Analysis, Chapter10: Forging Analysis. ISBN 0-521-64267-1 (2001)
26. Wan, J., Kocak, S., Shephard, M. S.: Automated Adaptive Forming Simulations. In Proceedings of 12th Interational Meshing Roundtable. (2003) 323–334
27. Yamakawa, S., Shimada, K.: Hex-Dominant Mesh Generation with Directionality Control via Packing Rectangular Solid Cells. IEEE Geometric Modeling and Processing - Theory and Applications. (2002)
28. Yamakawa, S., Shimada, K.: Fully-Automated Hex-Dominant Mesh Generation with Directionality Control via Packing Rectangular Solid Cells, International Journal for Numerical Methods in Engineering 57, (2003) 2099–129
29. Yamakawa, S., Shimada, K.: Increasing the number and volume of hexahedral and prism elements in a hex-dominant mesh by topological transformations. In Proceedings of 12th International Meshing Roundtable. (2003) 403–413
30. Zienkiewicz, O. C., Zhu, J. Z.: The Superconvergent Patch Recovery and A Posteriori Error Estimates. Part 1: The Recovery Technique. International Journal for Numerical Methods in Engineering. **33** (1992) 1331–1364
31. Zienkiewicz, O. C., Zhu, J. Z.: The Superconvergent Patch Recovery and a Posteriori Error Estimates. Part 2: Error Estimates and Adaptivity. International Journal for Numerical Methods in Engineering. **33** (1992) 1365–1382

Preserving Form-Features in Interactive Mesh Deformation

Hiroshi Masuda[1], Yasuhiro Yoshioka[1], and Yoshiyuki Furukawa[2]

[1] The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
masuda@nakl.t.u-tokyo.ac.jp
http://www.nakl.t.u-tokyo.ac.jp/~masuda
[2] National Institute of Advanced Industrial Science and Technology, 1-2 Namiki, Tsukuba-shi, Ibaraki 305-8564, Japan

Abstract. Interactive mesh editing techniques that preserve discrete differential properties are promising to support the design of mechanical parts such as automobile sheet metal panels. However, existing methods lack the ability to manipulate form-features and hard constraints, which are common in engineering applications. In product design, some regions on a 3D model are often required to precisely preserve the surface types and parameters during deformation. In this paper, we propose a discrete framework for preserving the shapes of form-features using hard constraints in interactive shape deformation. Deformed shapes are calculated so that form-features translate and rotate while preserving their original shapes according to manipulating handles. In addition, we show how to constrain the motion of form features using linear constraints. The implemented system can achieve a real-time response for constrained deformation.

1 Introduction

In product design, 3D models are often created in the early stage of product development, because such models are very effective for preliminary design evaluation by the development team. The evaluation of design concepts in the early stage helps products to meet requirements for manufacturing, cost, safety, quality, maintenance, and so on.

Interactive free-form surface modeling techniques are very important in the early stage of design. Since design concepts are very often changed or discarded in the early stage, it is not reasonable to spend a lot of time on creating detailed 3D models. Although non-uniform rational B-spline (NURBS) surfaces have been widely used to represent free-form shapes in CAD applications, it is very tedious and time-consuming to manipulate many surface patches with a large number of control points.

Free-form deformation (FFD) is a popular interactive technique in computer graphics applications. FFD changes geometric shapes by deforming the space in which the object lies. However, FFD is not necessarily convenient for supporting product design, because it often modifies product shapes in unintended ways, for example, circular holes are deformed to ellipses.

In the last few years, several discrete deformation techniques based on differential properties have been published [1,2,3]. They represent the differential properties of a given surface as a linear system and deform the surface so that the differential properties are preserved. In the typical mesh editing technique, the user first selects the fixed

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 207–220, 2006.
© Springer-Verlag Berlin Heidelberg 2006

region, which remains unchanged, and the handle region, which is used as the manipulation handle, and interactively deforms the shape by dragging the handle in the screen.

Our motivation for studying interactive deformation stems from requirements for the deformation of automobile sheet metal panels. In sheet metal panels, some curves and surfaces are often required to keep their original types, such as circles and cylinders, for manufacturing and assembly reasons. Such partial regions are called *form features* in product design. Sheet metal parts typically consist of the combination of free-form surfaces and form-features, each of which have different characteristics; while free-form surfaces are characterized by the distribution of curvature, form-features are defined by surface types and their parameters[4,5].

Therefore, it is required for engineering design to maintain the curvature of surfaces and the shapes of form-features.

In this paper, when constraints are approximately satisfied in the least squares sense, we call them *soft constraints*, and when constraints are precisely satisfied, we call them *hard constraints*. In typical shape design, while differential properties are not explicitly specified by the user, the surface types and their parameters are directly specified according to engineering requirements. It is obviously reasonable to treat differential properties as soft constraints, and user-defined constraints as hard constraints.

However, existing methods do not allow the combination of hard and soft constraints. They solve all constraints using the least-squares method and produce compromised solutions, which are not accepted in engineering sense. It is possible to put large weights on certain constraints, but it is difficult to predict weight values that satisfy the allowable margin of error, especially when differential properties are represented using the cotangent weighting method[6], which approximates the mean curvature very well.

In this paper, we propose a novel mesh-editing framework that can manage form-features using hard constraints. Our main contribution in this paper is:

- a novel deformation framework in which hard and soft constraints are incorporated and mean curvature normals are rotated by interpolating quaternion logarithms;
- the introduction of new constraints for translating and rotating form-features while preserving their original shapes; and
- the introduction of new constraints for maintaining the motion of form-features on a straight line or a plane.

In the following section, we review the related work on mesh editing. In Section 3, we describe our mesh deformation framework, in which hard constraints are incorporated using Lagrange multipliers and a new rotation method is introduced based on quaternion logarithms. In Section 4, we introduce a method for preserving form-features by constraining the relative positions and rotations of vertices, and then propose new feature constraints which maintain the motion of a form-feature on a line or a plane. We also show a simple feature extraction method. In Section 5, we evaluate our framework and show experimental results. We conclude the paper in Section 6.

2 Related Work

Interactive mesh-editing techniques have been intensively studied. Such research aims to develop modeling tools that intuitively modify free-form surfaces while preserving

the details of shapes. There are several types of approach for mesh editing, which are based on space deformation (FFD), multiresolution, and partial differential equations (PDE).

FFD is very popular in computer graphics. Such methods modify shapes by deforming 3D space in which objects lie [7,8,9,10]. Cavendish [5] discussed FFD approaches in the context of design support and described that FFD could be used for designing automotive sheet metal panels. We also aim at supporting the design of automotive sheet metal panels, but in our empirical investigation of automobile industry, problems arise when form-features in a 3D model are modified in unintended ways. It is difficult for FFD approaches to manage constraints on form-features, because the manipulation handles do not work directly on geometric shapes.

Multiresolution approaches [11,12,13,14,15] decompose a surface into a base mesh and several levels of details, each of which is represented as the difference between successive resolution levels. A shape is globally deformed at low resolution and locally deformed at high resolution. Botsch and Kobbelt [2] applied this technique to interactive mesh editing. A mesh model is decomposed into two-level resolutions and the smooth base is interactively deformed using energy minimization techniques. Geometric details are then recovered on the modified smooth shape. However, it is difficult to control the shapes of form-features precisely in a multiresolution framework.

PDE-based approaches directly deform the original mesh based on geometric constraints. These methods are categorized as non-linear and linear methods. Non-linear methods typically solve Laplacian or Poisson equations using non-linear iterative solvers [16,17,18,19,20]. Catalano et al. [21] investigated support tools for aesthetic design and showed non-linear PDE approaches are popular in computer-aided design. These methods produce fair surfaces, but it is time-consuming and difficult to deform shapes interactively.

We believe that linear PDE-based approaches are also useful for product design when product shapes undergo frequent design modifications. Linear PDE-based approaches represent differential properties and vertex positions in a linear system. Discrete Laplacian operators are often used to represent differential properties [1,2]. Yu et al. [3] introduced a similar technique called Poisson editing, which manipulates the gradients of the coordinate functions of the mesh. Zhou et al. [22] proposed volumetric Laplacian operators for large deformations.

A discrete Laplacian operator on a mesh is defined as the difference vector between a vertex position and the weighted average position of its one-ring neighbors. Since Laplacians are defined in the local coordinate systems[23,1], one or more vertices must be specified in the global coordinate system to determine all vertex positions. When each vertex is constrained by the differential property, additional constraints for vertex positions lead to over-constraint situations. In existing methods, the least squares method is typically used for calculating compromised solutions. When Laplacians and positional constraints are described as linear system $M\mathbf{x} = \mathbf{b}$, the least squares system is represented as $M^tM\mathbf{x} = M^t\mathbf{b}$, and therefore vertex positions are calculated by $\mathbf{x} = (M^tM)^{-1}M^t\mathbf{b}$. Since M^tM is a sparse symmetrical positive definite matrix, it can be efficiently factorized[24]. After the matrix is factorized once, $\mathbf{x}$ is interactively calculated according to the modification of $\mathbf{b}$.

On the other hand, Welch and Witkin[25] introduced the combination of soft and hard constraints in variational surface modeling and solved them using Lagrange multipliers, although they did not apply them for interactive mesh editing. Yoshioka et al. [26] solved hard constraints using equality-constrained least squares. This method is very effective when constraints are restricted to simple positional constraints. However, it is not useful for form-feature constraints, because a large number of hard constraints with two or more variables make equality-constrained least squares systems less sparse. In this paper, we introduce hard constraints and new form-feature constraints for interactive mesh editing and solve them using the Lagrange multiplier.

Several authors have discussed methods for rotating the Laplacian vectors following the deformation of surfaces. Since the rotation is applied to $\mathbf{b}$ on the right-hand side of the least squares system, Laplacian vectors are rotated in an interactive way. Lipman et al. [27] estimated the local rotations on the underlying smooth surface. Sorkine et al. [1] linearized elements in a rotation matrix assuming that rotation angles are very small and solved them as a linear system. Lipman et al. [28] encoded rotations and positions using relative positions on local frames and solved them as two separate linear systems. Zayer et al. [29] rotated Laplacian vectors using harmonic functions in $[0,1]$ based on discrete Laplace-Beltrami operators. They defined a unit quaternion at each vertex and interpolated four components of unit quaternions by assigning a single weight 1.0 to all handle vertices. Our approach is similar to Zeyer's method, although we incorporate constraints for the rotations of form-features and assign the logarithms of multiple unit quaternions for handle vertices.

3 Framework for Constrained Deformation

3.1 Constraints on Positions

Let the mesh $\mathbb{M}$ be a pair $(K,\mathbb{P})$, where $\mathbb{P}=\{\mathbf{p_1},\ldots,\mathbf{p_n}\}$ and $\mathbf{p}_i=(x_i,y_i,z_i)\in\mathbb{R}^3$; K is a simplicial complex that contains vertices i, edges (i,j), and faces (i,j,k). The adjacent vertices of vertex i are denoted by $N(i)=\{j|(i,j)\in K\}$. The original position of $\mathbf{p}_i$ is referred to as $\mathbf{p_i^0}=(x_i^0,y_i^0,z_i^0)$.

When the normal vector and mean curvature of vertex i are referred to as κ_i and $\mathbf{n}_i$, the mean curvature vector $\kappa_i\mathbf{n}_i$ can be approximated using the following discrete form [6];

$$\kappa_i\mathbf{n}_i=L(\mathbf{p}_i)=\frac{1}{4A_i}\sum_{j\in N(i)}(cot\alpha_{ij}+cot\beta_{ij})(\mathbf{p}_i-\mathbf{p}_j), \tag{1}$$

where α_{ij} and β_{ij} are the two angles opposite to the edge in the two triangles that share edge (i,j), as shown in Figure 1. We denote the mean curvature vectors of the original mesh as $\delta_i(i=1,2,\ldots,n)$.

In this paper, we describe additional constraints other than mean curvature vectors as $\mathbf{f}_j(\mathbb{P}_j)=\mathbf{u}_j\ \ (\mathbb{P}_j\subset\mathbb{P})$. Then constraints for vertices can be described as the following linear equations:

$$\begin{cases} L(\mathbf{p}_i)=R(\mathbf{n}_i,\theta_i)\delta_i \quad (i=1,2,\ldots,n) \\ \mathbf{f}_j(\mathbb{P}_j)=R(\mathbf{m}_j,\phi_j)\mathbf{u}_j \quad (j=1,2,\ldots,m), \end{cases} \tag{2}$$

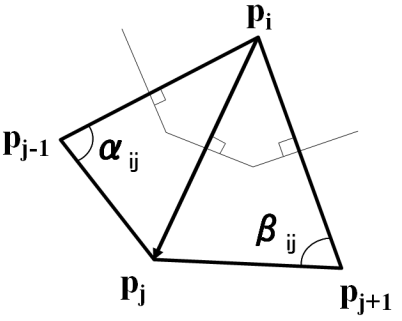


Fig. 1. Definition of α_{ij} and β_{ij}

where n is the number of vertices; m is the number of additional constraints; $R(\mathbf{n},\theta)$ represents a rotation matrix that rotates a vector around axis $\mathbf{n} \in \mathbb{R}^3$ by angle $\theta \in \mathbb{R}$. $R(\mathbf{n}_i,\theta_i)$ and $R(\mathbf{m}_j,\phi_j)$ are calculated before Equation 2 is solved. We will describe a method for calculating rotation matrices in the next section.

Since the number of constraints in Equation 2 is larger than the one of variables, the exact solution does not exist. Therefore, we classify constraints in Equation 2 into soft and hard constraints, because the positional constraints need to be satisfied as precisely as possible in most engineering applications.

When we represent soft constraints as $A\mathbf{x} = \mathbf{b}$ and hard constraints as $C\mathbf{x} = \mathbf{d}$ in matrix forms, variables $\mathbf{x}$ that minimizes $||A\mathbf{x}-\mathbf{b}||^2$ subject to $C\mathbf{x} = \mathbf{d}$ can be calculated using the Lagrange multiplier as:

$$\min_{\mathbf{x}}(\frac{1}{2}||A\mathbf{x}-\mathbf{b}||^2 + \mathbf{y}^t(C\mathbf{x}-\mathbf{d})), \tag{3}$$

where $\mathbf{y} = (y_1, y_2, \ldots, y_m)^t$ are Lagrange multipliers. This minimization can be calculated using the following linear system:

$$M\tilde{\mathbf{x}} = \tilde{\mathbf{b}} \tag{4}$$

$$M = \begin{pmatrix} A^tA & C^t \\ C & 0 \end{pmatrix}, \tilde{\mathbf{x}} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}, \tilde{\mathbf{b}} = \begin{pmatrix} A^t\mathbf{b} \\ \mathbf{d} \end{pmatrix}$$

This linear system determines the unique solution that satisfies hard constraints exactly. Matrix M can be factorized using sparse direct solvers for linear symmetric systems[30].

We will note that when conflicting or redundant constraints are involved in the linear system of hard constraints, they lead to the rank deficiency of linear systems, and the solver may halt the computation. Such over-constraint problems can be resolved by applying Householder factorization to each column in matrix C, as shown by Yoshioka et al. [26]. If the jth column in C^T is a redundant constraint, the diagonal and lower elements of the jth column are equal to zero after the previous $j-1$ columns are processed. Therefore, we can detect the redundant constraints. This process can be calculated very efficiently. See [26] for more detail.

3.2 Constraints on Rotations

In this section, we describe our new rotation-propagation method for calculating $R(\mathbf{n}_i, \theta_i)$ in Equation 2.

In our framework, we assign the logarithms of unit quaternions to all vertices. A quaternion can be written in the form:

$$Q = (w,x,y,z) = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}, \tag{5}$$

where $w,x,y,z \in \mathbb{R}$ and $\mathbf{i},\mathbf{j},\mathbf{k}$ are distinct imaginary numbers. When a quaternion has unit magnitude, it is called a unit quaternion and corresponds to a unique rotation matrix. A unit quaternion can be represented using rotation axis $\hat{\mathbf{n}}$ and rotation angle θ:

$$\hat{Q} = e^{\hat{\mathbf{n}}\frac{\theta}{2}} = cos\frac{\theta}{2} + \hat{\mathbf{n}}sin\frac{\theta}{2}, \tag{6}$$

where $\hat{\mathbf{n}}$ is a pure quaternion. The logarithm of a unit quaternion is defined as the inverse of the exponential:

$$\mathbf{q} = ln\hat{Q} = \frac{\theta}{2}\hat{\mathbf{n}}. \tag{7}$$

We assign logarithm $\mathbf{q}_i \in \mathbb{R}^3$ to vertex i and denote the logarithms assigned to all vertices as $\mathbb{Q} = \{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n\}$. When $\mathbf{q}_i$ is equal to 0, the mean curvature vector is not rotated; when $\mathbf{q}_i = \mathbf{v}_j$ is specified, the mean curvature vector is rotated around axis $\mathbf{v}_j/|\mathbf{v}_j|$ by angle $2|\mathbf{v}_j|$.

Shoemake [31] proposed the spherical linear interpolation between two unit quaternions. Johnson[32] applied the spherical linear interpolation to multiple unit quaternions using the logarithms of unit quaternions. Pinkall and Polthier [33] proposed an interpolation technique using discrete conformal mapping. Zayer et al. [29] applied discrete conformal mapping for interpolating unit quaternions. We introduce similar constraints on the logarithms of unit quaternions:

$$L(\mathbf{q}_i) = \frac{1}{4A_i}\sum_{j\in N(i)}(cot\alpha_{ij} + cot\beta_{ij})(\mathbf{q}_i - \mathbf{q}_j) = 0. \tag{8}$$

Then we introduce additional linear equations other than Equation 8, and describe them as:

$$\mathbf{g}_j(\mathbb{Q}_j) = \mathbf{v}_j \quad (\mathbb{Q}_j \subset \mathbb{Q}, \mathbf{v}_j \in \mathbb{R}^3). \tag{9}$$

As a result, linear equations for rotations can be described as:

$$\begin{cases} L(\mathbf{q}_i) = 0 \quad (i = 1,2,\ldots,n) \\ \mathbf{g}_j(\mathbb{Q}_j) = \mathbf{v}_j \quad (j = 1,2,\ldots,r), \end{cases} \tag{10}$$

where n is the number of vertices; r is the number of additional constraints on rotations. These equations construct a sparse linear system and can be solved using sparse direct solvers[30]. The solution of the linear system generates certain energy-minimization surfaces [33] in three dimensional space spanned by the logarithms of unit quaternions.

When all components of $\mathbb{Q}$ are calculated, rotation matrix $R(\mathbf{q}_i/|\mathbf{q}_i|, 2|\mathbf{q}_i|)$ is uniquely determined at each vertex. Figure 2 is an example of a deformed shape, in which the mean curvature vectors are rotated.

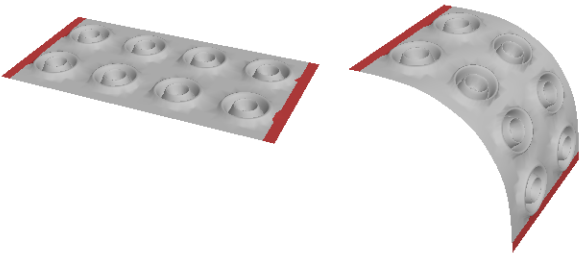


Fig. 2. Deformation using rotated mean curvature vectors. Left: original shape; right: deformed shape.

4 Preserving Form-Features

4.1 Preserving the Shape of Form-Feature

We define a form-feature as a partial shape that has an engineering meaning, such as a hole and a protrusion. In mesh model $\mathbb{M}$, a form-feature consists of a subset of vertices $\mathbb{P}$. In this section, we denote a form-feature as f, the index set of vertices in form-feature f as Λ_f.

When form-feature f is translated and rotated according to the motion of handles while preserving its original shape, the following constrains regarding rotations and positions maintain the shape of the form-feature:

$$\begin{cases} \mathbf{q}_i - \mathbf{q}_j = 0 \quad (i,j \in \Lambda_f; (i,j) \in K) \\ \mathbf{p}_i - \mathbf{p}_j = s_f R(\mathbf{n}_i, \theta_i)(\mathbf{p}_i^0 - \mathbf{p}_j^0) \quad (i,j \in \Lambda_f; (i,j) \in K), \end{cases} \tag{11}$$

where s_f is a scaling factor. These equations are added to Equation 2 and 10. Since each vertex in the form-feature has the same rotation matrix, $\mathbf{n}_i = \mathbf{n}_j$ and $\theta_i = \theta_j$ $(i,j \in \Lambda_f)$ in Equation 11.

In some cases, a form-feature has to keep the original direction depending on design intent. Then the following equations are added to Equation 2 and 10 instead of Equation 11:

$$\begin{cases} \mathbf{q}_i = 0 \quad (i \in \Lambda_f) \\ \mathbf{p}_i - \mathbf{p}_j = s_f(\mathbf{p}_i^0 - \mathbf{p}_j^0) \quad (i,j \in \Lambda_f; (i,j) \in K), \end{cases} \tag{12}$$

Figure 3 shows deformed shapes that preserve the shapes of holes as circles. As shown in Figure 3c-e, constraints on relative positions maintain the shapes of form-features. In Figure 3e, four small holes are constrained by Equation 11, but the center hole is constrained to preserve the original direction using Equation 12.

4.2 Constraining the Motion of Form-Feature

In computer-aided design, it is useful to constrain the motion of a form-feature. The positions of cylindrical form-features are often specified using the center lines, and the positions of planar faces are often constrained on a specified plane.

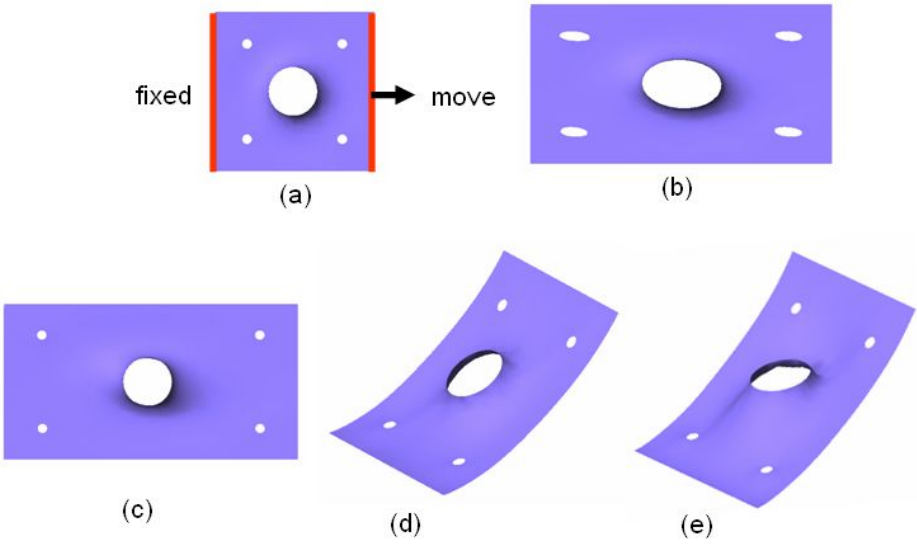


Fig. 3. Constrained deformation. (a) Original shape; (b) deformed shape with no form-feature constraints; (c) stretched while preserving the shapes of circles; (d) deformed shape with five rotated circles; and (e) deformed while preserving the direction of the center circle.

In linear constraints, the motion of a form-feature can be maintained on a straight line or a plane by constraining a point in the form-feature. A constrained point can be specified as a linear function of vertex positions in the form-feature. For example, the center position of a circle can be specified as function $0.5(\mathbf{p}_i + \mathbf{p}_j)$ using two vertex positions on the opposite sides.

Here, we generally represent a linear combination of vertex positions as $\mathbf{x} = (x, y, z)$, where each of x, y and z is a linear function of coordinates $\{\mathbf{p}_i\}(i \in \Lambda_f)$, respectively.

On-Plane Constraints. A plane is uniquely determined by its normal vector and a point on the plane. We represent the equation of a plane as $\mathbf{n}(\mathbf{x} - \mathbf{p}) = 0$, where $\mathbf{n} = (n_x, n_y, n_z)$ is the normal vector and $\mathbf{p} = (p_x, p_y, p_z)$ is a point on the plane. Then the following equation constrains the motion of a form-feature on a plane:

$$n_x x + n_y y + n_z z = n_x p_x + n_y p_y + n_z p_z \tag{13}$$

Since position $\mathbf{p}$ appears on the right-hand side of Equation 13, planes can be interactively moved to their normal directions. This capability is useful for the modification of an allowable margin to avoid interference.

Figure 4a shows deformed shapes in which a hole feature is constrained to preserve the shape with no rotations and to move on a plane.

On-Line Constraints. A straight line can be represented as $\mathbf{x} = k\mathbf{n} + \mathbf{p}$. Let $\mathbf{l}$, $\mathbf{m}$ and $\mathbf{n}$ be unit vectors that are perpendicular each other. Then position (x, y, z) moves on the straight line by using the following constraints:

$$\begin{cases} l_x x + l_y y + l_z z = l_x p_x + l_y p_y + l_z p_z \\ m_x x + m_y y + m_z z = m_x p_x + m_y p_y + m_z p_z \end{cases} \tag{14}$$

Straight lines can be moved interactively to directions that are perpendicular to **n**. This capability is also useful, because it corresponds to the movement of the center positions of circles on 2D drawings.

Figure 4b-c show the constrained motion of a form-feature. The form-feature is constrained to preserve the direction in Figure 4b and to rotate in Figure 4c while moving on a straight line.

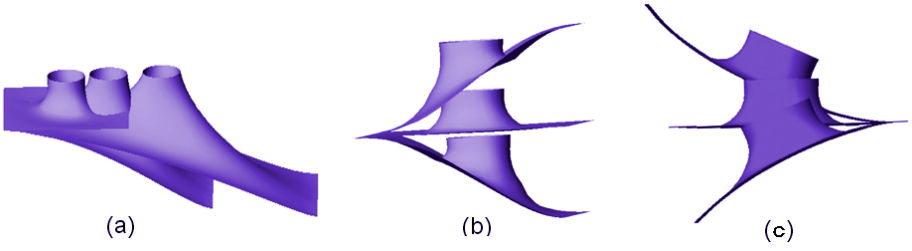


Fig. 4. (a) Deformed shapes with a form-feature that moves on a plane. (b) Deformed shapes with a form-feature that moves on a line. (c) Deformed shapes with a rotated form-feature that moves on a line.

4.3 Feature Extraction

In our system, the user first selects the region of a form-feature, and then adds linear constraints to the form-feature. It may be tedious work for the user to carefully select the region before specifying constraints. We introduce an interactive mesh segmentation technique for easy selection of feature regions.

So far, many algorithms have been reported for the segmentation of feature regions [34,35,36,37,38]. Our segmentation algorithm is based on the method proposed by Katz et al. [37], but we apply the method only to user-specified regions[39].

Figure 5 shows a feature extraction process. First, the user selects a region that includes the boundary of the feature region, as shown in Figure 5b. The region must be selected so that the mesh model is separated into exactly two regions. Then the optimal cut-set is calculated by the maximum flow, minimum cut algorithm. Finally, the feature region is separated, as shown in Figure 5c.

5 Experimental Results

Figure 6 shows examples of deformed shapes. In industrial design, character lines are essentially important. If a deformation process modifies the character lines of a product shape, the resultant shape is not accepted by designers. While the character lines are warped in Figure 6a, they are maintained in Figure 6b by defining constraints on the character lines.

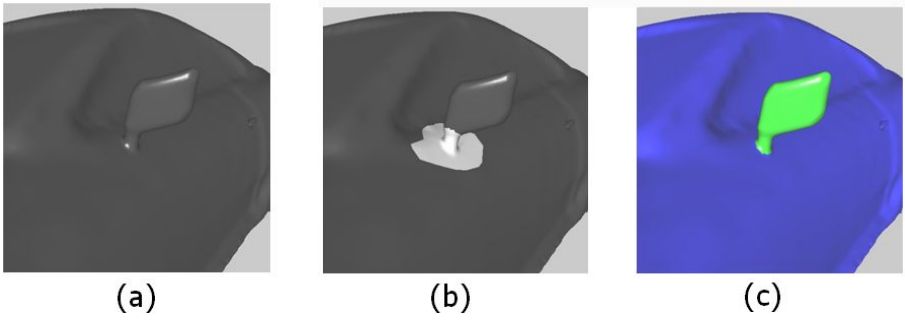


Fig. 5. Feature extraction. (a) Original shape; (b) The region selected by the user; (c) The extracted region.

Figure 7 shows a front grill part of an automobile model. While Figure7b contains no form-feature constraints, Figure 7c has form-feature constraints at the cavities. Deformation in Figure 7b destroys the design intent, but Figure 7c maintains the characteristic features. In Figure 7d, scaling factors in Equation 12 are modified in an interactive manner.

Figure 8 is a sheet metal panel. 16 cavities shown by arrows are constrained so that they rotate while preserving the original shapes of cavities.

Table 1 shows CPU time for calculating the deformed models in Figure 6-8. The CPU time was measured for setting up matrices and factorizing them on a PC with 1.50-GHz Pentium-M and 1 GB of RAM. Once the matrix was set up and factorized, the shape could be deformed in interactive rate. This result shows that the performance of our framework is good for practical use.

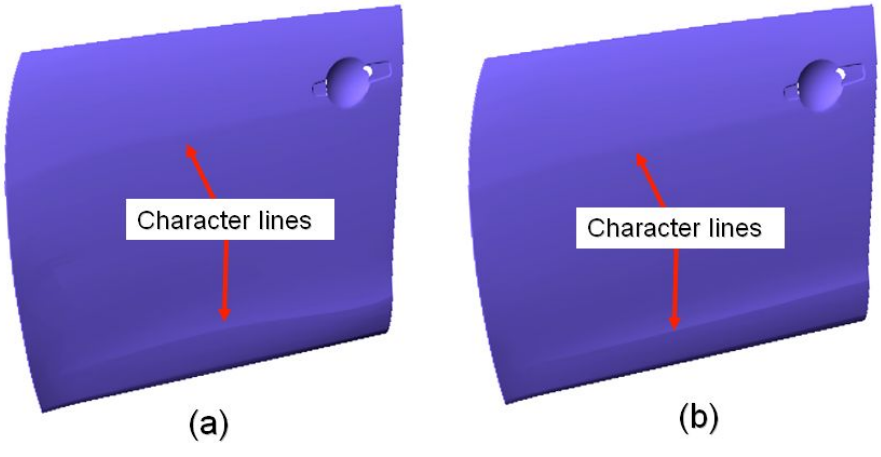


Fig. 6. Door panel. (a) Shape deformed without form-feature constraints. The character lines are warped. (b) Shape deformed with form-feature constraints on character lines.

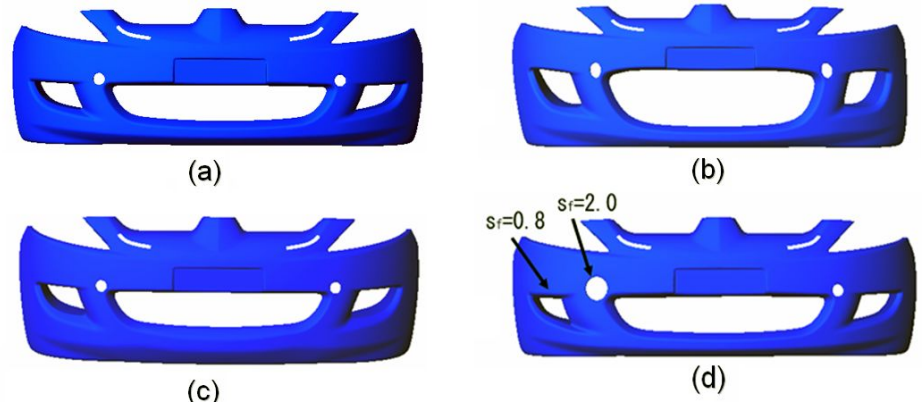


Fig. 7. Front grill. (a) Original shape. (b) Shape deformed without form-feature constraints. (c) Shape deformed with form-feature constraints. (d) Interactive scaling of form-features.

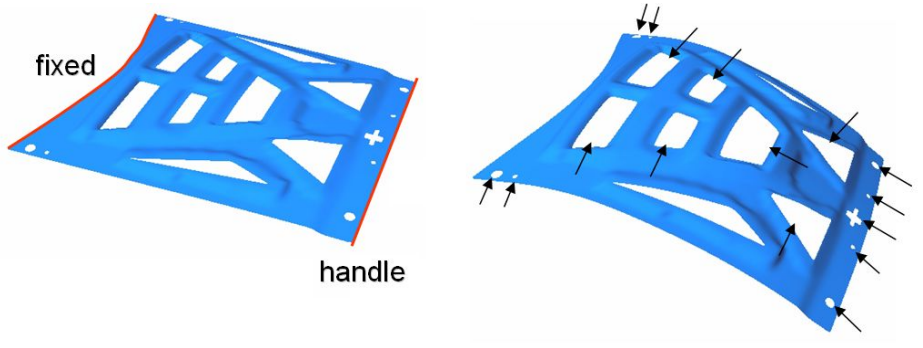


Fig. 8. Sheet metal part. Top: original planer shape. Bottom: deformed shape with rotated form-features shown by arrows.

Table 1. CPU time for computation. [Vert]: the number of vertices; [Soft]: the number of soft constraints; [Hard]: the number of hard constraints; [Feat]: the number of form-feature constraints; [Time]: CPU time(sec).

	Vert	Soft	Hard	Feat	Time
Figure 7	3337	5796	3826	3702	0.86
Figure 8	13974	25458	9334	1072	4.69
Figure 9	2982	6084	3308	3202	0.99

6 Conclusions and Future Work

We have presented a discrete framework for incorporating constraints for form-features using hard constraints. Deformed shapes are calculated so that constraints on rotations and positions are satisfied. Constraints on rotations and positions are separately solved

as two sparse symmetrical matrices, which are known for the existence of efficient solvers. We showed how to constrain the shape and motion of form-features; linear shape constraints can constrain the shapes of form-features to the same shape and linear motion constraints confine motion to movement on a plane or a straight line. These constraints are convenient for deforming the 3D models of sheet metal panels.

In future work, it will be important to develop more intuitive tools for specifying form-features and constraints on mesh models. Since a complex product shape contains a considerable number of form-features, the automatic detection of features is preferable. In addition, it will be useful to incorporate a geometric reasoning engine into our framework, since commercial geometric engines are available and widely used. Well-known problems on hard constraints include the management of inconsistent and redundant constraints. Since we have developed a mechanism for resolving such constraints, we will incorporate it into our framework.

Acknowledgements

This work was partly funded by Mitsubishi Motor Corporation (MMC) and 3D models of automobile parts in Figure 6 and 7 are courtesy of MMC.

References

1. Sorkine, O., Lipman, Y., Cohen-Or, D., Alexa, M., Rössl, C., Seidel, H.-P.: Laplacian surface editing. in: SGP 2004: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, 2004, pp. 175–184
2. Botsch, M., Kobbelt, L.: An intuitive framework for real-time freeform modeling. ACM Transactions on Graphics **23 (3)** (2004) 630–634
3. Yu, Y., Zhou, K., Xu, D., Shi, X., Bao, H., Guo, B., Shum,H.-Y.: Mesh editing with poisson-based gradient field manipulation. ACM Transactions on Graphics **23 (3)** (2004) 644–651
4. Fontana, M., Giannini, F., Meirana, M.: A free form feature taxonomy. Computer Graphics Forum **18 (3)** (1999) 703–711.
5. Cavendish, J. C.: Integrating feature-based surface design with freeform deformation. Computer-Aided Design **27 (9)** (1995) 703–711
6. Meyer, M., Desbrun, M., Schröder, P., Barr, A. H. : Discrete differential-geometry operators for triangulated 2-manifolds. in: Visualization and Mathematics III, 2003, pp. 35–57
7. Sederberg, T. W. , Parry, S. R. : Free-form deformation of solid geometric models. in: Proceedings of SIGGRAPH 1986, 1986, pp. 151–160
8. Coquillart,S.: Extended free-form deformation: a sculpturing tool for 3D geometric modeling. in: Proceedings of SIGGRAPH 1990, 1990, pp. 187–196
9. MacCracken, R., Joy, K. I. : Free-form deformations with lattices of arbitrary topology. in: Proceedings of SIGGRAPH 1996, 1996, pp. 181–188
10. Hu, S.-M., Zhang,H., Tai, C.-L., Sun, J.-G. : Direct manipulation of ffd: Efficient explicit solutions and decomposible multiple point constraints. The Visual Computers **17 (6)** (2001) 370–379
11. Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., Stuetzle, W.: Multiresolution analysis of arbitrary meshes. in: Proceedings of SIGGRAPH 1995, 1995, pp. 173–182

12. D. Zorin, P. Schröder, W. Sweldens, Interactive multiresolution mesh editing. in: Proceedings of SIGGRAPH 1997, 1997, pp. 259–268.
13. Kobbelt, L., Campagna, S., Vorsatz, J., Seidel, H.-P. : Interactive multi-resolution modeling on arbitrary meshes. in: Proceedings of SIGGRAPH 1998, 1998, pp. 105–114
14. Guskov, I., Sweldens, W., Schröder, P.: Multiresolution signal processing for meshes. in: Proceedings of SIGGRAPH 1999, 1999, pp. 325–334
15. Lee, S.: Interactive multiresolution editing of arbitrary meshes. Computer Graphics Forum **18 (3)** (1999) 73–82.
16. Bloor, M. I. G. , Wilson, M. J. : Using partial differential equations to generate free-form surfaces. Computer-Aided Design **22 (4)** (1990) 202–212
17. Schneider, R., Kobbelt, L.: Generating fair meshes with g1 boundary conditions. in: GMP 2000: Proceedings of the international conference on Geometric Modeling and Processing, 2000, pp. 251–261
18. Desbrun, M., Meyer, M., Schröder, P., Barr, A. H. : Implicit fairing of irregular meshes using diffusion and curvature flow. in: Proceedings of SIGGRAPH 1999, 1999, pp. 317–324
19. Yamada, A., Furuhata, T., Shimada,K., Hou, K.-H. : A discrete spring model for generating fair curves and surfaces. in: Pacific Conference on Computer Graphics and Applications, 1999, pp. 270–279
20. Taubin, G.: A signal processing approach to fair surface design, in: Proceedings of SIGGRAPH 1995, 1995, pp. 351–358.
21. Catalao, C. E. , Falcidieno, B., Giannini, F., Monti, M.: A survey of computer-aided modeling tools for aesthetic design. Journal of Computer and Information Science in Engineering **2 (11)** (2002) 11–20.
22. Zhou, K., Huang, J., Snyder, J., Liu, X., Bao, H., Guo, B., Shum, H.-Y. : Large mesh deformation using the volumetric graph laplacian. ACM Transactions on Graphics **24 (3)** (2005) 496–503.
23. Alexa, M.: Differential coordinates for local mesh morphing and deformation. The Visual Computer **19 (2-3)** (2003) 105–114.
24. Botsch, M., Bommes, D., Kobbelt, L.: Efficient linear system solvers for mesh processing. in: IMA Conference on the Mathematics of Surfaces, 2005, pp. 62–83.
25. Welch, W., Witkin, A.: Variational surface modeling. in: Proceedings of SIGGRAPH 1992, 1992, pp. 157–166.
26. Yoshioka, Y., Masuda, H., Furukawa, Y.: A constrained least-squares approach to interactive mesh deformation. in: SMI 2006: Proceedings of the international conference on Shape Modeling and Applications, 2006 (to apprar).
27. Lipman, Y., Sorkine, O., Cohen-Or, D., Levin, D., Rössl, C., Seidel, H.-P. : Differential coordinates for interactive mesh editing. in: SMI 2004: Proceedings of the international conference on Shape Modeling and Applications, 2004, pp. 181–190.
28. Lipman, Y., Sorkine, O., Levin, D., Cohen-Or, D.: Linear rotation-invariant coordinates for meshes, ACM Transactions on Graphics **24 (3)** (2005) 479–487.
29. Zayer, R., Rössl, C., Karni, Z., Seidel, H.-P. : Harmonic guidance for surface deformation, Computer Graphics Forum **24 (3)** (2005) 601–609.
30. Gould, N. I. M. , Hu, Y., Scott, J. A.: A numerical evaluation of sparse direct solvers for the solution of large sparse, symmetric linear systems of equations. Tech. Rep. RAL-TR-2005-005, Council for the Central Laboratory of the Research Councils (2005).
31. Shoemake, K.: Animating rotation with quaternion curves. in: Proceedings of SIGGRAPH 1985, 1985, pp. 245–254.
32. Johnson, M. P. : Exploiting quaternions to support expressive interactive character motion, Ph.D. thesis, Massachusetts Institute of Technology, School of Architecture and Planning, Program in Media Arts and Sciences (February 2003).

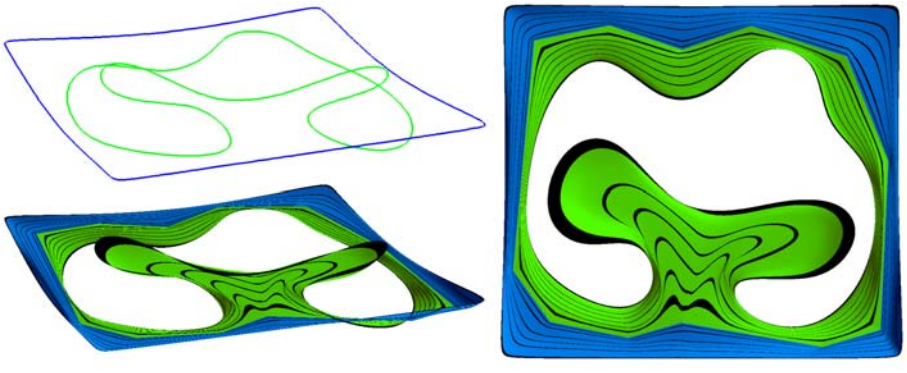
33. Pinkall, U., Polthier, K.: Computing discrete minimal surfaces and their conjugates, Experimental Mathematics **2 (1)** (1993) 15–36.
34. Mangan, A. P. , Whitaker, R. T. : Partitioning 3D surface meshes using watershed segmentation. IEEE Transaction on Visualization and Computer Graphics **5 (4)** (1999) 308–321.
35. Chazelle, B., Dobkin, D. P., Shouraboura, N., Tal, A.: Strategies for polyhedral surface decomposition: An experimental study. CGTA: Computational Geometry: Theory and Applications **7 (4-5)** (1997) 327–342.
36. Shlafman, S., Tal, A., Katz, S.: Metamorphosis of polyhedral surfaces using decomposition. Computer Graphics Forum **21 (3)** (2002) 219–228
37. Katz, S., Tal, A.: Hierarchical mesh decomposition using fuzzy clustering and cuts, ACM Transactions on Graphics **22 (3)** (2003) 954–961.
38. Katz, S., Leifman, G., Tal, A.: Mesh segmentation using feature point and core extraction. The Visual Computer **21 (8-10)** (2005) 649–658.
39. Masuda, H., Furukawa,Y., Yoshioka, Y., Yamato, H.: Volume-based cut-and-paste editing for early design phases. in: ASME/DETC2004/CIE: Design Engineering Technical Conferences and Computer and Information Engineering Conference, 2004.
40. Nealen, A., Sorkine, O., Alexa, M., Cohen-Or, D.: A sketch-based interface for detail-preserving mesh editing, ACM Transactions on Graphics **24 (3)** (2005) 1142–1147.
41. Hu, S.-M. , Li,Y., JU, T., Zhu, X.: Modifying the shape of nurbs surfaces with geometric constraints. Computer Aided Design **33 (12)** (2001) 903–912.
42. Sorkine, O.: Laplacian mesh processing, in: STAR Proceedings of Eurographics, 2005, pp. 53–70.
43. Toledo, S., Chen, D., Rotkin, V.: Taucs: A library of sparse linear solvers, http://www.tau.ac.il/ stoledo/taucs/ (2003).
44. Dam, E. B. , Koch, M., Lillholm, M.: Quaternions, interpolation and animation. Tech. Rep. DIKU-TR-98/5, Department of Computer Science, University of Copenhagen (1998).

Surface Creation and Curve Deformations Between Two Complex Closed Spatial Spline Curves

Joel Daniels II and Elaine Cohen

University of Utah

Abstract. This paper presents an algorithm to generate a smooth surface between two closed spatial spline curves. With the assumption that the two input curves can be projected to a single plane so that the projections do not have any mutual or self intersections, and so that one projection completely encloses the other. We describe an algorithm that generates a temporal deformation between the input curves, one which can be thought of as sweeping a surface. Instead of addressing feature matching, as many planar curve deformation algorithms do, the deformation method described generates intermediate curves that behave like wavefronts as they evolve from the shape of one boundary curve to a medial type curve, and then gradually take on the characteristics of the second boundary curve. This is achieved in a manner that assures there will be neither singularities in the parameterization nor self-intersections in the projected surface.



1 Introduction

This paper presents an algorithm to generate a parameterized spline surface between two closed spatial spline curves. The input curves are allowed to have differing degrees, parameterizations and shapes; however, they must project onto a single plane P without mutual or self intersections so that one projected curve completely encompasses the other. The algorithm creates a parameterized smooth spline surface, without any self-intersections, with the original closed curves as the boundaries.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 221–234, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Techniques to generate a surface from boundary curves are often used in geometric modeling. Usually, the interior of a surface is blended from four distinct boundary curves that correspond to mappings of the edges of the unit square. Our algorithm is quite different since it creates a surface between two quite dissimilar closed curves. This solution is related to surface completion solutions that build triangular patches to fill holes within a mesh [6,7]. These methods are useful in resolving gaps of a model when it is important to make it *water tight*.

Designing parting surfaces for injection molds is another modeling application that requires generating a surface between two closed spatial curves. An injection mold clamps two components together, forming a cavity that defines the object being molded. The parting curve of an object separates which parts of the object are represented in each of the mold pieces. The parting surface is the interface between the injection mold pieces that joins this parting curve with the mold's boundary curve. When these two curves are non-planar and when creating a ruled surface is not straightforward, then defining the parting surface can be technically challenging. Our surface generation solution creates surfaces under these very conditions.

Technique Overview. Conceptually, the surface generation algorithm builds two continuously deforming curves that sweep the output surface between the input spatial splines over time. Each curve deformation is constructed on P, the projection plane, and originates at the projection of one of the two input curves. The deformations can be developed between curves with little or no common feature characteristics. The properties of our planar deformations guarantee that the surface swept by the temporal curve deformation on P has no singularities in the parameterization and never self-intersects. Therefore, the final output is a smooth height field over P that defines a surface in $\mathcal{R}^3$ blending the two input spline curves.

2 Related Work

Surface Completion. Hole filling or surface completion techniques investigate problems that are related to the surface generation problem addressed here. There are two classes of research. One addresses holes bounded by piecewise linear curves and derives piecewise planar triangulated results while the other addresses holes or regions bounded by curves.

Triangulation methods exist for simple holes bounded by polylines; however, these solutions do not apply in more complex geometric configurations. Many techniques have addressed these challenges, for example, Davis et al. [7]; however, we are interested in higher degree boundary curves for which piecewise linear representations are not adequate.

One aspect of producing a spline blend surface between two closed curves requires constructing a tensor product parameterization. Using medial axis information to aid in forming a parameterization, [15] develops a radial parameterization of a region inside a planar boundary curve.

Curve Deformations. Curve deformation research can be divided into two separate topic areas: feature matching and vertex paths. Feature matching establishes a correspondence between the major features of two curves, for example [5,9,11,17]. These techniques are able to pair similarities between two planar curves. However, considering the deformations as surfaces, self-intersections frequently occur.

Consequently, the vertex path problem of curved deformations is more relevant to this research. Examples of this includes methods to prevent self-intersections intermediate shapes both locally [16] and globally [12]. While these methods produce better deformation paths, the surfaces swept by their algorithms may contain singularities. This research is constrained in that no self-intersections can occur on any intermediate shape, nor can a single point be on more than one temporal projected deformation curve.

Snakes and Active Contours. In this literature image processing methods place curves within an image, then evolve them until they are oriented along the edges of contrast between neighboring pixels. These curves, termed *snakes* or *active contours* [3,13,18,19], minimize energy functions that are computed over the image's pixels by using gradient descent methods. In the end, the active contours are used to segment the image into a set of coherent components.

Our method for generating the surfaces is similar to these curve evolution methods. Conceptually, we initialize active contours to the projection of the input curves and deform each snake towards the other. Because our work uses the deformation paths to parameterize a surface, it is necessary to address issues that do not concern the image processing community. These challenges include resolution of the surface singularities that occur from stationary pieces of an active contour, parameterization matching of multiple active contours, and height computations to create non-planar surfaces.

Robot Path Planning. Related to the vertex path problem is the robotic path planning problem of computing paths for robot seeds to navigate through an environment filled with obstacles represented by polygonal shapes. Chuang [4] assigns each obstacle a potential function by defining a scalar value that drops exponentially as the distance to the obstacle grows. The summation of these functions over all obstacles in the environment defines a *potential field*. The robot's motion is computed by gradient descent techniques.

To define a global minimum within the potential field, goal destinations have been assigned negative potential functions [10]. However, some formations of obstacles can create local minima that prevent trajectories from reaching their target destinations. In [14] Lewis and Weir use intermediate goals as *way-points* that can be used to navigate away from local minima.

Robot path planning research has also investigated the collaborated movement of multiple agents. Balch and Hybinette [1] extend potential fields for group path planning by locally repelling robots from each other and developing social potentials. Social potentials are attraction locations associated with the robots that show good results in achieving formations during travel. We extend the

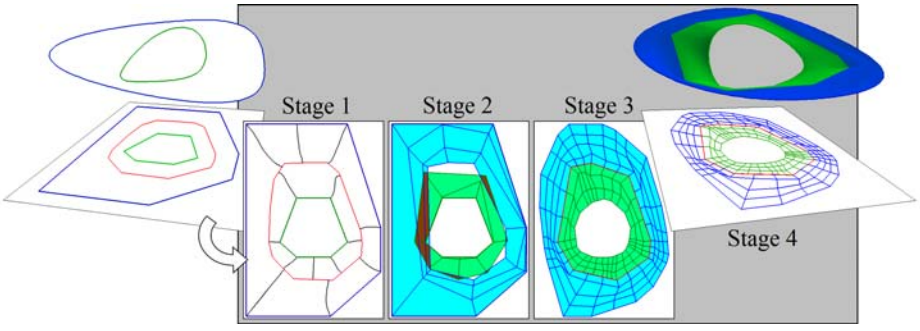


Fig. 1. The surface generation algorithm, composed of four stages, uses the projection of the input curves. (1) Compute vertex trajectories, (2) build a planar surface, (3) establishes C^0 continuity between the two surfaces, and (4) lift the surfaces into 3D.

concepts of robotic path planning techniques to address the vertex path problem of curve deformations within our algorithm for surface generation.

Medial Axis Approximations. The medial axis for a planar polygon shape is the set of points equidistant to 2, or more, points on the polygon [8]. There are many techniques to quickly compute piecewise linear medial axis approximations for planar polygonal shapes, which are adequate for our purposes as we project the curves to a plane. We assume that we are given an approximation of the medial axis of the region bounded by the projected curves.

3 Parametric Surface Generation

The algorithm generates a parameterized spline surface between the input curves, $\gamma_1(u)$ where $u \in I_u$ and $\gamma_2(v)$ where $v \in I_v$, by developing a curve deformation based on $\tilde{\gamma}_1(u)$ and $\tilde{\gamma}_2(v)$, the curves that have been projected on to P. Let $\tilde{C}_i = \{\tilde{c}_{i,j}\}_j$ denote the control polygon for $\tilde{\gamma}_i$. The technique, illustrated in Figure 1, can be decomposed into four main stages:

1. *Compute vertex paths from the control points of* $\tilde{\gamma}_1(u)$ *and* $\tilde{\gamma}_2(v)$ *to a medial axis approximating polyline between the projected curves.*
2. *Compute two planar curve deformations,* $\tilde{\sigma}_1(u,t)$ *and* $\tilde{\sigma}_2(v,t)$*, from the trajectories that sweep surfaces without overlaps.*
3. *Refine and relax* $\tilde{\sigma}_1(u,t)$ *and* $\tilde{\sigma}_2(v,t)$ *until* $\exists f : I_u \rightarrow I_v$ *such that* $\tilde{\sigma}_1(u,1) = \tilde{\sigma}_2(f(u),1)$ *and* f *is continuous and a bijection.*
4. *Lift* $\tilde{\sigma}_1(u,t)$ *and* $\tilde{\sigma}_2(v,t)$ *to define* $\sigma_1(u,t)$ *and* $\sigma_2(v,t)$ *as height fields above* P*, such that* $\sigma_1(u,0) \equiv \gamma_1$*,* $\sigma_2(u,0) \equiv \gamma_2$*, and* $\sigma_1(u,1) \equiv \sigma_2(f(u),1)$*.*

Consequently, the output surface is composed of two spline surfaces, $\sigma_1(u,t)$ and $\sigma_2(v,t)$. The following subsections describe each of these stages in further detail.

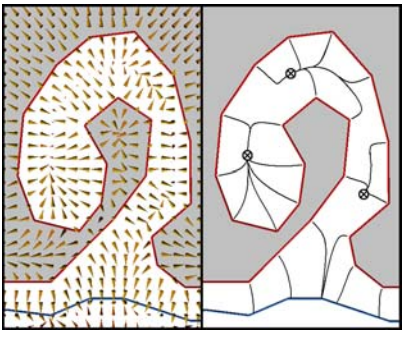


Fig. 2. (left) A visualization of V_r in a complex concavity, (right) where vertex trajectories travel towards local minima

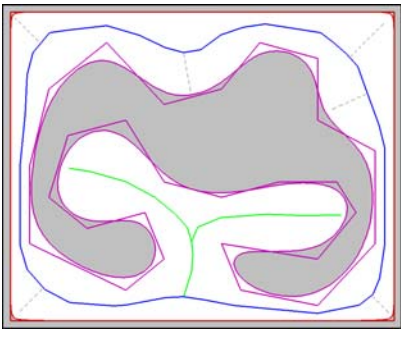


Fig. 3. Classified skeleton segments: *core edges* (blue), *branch edges* (green), and *extraneous edges* (dotted grey)

3.1 Stage 1: Vertex Trajectories

The goal of Stage 1 is to compute *vertex trajectories* for all $\tilde{c}_{i,j}$ of the two projected curves. These trajectories are defined as paths over time through a vector field and serve as a basis for the control mesh of the final output surface. The vertex trajectories begin at $\tilde{c}_{i,j}$ and traverse the vector field V towards an intermediate shape B which is piecewise linear. The following subsections further define B and explain how to construct a vector field in which the vertex trajectory curves are guaranteed to reach B and not intersect each other.

Vector Field Without Local Minima. Similar to the obstacles in a robot path planning environment, the segments of $\tilde{C}_1$ and $\tilde{C}_2$ are assigned a repulsion vector field. The i, j^{th} repulsion field is defined as:

$$v_{r,i,j}(\bar{x}) = \|\bar{x} - \bar{p}\|^{-k}\left(\frac{\bar{x} - \bar{p}}{\|\bar{x} - \bar{p}\|}\right),$$

where, the point $\bar{p}$ is the closest point on the line segment connecting $\tilde{c}_{i,j}$ and $\tilde{c}_{i,j+1}$ to the point, $\bar{x}$, at which the force is being evaluated, and the exponent, k, controls the exponential fall off. The final repulsion field, V_r, is the sum all $v_{r,i,j}$: $V_r(\bar{x}) = \sum_l v_{r,1,l}(\bar{x}) + \sum_k v_{r,2,k}(\bar{x})$.

Unfortunately, vertex trajectories that traverse V_r are not guaranteed to reach B. As in robot path planning, local minima may arise within the complex concavities formed by $\tilde{\gamma}_1(u)$ and/or $\tilde{\gamma}_2(v)$, illustrated in Figure 2.

Consequently, our algorithm requires a third input: a skeleton S of $\tilde{C}_1$ and $\tilde{C}_2$ on P. S is a medial axis approximation of the environment that is defined by a collection of polylines. The line segments, or edges, of S can be classified, illustrated in Figure 3, as one of three types: *core*, *branch*, or *extraneous*. A core edge is a line segment of S that divides $\tilde{C}_1$ from $\tilde{C}_2$ on P. A branch edge is a segment that separates $\tilde{C}_i$ from itself and is part of a polyline that originates at a core edge and terminates within a region of P that contains a local minimum. Lastly, an extraneous edge is a line segment that separates $\tilde{C}_i$ from itself but

is not part of a polyline ending near a local minimum. The vertex trajectory computation uses the classified edges of S to help define two additional vector fields V_a and V_f that overcome the motion towards local minima. We define a new vector field, $V = V_r + V_a + V_f$, without local minima through which vertex trajectories can be computed for $\tilde{c}_{i,j}$.

The set of core edges of S is a closed polyline on P between $\tilde{C}_1$ and $\tilde{C}_2$ that defines B with vertices $\{b_i\}_i$, the initial goal curve for the vertex trajectories. These edges are assigned global and local attraction functions, $v_{g,i}$ and $v_{l,i}$, to pull seed points towards their location. The final attraction vector field V_a is the sum of all $v_{g,i}$ and $v_{l,i}$: $V_a(\bar{x}) = \sum_i v_{g,i}(\bar{x}) + v_{l,i}(\bar{x})$.

The global attraction function attracts seed points no matter their distance from the segment and is defined as:

$$v_{g,i}(\bar{x}) = \|\bar{p} - \bar{x}\|^{-k}(\frac{\bar{p} - \bar{x}}{\|\bar{p} - \bar{x}\|}).$$

The $v_{g,i}$ is assigned to the segment between b_i and b_{i+1}, where $\bar{p}$ is the closest point on that segment to the input $\bar{x}$.

The second attraction function is designed to increase the local attraction field so that vertex trajectories within this region will snap to the segment in an appropriate place. The influence of the function defined on a segment of B is restricted to the viewable range of the segment. In this manner, the local attraction force disallows points from traveling parallel to a segment if they can be snapped immediately to B. For the segment connecting b_i to b_{i+1}, the local attraction function is defined as:

$$v_{l,i}(\bar{x}) = \begin{cases} (s(\bar{p})\cos(\frac{(r(\bar{p}) - \|\bar{p} - \bar{x}\|)}{r(\bar{p})}) + \frac{s(\bar{p})}{2.0})(\frac{\bar{p} - \bar{x}}{\|\bar{p} - \bar{x}\|}) & for\ \|\bar{p} - \bar{x}\| \leq r(\bar{p}), \\ (0,0) & otherwise, \end{cases}$$

where, $\bar{p}$ is the closest point on the segment to $\bar{x}$, $r(\bar{p})$ determines the radius of influence of the segment at $\bar{p}$, and $s(\bar{p})$ is a scalar value that controls the magnitude of the vectors. The linear interpolating function r is defined on each segment such that $r(b_i) = d_{curve}(b_i)$ and $r(b_{i+1}) = d_{curve}(b_{i+1})$, where $d_{curve}(\bar{y})$ is equal to the distance from the point, $\bar{y}$, to the closest point on $\tilde{C}_1$ and $\tilde{C}_2$. Similarly, the magnitude scalar function, s, is a linear interpolant between $s(b_i)$ and $s(b_{i+1})$. In practice we achieve good results by assigning $s(b_i) = r(b_i)$ and $s(b_{i+1}) = r(b_{i+1})$.

Branch segments of S are used to guide seed points out of complex concavities by defining *flow fields*. The i^{th} flow field is a locally defined vector field parallel assigned to a branch segment that connects points q_i to q_j of S:

$$v_{f,i}(\bar{x}) = \begin{cases} (s(\bar{p})\cos(\frac{(r(\bar{p}) - \|\bar{p} - \bar{x}\|)}{r(\bar{p})}) + \frac{s(\bar{p})}{2.0})(\frac{\bar{q_j} - \bar{q_i}}{\|\bar{q_j} - \bar{q_i}\|}) & for\ \|\bar{p} - \bar{x}\| \leq r(\bar{p}), \\ (0,0) & otherwise, \end{cases}$$

where, $\bar{p}$ is the closest point on the segment to $\bar{x}$. The orientation of q_i and q_j is chosen so the flow field is parallel to the polyline and directed towards a core edge of S (i.e. B). The maximum magnitude function, s, is evaluated for each

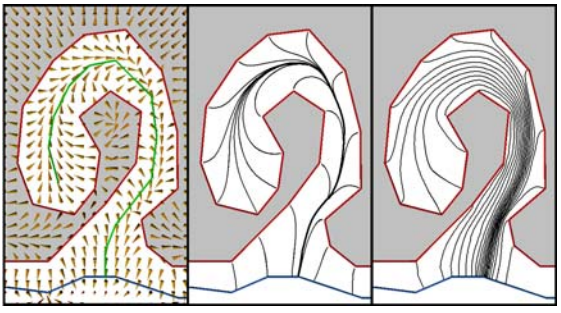


Fig. 4. The same concave section of curve as Figure 2 with (left) a visualization our vector field construction without local minima, (center) vertex trajectories trace to B, (right) seed repulsion avoid intersections that may result from numerical integration

segment to guarantee that $s(q_i) > \|V_r(q_i)\|$ and $s(q_j) > \|V_r(q_j)\|$. Therefore, the flow field, $V_f(\bar{x}) = \sum_i v_{f,i}((x))$, overpowers the motion towards local minima of V_r and guarantees that trajectories will be able to reach B. The radius of influence of each $v_{f,i}$ is measured at the endpoints, in the same manner as $v_{l,i}$, such that $r(q) = d_{curve}(q)$. Consequently, the influence of a flow field is restricted to the concave region in which the branch edge resides.

The extraneous edges of S are ignored. Since no local minima exist in these regions of V_r, seed points are not be trapped in their neighborhood.

The resulting vector field, V, is summed over all assigned forces, $V(\bar{x}) = \sum_k v_{r,1,k}(\bar{x}) + \sum_l v_{r,2,l} + \sum_m (v_{l,m}(\bar{x}) + v_{g,m}(\bar{x})) + \sum_n v_{f,n}(\bar{x})$, and does not contain local minima (see Figure 4). In this way, seed points placed within the environment, on P, can be guaranteed to trace successfully to B.

Neighbor Avoidance Path Computation. Vertex trajectories computed within a continuous vector field can be guaranteed not to cross in their limit by the *Uniqueness Theorem* [2]. We modify the vector field functions, $v_{r,i,j}$, $v_{g,i}$, $v_{l,i}$, and $v_{f,i}$, to satisfy the constraints of the Uniqueness theorem in Appendix A.

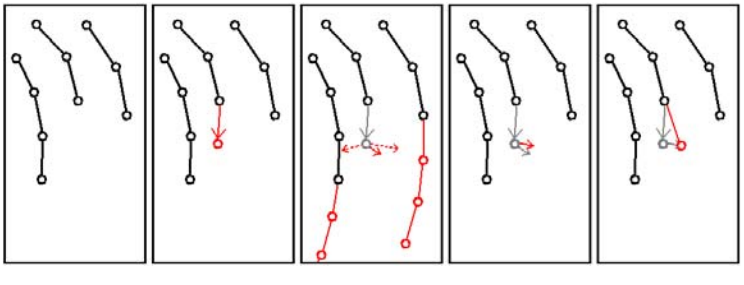


Fig. 5. Discrete vertex step computation for neighbor avoidance

Then in theory, the vertex trajectories will not intersect each other. However, numerical integration methods used to compute these paths take discrete sampling steps. Even with fifth order Runge-Kutta approximation, there is enough error in some pathological cases to cause path crossings. Consequently, additional efforts are required to prevent such anomalous path intersections.

To avoid this problem, we assigns a repulsion force, identical to a scaled versions of that assigned to the core edges of S, to the vertex trajectories based on proximity of nearest neighbors. While these forces are able to prevent vertex path intersections, they may introduce new local minima within V. The algorithm, illustrated in Figure 5, presents a method to compute the discrete steps of the vertex trajectories that avoids the introduction of local minima. It applies these forces only in the direction perpendicular to the desired direction of travel as a 4 step process:

1: *Approximate the desired step vector, $\bar{s}$, of a vertex point, $\bar{p}$, through V with fifth order Runge-Kutta (see Figure 5a).*
2: *Extend the two neighbor paths of the vertex by estimating a number of future steps, then compute their repulsion force, $\bar{r} = \sum v'_g(\bar{s}+\bar{p})$, at the desired step point (see Figure 5b).*
3: *Compute the component of $\bar{r}$, $\bar{r}'$, perpendicular to $\bar{s}$ (see Figure 5c).*
4: *Move the vertex point, $\bar{p} = \bar{s} + \bar{r}'$ (see Figure 5d).*
5: *Repeat steps 1-4 until all trajectories intersect B.*

This new vertex trajectory computation method relaxes the paths as they travel through V. Forward progress is not hindered by the repulsion forces as only the component perpendicular $\bar{s}$ is considered. Additionally, the estimation of future steps for the neighbor paths aids in ensuring that the $\bar{r}'$ will not allow the vertex path to cut in front of another trajectory. The neighbor avoidance and step computation successfully navigates groups of vertices through V, while evenly spreading the distances between paths that had been previously congested, as illustrated in Figure 4.

3.2 Stage 2: Planar Spline Surface Creation

Stage 2 generates a planar parameterization of a spline surface, $\tilde{\sigma}_i$ from the vertex trajectories associated with the control points of $\tilde{\gamma}_i$. The knot vector and degree of $\tilde{\sigma}_i$ equals that of $\tilde{\gamma}_i$ in the first parametric variable. This section describes the construction of a control mesh, $\tilde{M}_i$, that will be associated with $\tilde{\sigma}_i$. Consequently, it is necessary that $\tilde{M}_i$ does not contain any intersections between its row and column edges so the spline variation diminishing property can be used to guarantee that $\tilde{\sigma}_i$ is without self-intersections.

We extend a greedy data reduction algorithm to approximate the vertex trajectories with polylines that define the columns of $\tilde{M}_i$. Greedy data reduction of a curve is done by fitting the longest line segments to the curve without violating

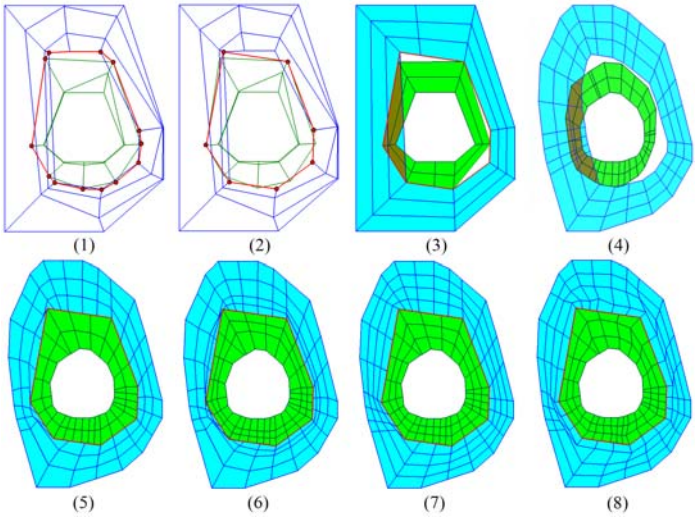


Fig. 6. 1-5 compute a C^0 boundary between $\tilde{\sigma}_1$ and $\tilde{\sigma}_2$, and 6-8 align tangents

a curvature approximation tolerance. Our extension performs the data reduction on the vertex trajectories simultaneously and adds a new violation metric.

The algorithm initializes $\tilde{M}_i$ to $\tilde{C}_i$. We advance the data reducing line segments over all vertex trajectories in parallel. The approximation line segments are appended to the columns of $\tilde{M}_i$ if advancing these line segments one more position will violate one of two conditions on any vertex trajectory. The first trigger condition is the classic curvature threshold test of the greedy algorithm. This guarantees that the polylines determined by the data reduction closely approximate the computed vertex trajectories. In fact, the column polylines will converge towards the computed trajectories as the threshold value is lowered to zero. The second trigger condition tests that the approximating line segments do not intersect each other, nor the established rows and columns of the growing $\tilde{M}_i$. This condition guarantees that the output mesh will not have any intersections. While this process is described as a second independent stage, we run the creation phase in parallel with Stage 1.

3.3 Stage 3: Boundary Computation

The goal of Stage 3 is to modify the control meshes, $\tilde{M}_1$ and $\tilde{M}_2$, to have $\tilde{\sigma}_1(u,1) \equiv \tilde{\sigma}_2(v,1)$ as they meet along the common boundary curve B'. Because there is no restriction on the parameterizations and degrees of γ_i, B' must be a piecewise linear. Each surface will degree-raise B', so $\tilde{\sigma}_i$ and the degree-raised B' will have matching knot vectors. Each column that terminates at a vertex of B' will have a knot value with multiplicity equal to the degree of $\tilde{\sigma}_i$. These columns will be referred to as *multi-knot columns*. At the multi-knot columns, $\tilde{\sigma}_i$ breaks C^1 continuity as a function of u (or v) to form the piece-wise linear shape of B' despite the degree and parameterization of $\tilde{\sigma}_i$.

Stage 3 consists of eight steps, illustrated in Figure 6. The first five steps align σ_1 and σ_2 along a boundary with C^0 continuity, while the final three steps define tangents across this boundary. The tangents are defined such that they are equal in direction (but not magnitude) to facilitate the construction of a G^1 continuous boundary in Stage 4. The boundary alignment steps are as follows:

1: *Compute a polyline, B', along B that connects all control points in the final row of $\tilde{\sigma}_1$ and $\tilde{\sigma}_2$.*

2: *Data reduction method removes extraneous points from B'.*

3: *For each vertex of B', find the closest control point in the final row of each $\tilde{\sigma}_i$. If point within a threshold distance, slide it to the vertex and relax the column.*

4: *Perform knot insertion of degree multiplicity on $\tilde{\sigma}_i$ to create the multi-knot columns that will align closely to the vertices of B'.*

5: *Slide the new multi-knot columns to the vertices of B'.*

6: *Insert a new row between the last and second to last rows of $\tilde{M}_i$.*

7: *For each multi-knot column, compute the halfway vector, $\bar{h}$, of the neighbor edges in the final row. Slide the inserted control point along $\bar{h}$ until the length of the growing column edge is equal to original length or an intersection in the mesh is created.*

8: *For each non-multi-knot column, slide the inserted control point along the vector, $\bar{h}'$, computed as a linear combination of the previous and next multi-knot columns' $\bar{h}$.*

Stage 3 modifies $\tilde{\sigma}_1$ and $\tilde{\sigma}_2$, so they continuously span the region between $\tilde{\gamma}_1$ and $\tilde{\gamma}_2$ without self or mutual intersections. They meet along a polyline, B', such that a C^0 boundary can be achieved despite the parameterizations and degree of each $\tilde{\sigma}_i$, as shown in Figure 6 (step 8). Stage 3 performs the point sliding and relaxation routines incrementally while modify surrounding rows and columns to ensure that no overlaps are created.

3.4 Stage 4: 3D Surface Creation

The final stage lifts the planar surface parameterizations into 3D as height fields above P, who's unit normal is N. The surfaces, $\tilde{\sigma}_1$ and $\tilde{\sigma}_2$ do not contain foldovers on P, and therefore, their corresponding 3D height fields, σ_1 and $\sigma 2$, are well defined.

The computation of height components for the planar parameterizations is segmented into four steps as listed and illustrated in Figure 7. The final height field, the surface generation output, is composed of σ_1 and σ_2. The height computations smooth the output surface so that it is G^1 continuous at all points except for small regions around the vertices of the piece-wise linear boundary curve shared by σ_1 and σ_2.

9: *Add a vector in the direction of N to each control point in the first row of $\tilde{\sigma}_i$, such that $\sigma_i(u,0) = \gamma_i(u)$.*

10: *Add a vector in the direction of N to each control point in the final row of the two surfaces, so $\sigma_1(u,1) \equiv \sigma_2(v,1)$, and is a weighted average of the heights assigned to $\sigma_1(u,0)$ and $\sigma_2(v,0)$.*

11: *Add a vector in the direction of N to the remaining control points as the weighted average of the heights of previously computed control points. The height components computed in the previous steps for the first and last rows of M_i serve as the original sources for this height diffusion.*

12: *Relax heights across multi-knot columns of σ_i and across the boundary shared by σ_1 and σ_2 for G^1 continuity where possible.*

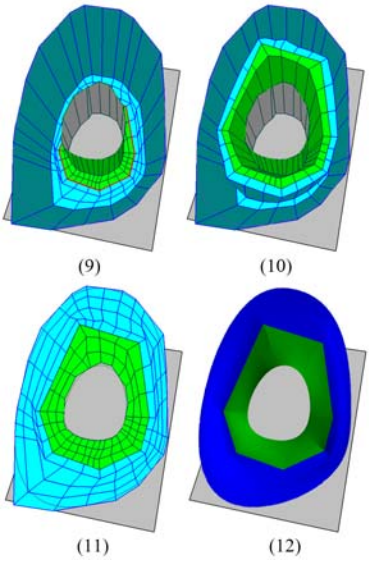


Fig. 7. Steps 9-12 raise the planar parameterizations into 3D

4 Conclusion

This paper presents a novel method of spline surface generation between two closed spatial boundary curves. The algorithm develops a spatial curve deformation solution that sweeps the well-behaved surfaces illustrated in Figure 8,

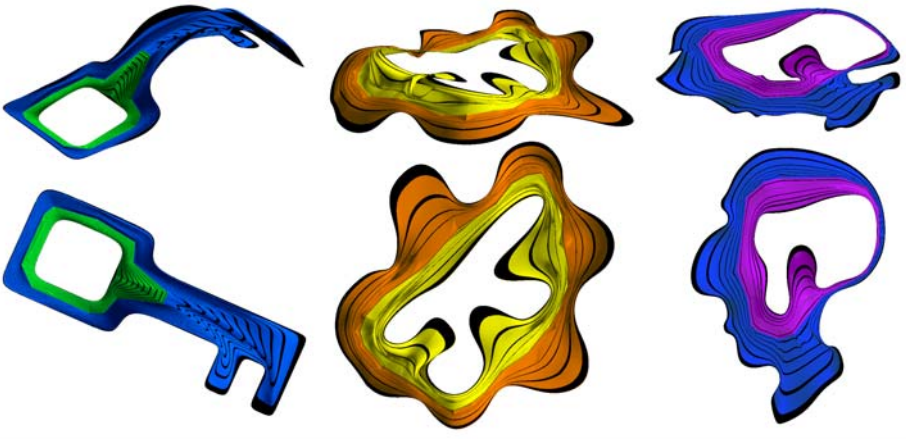


Fig. 8. 3D generated surfaces (and curve deformations as black iso-curves) spanning between spatial curves, viewed from an angle and from above

by guaranteeing that intermediate curve shapes do not contain local or global intersections between themselves nor each other.

The algorithm builds upon concepts traditionally found in robot path planning, addressing navigational challenges associated with their gradient descent methods. Our approach develops vector fields without local minima that guarantee multiple vertices can simultaneously navigate through the environment to their goal without intersections.

This surface generation solution is contains concepts that are applicable to navigational challenges and curve deformations, but is also useful in geometric modeling areas. Our algorithm, through implementation decisions and guarantees, is able to automatically generate smooth parting surfaces for injection molds. The results directly parameterize the surface without the use of trimmed NURBS, while handling the complex concavities cause most other techniques to behave poorly.

Limitations. Our algorithm requires a medial axis approximation to construct a vector field without local minima and where the global minima represents the goal destinations. The medial axis can be computed quickly for planar polygonal shapes, but may be unstable in dynamic scenes. Many of the changes translate to modifications of extraneous edges, in which case our algorithm operates unchanged. However, the birth or death of a branch segment may pose a computational challenge.

Implementation decisions limit the application of our algorithm. Because we built this system in order to produce parting surfaces for automated mold designing, the algorithm must guarantee that the projection of the resulting surface is without self-intersections. We plan to remove this restriction for future application.

Future Work. This is a novel approach to surface generation and curve deformations that is suggestive of multiple avenues of future work. We plan to improve the algorithm allowing for wider range of input curve types and to develop parameterizations directly in $\mathcal{R}^3$ rather than on P. The concepts that deal with curve deformation can benefit enforcement of a feature match, as well as extension to surface-to-surface deformations.

Acknowledgments. This work was supported in part by NIH 573996 and NSF IIS0218809. All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

References

1. Balch, T., and Hybinette, M. Social potentials and scalable multi-robot formations. *ICRA* (2000).
2. Blanchard, P., Devaney, R., and Hall, G. *Differential Equations.* International Thomson Publishing Inc., 1998.
3. Caselles, V., Kimmel, R., and Sapiro, G. Geodesic active contours. *International Journal of Computer Vision 22*, 1 (February 1997), 61–79.

4. Chuang, J. Potential-based modeling of three-dimensional worksapce for obstacle avoidance. *IEEE Transactions on Robtics and Automation 14*, 5 (October 1998), 778–785.
5. Cohen, S., Elber, G., and Yehuda, R. Matching of freeform curves. *Computer Aided Design 29*, 5 (1997), 369–378.
6. Curless, B., and Levoy, M. A volumetric method for building complex models from range images. In *Proceedings of SIGGRAPH '96* (1996), pp. 303–312.
7. Davis, J., Marschner, S., Garr, M., and Levoy, M. Filling holes in complex surfaces using volumetric diffusion. In *Proceedings of the First International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT-02)* (June 2002), IEEE Computer Society, pp. 428–438.
8. de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. *Computational Geometry: Algorithms and Applications*. Springer-Verlag Berlin Heidelberg, 1997.
9. Elber, G. Metamorphosis of freeform curves and surfaces. *Computer Graphics International '95* (June 1995), 29–40.
10. Ge, S., and Cui, Y. New potential functions for mobile path planning. *IEEE Transactions on Robotics and Automation 16*, 5 (2000), 615–620.
11. Goldstein, E., and Gotsman, C. Polygon morphing using a multiresolution representation. *Graphics Interface* (1995), 247–254.
12. Gotsman, C., and Surazhsky, V. Guaranteed intersection-free polygon morphing. *Computers and Graphics 25* (2001), 67–75.
13. Kass, M., Witkin, A., and Terzopolous, D. Snakes: Active contour models. *International Journal of Computer Vision 1*, 4 (January 1988), 321–331.
14. Lewis, J., and Weir, M. Using subgoal chaining to address the local minimum problem. In *Proceedings of Second International ICSC Symposium on Neural Computation* (2000).
15. Martin, W., and Cohen, E. Surface completion of an irregular boundary curve using a concentric mapping. In *Fifth International Conference on Curves and Surfaces* (June 2002).
16. Sederberg, T., Gao, P., Wang, G., and Mu, H. 2d blending shapes: An intrinsic solution to the vertex path problem. In *Proceedings of SIGGRAPH '93* (1993), vol. 27, pp. 15–18.
17. Sederberg, T., and Greenwood, E. A physically based approach to 2d shape blending. *Computer Graphics 26*, 2 (July 1992), 25–34.
18. Tsai, A., Yezzi, A., and Willsky, A. Curve evolution implementation of mumford-shah functional for image segmentation, denoising, interpolation, and magnification. *IEEE Transactions on Image Processing 10*, 8 (August 2001).
19. Xu, C., and Prince, J. Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing 7* (March 1998), 359–369.

5 Appendix A. Continuous Vector Field

The advantage of computing vertex trajectories within a vector field is that it can be guaranteed that no two paths will intersect through the *Uniqueness Theorem*[2] if the vector field is C^1 continuous. Therefore, we modify the functions, v_r, v_g, v_l, and v_f, to guarantee that $\frac{\partial V}{\partial t}$ is continuous. The vector functions are

re-defined in terms of a new coordinate system that aligns the line segment along the x-axis such that $\bar{q_1} = (0,0)$ and $\bar{q_2} = (a,0)$, for example:

$$v_{r,i,j}(x,y) = \begin{cases} \|(x,y)\|^{-k}(\frac{(x,y)}{\|(x,y)\|}) \; for \; x < 0 \\ (|y|)^{-k}(0, \frac{y}{|y|}) \; for \; 0 \leq x < a \\ \|(x-a,y)\|^{-k}(\frac{(x-a,y)}{\|(x-a,y)\|}) \qquad otherwise, \end{cases}$$

Each of the transformed branch functions share a common structure:

$$F(x,y) = \begin{cases} f_1(x,y) \; for \; x < a \\ f_2(x,y) \; for \; a \leq x < b \\ f_3(x,y) \qquad otherwise. \end{cases}$$

However, for each function it can be shown that $\frac{\partial f_1(a,y)}{\partial x} \neq \frac{\partial f_2(a,y)}{\partial x}$, and $\frac{\partial f_2(b,y)}{\partial x} \neq \frac{\partial f_3(b,y)}{\partial x}$. Therefore, two transition functions are defined to obtain C^1 continuity between the consecutive branch functions at $x = a$ and $x = b$. The new vector field functions will have the form:

$$G(x,y) = \begin{cases} f_1(x,y) \; for \; x < p_{1b} \\ t_{1,2}(x,y) \; for \; p_{1b} \leq x < p_{2a} \\ f_2(x,y) \; for \; p_{2a} \leq x < p_{2b} \\ t_{2,3}(x,y) \; for \; p_{2b} \leq x < p_{3a} \\ f_3(x,y) \qquad otherwise, \end{cases}$$

where, the transition function, $t_{i,j}(x,y)$, creates a C^1 continuous bridge between functions $f_i(x,y)$ and $f_j(x,y)$. In other words, a transition function, $t_{i,j}(x,y)$, must satisfy the following set of conditions:

$$t_{i,j}(p_{ib},y) = f_i(p_{ib},y), \;\; \frac{\partial t_{i,j}}{\partial x}(p_{ib},y) = \frac{\partial f_i}{\partial x}(p_{ib},y), \;\; \frac{\partial t_{i,j}}{\partial y}(p_{ib},y) = \frac{\partial f_i}{\partial y}(p_{ib},y),$$
$$t_{i,j}(p_{ja},y) = f_j(p_{ja},y), \; \frac{\partial t_{i,j}}{\partial x}(p_{ja},y) = \frac{\partial f_j}{\partial x}(p_{ja},y), \; \frac{\partial t_{i,j}}{\partial y}(p_{ja},y) = \frac{\partial f_j}{\partial y}(p_{ja},y).$$

It can be shown that a transition function defined as:

$$t_{i,j}(x,y) = f_i(x,y)(\frac{x-p_{ja}}{p_{ib}-p_{ja}})^2 + s(x,y)(\frac{x-p_{ja}}{p_{ib}-p_{ja}})(\frac{x-p_{ib}}{p_{ja}-p_{ib}}) + f_j(x,y)(\frac{x-p_{ib}}{p_{ja}-p_{ib}})^2,$$
$$s(x,y) = -2f_i(x,y)\frac{(x-p_{ja})(p_{ja}-p_{ib})}{(p_{ib}-p_{ja})^2} - 2f_j(x,y)\frac{(x-p_{ib})(p_{ib}-p_{ja})}{(p_{ja}-p_{ib})^2},$$

satisfies the required list of conditions. Therefore, re-writing the vector field functions, v_r', v_g', v_l' and v_f', such that they are C^1 continuous only requires substituting into the generic branch function, $G(x,y)$, where the transition functions are defined above. The transition functions are used to span over some ϵ distance such that $p_{1b} = -\epsilon$, $p_{2a} = 0$, $p_{2b} = a$, and $p_{3a} = a + \epsilon$.

Because the final vector field functions are each individually C^1 continuous, their sum, V, will also be continuous. The uniqueness theorem guarantees that all seeds dropped within V will trace non-intersecting paths.

Computing a Family of Skeletons of Volumetric Models for Shape Description

Tao Ju[1], Matthew L. Baker[2], and Wah Chiu[2]

[1] Washington University, St. Louis, MO
[2] Baylor College of Medicine, Houston, TX

Abstract. Skeletons are important shape descriptors in object representation and recognition. Typically, skeletons of volumetric models are computed via an iterative thinning process. However, traditional thinning methods often generate skeletons with complex structures that are unsuitable for shape description, and appropriate pruning methods are lacking. In this paper, we present a new method for computing skeletons on volumes by alternating thinning and a novel skeleton pruning routine. Our method creates a family of skeletons parameterized by two user-specified numbers that determine respectively the size of curve and surface features on the skeleton. As demonstrated on both real-world models and medical images, our method generates skeletons with simple and meaningful structures that are particularly suitable for describing cylindrical and plate-like shapes.

1 Introduction

Representing and understanding shapes play central roles in many of today's graphics and vision applications. These applications typically benefit from some form of shape descriptors, one of which is known as skeletons. A skeleton is a compact, medial structure that lies within a solid object [1]. A skeleton of a 2D object consists of 1D (e.g., curve) elements, whereas the skeleton of a 3D object may consist of both 1D and 2D (e.g., surface) elements. A skeleton that captures essential topology and shape information of the object in a simple form is extremely useful in solving various problems such as character recognition, 3D model matching and retrieval, and medical image analysis.

Here we consider computing skeletons of a 3D object consisting of lattice points in a binary volume. While there has been numerous work on 2D skeletonization (see an excellent survey in [2]), methods for computing 3D skeletons are relatively scarce. Skeletonizing volumetric models often utilizes a *thinning* process that iteratively removes deletable object points until a thin, skeletal structure is left. To preserve the object's topology during thinning, deletable points must be *simple* [3,4,5] in that their removal does not invoke topology change. In addition, to prevent shrinking of curves or surfaces that may carry the object's shape information, various types of *curve-end points* or *surface-end points* [3,6,7,8,9,10] can be identified and preserved during thinning. Thinning methods can also be classified as sequential [11], meaning deletable points are identified and removed one after another, or parallel (see review in [12]), meaning all deletable points are collected and removed at once in each thinning iteration.

Unfortunately, the skeletons computed by thinning methods often exhibit a complex structure. For example, Figure 1 (b) shows the result of the parallel thinning method

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 235–247, 2006.
© Springer-Verlag Berlin Heidelberg 2006

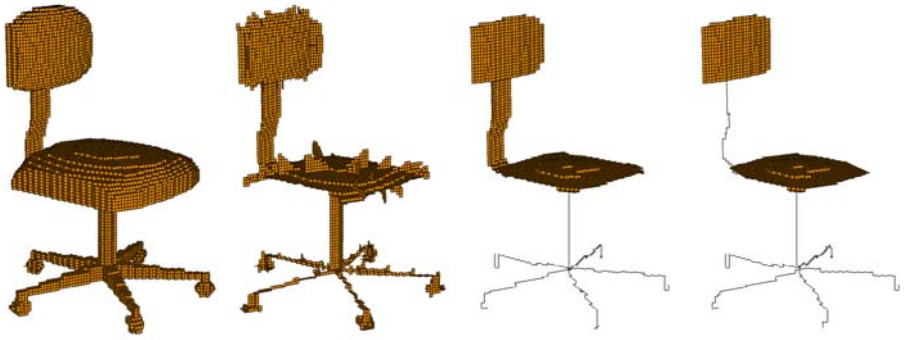


Fig. 1. A volumetric chair model (a), the skeleton computed using the parallel thinning method of Bertrand [9] (b), and two skeletons computed using our method with parameters $d_1 = 20, d_2 = 4$ (c) and $d_1 = 20, d_2 = 7$ (see explanations in Section 3.2)

of Bertrand [9] on the chair model in (a). Note in particular that the spurious surface branches do not provide meaningful information about the original object. Moreover, the chair legs have cylindrical shapes and can be represented in a simpler and more descriptive manner by 1D curves. Typically, a thinning method computes one skeleton for each object, leaving little room for adjustment. Although numerous pruning methods exist for 2D skeletons [13,14], for skeletons of 3D point-set and polygonal models [15,16,17,18,19,20,21] or for removing only curve branches from skeletons of volumetric models [22,23], pruning both redundant curve and surface features from skeletons of volumetric models has not been addressed so far.

Our main contribution is a new method for computing skeletons of a volumetric model by incorporating iterative thinning with skeleton pruning. The core of our method is a simple and efficient pruning algorithm capable of removing redundant curve features as well as surface features from an arbitrary skeleton. Pruning involves morphological erosion and dilation, and is governed by two user-specified parameters that allow flexible control over the size of skeleton curves and surfaces. Pruning and thinning are combined in alternating steps to compute simple, topology-preserving and shape-depicting skeletons. Figure 1 (c,d) give two example skeletons computed using our method with different pruning parameters.

An interesting application that we shall explore is the use of our computed skeletons in describing cylindrical and plate-like shapes. Models composed of these two shapes are not only common in our everyday life, such as chairs and tables, but also appearing often in medical imaging, such as the bone matrix [24,25] and the protein images that we shall see in Section 5. While varying the parameters in our method will produce a large family of skeletons, reasonable choices of parameters yield descriptive skeletons whose curves and surfaces correspond well to the cylindrical and plate-like shape components of the object, as seen in Figure 1 (c,d). In particular, the pruning parameters in our method enable flexible adjustment in the differentiation between cylinders and plates. We note that there have been previous attempts for identifying these two types of shapes directly using an un-pruned skeleton via classifying skeleton points [24,25].

However, as seen in Figure 1 (b), such attempts may easily yield incorrect results due to the complexity of a skeleton computed by thinning.

2 Digital Topology and Discrete Geometry

Here we briefly review basic concepts in digital topology that are fundamental to thinning-based skeletonization algorithms. As part of our own contribution, we follow by introducing several new notions for describing curves and surfaces on volumes.

2.1 Simple Points

We consider a volumetric model as a uniform 3D lattice consisting of *object points* V and *background points* $\overline{V}$. For topology analysis, we classify the $3\times3\times3$ neighborhood of each point x into three (overlapping) sets, $N_6(x), N_{18}(x)$ and $N_{26}(x)$, each consisting of points (other than x) that share a common grid edge, face or cell with x. The three sets are illustrated in Figure 2 (a). In addition, we denote $N_k(x,V) = N_k(x) \cap V$.

The topology of the object V is determined by the connectivity of points in V. We regard two points x, y as k-connected if $y \in N_k(x)$ for $k = 6, 18, 26$. In this paper we assume that the object points are 6-connected and the background points are 26-connected, so that the topology of the set V agrees with that of the iso-surface of the whole volume generated by mainstream contouring techniques, such as the Marching Cubes method and its variants [26,27]. In order to maintain topology of V during skeletonization, we follow the result of Bertrand [5] to determine if a point $x \in V$ is *simple* with respect to V, that is, if $V \setminus \{x\}$ preserves the topology of V:

Proposition 1. *Let $s(V)$ denote the set of simple points of V, then $x \in s(V)$ if and only if $c_6(N_6^2(x,V)) = 1$ and $c_{26}(N_{26}(x,\overline{V})) = 1$. Here $c_k(X)$ computes the number of k-connected components in point set X, and*

$$N_6^2(x,V) = N_6(x,V) \cup \bigcup_{y \in N_6(x,V)} M(x,y,V),$$

where $M(x,y,V) = N_{18}(x,V) \cap N_6(y,V)$.

2.2 Discrete Curves and Surfaces

A curve in an object V is a connected collection of object *edges*, each composed of two 6-connected object points, and a surface in V is a connected collection of object *faces*, each composed of four 18-connected object points. In Figure 2 (b,c), edges are shown as dark lines and faces as orange polygons.

Knowledge of curves and surfaces is critical as they are the components of a skeleton that carry shape information of the object. To prevent loss of such information during thinning, it is important to avoid shrinking of the curves and surfaces. While various definitions of curve-end and surface-end points have been proposed in previous work, few are designed for 6-connected objects (except in [9]). In our new definition, a set $S \in N_6(x,V)$ is called *locally 1-manifold* if $S \cup \{x\}$ forms two edges containing x, and

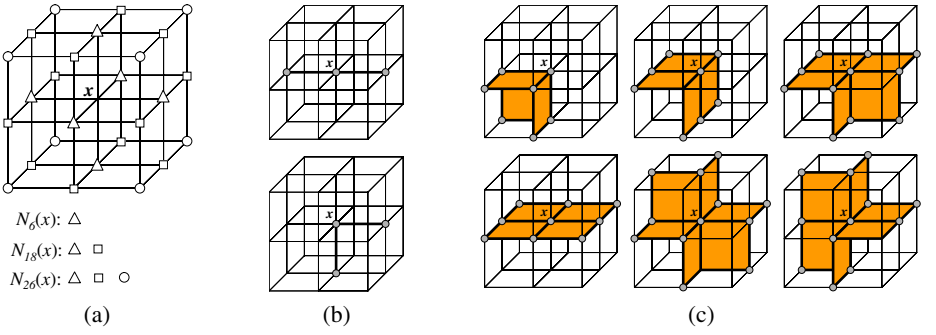


Fig. 2. (a): The 6, 18 and 26 neighborhoods $N_k(x)$. (b): Examples of locally 1-manifold subset of $N_6(x,V)$. (c): Examples of locally 2-manifold subset of $N_{18}(x,V)$.

$S \in N_{18}(x,V)$ is called *locally 2-manifold* if $S \cup \{x\}$ forms a ring of faces containing x that are topologically equivalent to a disk. Figure 2 (b,c) enumerate all rotationally distinct cases of local manifolds. A point $x \in V$ is therefore a *curve-end* (or *surface-end*) point if $N_6(x,V)$ (or $N_{18}(x,V)$) does not contain any locally 1-manifold (or 2-manifold) subset. Intuitively, there exists no curve that passes through a curve-end point, and no surface that completely surrounds a surface-end point. Both curve-end and surface-end points can be efficiently identified as follows (see proof in Appendix A):

Proposition 2. *Let $\partial_m(V)$ denote the set of curve-end ($m = 1$) or surface-end ($m = 2$) points of an object V, for any $x \in V$ we have*

1. *$x \in \partial_1(V)$ if and only if $\#N_6(x,V) < 2$.*
2. *$x \in \partial_2(V)$ if and only if $r(x,V) = 0$ where*

$$r(x,V) = \#N_6^2(x,V) - c_6(N_6^2(x,V)) - \sum_{y \in N_6(x,V)} \#M(x,y,V)$$

Here # computes set cardinality and $N_6^2(x,V), M(x,y,V)$ are defined in Proposition 1.

Proof

1. By definition, x is not a curve-end point if $N_6(x,V)$ contains more than two points, which form at least two edges with x.
2. First we observe that the quantity $\sum_{y \in N_6(x,V)} \#M(x,y,V)$ equals to the number of edges formed by points in the set $N_6^2(x,V)$. Consider the graph formed by the points and edges in $N_6^2(x,V)$, $r(x,V)$ in fact computes the number of closed rings of edges in the graph. Furthermore, each such ring is a locally 2-manifold subset of $N_{18}(x,V)$. Hence $x \in \partial_2(V)$ if and only if $N_6^2(x,V)$ does not contain any rings, or $r(x,V) = 0$. □

In addition to curve-end and surface-end points, we are also interested in the local neighborhood of each interior point on a curve or surface. An object point y is called a *curve-neighbor* (or *surface-neighbor*) of x if y lies in a locally 1-manifold (or 2-manifold) subset of $N_6(x,V)$ (or $N_{18}(x,V)$). Like end points, we present an explicit characterization of neighbor points (see proof in Appendix A):

Proposition 3. *Let $\omega_m(x,V)$ denote the set of curve-neighbor ($m=1$) or surface-neighbor ($m=2$) points of $x \in V$, for any $y \in V$ we have*

1. *$y \in \omega_1(x,V)$ if and only if $y \in N_6(x,V)$ and $\#N_6(x,V) \geq 2$.*
2. *$y \in \omega_2(x,V)$ if and only if $y \in N_{18}(x,V)$ and $r(x,V) > r(x,V \setminus \{y\})$.*

Here $r(x,V)$ is defined in Proposition 2.

Proof

1. By definition, y is a curve-neighbor of x only if y lies in a locally 1-manifold subset of $N_6(x,V)$, which requires both the existence of such a subset, i.e. $\#N_6(x,V) \geq 2$, and that $y \in N_6(x,V)$.
2. From the proof of Proposition 2, we know that $r(x,V)$ counts the number of possible locally 2-manifold subsets of $N_{18}(x,V)$. y lies in a locally manifold subset of $N_{18}(x,V)$ if and only if removing y from V reduces the number of manifold subsets. □

3 The Algorithms

The skeleton is computed by alternating two morphological operations: thinning and skeleton pruning. For thinning, we adopted the standard iterative paradigm while incorporating our new end-points definition for shape preservation. For pruning, we introduce a simple and effective procedure that extends the dilation and erosion operations often used in image denoising. The algorithms of thinning and pruning are discussed, and the complete method is presented last.

3.1 Iterative Thinning

Thinning begins with a volumetric object V with boundary points $\partial(V)$. Assuming 6-connectivity of the object (as explained in Section 2.1), the boundary is defined as

$$\partial(V) = \{x | x \in V \text{ and } N_6(x,\overline{V}) \neq \emptyset\}$$

Thinning removes each boundary point from V except for *critical* points, whose removal will alter the topology of V or result in loss of shape information. In our context, a critical point x can be either a non-simple point, a curve-end point or a surface-end point. We can write this definition of a critical point as a boolean expression:

$$IsCritical_m(x,V) = x \notin s(V) \text{ or } x \in \partial_m(V)$$

Here, the subscript $m = 0,1,2$ determines whether discrete surfaces ($m=2$) or curves ($m=1$) in addition to the topology of V are to be preserved during thinning, or only topology information will be preserved ($m=0$, where we let $\partial_0(V) = \emptyset$).

Note that based on the above definition, a curve-end or surface-end point that forms a small surface bump can be identified as critical points, such as those shown in Figure 3 (a,c) (end-points are drawn as triangles). These bumps are often resulted from discretizing a smooth surface. To distinguish discretization errors from shape features, in

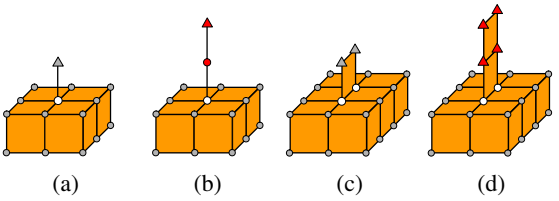


Fig. 3. Defining critical points: A curve-end or surface-end point (shown as triangles) is critical (colored red) only if it is shared by an edge or face that does not contain any non-boundary points (colored white)

practice, we add an additional restriction such that a curve-end (or surface-end) point is critical only if it is contained in an edge (or face) that contains only boundary points. As a result, only curve-end and surface-end points in Figure 3 (b,d) are considered critical.

The pseudo-code of thinning is shown in Figure 4 left. Thinning is performed iteratively. At each iteration, all boundary points are placed in a queue Q and are visited and removed sequentially. Note that program $Thin_m$ takes an extra parameter S, which is a subset of V that will be "protected" during thinning, i.e., no point of S will be removed from V. The use of S will become clear when the complete method is revealed in Section 3.3.

```
// Thinning object V while preserving S
// m: 0 (topology), 1 (curve), 2 (surface).
Thin_m(V, S)
    Repeat
        Q ← ∂(V)
        n ← 0
        Repeat while Q ≠ ∅
            x ← POP(Q)
            If x ∉ S and IsCritical_m(x,V)= False
                    V ← V \ {x}
                    n ← n+1
        If n = 0
            Return V
```

```
// Pruning skeleton V with depth d_m
// m: 1 (curve), 2 (surface).
Prune_m(V, d_m)
    If d_m ≤ 0
        Return V
    V' ← V \ ∂_m(V)
    S ← Prune_m(V', d_m − 1)
    Q ← ∂_m(S)
    Repeat for all x ∈ Q
        Repeat for all y ∈ N_18(x,V)
            If y ∈ ω_m(x,V) and y ∉ S
                S ← S ∪ {y}
    Return S
```

Fig. 4. Pseudo code for iterative thinning (left) and skeleton pruning (right)

For convenience, we refer to program $Thin_m$ as *surface thinning* ($m = 2$), *curve thinning* ($m = 1$), or *topology thinning* ($m = 0$). Figure 5 shows the result of thinning on three primitive shapes: a sphere, a cylinder and a plate. Observe in particular that surface thinning generates large surfaces for plate-like shapes, while curve thinning results in long curves for cylindrical shapes.

3.2 Skeleton Pruning

Pruning is the process of removing redundant curve or surface features from a skeleton. Examples of redundant features that we wish to remove include short curve branches,

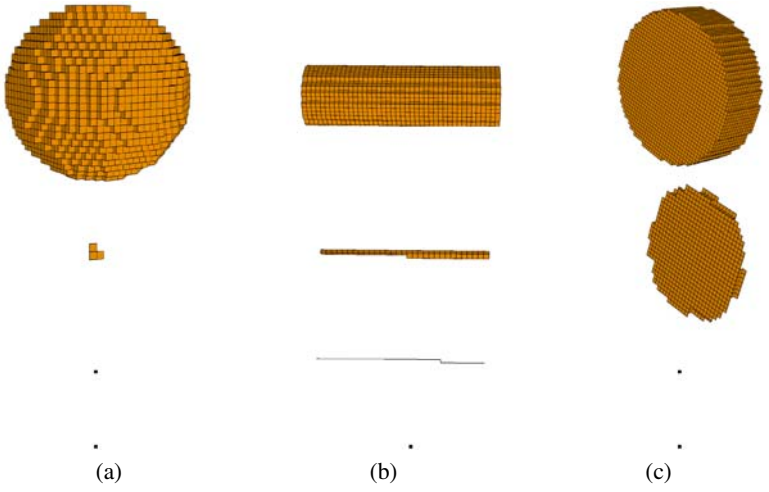


Fig. 5. Volumetric objects (top row) and their skeletons after surface thinning (second row), curve thinning (third row) and topology thinning (bottom row). The black dot represents a single skeleton point.

small surface branches, jagged surface borders and narrow surface bands. Our algorithm extends two morphological operators, namely erosion and dilation, which are often coupled (known as *opening*) in image processing to remove small and thin image artifacts and to smoothen jagged object borders. Whereas image denoising takes place on a 2D or 3D grid, skeleton pruning takes place on discrete curves and surfaces, and hence the two operators are adapted accordingly.

Given a skeleton object V and a super-set $V' \supset V$, our definition of erosion and dilation is based on end-points and neighbor-points introduced in Section 2.2:

$$\begin{aligned} Erode_m(V) &= V \setminus \partial_m(V) \\ Dilate_m(V,V') &= V \cup \textstyle\bigcup_{x \in \partial_m(V)} \omega_m(x,V') \end{aligned}$$

where $m = 1$ or 2, $\partial_1(V), \partial_2(V)$ are curve-end and surface-end points, and $\omega_1(x,V)$, $\omega_2(x,V)$ are curve-neighbor and surface-neighbor points. In words, erosion retracts V along its curve ($m = 1$) or surface ($m = 2$) border, while dilation expands V towards a larger set V' by growing manifold neighborhoods from the curve ($m = 1$) or surface ($m = 2$) border of V. Note that unlike thinning, erosion and dilation are topology-altering. Figure 6 shows an example of applying two rounds of surface erosion (b,c) followed by two rounds of surface dilation (d,e). The skeleton after i rounds of erosion ($i = 1,2$) is used as the super-set V' for the $(2-i)$th round of dilation.

Observe in Figure 6 that the combination of erosion and dilation has the effect of removing small, narrow features and smoothing borders. Hence we couple the two operators to form pruning, which is recursively defined as

$$Prune_m(V, d_m) = Dilate_m(Prune_m(Erode_m(V), d_m - 1), V).$$

where $d_m \geq 0$ is the *pruning parameter*, and $Prune_m(V,0) = V$.

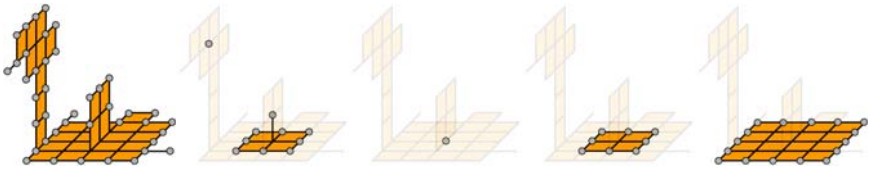


Fig. 6. A skeleton object (a), the result of applying two rounds of surface erosion (b,c), followed by two rounds of surface dilation (d,e). Eroded or dilated points at each step are highlighted.

The pseudo-code of pruning is provided in Figure 4 right. Pruning is performed recursively. At each recursion level, the eroded skeleton $V \setminus \partial(V)$ is first passed to the next level. Given the pruned skeleton S returned from the recursive call, the boundary points of S are then placed in a queue Q and processed sequentially for dilation. For each point in Q, dilation searches within the point's 18-neighborhood for curve-neighbor (or surface-neighbor) points and adds them to S. Note that, instead of storing multiple copies of the skeleton V during recursion, an integer value $d[x]$ can be computed and stored at each point $x \in V$ keeping track of recursion levels. Initially, all $d[x]$ are initialized as ∞. During recursion, $d[x]$ is first set to the level of recursion where x is eroded, and later reset to ∞ when it is dilated.

Like thinning, we refer to program $Prune_1(V, d_1)$ as *curve pruning* and $Prune_2(V, d_2)$ as *surface pruning*. Examples of pruning can be found as part of Figure 7 on the right, where surface pruning (top) and curve pruning (bottom) are applied with various parameters. Observe that larger values of d_1 remove longer curve branches, while larger values of d_2 remove wider surface bands and produce smoother surface borders. An important feature of our pruning method is that it does not shrink major curves or surfaces as a result of removing noises, similar the effect of opening in image denoising.

3.3 The Complete Method

We have presented two independent algorithms so far: a thinning operation that preserves the topology of the object yet may result in redundant features on the skeleton, and a pruning operation that removes redundant features from a skeleton yet may alter its topology. To create a skeleton that is both topology-preserving and composed of meaningful features, we combine thinning and pruning in alternating steps. Given an initial object V and pruning parameters d_1, d_2, we compute the final skeleton S in three steps:

Step 1. Extract major surface features by surface thinning followed by surface pruning (Figure 7 top):

$$S \leftarrow Prune_2(Thin_2(V, \emptyset), d_2).$$

Step 2. Extract major curve features by curve thinning followed by curve pruning (Figure 7 bottom):

$$S \leftarrow Prune_1(Thin_1(V, S), d_1).$$

Step 3. Ensure topology preservation through topology thinning:

$$S \leftarrow Thin_0(V, S).$$

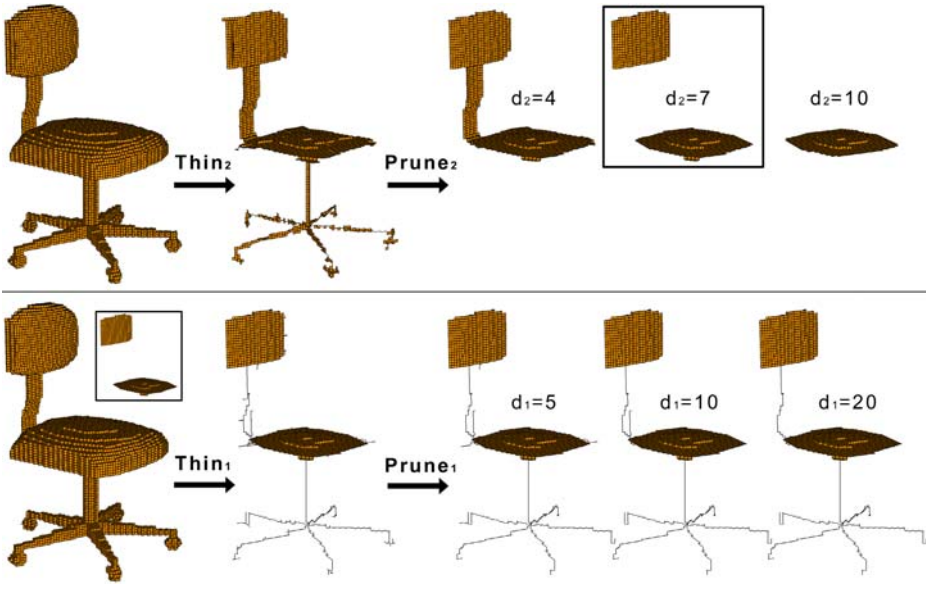


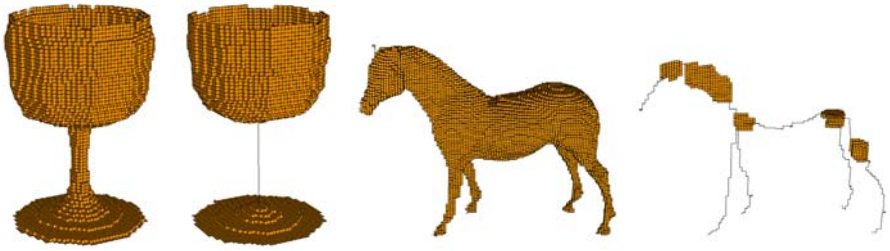
Fig. 7. Step 1 (top) and 2 (bottom) in generating a skeleton. Note that in the second step, curve thinning preserves the surfaces (shown in the insert) computed by the first step.

Note that in each step thinning is applied to the *original* object V. Except for step 1, thinning in both steps 2 and 3 also preserve major surfaces and/or curves computed in the previous step (through the use of the argument S in program $Thin_m$(V, S)). The final thinning step is necessary to ensure a topology-preserving skeleton, as pruning may alternate the topology of the object V.

The combination of thinning and pruning allows a whole spectrum of skeletons being generated by changing the two parameters d_1, d_2. For example, we can reproduce the result of directly applying surface thinning ($Thin_2(V,\emptyset)$), curve thinning ($Thin_1(V,\emptyset)$) and topology thinning ($Thin_0(V,\emptyset)$) using parameters $\{d_1,d_2\} = \{0,0\},\{\infty,0\}$ and $\{\infty,\infty\}$. More importantly, as observed in Figure 7, reasonable choices of d_1, d_2 would yield skeletons whose curves and surfaces correspond well to cylindrical and plate-like shapes of the original object. In addition, varying the parameters allow the user to adjust the differentiation between these two shape types (Figure 1 is such an example).

4 Results

We demonstrate our skeletonization method on several artificial and scanned models, as shown in Figure 8 and 9. Each model is originally represented in polygonal formats and converted into volumetric forms using the PolyMender software [28]. Observe that our skeleton accurately captures the three structural components of the goblet, which is composed of 2 plates and 1 cylinder. Although the horse and the bunny do not consist of obvious cylinders and plates, the skeletons still provide a descriptive view of its overall shape as well as the difference among local shapes. Note from Figure 9 that varying the



(a) (b) (c) (d)

Fig. 8. A goblet (a) and a horse (c) with their skeletons (b,d). Pruning parameters $d_1 = 20$ and $d_2 = 3$ are used in both examples.

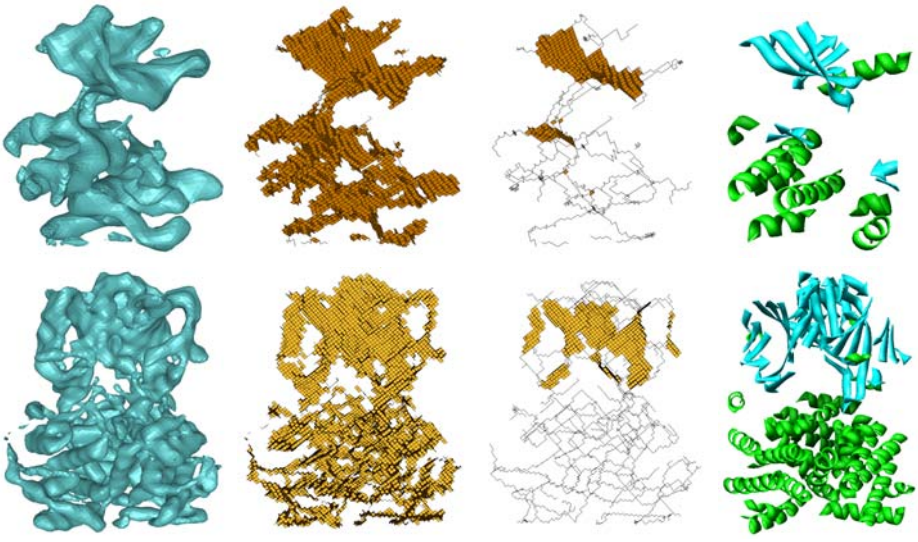
(a) (b) (c) (d)

Fig. 9. A bunny model (a) and its skeletons with parameters $\{d_1, d_2\}$ at $\{50, 2\}$ (b), $\{50, 4\}$ (c) and $\{50, 10\}$ (d)

pruning parameter gives multiple choices of skeletons that may be useful in different applications.

We have also applied our method in the bio-medical setting for identifying β-sheet elements from low-resolution protein images. Figure 10 (a) shows protein density models reconstructed from cryo-electron microscopy (cryo-EM). Unlike X-ray crystallography, cryo-EM is capable of imaging large molecular assemblies in nearly native states [29]. However, the drawback of cryo-EM is that the reconstructed model has a much lower resolution (typically 6 to 10 Å) than that of X-ray (less than 3 Å), and resolving atomic details is not possible. Nevertheless, secondary structures of the protein, such as α-helices, β-sheets and loops, are discernable at this resolution and can be recognized as cylinders (loops and α-helices) and curved plates (β-sheets). We applied our method to cryo-EM models, and the resulted skeletons (shown in Figure 10 (c)) differentiate cylindrical and plate-like shapes as skeleton curves and surfaces. Such differentiation allows β-sheets to be identified as skeleton surfaces in an automatic manner. We confirm our result by comparing our skeleton with the actual protein structure revealed by X-ray experiments, shown in Figure 10 (d), where β-sheets are drawn as parallel blue arrows. In contrast, applying Bertrand's parallel technique [9] yields far more surface components on the skeleton than our method, as shown in Figure 10 (b), making it impossible to differentiate plate-like and cylindrical shapes.

In contrast to previous skeletonization algorithms that typically involve one pass of thinning, our method involves three passes of thinning and two passes of pruning. As



(a) (b) (c) (d)

Fig. 10. The iso-surface models of proteins IRK and 1BVP (a), the skeletons computed by [9] (b), the skeletons computed using our method with $d_1 = d_2 = 3$ (c), and the actual structures of these proteins determined by X-ray crystallography (d), where blue stripes denote β-sheets and green spirals denote α-helices. Observe that the skeleton surfaces computed by our algorithm correspond well to the actual β-sheets in the protein structure.

Table 1. Performance of skeletonization for the examples in this paper. The timings are recorded in seconds, and are broken down into various thinning and pruning passes, see Section 3.3.

Model	Grid Size	# Object Voxels	Step 1 (Thin)	Step 1 (Prune)	Step 2 (Thin)	Step 2 (Prune)	Step 3 (Thin)	Total Time	Bertrand [9]
Goblet	64	15911	0.79	0.08	0.24	0.02	0.15	1.28	0.15
Chair	128	34920	1.28	0.28	1.04	0.19	0.65	3.44	1.21
Horse	128	54172	1.69	0.28	1.62	0.18	1	4.77	1.92
Bunny	128	214898	7.12	0.29	5.47	0.19	3.42	16.49	4.94
Protein 1IRK	128	10887	0.47	0.25	0.43	0.17	0.3	1.62	0.31
Protein 1BVP	128	24741	1.4	0.28	0.93	0.17	0.56	3.34	0.45

a result, our method runs slower than most thinning methods, as shown in Table 1, yet still at a reasonable speed. All experiments were run on a PC with Pentium 4 CPU at 1.7GHz and 512MB memory. The timings are broken down into each thinning and pruning pass and are compared with the parallel thinning method of Bertrand [9].

5 Conclusion and Discussion

In this paper, we describe a method for computing a family of topology and shape preserving skeletons of a volumetric model. The method is compose of two algorithms:

iterative thinning and skeleton pruning. Our method allows flexible control over the size of curves or surfaces on the resulting skeletons, which can be used for describing shapes of objects composed of cylindrical and plate-like regions.

As a future direction of research, we are investigating more shape applications, such as segmentation, matching and recognition, using the skeleton generated by our method. In particular, we will explore the use of the skeletons of proteins in fast and accurate fold recognition and comparison, which is an important task demanded by researchers in structural biology. Additionally, we are interested in developing algorithms that automatically determine the appropriate pruning parameters to compute desirable skeletons with minimum or no human input.

References

1. Blum, H.: A transformation for extracting new descriptors of shape. In Wathen-Dunn, W., ed.: Models for the Perception of Speech and Visual Forms. MIT Press, Amsterdam (1967) 362–380
2. Lam, L., Lee, S.W., Suen, C.Y.: Thinning methodologies-a comprehensive survey. IEEE Trans. Pattern Anal. Mach. Intell. **14** (1992) 869–885
3. Lee, T.C., Kashyap, R.L., Chu, C.N.: Building skeleton models via 3-d medial surface/axis thinning algorithms. CVGIP: Graph. Models Image Process. **56** (1994) 462–478
4. Saha, P.K., Chaudhuri, B.B.: Detection of 3-d simple points for topology preserving transformations with application to thinning. IEEE Trans. Pattern Anal. Mach. Intell. **16** (1994) 1028–1032
5. Bertrand, G.: Simple points, topological numbers and geodesic neighborhoods in cubic grids. Pattern Recogn. Lett. **15** (1994) 1003–1011
6. Tsao, Y.F., Fu, K.S.: A parallel thinning algorithm for 3-d pictures. Comput. Graphics Image Process. **17** (1981) 315–331
7. Gong, W., Bertrand, G.: A simple parallel 3d thinning algorithm. In: ICPR90. (1990) 188–190
8. Bertrand, G., Aktouf, Z.: A 3d thinning algorithms using subfields. In: Proceedings, SPIE Conference on Vision Geometry III. Volume 2356. (1994) 113–124
9. Bertrand, G.: A parallel thinning algorithm for medial surfaces. Pattern Recogn. Lett. **16** (1995) 979–986
10. Ma, C.M.: A 3d fully parallel thinning algorithm for generating medial faces. Pattern Recogn. Lett. **16** (1995) 83–87
11. Saito, T., Toriwaki, J.: A sequential thinning algorithm for three-dimensional digital pictures using the euclidean distance transformation. In: Proceedings of the 9th Scandinavian Conference on Image Analysis. (1995) 507–516
12. Palágyi, K., Kuba, A.: A parallel 3d 12-subiteration thinning algorithm. Graph. Models Image Process. **61** (1999) 199–221
13. attali, D., Sanniti di Baja, G., Thiel, E.: Pruning discrete and semicontinuous skeletons. Lecture Notes in Computer Science, Image Analysis and Processing **974** (1995) 488–493
14. Shaked, D., Bruckstein, A.: Pruning medial axes. Comput. Vis. Image Underst. **69** (1998) 156–169
15. Ogniewicz, R.L., Kübler, O.: Hierarchic Voronoi skeletons. Pattern Recognition **28** (1995) 343–359
16. Attali, D., Montanvert, A.: Computing and simplifying 2d and 3d continuous skeletons. Comput. Vis. Image Underst. **67** (1997) 261–273

17. Amenta, N., Choi, S., Kolluri, R.K.: The power crust. In: SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications, New York, NY, USA, ACM Press (2001) 249–266
18. Dey, T.K., Zhao, W.: Approximate medial axis as a voronoi subcomplex. In: SMA '02: Proceedings of the seventh ACM symposium on Solid modeling and applications, New York, NY, USA, ACM Press (2002) 356–366
19. Foskey, M., Lin, M.C., Manocha, D.: Efficient computation of a simplified medial axis. In: SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications, New York, NY, USA, ACM Press (2003) 96–107
20. Tam, R., Heidrich, W.: Shape simplification based on the medial axis transform. In: Proceedings of IEEE Visualization. (2003)
21. Sud, A., Foskey, M., Manocha, D.: Homotopy-preserving medial axis simplification. In: SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling, New York, NY, USA, ACM Press (2005) 39–50
22. Mekada, Y., Toriwaki, J.: Anchor point thinning using a skeleton based on the euclidean distance transformation. In: ICPR '02: Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 3, Washington, DC, USA, IEEE Computer Society (2002) 30923
23. Svensson, S., Sanniti di Baja, G.: Simplifying curve skeletons in volume images. Comput. Vis. Image Underst. **90** (2003) 242–257
24. Saha, P., Gomberg, B., Wehrli, F.: Three-dimensional digital topological characterization of cancellous bone architecture. IJIST **11** (2000) 81–90
25. Bonnassie, A., Peyrin, F., Attali, D.: Shape description of three-dimensional images based on medial axis. In: Proc. 10th Int. Conf. on Image Processing, Thessaloniki, Greece (2001)
26. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. In: SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press (1987) 163–169
27. Natarajan, B.K.: On generating topologically consistent isosurfaces from uniform samples. The Visual Computer **11** (1994) 52–62
28. Ju, T.: Robust repair of polygonal models. ACM Trans. Graph. **23** (2004) 888–895
29. Chiu, W., Baker, M., Jiang, W., Zhou, Z.: Deriving the folds of macromolecular complexes through electron cryomicroscopy and bioinformatics approaches. Curr. Opin. Struct. Biol. **2** (2002) 263–269

Representing Topological Structures Using Cell-Chains*

David E. Cardoze[1], Gary L. Miller[2], and Todd Phillips[2]

[1] Tanner Research
cardoze@tanner.com
[2] Computer Science Department
Carnegie Mellon University
glmiller@cs.cmu.edu, tp517@cs.cmu.edu

Abstract. A new topological representation of surfaces in higher dimensions, "cell-chains" is developed. The representation is a generalization of Brisson's cell-tuple data structure. Cell-chains are identical to cell-tuples when there are no degeneracies: cells or simplices with identified vertices. The proof of correctness is based on axioms true for maps, such as those in Brisson's cell-tuple representation. A critical new condition (axiom) is added to those of Lienhardt's n-G-maps to give "cell-maps". We show that cell-maps and cell-chains characterize the same topological representations.

Keywords: computational topology, cell complex, cell tuple, cell chain.

1 Introduction

The ability to represent and manipulate topological structures in a computer is central to many areas, such as the finite element method in scientific computation, computational geometry, computer graphics, solid modeling, and scientific visualization. In general, a user has a particular decomposition of a domain into a collection of topological objects and needs a data structure for representing the connectivity information between various simple objects. We present a new data structure that addresses this task more generally than previous data structures, while providing a strong theoretical characterization of those objects that can be represented. This problem has been cited as an important open area of research in computational topology [BEA+99].

A classic example of this task occurs in meshing, wherein a complex geometric objects is decomposed into simple atomic objects. In two dimensions, this is the equivalent to the representation of a graph embedded on a surface. The embedding partitions the surface up into three types of *cells*: vertices, edges, and faces; called cells in general. Another very important example are CAD systems. Here the objects or cells may be much more complicated then simply a collection of simplices or d-dimensional cells, [Far99].

* This work was supported in part by the National Science Foundation under grants CCR-9902091, CCR-9706572, ACI 0086093, CCR-0085982 and CCR-0122581.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 248–266, 2006.
© Springer-Verlag Berlin Heidelberg 2006

The goal of the topological data structure is to maintain the cells and incidence relationship between these cells in such a way that topological and other information can be stored and retrieved correctly and efficiently. If the full dimensional cells, say d-dimensional cells, are simplices, then one can, in principle, just enumerate the shared $d-1$-simplices. In general for many applications the d-cells will not be simplices. As an example, for spectral or high-order methods, cube-like elements are used because sparse methods for representing the stiffness matrix are known [DFM02]. Other applications may use a mesh with a mix of element types since, in general, it may be easier to have a mesh with mostly cube elements and only a few tetrahedral elements.

For d-cells that are finite, one can decompose the cells into simplices using the barycentric subdivision. In two dimensions, most, if not all, proposed topological data structures represent the barycentric subdivision, either implicitly or explicitly.

The most popular data structure is the cell-tuple of Brisson [Bri89, Bri93]. In this structure, topological information is stored via **cell-tuples**, which are maximal paths in the **incidence graph** or **incidence poset** [Ede87], see Figure 1. This data structure yields extremely efficient and elegant implementations for many operations on topological arrangements, and it is widely used in practice [Hal97].

The two main limitations of Brisson's cell-tuple is that it can only represent a very regular class of structures and that the test for membership in this class is undecidable in higher dimensions. To address the first limitation, we propose an extension to cell-tuples that allows for representation of an **incidence multigraph**, thus allowing various types of degeneracies. We operate using **cell-chains** (Section 4), which are maximal paths in the multigraph. In Section 8 we discuss **clusters** of cell-chains that correspond to a single cell-tuple. We show that in two-dimensions, clusters can be of size at most four, whereas clusters can be arbitrarily large in three or higher dimensions. We discuss implementation details for cell-chains in Section 7.

The ability of the data structure to handle degeneracies is very useful. Several authors have decomposed 2D surfaces up into highly degenerate pieces for efficient processing purposes [Hop96]. We also have used these degeneracies in a biological application to specify cell and artery boundaries [CCM+04] succinctly. The alternate approach for dealing with degeneracies is to refine the cell decomposition to remove these degeneracies. If the software cannot handle these degeneracies then they must be removed manually.

The second limitation of Brisson's cell complexes is that he requires that the cell-tuple representation comes from a manifold, a space locally homeomorphic to $\mathbb{R}^d$, for a d-dimensional structure. But determining if a topological structure is locally homeomorphic to $\mathbb{R}^d$ is undecidable for $d \geq 6$ [Mar58, VKF74], it is open for $d = 5$, and only known to be in NP for $d = 4$ [Sch]. To address the undecidability problem inherent in Brisson's cell complexes, we do not require our representation be locally homeomorphic to $\mathbb{R}^d$. We only require cell complex to satisfy testable conditions that are also known to be true in Brisson's case.

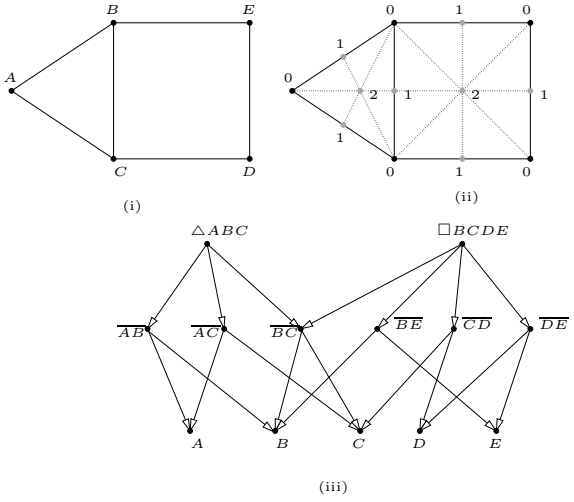


Fig. 1. (i) A simple cell complex made from a square adjacent to a triangle. (ii) The barycentric subdivision of this complex: The barycenter of each cell, vertex, edge, or face, is inserted and labeled with the dimension of the cell. The result is triangulated to form numbered simplices. (iii) The incidence graph (poset) for this complex. The vertices are cells edges connect neighboring cells whose dimension differs by 1.

These conditions are true for many degenerate structures also. The combinatorial axioms we give can be verified in $O(nd)$ where n is input size and d is dimension. Our axiomatic approach follows those developed for algebraic representations.

One of the early algebraic representations of graph connectivity on surfaces was given by Edmonds and Tutte [Edm60, Tut73]. Other examples of this type of representation include *maps* [Tut84, Vin83a, Vin83b, BS85], *n-G-maps* [Lie91, Lie94, LM], and the *quad-edge* structure [GS85]. Another similar algebraic structure is Tits notion of *buildings* [Tit81].

The benefit of algebraic representations is that they have axiomatic definitions for the combinatorial structure of the topological space that is being represented. The vast majority of algebraic representations (some in hindsight) use permutation groups that act on the simplices of the **barycentric subdivision** or **barycentric complex** (see Figure 1 as well as Section 2.) The barycentric complex is constructed bottom-up by "gluing" together numbered simplices. Axiomatic gluing rules are given so that the complex will be well-formed (in terms of whatever algebraic structure is involved.) Throughout the paper we will call a set of gluing rules a **map**. The most popular data structures of this form are n-G-maps, due to Lienhardt [Lie91, Lie94], which use a overly loose set of axioms to construct a very large class of topological structures.

We propose an additional axiom, to those of Lienhardt, the **Orthogonality Axiom** (Section 3), that restricts the possible combinatorial structures, but still allows a rich class of topological structures.

The central result of this work is Theorem 3 in Section 4, wherein we show that our cell-chain extension to Brisson's cell-tuple is precisely equivalent to our axiomatic restriction of Lienhardt's n-G-maps. This middle ground allows for an intuitive data structure that still represents a large group of topologies, thus combining the benefits of both major approaches to topological data structures. A two-dimensional implementation of cell-chains has been implemented within the TUMBLE software package. [Tum].

A plethora of similar data structures and representations exist in the literature. Cell-Chains, n-G-maps, and cell-tuples have much in common with other topological data structures [Bau72, Män88, GS85, Wei85] and [Wei86, DL87, RO89, Ros97]. An excellent review of existing work can be found in [LLLV05].

The rest of this paper is organized as follows. In Sections 2 and 3 respectively, we discuss the concepts of maps from a geometrical and combinatorial point of view. In Section 4 we introduce simplex-chains, cell-chains and prove our main result. In Section 5 we give a formal presentation of cells and show that they are well-defined for our cell-maps. In Section 6 we show that all of our axiom are true for a CW-complex thus justifying our axioms. In Section 7 we discuss some preliminary implementation issues. Finally in the last section we delineate the difference between the complexes we can represent and those of Brisson's cell-tuples.

2 Barycentric Complexes and Maps

In this section we discuss the concept of maps from a geometric point of view. This information should give enough background to introduce a set of axioms and to work in the more formal setting in Section 3. We start by defining some topological notions. For more background on topology refer to [RF90, Kin93, LW69, Moi77, Jan84].

The goal of a topological map is to partition a topological space up into regions that are homeomorphic to open balls. More formally, for $d \geq 1$ define

$$\begin{aligned}\mathbb{B}^d &= \{x \in \mathbb{R}^d : |x| < 1\},\\ \overline{\mathbb{B}^d} &= \{x \in \mathbb{R}^d : |x| \leq 1\},\\ \mathbb{S}^d &= \{x \in \mathbb{R}^{d+1} : |x| = 1\}.\end{aligned}$$

A space homeomorphic to $\mathbb{B}^d$ is called an **open d-cell**, a space homeomorphic to $\overline{\mathbb{B}^d}$ is called a **closed d-cell**, and a space homeomorphic to $\mathbb{S}^d$ is called an **d-sphere**. By convention we say that single points are both open and closed 0-cells.

A partition of a space into open cells is called a CW-complex. Recall that a d-dimensional **(finite, normal, homogeneous) CW-complex** is a pair (X, C) where X is a Hausdorff space and C is a finite partition of X into open cells such that for every open d-cell, c in C, there is a continuous map $h_c : \overline{\mathbb{B}^d} \to X$ whose restriction to $\mathbb{B}^d$ is a homeomorphism onto c and whose restriction to $\mathbb{S}^{d-1}$ is the union of open cells in C of dimension less than d. In addition it is

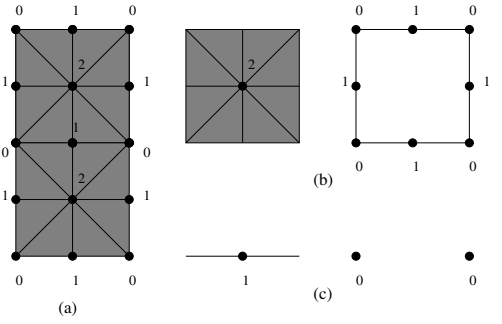


Fig. 2. (a) a barycentric complex with two 2-cells(faces), seven 1-cells(edges) and six 0-cells(vertices). (b) a 2-cell (left) and its boundary (right). (c) a 1-cell (left) and its boundary (right).

required that every open cell is either a d-cell or is on the boundary of a d-cell. A **sub-complex** of (X, C) is a CW-complex (Y, D) where $Y \subseteq X$ and $D \subseteq C$. We will usually refer to a CW-complex by its underlying space X alone and do not mention the partition C. A CW-complex is said to be a **pseudo-manifold** if every $(d-1)$-face is adjacent to at most two d-cells and every d-cell can be reached from every other d-cell traveling along $(d-1)$-faces.

Since we are interested in d-dimensional CW-complexes that represent generalized barycentric subdivisions, we add some additional constraints. The idea in representing a CW-complex is to further partition each i-cell into i-simplices. We do this by adding a new point interior to each cell. Each point is labeled by the dimension of the cell containing it. Using these points we now form d-simplices out of these points [Bri93] that partition up the d-cells. A **barycentric complex** X is a CW-complex where the sub-complex of every closed cell in X is a combinatorial simplex, and every vertex v in X is assigned a number $\nu(v)$ between 0 and d in such a way that no two vertices on the boundary of any given open cell have the same label, see Figure 2.

Since the closed cells of a barycentric complex are combinatorial simplices, by abuse of the notation we will refer to open cells as open simplices, and closed cells as closed simplices. Sometimes we will drop the open or closed adjective when the meaning is clear from the context or is irrelevant. A vertex whose label is i is said to be an i-vertex.

Instead of taking a manifold or surface and decomposing it into cells, we take numbered d-simplices and prescribe rules for gluing them into surfaces. In particular, a **map** is a set of numbered simplices and gluing rules that hopefully represent a connected barycentric complex that results from gluing numbered closed d-simplices along $(d-1)$-faces whose vertices have the same labels as prescribed by the gluing rules. The gluings must be induced by cell preserving homeomorphisms that respect the labeling. Every $(d-1)$-face can be involved in at most one gluing.

Throughout this paper we will consider only one type of gluing, namely, we will only glue or identify simplices of dimension $d-1$ and only if the set of

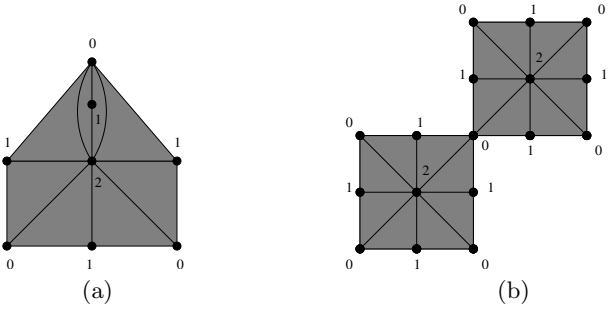


Fig. 3. (a) This map is not proper. The one-cell interior to the two-cell is not an open one-disk. (b) This barycentric complex is not a map. The vertex common to the two two-cells is a face of two triangles which cannot be reached from one another by traversing edges containing it.

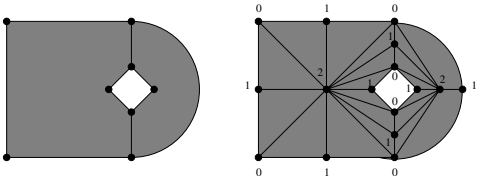


Fig. 4. A CW-complex and its barycentric decomposition

vertex numbers are the same. Thus if σ_1 and σ_2 are two numbered d-simplices containing, respectively, $d-1$-simplices σ_1' and σ_2'. To identify σ_1' and σ_2' it must be the case that the vertex labels agree and that they contain all the labels from $0 \ldots d$ except one number. If we identify them, then we will identify all simplices in σ_1' with all simplices in σ_2' containing the same vertex numbers.

Hence a map X can be represented by a tuple $(S, \alpha_0, \ldots, \alpha_d)$ where S is a set of numbered closed d-simplices and the α_i's are involutions on S that describe the gluing's. If α_i has no fixed points for $0 \leq i < d$ we say that X has a **proper boundary**. The barycentric complex in Figure 2 is a map while the one in Figure 3(b) is not.

Note that there are maps that are not proper but satisfy the conditions put forward in [Lie94]. See Figure 3(a) for an example.

One possibility for an even stronger condition would be that the cell decomposition itself be a CW-complex, unfortunately, this is undecidable in higher dimension.

3 Formal Presentation of Maps

In this section we present a formal approach to combinatorial maps, as motivated in Section 2. A more formal specification is necessary for at least three reasons. Firstly, a combinatorial structure is closer to what is actually stored

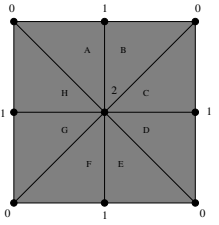


Fig. 5. If the A is glued to F and B is glued to E (both gluings by α_2), then the resulting object is a cylinder. If additionally, C is glued to H and G is glued to D (both gluings by α_2), then the resulting object is a torus.

and manipulated internally in the machine. Secondly, the proofs for topological data structures are all based on this formal specification. Third, since it is undecidable to determine if the structure we will consider is a well-formed manifold, we must therefore work in a formal combinatorial model.

A **k-simplex** σ is a set of $k+1$ atomic objects and all subsets of these objects. Each subset and all the subsets containing it is also a simplex (a subsimplex). We call an atomic object a **vertex** or **0-simplex**. Throughout this paper we deal only with numbered simplices. A **numbered k-simplex** σ is a k-simplex where each vertex is assigned a distinct integer label. A **consecutively numbered k-simplex** is a numbered simplex with numbers from the range $\{i, \ldots, i+k\}$. We call a consecutively numbered 1-simplex an **edge-simplex**. The **span** of a numbered simplex is the set of its integer labels.

Throughout this paper we consider only one type of gluing of two simplices, namely, we only glue or identify two subsimplices of dimension $d-1$ of two distinct d-simplices and only if the set of vertex numbers are the same. That is, if σ_1 and σ_2 are two numbered d-simplices containing, respectively, $d-1$-simplices σ_1' and σ_2' then to identify σ_1' and σ_2' it must be the case that the vertex labels agree and must contain all numbers but say i. If we identify them then we also identify all subsimplices in σ_1' with all subsimplices in σ_2' with the same vertex labels.

In the definition to follow one can think of the set Σ as n numbered d-simplices, but formally it is just a finite set of size n. By the discussion in the last paragraph to describe the $d-1$ simplices that are to be identified it will suffice to list pairs of d-simplices and, for each pair, the vertex that will not be identified. As has been the tradition since Tutte we do this in a group theoretic way [Tut73, Lie91]. There is yet another important way to view the gluing of the d-simplices described above. Here we think of going from σ_1 to σ_2 by reflecting σ_1 about the $d-1$-simplex common to σ_1 and σ_2. Thus the $\alpha_{i's}$ in Definition 1 are some kind of reflection and we are interested in the group generated by these "reflection".

We let $\langle a_1, \ldots, a_k \rangle$ denote the subgroup generate by the permutations $a_1, \ldots, a_k$. We say the permutation group G **acts transitively** on Σ if for every pair of elements $\sigma_1, \sigma_2 \in \Sigma$ there is a permutation $\alpha \in G$ such that $\alpha(\sigma_1) = \sigma_2$.

Definition 1. *A* **map** $\mathcal{M}$ *is a tuple* $(\Sigma, \alpha_0, \ldots, \alpha_d)$ *where* Σ *is a finite set of size* n *and each* α_i *is a permutation of* Σ *for* $0 \leq i \leq d$ *such that*

1. $\alpha_i^2 = ident$ *for* $0 \leq i \leq d$.
2. α_i *is fix-point-free (fpf) for* $0 \leq i < d$.

We say that $\mathcal{M}$ *is* **connected** *if* $\langle \alpha_0, \ldots, \alpha_d \rangle$ *acts transitively.*

If we view Σ as a disjoint set of numbered d-simplices and the $\alpha's$ as gluing rules as defined above then $\mathcal{M}$ gives a simplicial complex, denoted by $complex(\mathcal{M})$. One interprets the fixed points of α_d as determining the $d-1$-dimensional boundary faces of the complex.

We view the permutations $\alpha_0, \ldots, \alpha_d$ as acting on the d-simplices of $\mathcal{M}$. In general the permutations do not act in a well defined way on the i-simplices for $0 \leq i < d$. The first restriction that Lienhardt and others require is that many of the α_i commute. We shall call a map with the commutivity property a commuting-map. For complexes of commuting-maps, there will be a natural and well-defined action of permutations on simplices.

Definition 2 (Commuting-Map). *A* **commuting-map** $\mathcal{M} = (\Sigma, \alpha_0, \ldots, \alpha_d)$ *is a map with the further property:*

Commutivity: $\alpha_i \alpha_j = \alpha_j \alpha_i$ *whenever* $2 \leq i + 2 \leq j \leq d$.

The commuting condition is very natural one. In 2D it just says that exactly four numbered triangles contain an edge unless the edge is on the boundary in which case there are two. It is implicit in Tutte's axioms [Tut73]. A commuting-map is what Lienhardt calls an *n-bG*-map where his n is our d, [Lie94]. Lienhardt replaces our commutivity condition with the condition that $\alpha_i \alpha_j$ is an involution whenever $2 \leq i+2 \leq j \leq d$ which can easily be seen to be equivalent. He has also proposed that $\alpha_i \alpha_j$ is fpf whenever $2 \leq i+2 \leq j \leq d$. The orthogonality condition is a stronger condition and we will show that it is necessary and sufficient to prove Theorem 2. Again, in Section 6 a formal justification will be given for these axioms.

Denote the group $\langle \alpha_j, \ldots, \alpha_i \rangle$ for $0 \leq j \leq i \leq d$ by G_j^i. Given a numbered simplex λ we denote by G_λ the subgroup $\langle \alpha_i \mid i \notin \text{span}(\lambda) \rangle$.

Definition 3 (Cell-Map). *A* **cell-map** $\mathcal{M} = (\Sigma, \alpha_0, \ldots, \alpha_d)$ *is a commuting-map with the further property:*

Orthogonality: *For every* $0 < i < d$, $\sigma \in \Sigma$, $\alpha \in G_0^{i-1}$, *and* $\beta \in G_{i+1}^d$ *if* $\alpha\beta(\sigma) = \sigma$ *then* $\alpha(\sigma) = \beta(\sigma) = \sigma$.

Observe that for a cell-map the groups G_0^{i-1} and G_{i+1}^d commute and their intersection is the identity permutation. Thus the group generated by G_0^{i-1} and G_{i+1}^d is isomorphic to their direct product. The main goal of this section is to show that their action on the cell-chains(to be defined) is the natural direct product action.

In what follows we will first prove properties of maps. Using these properties prove properties of commuting-maps. Finally we consider properties of cell-maps. An important property held by any map is local strong connectivity.

Definition 4 (Local Strong Connectivity[1]). *A d-simplicial complex $\mathcal{C}$ is locally strongly connected or simply locally connected if whenever τ and γ are two d-simplices that share a nonempty simplex λ, then there exists a sequence $\sigma_1, \ldots, \sigma_t$ of d-simplices such that $\sigma_0 = \tau$, $\sigma_t = \gamma$, and σ_i and σ_{i+1} have a common $(d-1)$-simplex containing λ for $0 \leq i < t$.*

As is well known, the simplicial complex of a map is locally connected. In fact, the simplicial complex of a map is a pseudo-manifold.

Lemma 1. *If $\mathcal{C}$ is the simplicial complex of a map $\mathcal{M}$ then $\mathcal{C}$ is locally strongly connected.*

We prove something slightly stronger. Namely, if we glue d-simplices by identifying $d-1$-faces then the complex is locally connected. The proof is by induction on the number of identifications or gluings. Initially we have n disjoint d-simplices. In this base case we are done since no two distinct d-simplices can have a nontrivial intersection. Thus we may assume that after k identification the complex is locally connected. Suppose by induction that τ and γ are distinct d-simplices that share a simplex λ after $k+1$ identifications. There are two cases depending on whether or not $\lambda \subseteq \tau \cap \gamma$ before the $k+1$ first identification. In the former case we are done since we get a sequence by induction.

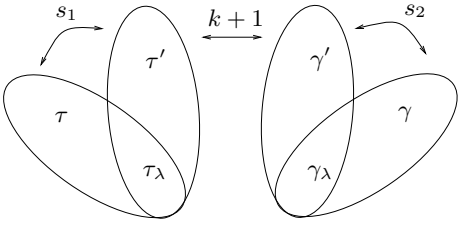


Fig. 6. Illustration of the proof of Lemma 1. s_1 and s_2 represent the inductively generated sequences. The $k+1$ gluing connects the sequence.

In this latter case, the common simplex λ, must have been formed by identifying two simplices during the $k+1$ gluing. Thus there must have been $d-1$-simplices τ_λ and γ_λ, contained in the simplices τ and γ, respectively, that were identified to form λ at the $k+1$ gluing. It is critical to the proof that a single identification must have formed this common simplex λ. Before this identification τ and γ would not contain this common simplex but after this single identification they do. Thus, it must be that case that at gluing $k+1$, τ_λ and γ_λ are identified. There must exist two d-simplices, τ' and γ' containing τ_λ and γ_λ, respectively, that are identified along a $d-1$-simplex λ' and the simplex λ' contains λ. Here we use the fact that we do not ever identify d-simplices. By induction there a sequences of d-simplices from τ to τ' and one from γ' to γ. By combining these

[1] *Locally Strongly Connected* has appeared in other topological literature in a different context.

two sequences with the $k+1$st gluing we get the prescribed sequence from τ to γ. □

Observe that the $k+1$ gluing in the proof above is obtained by applying the permutations α_i, where $i \notin G_\lambda$, such that $\alpha_i(\tau') = \gamma'$. Therefore by induction all the gluings on the path obtained from permutations in G_λ.

It is not hard to see that the sequence of d-simplices can be obtain one from another by applying permutations α_i, where $i \notin G_\lambda$, where λ is the intersection. Thus, we reformulate Lemma 1 group theoretically. This formulation of the locally connectivity lemma is very simple group theoretically and will be used throughout the paper.

Lemma 2 (Local Connectivity Lemma). *If τ and γ are d-simplices in the complex of a map that share a nonempty simplex λ then there exists a permutation $\alpha \in G_\lambda$ such that $\alpha(\tau) = \gamma$.*

It follows by induction for a complex of a map that every d-simplex has a unique subsimplex with a given subset of labels from the set $0, \ldots, d$. Using this fact we define a natural **boundary map**. If σ is a simplex and S is a set of labels then $[\sigma]_S$ denotes the sub-simplex of σ with label in S. When there is no confusion we let $[\sigma]_i$ denote the subsimplex $[\sigma]_{\{0,\ldots,i\}}$.

We say the permutation $\alpha \in \langle \alpha_0, \ldots, \alpha_d \rangle$ **acts in a well defined way** on a simplex σ if for all d-simplices τ and γ containing σ

$$[\alpha(\gamma)]_S = [\alpha(\tau)]_S$$

where S is the set of labels of σ

Lemma 3. *If $\mathcal{C}$ is the simplicial complex of a map $\mathcal{M}$, σ is a simplex of $\mathcal{C}$, and $\alpha \in G_\sigma$ then α acts in a well defined way on σ and $\alpha(\sigma) = \sigma$.*

Proof. Suppose that σ is an i-simplex with span S. It will suffice to prove the lemma for any permutation α_j for $j \notin S$. If $i = d$ the lemma is vacuously true. Suppose that $i < d$ and let γ be any d-simplex that contains σ. There must exist a $d-1$-simplex τ with span $S' = \{0, \ldots, d\} - \{j\}$ such that $\sigma \subseteq \tau \subset \gamma$. If $j = d$ and γ is a fixed point then so it τ. If not then there a a unique γ' such that $\alpha_j(\gamma) = \gamma'$ and γ' also contains τ. Thus in either case α_j fixes τ. That is $[\alpha_j(\gamma)]_{S'} = \tau$. This gives the following sequence of equalities:

$$[\alpha_j(\gamma)]_S = [[\alpha_j(\gamma)]_{S'}]_S = [\tau]_S = \sigma$$

□

Additionally, we need one more technical lemma that will be used to prove our main theorem, Theorem 3.

Lemma 4. *If $\mathcal{C}$ is a numbered simplicial-complex of a commuting-map and γ, γ', and λ be three simplices such that span(γ) = span$(\gamma') \subset \{0, \ldots, i\}$, $\lambda \subseteq \gamma \cap \gamma'$, and $\{j, \ldots, i\}$ = span(λ) for $j \leq i$ then there exits an $\alpha_L \in G_0^{j-1}$ such that $\alpha_L(\gamma) = \gamma'$.*

Proof. In general $\alpha(\gamma)$ may not be well-defined. Here all we mean is that there exists a d-simplex $\gamma \subset \sigma$ such that $\gamma' \subset \alpha(\sigma)$. By Lemma 1 there exists an $\alpha \in G_\lambda$ such that $\alpha(\gamma) = \gamma'$. By the Commutivity condition $\alpha = \alpha_L \alpha_H$ where $\alpha_L \in G_0^{j-1}$ and $\alpha_H \in G_{i+1}^d$. Thus, $\alpha_L(\gamma) = \alpha_H^{-1}(\gamma')$. Since α_H^{-1} fixes γ' by Lemma 3 we get that $\alpha_L(\gamma) = \gamma'$. □

4 Simplex-Chains and Cell-Chains

In this section we introduce the notation of simplex-chains. Simplex-chains are a natural generalization of Brisson's cell-tuples. Let $\mathcal{C}$ be a numbered simplicial complex.

Definition 5 (Simplex-Chain). *A sequence* $(X_{L_0}, \gamma_1, X_{L_1}, \ldots, \gamma_k, X_{L_k})$ *is a length* k **simplex-chain** *of* $\mathcal{C}$ *if:*

1. *Each* γ_i *is a consecutively numbered simplex of* $\mathcal{C}$ *of dimension* $d_i \geq 1$ *for* $1 \leq i \leq k$.
2. *Each* X_{L_i} *is 0-simplex with label* L_i *for* $0 \leq i \leq k$
3. $X_{L_i} = \gamma_i \cap \gamma_{i+1}$ *for* $0 < i < k$.
4. X_{L_0} *(* X_{L_k} *) is the 0-simplex in* γ_1 *(* γ_k *) with minimum (maximum) label.*

A simplex-chain is an **cell-chain** *if every* $d_i = 1$. *The* **dimension** *of the chain is* $d_1 + \cdots + d_k$. *In general we may drop some or all of the 0-simplices when they are not being considered since they are determined by the simplices containing them.*

Lemma 5. *If* $\mathcal{C}$ *is a numbered simplicial-complex of a commuting-map then every simplex-chain of length 2 and dimension* d' *is contained in a* d'*-simplex and thus a* d*-simplex.*

Proof. Let $(\lambda_L, X_i, \lambda_H)$ be a simplex-chain of length 2. There must exist d-simplices σ_L and σ_H containing λ_L and λ_H, respectively. Therefore $X_i \subset \sigma_L \cap \sigma_H$.

By the local connectivity property there exists a permutation $\alpha \in G_{X_i}$ such that $\alpha(\sigma_L) = \sigma_H$ By the commutivity property for cell-maps we know that $\alpha = \alpha_L \alpha_H$ where $\alpha_L \in G_0^{i-1}$ and $\alpha_H \in G_{i+1}^d$. Thus $\alpha_H(\sigma_L) = \alpha_L^{-1}(\sigma_H)$. We claim that $\alpha_H(\sigma_L)$ contains both λ_L and λ_H since $\alpha_H(\sigma_L)$ contains λ_L and $\alpha_L^{-1}(\sigma_H)$ contains λ_H by Lemma 3.

In a d-simplex every subset is a simplex thus there must exist a simplex of dimension that of (λ_L, λ_H) containing λ_L and λ_H. □

To prove that every simplex-chain is contained in a simplex we then use induction on the length of the chain. This gives the following Theorem:

Theorem 1. *If* $\mathcal{C}$ *is a numbered simplicial-complex of a commuting-map and* SC *is a simplex-chain of dimension* d' *then* SC *is contained in a* d'*-simplex.*

We now show that the simplex of minimum dimension containing a chain is unique. This will require that the map is a cell-map. We start by considering chains of length two.

Lemma 6. *If $\mathcal{C}$ is a numbered simplicial-complex of a cell-map and SC is a length two simplex-chain of dimension d' then SC is contained in a unique d'-simplex.*

Proof. Let $(X_i, \gamma_1, X_j, \gamma_2, X_k)$ be a simplex-chain of dimension d' and suppose by way of a contradiction that it is contained in two distinct d'-simplices λ and λ'. Let σ be some d-simplex that contains λ. Using the fact that $\gamma_2 \subset \lambda \cap \lambda'$ and Lemma 4 there exists $\alpha_L \in G_0^{j-1}$ such that $\lambda' \subset \alpha_L(\sigma) \neq \sigma$. Letting $\sigma' = \alpha_L(\sigma)$ and using the fact that $\gamma_1 \subset \lambda \cap \lambda'$ and the dual to Lemma 4 there exists $\alpha_H \in G_{j+1}^d$ such that $\alpha_H(\sigma') = \sigma$.

We have a contradiction since $\alpha_L \alpha_H(\sigma) = \sigma$ but $\alpha_L(\sigma) \neq \sigma$. □

Following by induction on the length of simplex-chains, we get the following theorem:

Theorem 2. *If $\mathcal{C}$ is a numbered simplicial-complex of a cell-map and SC is a d'-dimensional simplex-chain then SC is contained in a unique d'-simplex.*

We will use the following simple corollary:

Corollary 1. *For a complex of a cell-map there is a one-to-one correspondence between the cell-chains and d-simplices.*

We next show that the orthogonality condition is not only sufficient but necessary to get the uniqueness condition of Corollary 1.

Theorem 3. *If $\mathcal{C}$ is the simplicial complex of a commuting-map $\mathcal{M}$, then $\mathcal{M}$ is a cell-map if and only if there is a one-to-one correspondence between the cell-chains and d-simplices.*

Proof. The one-to-one correspondence is necessary by Corollary 1. We claim that it is also sufficient to make $\mathcal{M}$ a cell-map.
We must show orthogonality. To this end, let a d-simplex σ be given, and let $\alpha_L \in G_0^{j-1}$ and $\alpha_H \in G_{j+1}^d$ be given for some j, such that $\alpha_L \alpha_H(\sigma) = \sigma$.
Define $\sigma' = \alpha_H(\sigma)$. It then suffices to show that $\sigma = \sigma'$.
By assumption, we have a one-to-one correspondence between edge chains and d-simplices, so let us define X and X' as the unique edge chains associated with σ and σ' respectively. Let γ_i and γ_i' be the edges of X and X'.
Since $\sigma' = \alpha_H(\sigma)$, and since $\alpha_H \in G_{j+1}^d$, we have

$$[\sigma']_j = [\alpha_H(\sigma)]_j = [\sigma]_j$$

So that σ and σ' must have the same j-cell. Therefore $\gamma_i = \gamma_i'$ for $i < j$.
By a dual argument for the high-dimensional edges (using $\sigma = \alpha_L(\sigma')$), it holds that $\gamma_i = \gamma_i'$ for $i \geq j$. Hence $X = X'$, so by the assumed correspondence, $\sigma = \sigma'$, so orthogonality must hold. □

We can use this theorem to give an algorithm for testing if a map is a cell-map. Given a map $\mathcal{M} = (\Sigma, \alpha_0, \ldots, \alpha_d)$, we can first verify that it is a commuting-map. If $|\Sigma| = n$, then this can be done naively in $O(nd^2)$, since there are $O(d^2)$ permutations that must commute for each of the n elements.

We can then use a commuting-map to build the incidence multigraph as follows. Begin with a set of n disconnected cell-chains of length d, one for each element. Then for each α_i, for each element σ, we join the two chains associated with σ and $\sigma' = \alpha_i(\sigma)$. Joining these two chains consists of identifying all of their edges except γ_i and γ_{i+1}, this makes $O(d)$ edges that must be joined for each σ and for each α_i. Naively this takes $O(nd^2)$, but a simple divide-and-conquer approach can yield $O(nd \log d)$.

Once the incidence multigraph is constructed, we need simply count the number of maximal paths (number of cell-chains). Denote this as t. This path-counting can be accomplished by simple dynamic programming.

Since we have a commuting-map, by Theorem 1, each cell-chain is contained in some σ. Since each σ can contain at most one cell-chain, it follows by counting that the cell-chains and Σ are in one-to-one correspondence iff the $t = n$. Hence by Theorem 3, we have a cell-map iff $t = n$.

5 Cells

In this section we consider cells of a numbered simplicial complex of commuting-maps and cell-maps. We consider the natural decomposition of the complex into cells, Definition 6.

The main goal of this section is to show that for cell-maps each cell, in a natural way, can also be viewed as a complex of a cell-map. This justifies the definition of a **cell-map** (Definition 3). A CW-complex is a homotopy-theoretic generalization of the notion of a simplicial complex. A CW-complex is a space X which is partitioned in a collection of cells each homeomorphic to an open ball of some dimension. Under certain conditions a CW-complex can be partitioned into a simplicial complex using a barycentric decomposition. We are interested in simplicial complexes which admit a natural decomposition into cells. We will show that cell-maps do this for us.

We first show that many of the permutation that we have defined to act on the d-simplices actually act in a well defined way on subsimplices.

In this section we give a standard formal definition of a cell in a barycentric-complex. We then show that each cell can be viewed as barycentric-complex and if the original complex comes from a cell-map then the cells also can be viewed as coming from a cell-map.

Let $\mathcal{C}$ be a numbered simplicial-complex derived from a cell-map $\mathcal{M} = (\Sigma, \alpha_0, \ldots, \alpha_d)$.

Definition 6. *Let X_i be a 0-simplex of $\mathcal{C}$ with label i. The cell or i-cell C of X_i is the set of all simplices of $\mathcal{C}$ containing X_i with labels $\leq i$.*

Except for 0-cells a cell is not simplicial-complex (They are not closed under subsimplices). There are two natural closures, namely, the closure in $\mathcal{C}$ and the free-closure. The closure in $\mathcal{C}$ is simply the smallest simplicial-complex in $\mathcal{C}$ containing C. To obtain the **free-closure** we add new simplices to C to close it under subsimplices.

We first show that the elements of G_0^{i-1} act in a well defined way on the i-simplices of an i-cell C.

Lemma 7. *If $\mathcal{C}$ is a numbered simplicial-complex of a commuting-map and τ and γ are d-simplices that contain an i-simplex λ of an i-cell, and $\alpha \in G_0^{i-1}$ then $[\alpha(\tau)]_i = [\alpha(\gamma)]_i$*

Proof. It will suffice to prove the lemma for each α_j where $0 \leq j < i$. Let τ, γ, and λ be as in the lemma hypothesis. By the Local Connectivity Lemma, Lemma 2, there exists a permutation $\alpha \in G_{i+1}^d$ such that $\alpha(\tau) = \gamma$. This gives the following sequence of equalities:

$$[\alpha_j(\gamma)]_i = [\alpha_j(\alpha(\tau))]_i = [\alpha(\alpha_j(\tau))]_i = [\alpha_j(\tau)]_i$$

□

By the last lemma we see that we can define a natural action of $\alpha \in G_0^{i-1}$ on an i-simplex λ of an i-cell C. Namely, $\alpha(\lambda)$ equals the i-simplex λ' of C that is obtained by taking any d-simplex containing λ, applying α, and then restricting the image to an i-cell of C. The lemma says that it does not matter which d-simplex we pick.

Thus, if $\bar{C}$ is the free closure of an i-cell in $\mathcal{C}$ with Σ' being the i-simplices of $\bar{\mathcal{C}}$ then $(\Sigma', \alpha'_0, \ldots, \alpha'_i)$ is a map where α'_i is the identity permutation and α'_j for $0 \leq j < i$ is the permutation given by Lemma 7 as described above. If we want $(\Sigma', \alpha'_0, \ldots, \alpha'_i)$ to be a map then we must insure that α'_j is fixed-point-free for $0 \leq j < i$. This requires that $(\Sigma, \alpha_0, \ldots, \alpha_d)$ be a cell-map. Its will give us the nice property that $(\Sigma', \alpha'_0, \ldots, \alpha'_i)$ is also a cell-map.

Lemma 8. *If $(\Sigma, \alpha_0, \ldots, \alpha_d)$ is a cell-map then $(\Sigma', \alpha'_0, \ldots, \alpha'_i)$ as defined above is a cell-map.*

Proof. We first will show that $(\Sigma', \alpha'_0, \ldots, \alpha'_i)$ is map. It will suffice to show that α'_j for $0 \leq j < i$ is fpf. Suppose by way of contradiction that γ is a fix point of α'_j and σ is a d-simplex containing γ. By the local connectivity lemma there exists an $\alpha \in G_{i+1}^d$ such that $\alpha\alpha_j(\sigma) = \sigma$. Therefore by the orthogonality condition $\alpha_j(\sigma) = \sigma$, a contradiction. Thus $\alpha_j(\gamma) \neq \gamma$.

It should be clear that Σ' will inherit the commutivity property. We next show that it has the orthogonality property.

Suppose that $\alpha \in G_0^{j-1}$ and $\alpha' \in G_{j+1}^i$ and $\alpha\alpha'(\gamma) = \gamma$ for a i-simplex γ in the i-cell. Let σ be a d-simplex containing γ. We claim that $\alpha\alpha'$ fixes σ. For suppose that $\alpha\alpha'(\sigma) = \sigma'$. By the local connectivity lemma we know that there exists a $\alpha'' \in G_{i+1}^d$ such that $\alpha''(\sigma') = \sigma$. Thus $\alpha''\alpha\alpha'(\sigma) = \sigma$. By the orthogonality property $\alpha\alpha'(\sigma) = \sigma$. Again by the orthogonality property $\alpha(\sigma) = \alpha'(\sigma) = \sigma$. Therefore $\alpha(\gamma) = \alpha'(\gamma) = \gamma$.

Thus $(\Sigma', \alpha'_0, \ldots, \alpha'_i)$ has all the properties of a cell-map. □

Thus the free-closure of our cells are also cells-maps and their permutations are just the natural ones inherited from the full cell-map. We think of the permutation as also acting on the subsimplices when it is well-defined.

6 Axiom Justification

In this section we justify our axioms of a cell-map. In particular, using the work of Brisson [Bri93] we show that our combinatorial axioms hold for the generalized barycentric subdivision of a subdivided d-manifold (also known as a finite regular CW-complex). Thus the topological class of structures able to be represented by cell-complexes can be characterized combinatorially as cell-maps.

Throughout this section we will assume the manifold and subdivision has no boundary.

Brisson's notion of a cell-tuple can be defined it in the following terms, where $\mathcal{C}$ is a numbered d-simplicial complex.

Definition 7. *A sequence* $(X_0, \ldots, X_d)$ *is a* **cell-tuple** *if*

1. *Each* X_i *is a vertex with label* i.
2. *Each pair* X_i *and* X_{i+1} *is contained in a edge-simplex.*

A **cell-triple** is just a consecutive sub-tuple of length three.

Since each X_i is interior to a unique cell we could as Brisson does replace each vertex with its cell.

Brisson does not actually prove the following lemma but does state it to be true, page 404, line 8, [Bri93].

Lemma 9. *Every cell-tuple of* $\mathcal{C}$ *is contained in a unique* d*-simplex of* $\mathcal{C}$.

Following this lemma for each $0 \leq i \leq d$, there is a unique fixed-point free involution that Brisson calls $switch_i$, which takes a cell-tuple $(X_0, \ldots, X_i, \ldots, X_d)$ to another cell-tuple $(X_0, \ldots, \bar{X}_i, \ldots, X_d)$ where $X_i \neq \bar{X}_i$.

The main fact we need from Brisson is Corollary 1 from [Bri93]:

Lemma 10. *The action of* $switch_i$ *is well-defined on the cell-triple* (X_{i-1}, X_i, X_{i+1}).

Here well defined means that the image of X_i under $switch_i$ is only dependent on X_{i-1} and X_{i+1}.

We next claim that the switch acting on the cell-tuple is a cell-map.

Lemma 11. *Let* Σ *be the cell-tuples of a generalized barycentric subdivision of a subdivided* d*-manifold and* $switch_i$*'s be the switch operators acting on* Σ *then* $(\Sigma, switch_0, \ldots, switch_d)$ *is a cell map.*

Proof. The switch operators are fixed-point free involutions by Lemma 9. The commutivity and orthogonality conditions follow from Lemma 10. □

7 Implementation Issues

As we stated in the introduction, the use of Brisson's cell-tuples has the nice property that the cells are explicitly presented. This is why we propose using

cell-chains as defined in Section 4. The reason is that we would expect most edges (in the incidence multigraph) of a cell-chain to be determined by their vertices (cells). In particular if we label the edges between two consecutive vertices with integers starting from zero we would expect that all but a small number would be labeled with zero.

One very important feature of Brisson's representation for non-degenerate structures is that the permutation $switch_i$ is well defined and is determined solely by its action on cell triples of the form (X_{i-1}, X_i, X_{i+1}). For degenerate structures this is not the case. For cell-maps, by a simple extension of Lemma 7, the permutation α_i is well defined and determined by its action on consecutively numbered 2-simplices with labels $\{i-1, i, i+1\}$. By Theorem 3 there is a one-to-one correspondence between every cell-chain of length two and every consecutively numbered 2-simplex. Thus we have:

Lemma 12. *For cell-maps for each $0 \leq i \leq d$ the permutation α_i is well defined and determined by its action on cell-chains of the form $(X_{i-1}, \gamma_1, X_i, \gamma_2, X_{i+1})$*

Thus we need only store the action of each α_i on cell-chains $(X_{i-1}, \gamma_1, X_i, \gamma_2, X_{i+1})$. Since most cell-chains will be determined by their cells we propose to have two data structures for storing the value of length two cell-chains. One for the case when the cell-chain is determined by its cell-tuple and one for the others. In the prior case, we use a regular cell-tuples structure, and in the latter case, we use complete length two cell-chains structure. If the image of a cell-tuple is an cell-chain then we store the image in the cell-chain data structure costing two lookups.

The ability to store the action on triples instead of full chains leads to an important improvement in space efficiency for high dimensions. The number of cell-chains within a given cell is nominally $O(d!)$, but by only storing triples we see a vast decrease in complexity, allowing the switch structure to be stored in space smaller than that of the entire incidence multi-graph.

8 Two and Three Dimensional Clusters

In this section we show that 2D clusters have size at most 4. In 3D we give an example of a cell-map with unbounded size cluster. Recall a cluster is a set of d-simplices all containing the same cell-tuple. We say the cluster is nontrivial if it has size two or more.

Theorem 4. *A nontrivial cluster of a 2-dimensional simplicial complex of a cell-map has size either two or four. If the cluster has size four then the surface of the complex must be non-planar.*

Proof. Let (X_0, X_1, X_2) be a cell-tuple of the simplicial complex. We know by Theorem 2 that there is a one-to-one correspondence between the simplices containing the cell-tuple and cell-chains containing the cell-tuple. Thus it will suffice to count the containing cell-chains. Since a 1-cell contains exactly two vertices

there can be at most two edges from X_0 to X_1. By duality there are at most two edges from X_1 to X_2. Thus there are at most four cell-chains containing (X_0, X_1, X_2) and the number can either be one, two, or four.

Consider the case when there are four cell-chains containing a cell-tuple. The two edges containing (X_0, X_1) form one cycle and the two edges containing (X_1, X_2) form another one. These two cycles cross only at X_1 and in a fundamental way. Thus the surface must not be planar. □

In the case of 3D clusters we need to understand the cell-chains containing a cell-tuple (X_0, X_1, X_2, X_3). Based on the discussion in the proof of Theorem 4 there can be at most two edges containing (X_0, X_1) and at most two containing (X_2, X_3). We show by an example that there can be an unbounded number of edges containing (X_1, X_2).

Consider the example of the so-called "napkin" surface in 3D. This surface is formed by taking a flat sheet and folding together several corners, then pinching the surface where the corners meet. In Figure 7, we show a napkin with four corners, though in general the number may be increased without bound. The surface in Figure 7 has two vertices (A and B), and five edges. The vertical edge is denoted e, and the four loops around the top are denoted e_1, e_2, e_3, and e_4.

The surface is shown unfolded in Figure 8(a), where we see that this surface is homeomorphic to an open disk. It's barycentric decomposition is seen in Figure 8(b). Note that there are 8 simplices defined by C,e,B, and in general the

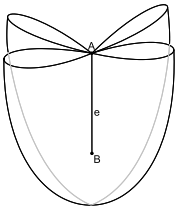


Fig. 7. The napkin example with four folds

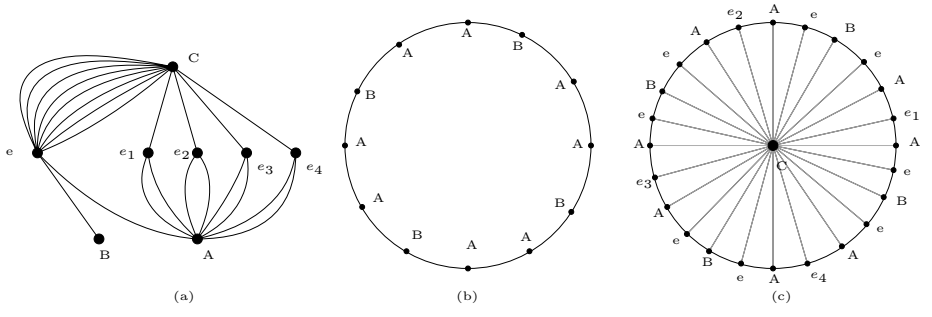


Fig. 8. (a) The incidence multigraph of the napkin. (b) The napkin surface and its boundary. (c) Barycentric decomposition of the napkin surface.

size of this cluster is twice the number of leaves chosen. The use of cell-chains allows us to distinguish within this cluster by numbering the different spokes connecting C and e in Figure 8(b).

References

[Bau72] B. Baumgardt. Winged-edge polyhedron representation. Technical Report CS-320, Stanford University, 1972.

[BEA+99] Marshall Bern, David Eppstein, Pankaj K Agarwal, Nina Amenta, Paul Chew, Tamal Dey, David P Dobkin, Herbert Edelsbrunner, Cindy Grimm, Leonidas J Guibas, John Harer, Joel Hass, Andrew Hicks, Carroll K Johnson, Gilad Lerman, David Letscher, Paul Plassmann, Eric Sedgwick, Jack Snoeyink, Jeff Weeks, Chee Yap, and Denis Zorin. Emerging challenges in computational topology, 1999.

[Bri89] E. Brisson. Representing geometric structures in d dimensions: Topology and order. In *Symposium on Computational Geometry*, pages 218–227, 1989.

[Bri93] E. Brisson. Representing geometric structures in d dimension: Topology and order. *Discrete and Computational Geometry*, 9:387–426, 1993.

[BS85] R. Bryant and D. Singerman. Foundations of the theory of maps on surfaces with boundaries. *Quarterly Journal of Mathematics*, 2(36):17–41, 1985.

[CCM+04] D. Cardoze, A. Cunha, G. Miller, T. Phillips, and N. Walkington. A Bezier-Based Approach to Unstructured Moving Meshes. In *20th Symposium on Computational Geometry*, 2004.

[DFM02] M.O. Deville, P.F. Fischer, and E.H. Mund. *High-Order Methods for Incompressible Fluid Flow*. Cambridge University Press, 2002.

[DL87] D. Dobkin and M. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. In *Third ACM Symosium on Computational Geometry*, pages 86–99, 1987.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, New York, 1987.

[Edm60] J.R. Edmonds. A combinatorial representation for polyhedral surfaces. *Notices Amer. Mac. Soc.*, 7, 1960.

[Far99] Rida T. Farouki. Closing the gap between cad model and downstream application. *SIAM News*, 32(5), June 1999.

[GS85] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions an the computation of voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[Hal97] Halperin. Arrangements. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 1997.

[Hop96] H. Hoppe. Progressive meshes. In *ACM SIGGRAPH*, pages 99–108, 1996.

[Jan84] K. Janich. *Topology*. Springer-Verlag, 1984.

[Kin93] L.C. Kinsey. *Topology of Surfaces*. Springer-Verlag, 1993.

[Lie91] P. Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23:59–82, 1991.

[Lie94] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry & Applications*, 4:275–324, 1994.

[LLLV05] Marcos Lage, Thomas Lewiner, Hlio Lopes, and Luiz Velho. Chf: A scalable topological data structure for tetrahedral meshes. In *In: Proceedings of the XVIII Brazilian Symposium on Computer Graphics and Image Processing*, pages 349–356. IEEE Press, 2005.

[LM] Bruno Lèvy and Jean-Laurent Mallet. Cellular modeling in arbitrary dimension using generalized maps.

[LW69] A. Lundell and S. Weingram. *The Topology of CW Complexes*. Van Nostrand Reinhold, 1969.

[Män88] Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Inc., 1988.

[Mar58] A.A. Markov. Insolubility of the problem of homeomorphy. In *Proceedings of the International Congress of Mathematics*, pages 300–306, 1958. Engish Translation by Afra Zomorodian.

[Moi77] E. Moise. *Geometric Topology in Dimensions* 2 *and* 3. Springer-Verlag, 1977.

[RF90] R. Piccinini R. Fritsch. *Cellular Structures in Topology*. Cambridge, 1990.

[RO89] J. Rossignac and M. O'Conner. Sgc: A dimension independent model for pointers with internal structures and incomplete boundaries. In *IFIP/NSF Workshop on Geometric Modeling*, pages 145–180, 1989.

[Ros97] J. Rossignac. Structured topological complexes: A featured-based api for non-manifold topologies. In *ACM Symposium on Solid Modeling and Applications*, pages 1–9, 1997.

[San] The Sangria Project. `http://cs.cmu.edu/~sangria`. Supported by NSF ITR ACI-0086093.

[Sch] Saul Schleimer. Sphere recognition lies in NP. `http://front.math.ucdavis.edu/math.GT/0407047`.

[Tit81] J. Tits. A local approach to buildings. In C. Davis, B. Grümbaum, and F.A. Sherk, editors, *The Geometric Vein*, pages 519–547. Springer-Verlag, 1981.

[Tum] The tumble software package. `http://rioja.sangria.cs.cmu.edu/`. Supported by NSF ITR ACI-0086093.

[Tut73] W.T. Tutte. What is a map? In *New Directions in the Theory of Graphs*, pages 309–325. Acad. Press, 1973.

[Tut84] W.T. Tutte. *Graph Theory*. Cambridge University Press, 1984.

[Vin83a] A. Vince. Combinatorial maps. *Journal of Combinatorial Theory B*, 34:1–21, 1983.

[Vin83b] A. Vince. Regular combinatorial maps. *Journal of Combinatorial Theory B*, 34:256–277, 1983.

[VKF74] I.A. Volodin, V.E. Kuznetsov, and A.T. Fomenko. The problem of discriminating the standard sphere. *Russian Mathematical Surveys*, 29(5):71–172, 1974.

[Wei85] K. Weiler. Edge-based data structures for solid modeling in curve-surface environment. *IEEE Computer Graphics and Applictions*, 5(1):21–40, 1985.

[Wei86] K. Weiler. *Topological Structures for Geometric Modeling*. PhD thesis, Rensselaer Polytechnic Institute, Troy. N.Y., 1986.

Constructing Regularity Feature Trees for Solid Models

M. Li, F.C. Langbein, and R.R. Martin

School of Computer Science, Cardiff University, Cardiff, UK
{M.Li, F.C.Langbein, R.R.Martin}@cs.cf.ac.uk

Abstract. Approximate geometric models, e.g. as created by reverse engineering, describe the approximate shape of an object, but do not record the underlying design intent. Automatically inferring geometric aspects of the design intent, represented by feature trees and geometric constraints, enhances the utility of such models for downstream tasks. One approach to design intent detection in such models is to decompose them into *regularity features*. Geometric regularities such as symmetries may then be sought in each regularity feature, and subsequently be combined into a global, consistent description of the model's geometric design intent. This paper describes a systematic approach for finding such regularity features based on recovering broken symmetries in the model. The output is a tree of regularity features for subsequent use in regularity detection and selection. Experimental results are given to demonstrate the operation and efficiency of the algorithm.

1 Introduction

Reverse engineering creates a geometric model from measured 3D data [25]. This model is not necessarily suitable for applications which need to modify or analyse it: it suffers from inaccuracies caused by sensing errors, as well as approximation and numerical errors arising during reconstruction. Such models are approximate in the sense that intended *regularities* like symmetries, congruent sub-parts, aligned cylinder axes, etc. within the model are not exactly represented. Furthermore, even if a regularity *is* preserved to within a sufficiently small tolerance, it can easily be destroyed by later editing operations, if it is not *explicitly* denoted as a property to be preserved. It is thus desirable to explicitly determine the geometric design intent of such models, as embodied by regularities. In this paper we consider the first step of this process, decomposing boundary representation (B-rep) models into *regularity feature trees* (RFTs). Note that such models may have come from reverse engineering, but could also be any other model whose design intent is not explicitly known or has been lost.

Our overall goal is to represent the geometric design intent of a model using a *feature tree* augmented with geometric constraints describing regularities, for processing with a feature-aware constraint solver such as Frontier [23]. Regularities are geometric properties the designer may have desired, e.g., global symmetries of vertices, or congruent sub-parts arranged in a regular manner, or plane normals and other directions forming an orthogonal system. We do not

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 267–286, 2006.
© Springer-Verlag Berlin Heidelberg 2006

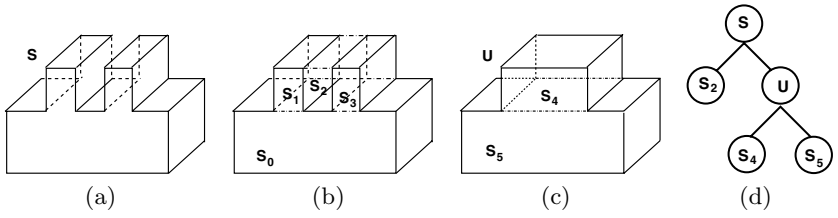


Fig. 1. A simple model and its regularity feature tree

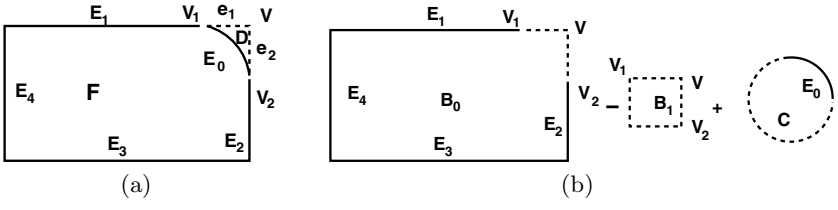


Fig. 2. Expressing a simple face with regularity features and CSG primitives

consider higher-level *functional* or *aesthetic* intent, nor do we look for features specifically useful for purposes such as machining.

Previous work proposed *beautification* of approximate reverse engineered geometric models to improve them with respect to design intent [3,7,8,14]. This approach determines candidate regularities of the *whole model* based on symmetries, and then imposes a consistent subset of these regularities by solving a geometric constraint system. This works well for models with a limited number of ambiguous interpretations in terms of regularities, e.g. models with a major rotational symmetry axis where the rotational symmetry is only broken by a few features. However, complex models often have too many *alternative* plausible approximate regularities for decision methods to be able to determine which regularities represent the original design intent of the whole model. Thus, in this paper we consider the construction of an RFT to simplify the problem by decomposing the whole model into manageable parts, hierarchically. Subsequent work will consider the regularity analysis in each regularity feature *separately*, so these can be combined gradually to form a global description of the intended overall shape. Furthermore, we will analyse the relations *between* regularity features to detect, e.g., congruences and symmetries. As a simple example, consider a rectangular block with many prisms attached to its faces. Analysing the *whole* model without finding the prismatic features creates many candidate plausible angles to enforce between planes in the model. By first identifying the individual prisms as features, we can detect their approximate prismatic symmetries, and separately determine potential regular arrangements of the prisms on the block. Only considering *parts* of the model at any one time will increase the speed of regularity analysis *and* provide more reliable results.

By *regularity features*, we mean simple volumes *derived from the model* which expose hidden approximate regularities in the model. Just like machining features [4],

they are not a class of pre-defined simple geometric primitives, but are determined by the model's geometry. However, in this case, they are constructed to expose intended regularities in the model, rather than describing how to manufacture it. For example, in Fig. 1, the entities of the model S are grouped into volumes S_0 to S_3, whose union gives an updated model U. U is further grouped into S_4 and S_5. As explained in detail below this groups S into regularity features S_2 and U, and U further into S_4, S_5 as shown in Fig. 1(d).

As many regularities can be expressed in terms of symmetries [8], our method for finding regularity features relies on recovering broken symmetries. While this means that some regularity features may become more symmetric, others may just represent the *symmetry break* in the model, i.e. some part which reduces the model's symmetry. To find regularity features, we look for *recoverable edges* and *recoverable faces*. A recoverable face is a newly generated face, with the same underlying geometry as an existing face, but different boundaries, and which, when added to or removed from the existing face, allows us to recover the broken symmetry of the face. We recover the symmetry of a face by modifying its boundary within its underlying geometry.

In order to find recoverable faces, we first detect recoverable edges: newly generated edges based on symmetry expectations derived from faces of the approximate model. The recoverable faces are bounded by a combination of original and recoverable edges. E.g. in Fig. 2(a), using the broken symmetry hints, the two recoverable edges V_1V and V_2V are first constructed. From these we can then determine the recoverable face D to produce the rectangular face B_0. The asymmetry of F with respect to B_0 is represented by D.

Once detected, the recoverable faces are used together with appropriate original faces of the model to find regularity features as *cells*, closed volumes which do not contain any (new or original) faces in their interior. This leads to *negative* and *positive* regularity features: a regularity feature is negative if the orientation of its faces is inverse to any original faces with the same geometry in the model. E.g. for model S in Fig. 1(a), recoverable edges and faces illustrated by dot-dashed lines in Fig. 1(b) are detected and four regularity features are constructed: blocks S_0, S_1, S_2 and S_3; S_2 is negative and the others positive.

Using these regularity features, a more symmetric updated model is constructed which exposes partly recovered symmetries of the original model. Volumes corresponding to negative regularity features are added to the original model, while (certain) positive features are cut off. Note that only positive solids *not* adjacent to negative solids can be cut off from S. Other positive regularity features are ignored, because adding the negative solid may change the local connectivity and yield a simpler structure overall. E.g. the negative regularity feature S_2 in Fig. 1(b) is first added to the model S, resulting in an updated model U (Fig. 1(c)). Since all positive regularity features S_0, S_1 and S_3 have faces in common with S_2, no further updating is required. The original model is decomposed into the negative regularity feature S_2 and the updated regularity feature U, which become the children of S. In this way, all geometric entities of S are transmitted to its children, which are processed recursively to construct

the tree. Recursion stops when no further regularity features are found. For U this creates two additional regularity features S_4 and S_5 (Fig. 1(c)), while S_2 is not decomposed any further. The corresponding RFT is shown in Fig. 1(d).

For simplicity, throughout this paper, we assume that the approximate input model is a manifold 3D solid represented by a valid, watertight B-rep data structure, and is bounded by planar, spherical, cylindrical, conical and toroidal surfaces, which covers a wide range of mechanical components [15]. The only reason for this restriction is the difficulty of extending the geometry of free-form surfaces. We assume that blends have been identified and suppressed using existing blend-removal methods [21,30]. Finally, we assume that all geometries are represented parametrically and we denote the complete underlying parametric curve of an edge E as $\tilde{E}$.

The next section discusses how our novel ideas are related to earlier work. In Sect. 3 we describe recoverable edges and faces in detail. We then outline our algorithm in Sect. 4, and give further algorithmic details in Sect. 5. Section 6 presents some experimental results.

2 Related Work

Our proposed algorithm is clearly closely related to previous work on feature recognition, conversion of B-rep models to CSG models, and conversion of wireframe models to solid models. We discuss relevant results in these areas and compare them to our method.

The current work, and *feature recognition*, both detect local shape information in solid models. However, feature recognition mainly focuses on detecting information needed to manufacture a solid model, such as holes, slots and pockets [4,17,24]. More closely related to our method are feature recognition techniques which try to construct feature volumes for such applications as tool accessibility analysis and process planning [2,6,18,19,20,26,27,28,29].

Convex hull decomposition, also called *Alternating Sum of Volumes* (ASV), and its variations, produce a hierarchical volumetric representation of solids from boundary information. It then recognises feature volumes in this decomposition. This approach subtracts an object from its convex hull to produce a new object recursively until a convex object is produced [6,16,26]. Such methods are not directly applicable for producing RFTs, for two reasons. Firstly, while such approaches work for polyhedra, they are difficult to generalise to objects with curved surfaces. Secondly, and more importantly, in general the convex hull does not recover broken symmetries such as cut off corners of cubes—this approach does not explicitly consider regularities.

In our method of finding regularity features, new recoverable edges and faces are generated to find feature volumes. Similar ideas have been applied to create feature volumes from *feature face-sets* [2,19,20,29]. The latter have to be detected or are sometimes assumed to be provided in advance, reducing the complexity of the problem. Our approach is applied directly to a solid model: all the needed geometric information is obtained from the model.

The idea of face intersections has also been applied in *cell-based* methods: the model is decomposed into a set of minimal cells by intersecting all faces of the model, having extended them as necessary using their underlying geometry. These cells are then merged into subsets to form feature volumes [18,27]. However, whereas features are *local* parts of a model, this approach makes *global* use of local geometry, leading to a combinatorial explosion in the number of cells generated and merged; nevertheless, recent work has limited this problem by using localised face extensions, and cell collection using seed cells [28]. In contrast, our algorithm detects regularity features as minimal volumes, defined by recoverable faces generated via *local* extensions of existing geometry. In combination with recursive processing of regularity features, this greatly reduces the number of volumes to be considered.

In feature recognition, it is important to carefully choose the newly generated geometric entities used to form features. Regularity features are supposed to reveal symmetries of the model, an issue not been addressed in previous work. Our approach systematically constructs regularity features using carefully chosen newly generated entities based on the idea of recovering broken symmetries. To obtain all the possible regularity features and avoid the need for heuristics, all possible local entities are constructed, and selection amongst them occurs naturally during the entity construction process. Specifically, *only* those recoverable edges, faces or regularity features making a contribution towards generating a recoverable face, regularity feature or updated model respectively, are kept.

Our approach is also closely related to methods for converting B-rep models to CSG models [22], which try to determine how to construct a model from primitives using Boolean operations. Our method shifts the emphasis from the construction process to finding suitable primitives, the regularity features. In CSG models, the primitives are simple regular solids, whereas in our method, the regularity features are solids which, when added to or removed from other solids, produce a more regular solid. For example, the 2D face F in Fig. 2(a) is not difficult to express using Boolean operations between simple CSG primitives, e.g. rectangle B_0 minus rectangle B_1 plus circle C as shown in Fig. 2(b); original geometric entities in F are drawn with solid lines and added geometric entities are drawn with dashed lines. Such expressions are not directly required for regularity detection, and can involve extra unnecessary geometric primitives, such as the rectangle B_1 completely composed of dashed lines. Building up such expressions consequently can require additional unnecessary computation. In our method, we simply express F as regularity features B_0, the intended model symmetry, and D, the symmetry break. As a further difference, note that D could in principle intersect with another part of the model due to some unwanted *global*, rather than local, interaction. This must be taken into consideration when constructing a CSG model, but for regularity processing can simply be ignored.

Construction of regularity features from recoverable edges requires similar steps to *converting wire-frame models* into solid models [1,5,11,13]. Given a wire-frame model linking vertices and edges, these methods produce a corresponding volumetric description by deciding which edge loops are covered by faces. The

main issues here are algorithmic efficiency, and how to enumerate all multiple interpretations—or to choose the most appropriate one from amongst them. A general approach for polyhedra is based on ordering edges around each vertex and faces around each edge to create all possible solutions [13]. More general work to reconstruct curved solids from engineering drawings uses a maximal turning angle method to detect all possible faces in a wire-frame with straight and conic edges [11]. Graph theory has been applied to resolve the conversion problem for 0-, 2- and 3-connected wire-frames [1,5]. In our method, unlike in the wire-frame case, the face normals of the model are available, as well as those of any generated recoverable faces (determined uniquely by the original edge orientations), greatly simplifying the construction of regularity features.

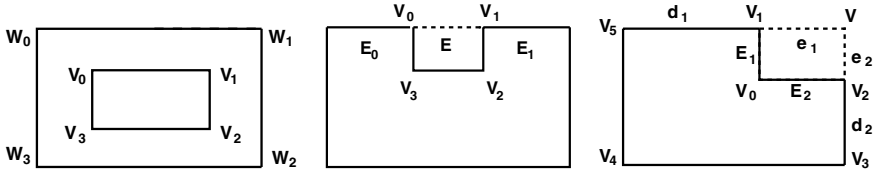
In summary, like previous work on feature recognition, B-rep to CSG conversion, ASV and related approaches, our method hierarchically decomposes a model into parts. These approaches are closely related and even sometimes produce similar results. However, our approach has the goal of recovering symmetries in the model, while previous work is based on other geometric aspects such as the convex hull, specific primitives, or machining features. For our application—detecting geometric design intent—symmetry is the important geometric aspect.

3 Recoverable Edges and Faces

Our algorithm is based on recovering symmetries that were broken during construction of the (ideal, rather than approximate) original model. Constructing geometric models by using modelling operations on geometric primitives usually destroys regularity in structures which were originally simple. One can understand this process as a sequence of symmetry breaking operations—see e.g. [10]. These operations often leave hints in the model from which they may be recovered. We recover broken symmetries by first analysing symmetry breaks in faces, from which symmetry breaks of the model are constructed. Specifically, the symmetries of a primitive face might be broken in its interior, across one or more edges, or surrounding one or more vertices—or some combination of these. By adding entities we may recover the original symmetries.

To simplify the processing required, we reduce complex cases of broken symmetry to combinations of elementary cases which we call *Missing Face Segment (MFS)*, *Missing Edge Segment (MES)*, and *Missing Vertex Segment (MVS)*, examples of which are shown in Fig. 3. For MFS cases, to regain the symmetry we must generate a new face bounded by an inner edge loop of a face of the input model. MES cases require a new edge, and MVS cases require a new vertex; other associated geometry is also required in these cases.

The new geometry required to recover the symmetry is composed of recoverable vertices, recoverable edges and recoverable faces. A *recoverable vertex* is a newly generated intersection vertex between certain edges, e.g. vertex V in Fig. 3(c). A *recoverable edge* is a newly generated edge that is not already represented in the model and connects two vertices (original or newly generated) that lie in the underlying geometry $\tilde{E}$ of some edge E of the model, e.g. edge



(a) Missing face segment1 (MFS) (b) Missing edge segment (MES) (c) Missing vertex segment (MVS)

Fig. 3. Three elementary cases of breaking face symmetries

E in Fig. 3(b) or e_1 and e_2 in 3(c). A *recoverable face* is a newly generated face with an underlying surface derived from a face in the model, bounded by recoverable or original edges, e.g. the face bounded by the loop V_0, V_1, V_2, V_3 or W_0, W_1, W_2, W_3 in Fig. 3(a), the face bounded by the loop V_0, V_1, V_2, V_3 in Fig. 3(b) or the face bounded by the loop V_1, V, V_2, V_0 in Fig. 3(c). We distinguish between two recoverable edge types depending on whether they relate to the MES or MVS case as explained next.

3.1 MES Recoverable Edges

MES recoverable edges are straight-forward to define. Let V, V' be two vertices of the input model M such that

(1) these vertices lie on the underlying geometry of some edge E of the model M, i.e. $V, V' \in \tilde{E}$;
(2) the segment S of the underlying curve $\tilde{E}$ bounded by V, V' contains no other vertex of M in its interior;
(3) there is no edge E' of M with underlying geometry $\tilde{E}$ bounded by V, V'.

Then the edge E^* with underlying geometry $\tilde{E}$, bounded by V and V', is called an *MES recoverable edge*. E.g., in Fig. 3(b), the newly generated edge E bounded by V_0 and V_1 is an MES recoverable edge with underlying geometry $\tilde{E}_0$.

3.2 MVS Recoverable Edges

MVS recoverable edges are more complex than MES ones, as we have to decide which two vertices, original or newly generated, to use to construct the edges needed when recovering the face's symmetry. We first show how this can be done for a vertex on a planar face at which straight edges meet, and then how to extend the idea to more general cases.

We first introduce some terminology. A vertex on a planar face bounded by two straight edges at which the interior angle is greater than π is called *concave*, e.g. V_0 in Fig. 3(c). We call the two incident edges at a concave vertex V_0 on a planar face F *boundary edges*, e.g. E_1, E_2 in Fig. 3(c). The end-points of boundary edges, other than V_0, are called *boundary vertices*, e.g. V_1, V_2. An edge of F that is not a boundary edge but contains a boundary vertex is called an *external edge*, e.g. d_1, d_2. A line normal to an external edge at the corresponding boundary vertex is called an *associated external line*, e.g. n_1, n_2 in Fig. 4 at the concave vertex V_0.

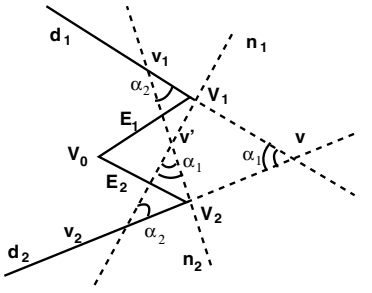


Fig. 4. MVS recoverable vertex and edge generation

A face of the model sharing the external edge with F is called an *external face*, e.g. F_1, F_2 in Fig. 6(a). As the input model M is a manifold solid, the number of boundary vertices, boundary edges, external edges, associated external lines and external faces is always two. Any edge containing a boundary vertex but not lying on F is called an *associated edge*, e.g. E_3 and E_4 in Fig. 6(a). An *associated external plane* is a new plane containing the associated external line and an associated edge originating from a boundary vertex. More than three edges can originate from one vertex, e.g. at the apex of a pyramid, so there may be more than one associated edge or associated external plane originating from a boundary vertex.

The external edges and faces usually give some indication of the local symmetry at a concave vertex, and are helpful in recovering the broken symmetry of the face. Furthermore, in engineering, rectangles play a particularly important role and are often present. In the following definitions of MVS recoverable vertex and edge, we seek the 'most plausible' way of completing a rectangle using the external edges and associated external lines. Two principles are used. Firstly, the two generated recoverable edges should be those that form the closest possible angle to a right angle. This avoids having to use a tolerance value to determine whether or not two lines are orthogonal—choosing such a tolerance is non-trivial given an approximate input model. Secondly, the constructed recoverable vertex should lie on the opposite side of the line V_1V_2 determined by the two boundary vertices V_1V_2 from the concave vertex V_0. This ensures that the constructed recoverable edges do not intersect the two boundary edges, which must lie on the same side of line V_1V_2 as V_0 due to the concavity at V_0. See also Fig. 4.

With these considerations, a unique line pair (l_1, l_2) is selected from amongst the external edges and associated external lines around one concave vertex; their intersection point is the MVS recoverable vertex. For any two lines l_1, l_2, we denote their intersection vertex as $V_I(l_1, l_2)$ and their intersection angle ($\leq \pi/2$) as $A_I(l_1, l_2)$. Suppose V_0 is a concave vertex with boundary vertices V_1, V_2, external edges d_1, d_2 and associated external lines n_1, n_2 (see Fig. 4). Since n_i is orthogonal to $\tilde{d_i}$, $i = 1, 2$, it can be seen that $A_I(\tilde{d_1}, \tilde{d_2}) = A_I(n_1, n_2)$, $A_I(\tilde{d_1}, n_2) = A_I(n_1, \tilde{d_2})$, and $A_I(\tilde{d_1}, \tilde{d_2}) + A_I(\tilde{d_1}, n_2) = \pi/2$. Taking the four line pairs, we select (l_1, l_2) using the following method:

1. If $A_I(\tilde{d_1}, \tilde{d_2}) < \pi/4$:
   - If $V_I(\tilde{d_1}, n_2)$ lies on the opposite side of line V_1V_2 from V_0 and $V_I(n_1, \tilde{d_2})$ does not, set $l_1 = \tilde{d_1}$ and $l_2 = n_2$.
   - If $V_I(n_1, \tilde{d_2})$ lies on the opposite side of line V_1V_2 from V_0 and $V_I(\tilde{d_1}, n_2)$ does not, set $l_1 = n_1$ and $l_2 = \tilde{d_2}$.
2. If l_1 and l_2 do not yet have values:
   - If $V_I(\tilde{d_1}, \tilde{d_2})$ lies on the opposite of line V_1V_2 from V_0, set $l_i = \tilde{d_i}$, $i = 1, 2$;
   - otherwise set $l_i = n_i$, $i = 1, 2$.

A unique pair (l_1, l_2) is always obtained using the above definition: the second step always provides l_1 and l_2 whenever $\tilde{d_1}$ is not parallel to $\tilde{d_2}$, because $V_I(\tilde{d_1}, \tilde{d_2})$ and $V_I(n_1, n_2)$ must lie on different sides of line V_1V_2. Whenever $\tilde{d_1}$ is parallel to $\tilde{d_2}$, there always exists a unique point $V_I(\tilde{d_1}, n_2)$ or $V_I(n_1, \tilde{d_2})$ lying on the opposite of line V_1V_2 from V_0, and hence the first rule applies in this case.

The vertex $V_I(l_1, l_2)$ is the desired *MVS recoverable vertex*, e.g. vertex V in Fig. 3(c). Edges bounded by the recoverable vertex and the boundary vertices, with underlying curves l_1 or l_2, are called the *MVS recoverable edges*, e.g. edges e_1, e_2 in Fig. 3(c). Fig. 5 shows some examples of recoverable vertices V with recoverable edges e_1, e_2 obtained by this selection method.

More generally, for a face with at least two straight edges, we try to convert the involved curved edges into straight edges and then analyse them using the above approach, following these principles:

1. If one or more connected curved edges lie between two straight edges on the face, we treat them in the same way as we earlier treated a concave vertex lying between two straight edges; the two straight edges are considered to be the corresponding external edges. This is reasonable as the straight edges in the face may be hints for a broken symmetry in the input model. E.g. in Fig. 2(a) the curved edge E_0 is adjacent to two straight edges E_1 and E_2. Thus, E_1, E_2 are treated as the corresponding external edges, resulting in recoverable vertex V and recoverable edges e_1, e_2, which ultimately give a rectangle. In Fig. 6(b), the external edges for the curved edge C_0 are E_1 and E_2, resulting in a recoverable vertex V and recoverable edges e_1 and e_2.
2. If a concave vertex has straight boundary edges but a curved external edge, the external edge is replaced by its tangent line at the boundary vertex during generation of MVS recoverable edges. Generally this is a more plausible

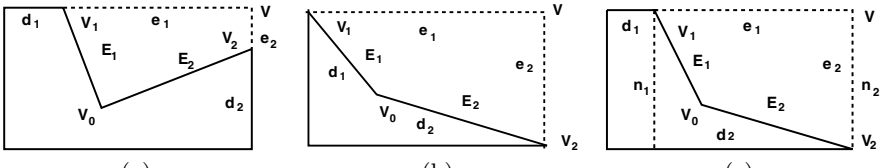
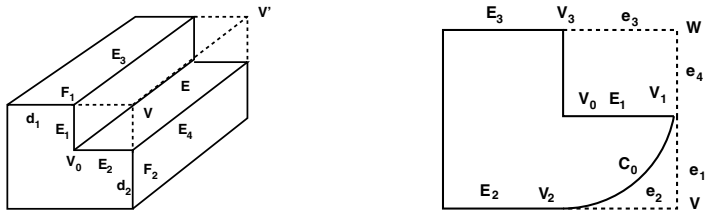


Fig. 5. Different intersection cases of external edges on a face and corresponding MVS recoverable vertex V and edges e_1 and e_2



(a) External faces for additional MVS recoverable edges

(b) MVS recoverable edge detection for general cases

Fig. 6. More general MVS cases

way of recovering model symmetry—leading to a rectangle—than using the curved edge directly. E.g. for the concave vertex V_0 in Fig. 6(b), the tangent line of C_0 at V_1 is used instead of C_0 in the generation of MVS recoverable edges, resulting in a recoverable vertex W and recoverable edges e_3 and e_4.

The model building operations used to construct the original model may have removed certain edges completely, e.g. edge E bounded by vertices V and V' in Fig. 6(a). External faces or associated external planes help to find the solution in such cases. The vertices V, V' which bound an MES recoverable edge may be MVS recoverable. If the MVS recoverable vertices are constructed from an external edge, the underlying curve of the MES recoverable edge is the intersection of the external faces. If the MVS recoverable vertices are constructed from an external line (instead of an external edge), the curve underlying the MES edge is the intersection of the associated external planes.

In the case of faces having at most one straight edge, it is difficult to design a universally useful method for finding MVS recoverable edges. We do not process them in our current approach, and leave a more sophisticated approach for future work. Our current implementation detects all MVS recoverable edges on planar faces. However, our concept of MVS recoverable edges is not limited to such cases in principle. For curved surfaces, we could consider the geometry of the boundary loop inside the underlying surface, and follow geodesics instead of straight lines.

4 Algorithm Overview

We now give an overview of our algorithm for constructing an RFT from an input manifold B-rep model M. Further details are presented in the next section. The RFT for M is built up by first constructing regularity features for M and then processing the resulting features recursively. In the following we refer to the solid currently being processed in this recursion as S.

We decompose S, if possible, into *edge-connected* solids, each of which is determined by a *separate component* of the edge graph of S. Each edge loop representing a boundary of an edge-connected solid is closed with a face as appropriate. If S consists of several edge-connected solids, they become children of S in the RFT, and each is processed by the following steps in turn as the current solid.

Algorithm. RFTConstruction (M)
Input: M—a manifold B-rep solid (with approximate geometry)
Output: T—a Regularity Feature Tree for M

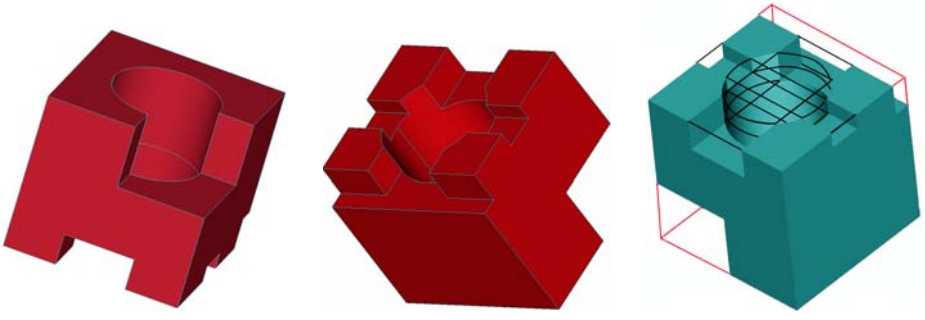
1. $T \leftarrow$ tree with root M to store regularity feature tree
2. $Q \leftarrow$ FIFO queue of solids, initially containing M
3. $C \leftarrow$ GroupFaces (M)
4. **while** $Q \neq$ *empty* **do**
5. $\quad S \leftarrow$ **pop** (Q)
6. $\quad R \leftarrow$ EdgeConnectedSolids (S)
7. $\quad$ **if** $|R| = 1$ **then**
8. $\quad\quad \mathcal{E}_{\mathrm{E}} \leftarrow$ MESRecoverableEdges (S, C)
9. $\quad\quad \mathcal{E}_{\mathrm{V}} \leftarrow$ MVSRecoverableEdges (S, $\mathcal{E}_{\mathrm{E}}$, C)
10. $\quad\quad \mathcal{F}_{\mathrm{R}} \leftarrow$ RecoverableFaces (S, $\mathcal{E}_{\mathrm{E}} \cup \mathcal{E}_{\mathrm{V}}$)
11. $\quad\quad$ **if** $\mathcal{F}_{\mathrm{R}} =$ *empty* **then break** (continue at line 4)
12. $\quad\quad D \leftarrow$ RegularityFeatures (S, $\mathcal{F}_{\mathrm{R}}$)
13. $\quad\quad$ **if** $D =$ *empty* **then break** (continue at line 4)
14. $\quad\quad R \leftarrow (U, D_0, \ldots, D_d) \leftarrow$ UpdateSolid (S, D)
15. $\quad$ **for** $N \in R$ **do**
16. $\quad\quad$ Add N as child of S to T
17. $\quad\quad$ **push** (Q, N)
18. **return** (T)

Fig. 7. RFT construction algorithm

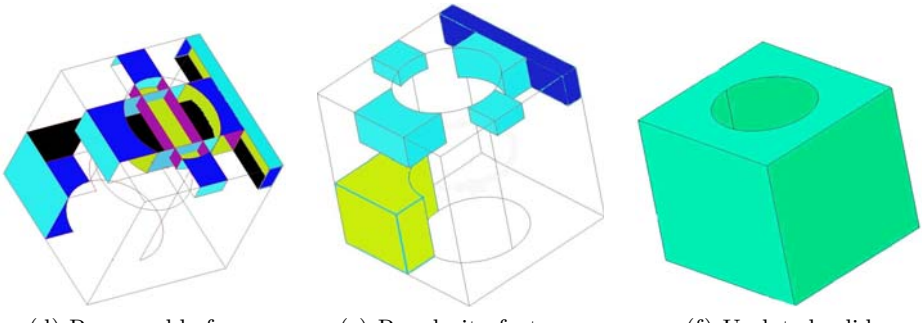
Next, MES and MVS recoverable edges of S are detected. Existing edges, together with newly generated recoverable edges, create new edge loops from which subsequent recoverable faces are constructed. The orientations of the faces constructed using these loops are determined from orientations of the edges. After extending the solid S using these newly constructed recoverable faces, the regularity features are identified as closed cells, which are used to recover broken symmetries of S and form the basis of the construction of regularity features of S. The negativity or positivity of each regularity feature comes naturally from the orientations of the faces bounding them.

Once the regularity features of S have been identified, we update S to create an updated model U by first adding all negative regularity features to S and then removing all positive regularity features from S except for those adjacent to negative regularity features (see Sect. 1). The added negative and removed positive regularity features (*not all* positive ones), together with U, become the children of S in the RFT, and are recursively processed. Recursion stops when all solids in the RFT have been processed.

Pseudo-code for the algorithm is given in Fig. 7. The tree T contains the RFT as it is built up; its root is set to M. Recursion is implemented via a queue Q storing solids still to be processed; it initially contains M (Lines 1–2). In order to determine recoverable edges lying in the same surface geometry, we first group faces of M sharing the same underlying surface to within tolerance (Line 3). For this we cluster the faces according to a similarity measure based on the symmetric Hausdorff distance. Using the face clusters we can also determine whether other geometric entities in the model are the same. E.g. we check whether two edges



(a) Top view (b) Bottom view (c) Recoverable edges



(d) Recoverable faces (e) Regularity features (f) Updated solid

Fig. 8. Example model

share the same underlying curve by checking whether they are the intersection of two face pairs which come from the same two underlying surfaces.

Repeatedly, until the queue is empty, the first solid is removed from the queue and its regularity features are constructed (Lines 4–17). First, the solid S is analysed to determine whether its edge graph consists of a single connected component (Line 6). If not, it is decomposed into edge-connected solids, which are added as children of S to the tree and queued for further processing (Lines 15–17). For efficiency, we tag edge-connected solids detected to avoid re-checking them when they are processed recursively.

If S consists only of one edge-connected solid, we determine its regularity features (Lines 7–14). First all the MES and MVS recoverable edges in S are constructed using the ideas in Sect. 3 (Lines 8–9). From these we then detect the recoverable faces as described in Sect. 5.2 (Line 10). If no recoverable faces are detected, we stop processing S and proceed to the next element of the queue (Line 11). Otherwise regularity features are found by detecting cells bounded by recoverable faces and the original faces of S (Line 12). If no regularity features are detected, we stop processing S and proceed to the next element of the queue (Line 13). Otherwise, an updated solid U for S is computed based on the detected regularity features, together with the added negative and removed positive regularity features (Line 14). They are added as children to S and scheduled for further processing (Line 15–17).

At each step we simplify the solid being processed by filling in or cutting off material as determined by the recoverable faces. Each new solid is simpler, having less recoverable edges. As the overall number of recoverable edges in a model is finite, recursion eventually stops.

As an example, consider the model M shown from above and below in Figs. 8(a) and (b). It becomes the root of the output RFT T. M is edge-connected, so we construct its MES (red) and MVS (black) recoverable edges as shown in Fig. 8(c). The recoverable faces and the regularity features are then generated in turn as shown in Figs. 8(d) and (e). As all the generated regularity features are negative, they are added to M, producing the updated model shown in Fig. 8(f). Both the constructed regularity features and the updated model become the children of M in T. Except for the updated solid in Fig. 8(f), no further regularity features are obtained when the initial regularity features are reconsidered. The updated model is further processed to give two edge-connected solids: a cube and a cylinder, which become the children of the updated model. No further regularity features are found on considering them.

5 Algorithm Details

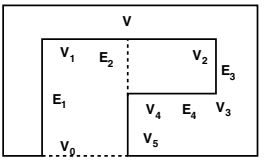
This section discusses further important details of our algorithm. As Lines 8–14 only process edge-connected solids, we assume in the following that we have an edge-connected solid S with edge set $\mathcal{E}$ and face set $\mathcal{F}$.

5.1 Recoverable Edge Construction

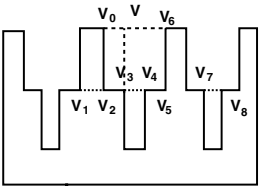
The recoverable edge set $\mathcal{E}_{\mathrm{R}} = \mathcal{E}_{\mathrm{E}} \cup \mathcal{E}_{\mathrm{V}}$ consisting of MES and MVS recoverable edges is constructed in sequence using the ideas in Sect. 3 (Lines 8–9). This sequence is required to resolve conflicts between MES and MVS cases (see below). For constructing MVS recoverable edges, two issues must be considered carefully.

Firstly, if several concave vertices are consecutively linked to each other by edges in $\mathcal{E}$, we have to decide which of these vertices to process. We choose the shortest edge having one or two concave vertices as end-points. If this edge has two concave vertices as end-points, we choose the vertex for which the adjacent edge is shorter. Otherwise, we simply choose the concave vertex. This way we expect to find the 'smallest' broken symmetry first. We process this vertex as a standard MVS vertex and leave the other vertices for later processing in the resulting regularity features. For instance, vertices V_1, V_2 and V_3 in Fig. 9(a) are all concave vertices. Only V_3 is selected for construction of MVS recoverable edges.

Secondly, if a concave vertex or the boundary vertex of an MVS case also serves as an end-point of an MES recoverable edge, we have to decide which to select such that similar sub-parts create similar decomposition results. This is particularly important for models where all geometric relations between model entities are only approximately satisfied. In Fig. 9(b), for example, if we construct edge V_7V_8 as an MES recoverable edge for the right-hand feature, while edges



(a) Choice between multiple concave vertices for MVS



(b) Ambiguity between MES and MVS recoverable edges

Fig. 9. Ambiguous recoverable edge cases

V_0V, V_3V are found as MVS recoverable edges for the central feature, the bottom of the right slot will be filled in, while for the central feature V, V_0, V_2, V_3 is filled. Such inconsistency in the construction will hamper application of our ideas to design intent detection. To avoid this problem, we always give preference to MES recoverable edges, since they are constructed from clear hints for broken symmetries, and usually produce fewer regularity features. Thus, if a concave vertex or the boundary vertices of an MVS case serve as an end-point of an MES recoverable edge, we do not construct MVS recoverable vertices and edges. For the model in Fig. 9(b), recoverable edges V_3V_4 and V_7V_8 are constructed as MES recoverable edges while VV_0 and VV_3 are not taken into consideration.

Finally, note that an MVS recoverable vertex might be the same as another MVS recoverable vertex, or an original vertex, within tolerance. We combine such vertices using the face grouping information gathered in Line 3 of our algorithm.

5.2 Recoverable Face Construction

After the recoverable edge set $\mathcal{E}_\mathrm{R}$ of the solid S has been obtained as above, the recoverable face set $\mathcal{F}_\mathrm{R}$ is constructed from the union of the edges in $\mathcal{E}$ and $\mathcal{E}_\mathrm{R}$ based on taking minimal turning angles between these edges (Line 10).

Using an edge ordering algorithm, for each recoverable edge E in $\mathcal{E}_\mathrm{R}$, we find all the minimal edge loops lying on the same surface and containing E in the edge graph given by the union of $\mathcal{E}$ and $\mathcal{E}_\mathrm{R}$. Each such non-self-intersecting loop bounds a new recoverable face. Such loops must be detected on *both* surfaces on which E lies, proceeding in *both* clockwise and counter-clockwise directions. If a loop does not contain any edge in $\mathcal{E}$, it makes no contribution to recovering symmetries of existing faces and hence is ignored.

5.3 Regularity Feature Construction

Having obtained the recoverable face set $\mathcal{F}_\mathrm{R}$, we generate all possible regularity features of the model S (Line 12). These are the cells determined by original faces $\mathcal{F}$ together with the recoverable faces $\mathcal{F}_\mathrm{R}$. As the input solid S already has proper loops and face normals, generating these cells is much simpler than the general problem of converting wire-frame models to solid models [1,5,11,13].

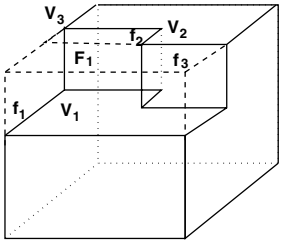


Fig. 10. Two recoverable faces to be considered in sequence

To ensure that the resulting solids are manifold and closed, the Möbius rule is applied: each edge belongs to *two* faces and the orientation of the edge is *opposite* on each face [9]. The negativity or positivity of each resulting regularity feature is determined from the face orientations.

Note that each original edge can only lie on the intersection of two faces as S is manifold. Moreover, our algorithm guarantees that at most two recoverable faces are generated around a given edge on one surface. Hence, the number of faces around one edge, both in $\mathcal{F}_{\mathrm{R}}$ and $\mathcal{F}$, is at most six, two of which are original faces while others are recoverable faces. All faces having the same underlying surface also have the same or opposite normals. If the normals of two faces point in the same direction, we cannot uniquely order them around their common edge. To resolve this issue, we note that: (i) if the two faces are lying on different sides of an edge on the surface, the unique one contributing to the Möbius rule is considered; (ii) if the two faces lie on the same side of an edge on the surface, each of them is selected in sequence to generate all possible volumes. E.g. in Fig 10 we start with recoverable face f_1 at the left which is expanded to F_1 via edge V_1V_3. If we expand F_1 via V_2V_3, both recoverable faces f_2 and f_3 give the same turning angle. Considering f_2 first, no volume is detected. Then considering f_3, a volume cut off from the cube is produced.

5.4 Model Update

Having found all the regularity features, negative and positive, we compute an updated, more symmetric model U incorporating geometric information from the original solid S and the regularity features (Line 14). Firstly, if the intersection of the interior of two regularity features is not empty, the one with smaller volume is kept while the other one is discarded. Secondly, volumes determined by negative regularity features are added to S, while volumes corresponding to the positive solids are cut from S, except for those adjacent to negative features. Both rules are aimed at constructing simple updated models and avoid the inclusion of unnecessary regularity features in the RFT.

We create the updated model using *face combination operations* (rather than potentially non-robust Boolean operations) by analysing the geometric and topological relations between the regularity features and their parent. This is feasible as the faces of a regularity feature come from its parent's faces or underlying surfaces. This approach fills in a set of negative regularity features or splits off

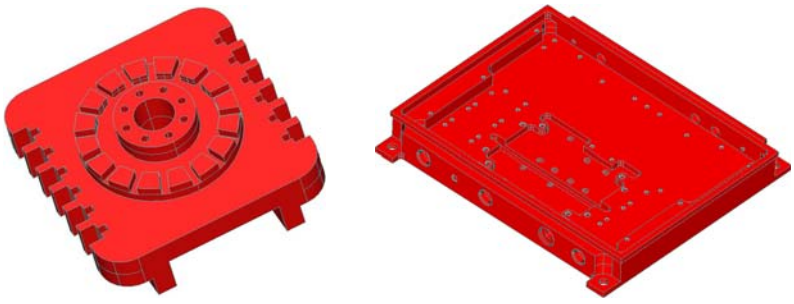


Fig. 11. Two example models

a set of regularity features by changing the topology of the model; both can be done in the same way. We simply combine the regularity features and the parent solid along their shared faces; similarly, adjacent faces on the same underlying surface are further combined along their common edges. Finally, all connected edges lying on the same curve are combined into a single edge.

6 Examples

Our algorithm has been implemented under Linux using OpenCASCADE and experiments were run on a 3.4GHz Pentium 4E with 1GB RAM. This section shows the RFTs constructed for two complicated example models shown in Fig. 11, and considers the algorithm's performance.

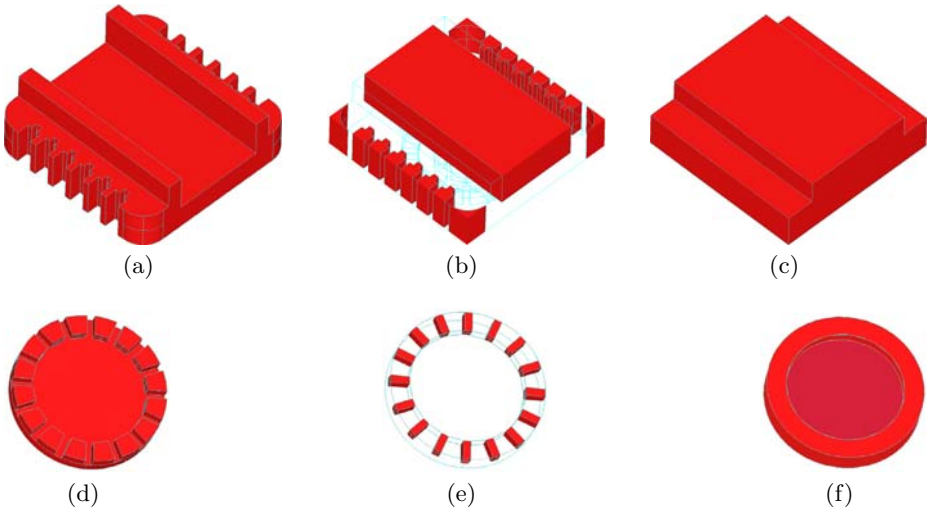


Fig. 12. Decomposition of the model in Fig. 11(a)

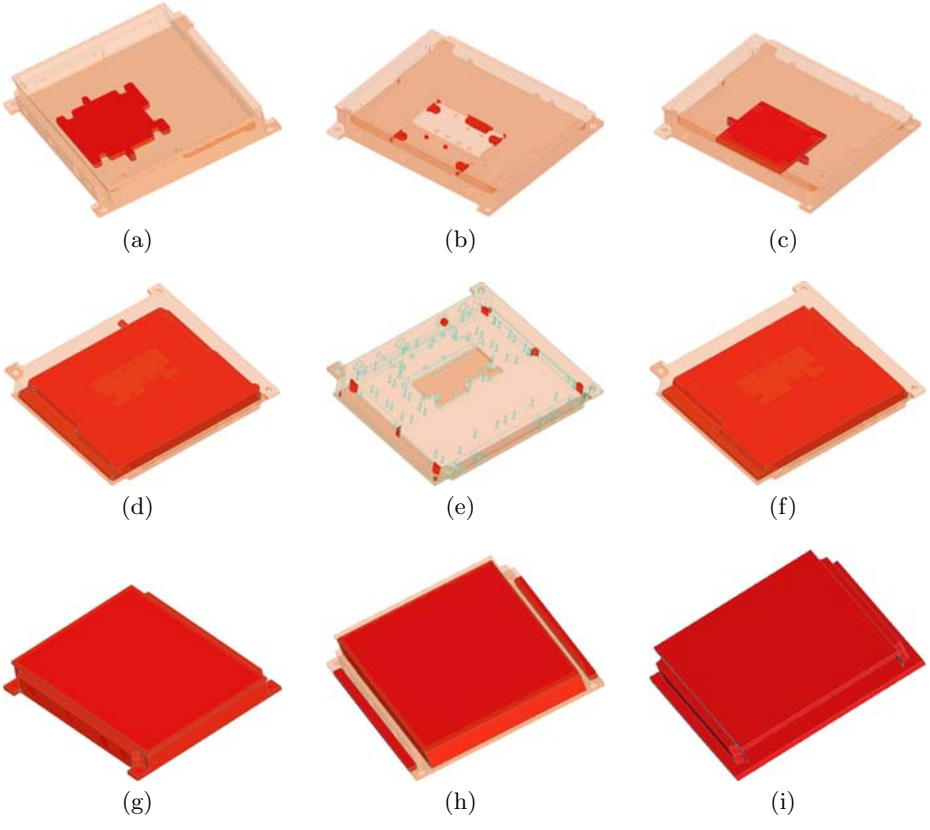
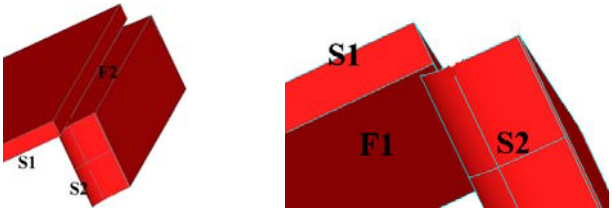


Fig. 13. Decomposition of the model in Fig. 11(b)

The decomposition for the model in Fig. 11(a) with 281 faces is shown in Fig. 12. On the first decomposition level 10 edge-connected solids are produced, two of which are (a) and (d). The other edge-connected solids are simple cylinders which do not decompose further. Sub-part (a) leads to 29 negative regularity features at the next level in the tree as shown in (b), and results in the updated model (c), which is further decomposed into two rectangular blocks. Note that (b) properly identifies the congruent negative regularity features in a regular translational arrangement. Similarly, (d) leads to 16 negative solids as shown in (e). The resulting updated model is shown in (f), which is further decomposed into a positive and a negative cylinder. In (e) the symmetric arrangement of the negative regularity features has been extracted explicitly. It takes 27.50 seconds to produce the final RFT of depth 4 consisting of 59 regularity features. Overall the decomposition reveals the rotational symmetries of the main cylinder as well as the regular translational arrangement of congruent regularity features at the sides. The regularity features at the leaves of the tree are simple and regular, which would allow easy subsequent regularity analysis.



(a) (b)

Fig. 14. Close-up of cause for incomplete decomposition of Fig. 13(i)

Fig. 11(b) shows another complicated example with 355 faces; its decomposition is shown in Fig. 13. On the first decomposition level 93 edge-connected solids are produced, three of which are (a), (d) and (g). The other edge-connected solids are simple, non-decomposable, cylinders. Sub-part (a) is further decomposed into 11 regularity features at the next level in the tree as shown in (b), resulting in the updated model (c). Sub-part (d) leads to 11 regularity features at the next level, as shown in (e); the updated model is shown in (f). The resulting regularity features are further decomposed but are straight-forward and not shown due to space limitations. It takes 18.96 seconds to produce the final RFT of depth 5 consisting of 155 regularity features. However, the updated model in Fig. 13(i) was only decomposed into a bottom block and a second part. We expected the latter to be decomposed further into three blocks. Fig. 14 shows a closeup of the regularity feature revealing the cause of the incomplete decomposition: blocks S_1, S_2 touch each other at faces F_1, F_2, but their interiors do not intersect. Both faces would have to be extended to recover the original blocks. However, in such contact cases, our current algorithm cannot construct recoverable faces to separate the two blocks due to the lack of suitable recoverable edges. In order to address this problem, original edges which form loops around 'holes' would have to be determined efficiently so they can be filled in by the geometry of neighbouring faces. Several further examples of realistic industrial parts are provided at `http://www.langbein.org/research/DID/rftexamples`. In each case the computation took less than 30 seconds for models with a maximum of 355 faces and 164 regularity features, demonstrating the algorithm's practical utility. In these tests, any rotationally symmetric arrangements and regular translational arrangements are clearly exposed by the decomposition; often the regularity features are congruent or similar to each other. Furthermore, the regularity features at deep levels in the tree show a high level of regularity as they are often the basic rectangular blocks, cylinders, etc. present in the model.

7 Conclusions

We have presented an algorithm for constructing *regularity feature trees* for B-rep solid models, for detecting geometric design intent. Our experiments indicate that it produces suitable RFTs for detecting regularities of a model's sub-parts and intended geometric relations between the sub-parts. The algorithm takes a

relatively short time, suitable for practical applications. We note that the ideas presented are applicable to more general curved models, although the decomposition method may need further refinement and generalisation. In future work we intend to detect potential geometric regularities more efficiently using the RFT and ultimately select a suitable subset of these regularities to describe the geometric design intent of approximate models.

Acknowledgements

This project was supported by EPSRC grant GR/S69085/01. The example models in Sect. 6 were obtained from the National Design Repository at Drexel University, `http://www.designrepository.com/`.

References

1. S. Bagali, J. Waggenspack. A shortest path approach to wireframe to solid model conversion. In: *Proc. 3rd ACM Symp. Solid Modeling and Appl.*, pp. 339–349, 1995.
2. X. Dong, M. Wozny. A method for generating volumetric features from surface features. In: *Proc. 1st ACM Symp. Solid Modeling and Appl.*, pp. 185–194, 1991.
3. C. H. Gao, F. C. Langbein, A. D. Marshall, R. R. Martin. Local topological beautification of reverse engineered models. *Computer-Aided Desisn*, 36(13):1337–1355, 2004.
4. J. Han, M. Pratt, W. Regli. Manufacturing feature recognition from solid models: a status report. *IEEE Trans. Robotics and Automation*, 6(6):782–796, 2000.
5. K. Inoue, K. Shimada, K. Chilaka. Solid model reconstruction of wireframe CAD models based on topological embeddings of planar graphs. *J. Mechanical Design*, 125(3):434–442, 2003.
6. Y. Kim. *Convex decomposition and solid geometric modeling*. PhD thesis, Starnford University, USA, 1990.
7. F. C. Langbein, B. I. Mills, A. D. Marshall, R. R. Martin. Approximate geometric regularities. *Int. J. Shape Modeling*, 7(2):129–162, 2001.
8. F. C. Langbein. *Beautification of reverse engineered geometric models*. PhD thesis, Cardiff University, UK, 2003.
9. R. Lequette. Automatic construction of curvilinear solids from wireframe views. *Computer-Aided Design*, 20(4):171–179, 1988.
10. M. Leyton. *A generative theory of shape*. Lecture Notes in Computer Science 2145, Springer, Berlin, 2001.
11. S. Liu, S. Hu, Y. Chen, J. Sun. Reconstruction of curved solids from engineering drawings. *Computer-Aided Design*, 33(14):1059–1072, 2001.
12. E. Lockwood, R. Macmillan. Geometric symmetry. *Mathematical Intelligence*, 6(3):63–67, 1984.
13. G. Markowsky, M. Wesley. Fleshing out wire frames. *IBM J. Research and Development*, 24(5):582–597, 1980.
14. B. Mills, F. Langbein, A. Marshall, R. Martin. Approximate symmetry detection for reverse engineering. In: *Proc. 6th ACM Symp. Solid Modeling and Appl.*, pp. 241–248, 2001.

15. B. Mills, F. Langbein, A. Marshall, R. Martin. Estimate of frequencies of geometric regularities for use in reverse engineering of simple mechanical components. Tech. report GVG 2001-1, Dept. Computer Science, Cardiff University, 2001.
16. A. Rappoport. The extended convex differences tree (ECDT) representation for n-dimensional polyhedra. *Intl. J. Comp. Geometry and Appl.*, 1(3):227-241, 1991.
17. W. Regli. *Geometric algorithms for recognition of features from solid models*. PhD thesis, University of Maryland, USA, 1995.
18. H. Sakurai, P. Dave. Volume decomposition and feature recognition, Part II: curved objects. *Computer-Aided Design*, 28(6–7):519–537, 1996.
19. D. Sandiford, S. Hinduja. Construction of feature volumes using intersection of adjacent surfaces. *Computer-Aided Design*, 33(6):455–473, 2001.
20. V. Sashikumar, S. Milind. Reconstruction of feature volumes and feature suppression. In: *Proc. 7th ACM Symp. Solid Modeling and Appl.*, pp. 60–71, 2002.
21. V. Sashikumar, S. Milind, R. Rahul. Removal of blends from boundary representation models. In: *Proc. 7th ACM Symp. Solid Modeling and Appl.*, pp. 83–94, 2002.
22. V. Shapiro, D. Vossler. Separation for boundary to CSG conversion. *ACM Trans. Graphics*, 12(1):35–55, 1993.
23. M. Sitharam, J.-J. Oung, Y. Zhou, A. Arbree. Geometric constraints within feature hierarchies. *Computer-Aided Design*, 38(1):22-38, 2006.
24. J. Vandenbrande. *Automatic recognition of Machinable Features in Solid Models*. PhD thesis, University of Rochester, USA, 1990.
25. T. Varady, R. Martin, J. Cox. Reverse engineering of geometric models - an introduction. *Computer-Aided Design*, 29(4):255–268, 1997.
26. D. Waco, Y. Kim. Geometric reasoning for machining features using convex decomposition. *Computer-Aided Design*, 26(6), 477-489, 1994.
27. Y. Woo, H. Sakurai. Recognition of maximal features by volume decomposition. *Computer-Aided Design*, 34(3):195–207, 2002.
28. Y. Woo. Fast cell-based decomposition and applications to solid modeling. *Computer-Aided Design*, 35(11):969–977, 2003.
29. X. Xu, S. Hinduja. Recognition of rough machining features in $2\frac{1}{2}$ components. *Computer-Aided Design*, 30(7):503–516, 1998.
30. H. Zhu, C. Menq. B-rep model simplification by automatic fillet/round suppressing for efficient automatic feature recognition. *Computer-Aided Design*, 34(2):109–123, 2002.

Insight for Practical Subdivision Modeling with Discrete Gauss-Bonnet Theorem

Ergun Akleman[1] and Jianer Chen[2]

[1] Texas A&M University, Visualization Sciences Program, Department of Architecture, C418 Langford Center, College Station, Texas 77843-3137 USA
ergun@viz.tamu.edu
www-viz.tamu.edu/faculty/ergun
[2] Texas A&M University, Computer Science Department, College Station Texas 77843-3112, USA
chen@cs.tamu.edu
http://faculty.cs.tamu.edu/chen/

Abstract. In this paper, we introduce an insight for practical subdivision modeling to improve the quality of control mesh structures. Our approach is based on a discrete version of Gaussian-Bonnet theorem on piecewise planar manifold meshes and vertex angle deflections that determines local geometric behavior. Based on discrete Gaussian-Bonnet theorem, summation of angle deflections of all vertices is independent of mesh structure and it depends on only the topology of the mesh surface. Based on this result, it can be possible to improve organization of mesh structure of a shape according to its intended geometric structure.

1 Motivation

Most professional modelers in animation and special effect industry slowly migrated subdivision surfaces. One of the crucial questions for practical modeling with subdivision surfaces is to identify the numbers and types of extraordinary vertices that needs to be used for a successful modeling. In the first observation, this question seems easy to answer based on genus of the goal surface. Using Euler's formula, we can compute the number of extraordinary vertices and their valences (also the number of faces and their valences) for the given genus.

However, extraordinary vertices do not result only from genus. In most practical cases, we need to consider the geometrical structure of the surface. Tree structures are especially the ones that cause problems. Tree structures are more common than real trees. For instance, even head of a human being can be considered genus-1 tree that has at least 3 branches that include nose and 2 eyes. The one hole come from digestive track that start from mouth. It is possible to model the head as a genus-0 tree with at least four branches by include mouth as a branch.

By relating topology and geometry, we provide an insight to answer such problems coming from practical modeling. Our results justify some of the intuitively developed practices in current modeling approaches. For instance, the modelers

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 287–298, 2006.
© Springer-Verlag Berlin Heidelberg 2006

introduce extraordinary vertices to model eyes, nose and mouth in facial modeling. Our results go further than and even defies common sense about subdivision modeling. We show that despite the common belief the quality of some surfaces can be improved by introducing extraordinary vertices.

2 Introduction

Our approach is based on the Gauss-Bonnet Theorem that says that the integral of the Gaussian curvature over a closed smooth surface is equal to $2\pi\times$ the Euler characteristic of the surface which is $2-2g$ [1]. The theorem requires a smooth surface. In this section, we provide a simple proof for Gauss-Bonnet Theorem on Piecewise Linear Meshes. There exist a variety of approaches for discrete versions of Gauss-Bonnet Theorem (see [2,3] for some examples).

2.1 Piecewise Linear Meshes

We say that a mesh M is *piecewise linear* if every edge of M is a straight line segment, and every face of M is planar. Piecewise linear meshes are useful since the surface of each face is well-defined [4]. Most common piecewise linear meshes are triangular meshes. However, piecewise linear meshes are a significant generalization of triangular meshes, which not only allow non-triangular faces, but also allow faces to have different face valence. In the discussion of the current paper, we will be focused on piecewise linear meshes.

We start by some intuitions. Let M be a piecewise linear mesh such that all vertices of M have the same vertex valence m and all faces of M have the same face valence n (such a mesh is called a *regular mesh*). Suppose that M has v vertices, e edges, and f faces. By Euler Equation [5,6],

$$v-e+f=2-2g \tag{1}$$

where g is the genus of the mesh M and from the equalities $mv=nf=2e$, we derive

$$e=\frac{nm(2-2g)}{2n+2m-nm}, \qquad v=\frac{2n(2-2g)}{2n+2m-nm}, \qquad f=\frac{2m(2-2g)}{2n+2m-nm}$$

We would like to generalize the above relations to piecewise linear meshes that are not regular. For this, we introduce the following definitions.

Definition 1. *Let M be a piecewise linear mesh, with vertex set $\{\mu_1,\ldots,\mu_v\}$ and face set $\{\phi_1,\ldots,\phi_f\}$. Moreover, for each i, $1\le i\le v$, let m_i be the valence of the vertex μ_i, and for each j, $1\le j\le f$, let n_j be the valence of the face ϕ_j. Then the average vertex valence m and the average face valence n are defined, respectively, to be*

$$m=(\sum_{i=1}^{v} m_i)/v \qquad \textit{and} \qquad n=(\sum_{j=1}^{f} n_j)/f$$

Note that m and n are not necessarily integers.

Based on these new concepts, the number v of vertices, the number e of edges, and the number f of faces in a piecewise linear mesh can be given as functions in terms of the average vertex valence, the average face valence, and the genus of the mesh, as shown in the following lemma.

Lemma 1. *Let M be a piecewise linear mesh. Suppose that M has v vertices, e edges, f faces, and genus g. Let m and n be the average vertex valence and average face valence, respectively, of the mesh M, then*

$$e = \frac{nm(2-2g)}{2n+2m-nm}, \qquad v = \frac{2n(2-2g)}{2n+2m-nm}, \qquad f = \frac{2m(2-2g)}{2n+2m-nm}$$

Proof. Suppose that the vertex set of M is $\{\mu_1, \ldots, \mu_v\}$ and that the face set of M is $\{\phi_1, \ldots, \phi_f\}$. Moreover, suppose that for each i, $1 \le i \le v$, the vertex μ_i has valence m_i, and that for each j, $1 \le j \le f$, the face ϕ_j has valence n_j. Since each edge in the mesh M is used exactly twice in vertex valences and is used exactly twice in face boundaries, we have

$$\sum_{i=1}^{v} m_i = \sum_{j=1}^{f} n_j = 2e$$

On the other hand, by our definitions of the average vertex valence m and the average face valence n, we have

$$mv = \sum_{i=1}^{v} m_i \qquad \text{and} \qquad nf = \sum_{j=1}^{f} n_j$$

Therefore, we still have $mv = nf = 2e$. Since the mesh M has genus g, Equation (1) holds. Combining Equation (1) and the equalities $mv = nf = 2e$, we get immediately

$$e = \frac{nm(2-2g)}{2n+2m-nm}, \qquad v = \frac{2n(2-2g)}{2n+2m-nm}, \qquad f = \frac{2m(2-2g)}{2n+2m-nm}$$

This proves the lemma.

2.2 Angle Deflection at a Vertex

To understand the local behavior around a vertex of a piecewise linear meshes, angle deflection are great tools. Angle deflection at a vertex are formally defined based on the corner angles around a vertex, as follows.

Definition 2. *Let μ_i be a vertex on a piecewise linear mesh M. The angle deflection at vertex μ_i is defined as*

$$\bar{\theta}(\mu_i) = 2\pi - \sum_{j=1}^{m_i} \theta_j$$

where θ_j is the internal angle of the j-th corner of vertex μ_i and m_i is the valence of the vertex μ_i.

The importance of angle deflections to describe local behavior can be best understood by viewing developable sculptures [7] of Sculptor Ilhan Koman. He created a set of sculptures by forcing the total angle in a given point larger than 2π [7]. As shown in Figure 1, when the angle goes beyond 2π, it is possible to create a wide variety of saddle shapes. We observe that Koman's developable structures can give very useful information about the local behavior at a vertex for piecewise linear meshes.

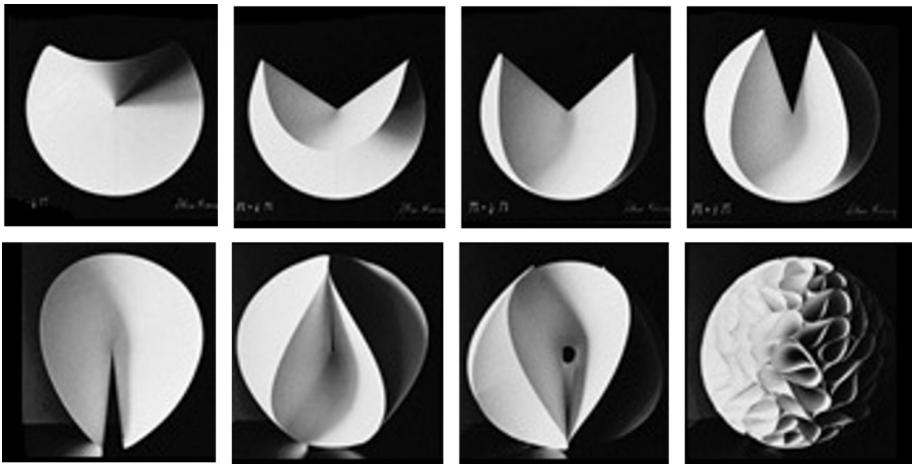


Fig. 1. Ilhan Koman's Developable Sculptures

It is interesting to point out that the value of $\bar{\theta}(\mu_i)$ is somewhat related to curvature and based on the values of corner angles around the vertex μ_i. More precisely,

- $\bar{\theta}(\mu_i) > 0$: then vertex μ_i is either convex or concave.
- $\bar{\theta}(\mu_i) = 0$: then vertex μ_i is planar.
- $\bar{\theta}(\mu_i) < 0$: then vertex μ_i is a saddle point.

The actual values for $\bar{\theta}(\mu_i) \geq 0$ are not particularly useful, they simply tell how sharp is the convex or concave area. The upper limit for positive values is 2π and it corresponds to the sharpest convex or concave regions. For negative values there is no lower limit. In addition, the actual value of the number gives information about the type of saddle. Since there is no lower limit, saddle points can be very wild, as it can be seen from Ilhan Koman's sculptures in Figure 1.

This is not unexpected since the most popular discrete version of Gaussian curvature introduced by Calladine for triangular meshes uses angular deflection [8]. Calladine's discrete Gaussian curvature given as

$$\frac{\text{Angular Deflection}}{\text{Area Associated with Vertex}}$$

2.3 Sum of Angle Deflections

Here, we will give a simple proof of discrete version of Gauss-Bonnet theorem as *Sum of angle deflections* (SAD), given as follows.

Definition 3. *Let M be a piecewise linear mesh with a vertex set $\{\mu_1, \ldots, \mu_v\}$. The sum of angle deflections (SAD) of the mesh M is defined as*

$$\bar{\theta}(M) = \sum_{i=1}^{v} \bar{\theta}(\mu_i)$$

Note that the SAD of a piecewise linear mesh is closely related to the geometric shape of the mesh M. On the other hand, as we will prove in the following theorem, the SAD of a mesh is a topological invariant.

Theorem 1. *Let M be a piecewise linear mesh of genus g. Then the SAD $\bar{\theta}(M)$ of the mesh M is equal to $2\pi(2-2g)$.*

Proof. *Let $\{\mu_1, \ldots, \mu_v\}$, $\{\epsilon_1, \ldots, \epsilon_e\}$, and $\{\phi_1, \ldots, \phi_f\}$ be the vertex set, edge set, and face set of the mesh M, respectively. Moreover, for each i, $1 \le i \le v$, let m_i be the valence of the vertex μ_i, and for each j, $1 \le j \le f$, let n_j be the valence of the face ϕ_j. Let m and n be the average vertex valence and the average face valence, respectively, of the mesh M.*

Consider each face ϕ_j of valence n_j. Since ϕ_j is a planar polygon, the sum $\sigma(\phi_j)$ of inner angles of ϕ_j is equal to $(n_j - 2)\pi$. Adding this up over all faces in the mesh M, we obtain

$$\sum_{j=1}^{f} \sigma(\phi_j) = \sum_{j=1}^{f} (n_j - 2)\pi = \left(\sum_{j=1}^{f} n_j - 2f \right) \pi = nf\pi - 2f\pi \tag{2}$$

where we have used the definition of the average face valence $n = (\sum_{j=1}^{f} n_j)/f$.

On the other hand, by the definition of angle deflections, the sum of corner angles around each vertex μ_i of the mesh M is equal to $2\pi - \bar{\theta}(\mu_i)$. Adding this up over all vertices in the mesh M, we get

$$\sum_{i=1}^{v} (2\pi - \bar{\theta}(\mu_i)) = 2v\pi - \bar{\theta}(M)$$

This value should be equal to the value in (2) since both of them are the sum of the total corner angles in the mesh M. Therefore, we get

$$\bar{\theta}(M) = 2v\pi - nf\pi + 2f\pi = 2\pi(v - nf/2 + f)$$

By the definition of the average face valence $n = (\sum_{j=1}^{f} n_j)/f$, and noticing that $2e = \sum_{j=1}^{f} n_j$, we get immediately $nf/2 = e$. This gives

$$\bar{\theta}(M) = 2\pi(v - e + f)$$

By Euler Equation (1), we get

$$\bar{\theta}(M) = 2\pi(2 - 2g)$$

This proves the theorem.

Theorem 1 is independent of the number of vertices, the number of edges, the number of faces, and the average vertex and face valences. It depends only on the genus of the piecewise linear mesh M. As a result, any homeomorphic operation (operations that do not change topology, in this case genus) that converts a piecewise linear mesh to a new piecewise linear mesh does not change the SAD of the mesh. In other words, the effect of a homeomorphic operation on the SAD of a mesh will be zero, which means if we gain angle deflections in some vertices, we will lose some angle deflections in some other vertices. This is the important result for practical polygonal mesh modeling.

2.4 Visual Intuition

To give a visual intuition about sum of angle deflections, we can relate it to critical points in Morse theory [9]. Consider the 3-D objects in Figure 2. The first two objects have genus $g = 0$, the next two objects have genus $g = 1$, while the last object has genus $g = 2$. If we assign $+1$ to each minima/maxima (convex/concave) type critical point and -1 to each saddle type of critical point (as marked in the figure), the total adds up to $(2 - 2g)$, as shown in Figure 2.

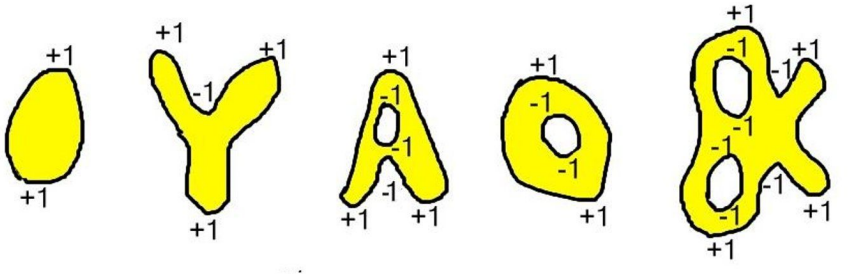


Fig. 2. The total adds up to $(2 - 2g)$ if we assign minima/maxima points $+1$ and saddles -1

Figure 2 also give a visual explanation of the sum of angle deflections. In figure, we assume that all angle deflections occur only at critical points. If that were really the case, the angle deflection in each critical point would have been $+2\pi$ for minima/maxima and -2π for each saddle point. So, the total will be equal to $2\pi(2 - 2g)$, which is the sum of angle deflections.

It is also clear from Figure 2 why branches do not change the total angle deflections. Each branch induces the same numbers of saddle points and maxima/minima points; so total effect becomes zero. On the other hand, each handle (i.e., each hole) introduces two saddle type critical points, which decreases the total angle deflections by 4π.

3 Homogenous Operations

For branch creations, professional modelers usually use extrusion [10] and wrinkle operations (see Section 4). These operations are homogenous, i.e. they do not change mesh topology. Subdivision schemes, which are commonly used to create smooth surfaces, are also homogenous operations.

To understand why homogenous operations do not change the total angle deflections, it is insightful to look at them from a mesh topological point of view.

Let us assume that the number of faces, edges and vertices of a mesh increase in each application of homogenous operations. Based on this assumption, let us consider the practice in which when homogenous operations are applied to a mesh while the genus of the mesh stays the same, the numbers of faces, edges and vertices of the mesh increase without a limit.

Remark 1. This is not an unrealistic assumption. In fact, all subdivision schemes, extrusions and wrinkle operations increase the number of faces, edges and vertices.

To analyze repeated applications of homogenous operations we can again use the Euler equation. Let us consider specifically meshes of genus g. Let n and m be the average face valence and the average vertex valence of a mesh of genus 1. Using Euler equation (1) and equations in Lemma 1, we have

$$\frac{2-2g}{e} = \frac{2}{m} + \frac{2}{n} - 1$$

By our assumption e can be arbitrarily large but the mesh genus stays the same, thus

$$\frac{2}{m} + \frac{2}{n} - 1 \approx 0$$

which gives

$$m \approx \frac{2n}{n-2}$$

Under the assumption that $m \geq 3$ and $n \geq 3$, and replacing the above approximation by an equality, we will get

$$3 \leq n \leq 6$$

This result says that the average face valence cannot exceed 6 regardless of what we do (if we do not allow valent-2 vertices and polygons with two sides).

In other words, if our homogenous operations create only quadrilateral faces (i.e., $n = 4$), then the average vertex valence m goes to 4. If the homogenous operations create only triangles (i.e., $n = 3$), then the average vertex valence m goes to 6. For pentagons (i.e., $n = 5$), the average vertex valence approaches to $10/3$ and for hexagons (i.e., $n = 6$) it goes to 3. Any mixed use of different face types will create a rational number for the average vertex valence.

This also explain why most subdivision schemes provide $(3, 6)$ [11,12], $(4, 4)$ [13,14,15,16] and $(6, 3)$ [17,18,19] regular regions in which integer tuple (n, m)

satisfy the equation $\frac{2}{m} + \frac{2}{n} - 1 = 0$. Note that in these subdivisions the number of regular vertices and faces (i.e vertices with valence m and faces with valence n) increases, while the number of extraordinary vertices and faces stays the same in each iteration. As a result, the regular regions dominate the mesh.

Remark 2. To guarantee that subdivided mesh is at least G^1 continuous each face of subdivided mesh must approach a convex planar surface. So, we can say that most subdivision surfaces consist of piecewise linear approximations of curved faces. Piecewise linear approximations usually are $(3, 6)$, $(4, 4)$ or $(6, 3)$ regular regions for most subdivision schemes [20].

4 Implications on Practical Control Mesh Modeling with Homogenous Operations

Gauss-Bonnet is particularly insightful for modeling better control mesh structures for subdivision surfaces that include branches. We do not create branches only for modeling trees. Branches are important for modeling and they are everywhere. Even a simple genus-0 human face model must include branches such as eye sockets, nose and mouth.

Remark 3. With better control mesh structures, we mean faces are convex and as regular as possible. It is not always possible to make faces regular. For instance, we cannot get an angle deflection larger than π with regular convex faces. However, it is still possible to make faces as regular as possible by using appropriate valence. For instance, if we force the regular mesh structures such as $(4, 4)$, we cannot avoid irregular faces.

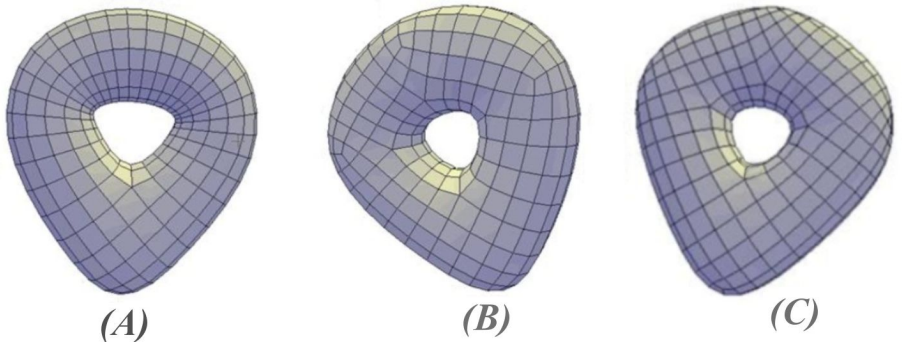


Fig. 3. This example shows that introducing non-regular structures can improve the mesh structure even for a simple genus-1 surface. In (A), top part is a regular (4,4) structure. See the thin quadrilaterals exists in regular $(4, 4)$ structure. As shown two non-regular structures in (B) and (C), it is possible to release the tension by increasing the valence of some vertices in saddle region and decreasing the valence of some vertices in convex region. As a result, quadrilaterals look more regular.

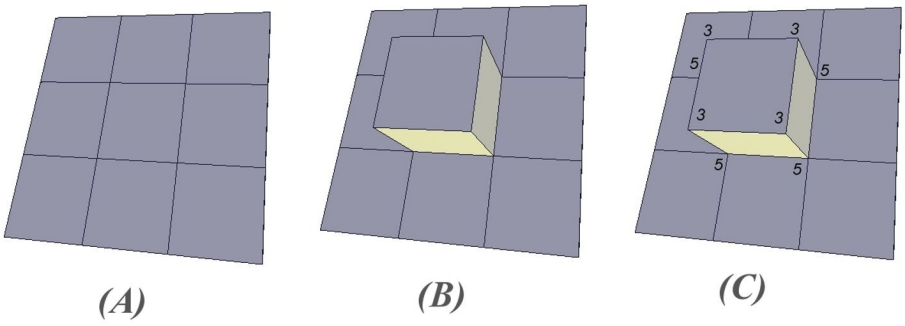


Fig. 4. Branch creation with extrusion operation. Before extrusion we have a quadrilateral with 4-valent vertices (See A). After extrusion 4 vertices have 5 valence and another 4 vertices have 3 valence. Average vertex valence added is 4 (See C).

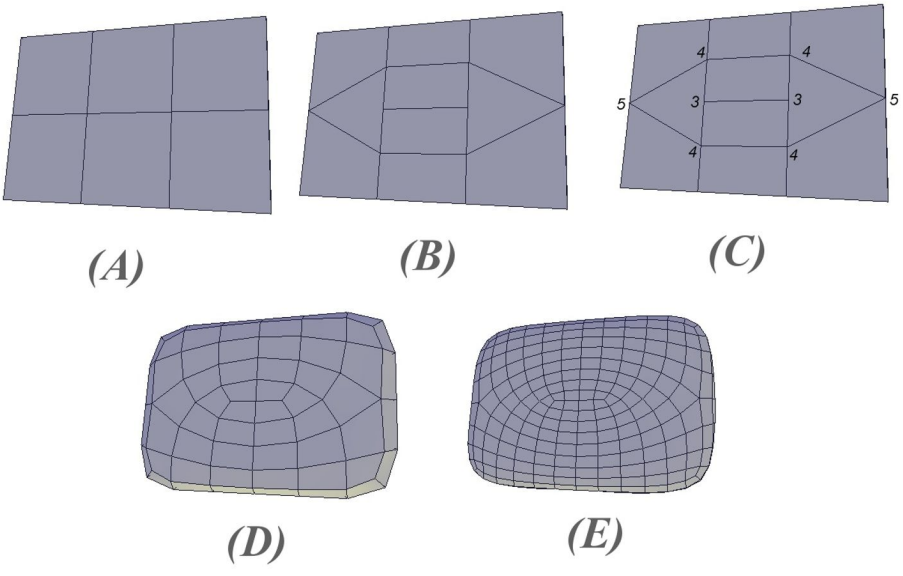


Fig. 5. Branch creation with wrinkle operation. (A) is the initial mesh. (B) is the wrinkle operation. As shown in (C), after wrinkle operation, 4 vertices have 4 valence, 2 vertices have valence 3 and another 2 vertices have 5 valence. So, added average vertex valence is 4. Catmull-Clark subdivided versions in (D) and (E) shows how an eye or wrinkle is automatically created with this operation.

By introducing saddle and minima/maxima type of critical points we can obtain better control mesh structure. In practice, that is what professional modelers do very efficiently. Based on our insight, it is easy to introduce saddles and minima/maxima. If we use the same type of faces, assuming that all the faces are almost regular, all we have to do is to carefully control the valences of vertices.

For instance, if we work only with triangles, we have to choose 3, 4 and 5 valent vertices in places we want to obtain minima or maxima. For saddle points, we have to choose valences of vertices larger than 6. Based on how complicated we want to design a saddle point, we can increase the valence.

Changing the valence can even improve the mesh structure of genus-1 surfaces. The Figure 3 shows an example of mesh improvement by decreasing and increasing valence in appropriate regions.

For professional modeling quadrilateral meshes are particularly important. In quadrilateral modeling, for minima and maxima, we have only one choice, 3 valence vertices. For saddles we can use any valence higher than 4. Extrusions and wrinkle operations as shown in Figures 4 and 5 introduce minima/maxima and saddle points simultaneously and therefore, they are commonly by professional modelers used to create branches.

5 Conclusions and Future Work

In this paper, we have introduced an approach to improve mesh structures to use in practical modeling with homogenous operations. We show that vertex angle deflection can give a powerful insight about local geometric behavior to professional modelers. Based on a discrete version of Gauss-Bonnet theorem, we have shown that it is be possible to improve the organization of mesh structures of shapes according to their geometric structure.

Based on the insights in this paper, it is possible to automatically create a better mesh from a given mesh. We think that vertex angle deflections can also be used to smooth the surfaces.

Despite the similarity, angle deflections are different than Gaussian curvature and its discrete versions. For instance, unlike sum of angle deflections, sum of discrete Gaussian Curvature may not be equal to $2\pi(2-2g)$. Both Gaussian curvature and angle deflections are rotation and translation invariant. However, Angle deflection is also scale invariant which can make it useful for shape retrieval.

References

1. Weisstein, E.W.: Gauss-Bonnet Formula. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Gauss-BonnetFormula.html (2005)
2. Etal, M.M.: Discrete differential-geometry operators for triangulated 2-manifolds. In: in H.-C. Hege and K. Polthier eds, Visualization and Mathematics III, Springer. (2003) 35–57
3. Watkins, T.: Gauss-Bonnet Theorem and its Generalization. (2005)
4. Williams, R.: The Geometrical Foundation of Natural Structures. Dover Publications, Inc. (1972)
5. Hoffmann, C.M.: Geometric & Solid Modeling, An Introduction. Morgan Kaufman Publishers, Inc., San Mateo, Ca. (1989)

6. Mantyla, M.: An Introduction to Solid Modeling. Computer Science Press, Rockville, Ma. (1988)
7. Koman, F.: Ilhan Koman - Retrospective. Yapi ve Kredi Kulture Sanat yayincilik, Beyoglu, Istanbul, Turkey (2005)
8. Calladine, C.R.: Theory of Shell Structures. Cambridge University Press, Cambridge (1983)
9. Milnor, J.: Morse Theory. Princeton University Press (1963)
10. Landreneau, E., Akleman, E., Srinivasan, V.: Local mesh operations, extrusions revisited. In: Proceedings of the International Conference on Shape Modeling and Applications. (2005) 351 – 356
11. Kobbelt, L.: $\sqrt{3}$-subdivision. In: Proceedings of SIGGRAPH 2000. Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH (2000) 103–112
12. Loop, C.: Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah (1987)
13. Doo, D., Sabin, M.: Behavior of recursive subdivision surfaces near extraordinary points. Computer Aided Design (10) (1978) 356–360
14. Catmull, E., Clark, J.: Recursively generated b-spline surfaces on arbitrary topological meshes. Computer Aided Design (10) (1978) 350–355
15. Peters, J., Reif, U.: The simplest subdivision scheme for smoothing polyhedra. ACM Transactions on Graphics **16**(4) (1997) 420–431
16. Sabin, M.: Subdivision: Tutorial notes. Shape Modeling International 2001, Tutorial (2000)
17. Claes, J., Beets, K., Reeth, F.V.: A corner-cutting scheme for hexagonal subdivision surfaces. In: Proceedings of Shape Modeling International'2002, Banff, Canada. (2002) 13–17
18. Oswald, P., Schröder, P.: Composite primal/dual $\sqrt{3}$-subdivision schemes. Computer Aided Geometric Design, CAGD (2003)
19. Akleman, E., Srinivasan, V.: Honeycomb subdivision. In: Proceedings of ISCIS'02, 17th International Symposium on Computer and Information Sciences. Volume 17. (November 2002) 137–141
20. Akleman, E., Srinivasan, V., Melek, Z., Edmundson, P.: Semi-regular pentagonal subdivision. In: Proceedings of the International Conference on Shape Modeling and Applications. (2004) 110–118
21. Srinivasan, V., Akleman, E.: Connected and manifold sierpinski polyhedra. In: Proceedings of Solid Modeling and Applications. (2004) 261–266
22. Zorin, D., Schröder, P.: A unified framework for primal/dual quadrilateral subdivision schemes. Computer Aided Geometric Design, CAGD (2002)
23. Prautzsch, H., Boehm, W.: Chapter: Box splines. The Hanbook of Computer Aided Geometric Design (2000)
24. Fomenko, A.T., Kunii, T.L.: Topological Modeling for Visualization. Springer-Verlag, New York (1997)
25. Ferguson, H., Rockwood, A., Cox, J.: Topological design of sculptured surfaces. In: Proceedings of SIGGRAPH 1992. Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH (1992) 149–156
26. Welch, W., Witkin, A.: Free-form shape design using triangulated surfaces. In: Proceedings of SIGGRAPH 1994. Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH (1994) 247–256
27. Dyn, N., Levin, D., Simoens, J.: Face-value subdivision schemes on triangulations by repeated averaging. In: Curve and Surface Fitting: Saint-Malo 2002. (2002) 129–138

28. Takahashi, S., Shinagawa, Y., Kunii, T.L.: A feature-based approach for smooth surfaces. In: Proceedings of Fourth Symposium on Solid Modeling. (1997) 97–110
29. Zorin, D., P. Schröder, e.: Subdivision for modeling and animation,. ACM SIGGRAPH 2000 Course #23 Notes (2000)
30. Gross, J.L., Tucker, T.W.: Topological Graph Theory. John Wiley & Sons, New York (1987)

Shape-Based Retrieval of Articulated 3D Models Using Spectral Embedding

Varun Jain and Hao Zhang

GrUVi Lab, School of Computing Sciences,
Simon Fraser University, Burnaby, British Columbia, Canada
{vjain, haoz}@cs.sfu.ca

Abstract. We present an approach for robust shape retrieval from databases containing articulated 3D shapes. We represent each shape by the eigenvectors of an appropriately defined affinity matrix, obtaining a spectral embedding. Retrieval is then performed on these embeddings using global shape descriptors. Transformation into the spectral domain normalizes the shapes against articulation (bending), rigid-body transformations, and uniform scaling. Experimentally, we show absolute improvement in retrieval performance when conventional shape descriptors are used in the spectral domain on the McGill database of articulated 3D shapes. We also propose a simple eigenvalue-based descriptor, which is easily computed and performs comparably against the best known shape descriptors applied to the original shapes.

1 Introduction

In recent years, there has been a tremendous advance in 3D model acquisition technology and a large number of 3D models have become available on the web or through other means. The problem of indexing and retrieval of 3D shapes [1] has become as important, both in practice and in terms of research interests, as that of indexing and retrieval of image or textual data. Formally, given a database of 3D shapes represented in the form of triangle meshes[1], and a query shape, a shape retrieval algorithm seeks to return shapes, ordered by decreasing visual similarity to the query shape, from the database that belong to the same class as the query, where the classification is done by human.

Since the process of object recognition by human has not been completely understood, we are still incapable of proving theoretically that one particular shape retrieval algorithm is the best. In practice, several benchmark data sets and their associated performance evaluations [1,2,3] are available to empirically measure the quality of existing shape retrieval algorithms. The most comprehensive comparative study of retrieval algorithms for 3D shapes to date is due to Shilane et al. [1], based on the now well-known Princeton shape benchmark.

A variety of retrieval algorithms have been proposed [4]. Typically, each shape is characterized by a shape descriptor. An appropriately defined similarity distance between the descriptors sorts the retrieved models. Commonly used quality

[1] A liberal use of the term mesh is adopted: the mesh can be non-manifold, open or closed, having disconnected parts, or a collection of disjoint soup of triangles.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 299–312, 2006.
© Springer-Verlag Berlin Heidelberg 2006

criteria for shape descriptors include invariance to rigid-body transformations, scaling, bending and moderate stretching, robustness against noise and data degeneracies, and storage and computational costs. The discriminative power of a shape descriptor and its similarity distance is most often judged by plotting the precision-recall (PR) curve [1] generated from a benchmark database.

Most state-of-the-art descriptors, including the twelve compared by Shilane et al. [1] on the Princeton benchmark, are designed to be invariant to only rigid-body transformations and uniform scaling. Hence, it is no surprise that they do not perform well when applied to shapes having non-rigid transformations such as bending or stretching, which are obviously harder to handle due to their non-linearity and increased degrees of freedom. In this paper, we propose a technique to render a descriptor invariant to bending, hence enhancing its performance over databases that contain articulated shapes. Our experiments will thus be conducted primarily on the McGill database of articulated shapes [3].

Given a shape represented as a triangle mesh, we first apply pre-processing to convert it into a connected weighted graph. Shortest graph distances between pairs of nodes, mimicking geodesic distances over the mesh surface, provide an intrinsic characterization of the shape structure. We filter these distances appropriately to remove the effect of scaling and then compute a spectral embedding of the shape in a low-dimensional space, where we attain invariance to bending. The spectral embeddings are given by the eigenvectors, properly scaled, of the matrix of filtered distances. The corresponding eigenvalues can be used to obtain a simple shape descriptor that works quite well on the McGill database.

Alternatively, any existing 3D shape descriptors can be applied to spectral embeddings in 3D, which would result in absolute performance improvements in the PR curves on the McGill database. In this paper, we demonstrate this for the spherical harmonic shape (SHD) descriptor of Kazhdan et al. [5] and the light field descriptor (LFD) of Chen et al. [2], two of the best-performing descriptors from the Princeton benchmark test [1]. Finally, it is worth noting that with the aid of sub-sampling and interpolation via Nyström approximation [6], the spectral embeddings are quite efficient to compute.

The rest of the paper is organized as follows. After briefly discussing previous work in Section 2, we describe efficient construction of the bending-invariant spectral embeddings from a given mesh, possibly with disconnected components and other degeneracies, in Section 3. In Section 4, we give a comparative study between various shape descriptors, including those derived from spectral embeddings, for shape retrieval. Experimental results and discussions are given in Section 5. Finally, we conclude in Section 6 and suggest possible future work.

2 Previous Work

It is quite conceivable that a great deal of prior knowledge is incorporated into the process of human object recognition and classification, perhaps with subpart matching playing an important role. In this paper however, we focus on purely shape-based approaches using global shape descriptors [4]. At a high level, a 3D

shape retrieval algorithm either works on the 3D models directly, e.g., [5,7], or relies on a set of projected images [2] taken from different views. Let us call these the object-space and the image-space approaches, respectively. The latter, e.g., the LFDs of Chen et al. [2], has a more intuitive appeal to visual perception and thus often result in better benchmark results for retrieval [1], but at the expense of much higher computational cost.

Many object-space shape descriptors construct one or a collection of spherical functions, capturing the geometric information in a 3D shape *extrinsically* [1]. These spherical functions represent the distribution of one or more quantities, e.g., distance from points on the shape to the center of mass [8], curvatures [9], surface normals [10], etc. The bins are typically parameterized by the sphere radius and angles. The spherical functions are, in most cases, efficient to compute and robust to geometric and topological noise, but they may be sensitive to the choice of sphere center or the bin structures. To align the bins for two shapes properly, these approaches require pre-normalization with respect to translation, rotation, and uniform, e.g., [8,10], or nonuniform scaling [11]. As an alternative, rotation-invariant measures computed from the spherical functions, e.g., energy norm at various spherical harmonic frequencies [5], can be utilized. However, non-rigid transformations are not handled by these approaches.

As a salient intrinsic geometric measure, surface curvature, as well as the principal curvature directions, have been used for shape characterization and retrieval [9]. These approaches are sensitive to noise and non-rigid transforms such as bending. Another intrinsic approach is the use of shape distributions [7], where a histogram of pairwise distances between the vertices of a mesh defines the shape descriptor. Other form of statistics, e.g. [12], can also be used and bending-invariance can obviously be achieved if geodesic distances are used in this context, but the discriminative power of the histograms is suspect.

The most common approach to handling shape articulation is via skeletal or other graph representation of the shapes, e.g., [13,14], and then apply graph matching. The cost of extracting the skeletons can be high, e.g., when medial-axes are used [14], and the subsequent graph matching is often computationally expensive and the shape descriptor itself is sensitive to topological noise. Our approach also uses a graph-based intrinsic characterization of the shapes. The spectral embeddings automatically normalize the shapes against rigid-body transformations, uniform scaling, and bending, and they are fast to compute. The resulting shape descriptors provide a more intuitive way of characterizing shapes, compared to shape distribution [7]. In addition, the spectral approach is quite flexible and allows for different choices of graph edge weights and distance computations, rendering the approach more robust against topological noise.

The idea of using spectral embeddings for data analysis is not new and clustering [15,16] and correspondence analysis [17,18,19] are the main applications. Past work that is most relevant to ours is the use of bending-invariant shape signatures by Elad and Kimmel [20]. They work on manifold meshes and compute spectral embeddings using multidimensional scaling (MDS) based on geodesic distances. A more efficient version of MDS is adopted to approximate the true

embeddings; this is different from Nyström approximation. They only tested shape retrieval on manifold, isometric shapes, e.g., models obtained by bending a small set of seed shapes. In practice, many 3D shapes are neither manifolds nor isometric to each other, thus a more robust approach, based on more general graphs and distance measures, and a more complete experiment, are called for.

3 Construction of Spectral Embeddings

Point correspondence between two images or extracted image features has been well studied in computer vision. Spectral technique is first applied to this problem by Umeyama [21], Scott and Longuet-Higgins [22], and Shapiro and Brady [18]. Since then, the use of spectral techniques for correspondence in 2D has received a great deal of attention [17]. In machine learning, spectral clustering [15] and its variants are well-known. However, the use of spectral methods for 3D geometry processing is relatively new. For example, Zhang and Liu [16] apply spectral embeddings to mesh segmentation, while Gotsman et al. [23] utilize the spectral properties of mesh Laplacians for spherical parameterization. Spectral analysis has also been applied to mesh compression [24]. To the best of our knowledge, the use of spectral embeddings for 3D shape retrieval has not been reported before. In this section, we describe the process of constructing spectral embeddings for a 3D mesh that can be subsequently used for shape retrieval.

3.1 Affinity Matrix and Spectral Embedding

Given a 3D triangle mesh with n vertices, we form an $n \times n$ affinity matrix A such that the ij^{th} entry of A is the affinity between the i^{th} and the j^{th} mesh vertices. Several possible choices for the affinities are discussed in Section 3.4. A is then eigen-decomposed as, $A = V \Lambda V^T$, where Λ is a diagonal matrix with eigenvalues $\lambda_1 \geq ... \geq \lambda_n$ along the diagonal and $V = [\mathbf{v_1}|\ldots|\mathbf{v_n}]$ is the $n \times n$ matrix of the corresponding eigenvectors, with $\mathbf{v_1}, \ldots, \mathbf{v_n}$ the eigenvectors.

As the eigenvectors are of unit length, their entries may vary in scale with change of the mesh size n. We normalize this variation by scaling the eigenvectors by the square-root of the corresponding eigenvalues [19] and then consider only the first k scaled eigenvectors to give a k-dimensional *spectral embedding*:

$$\hat{V}_k = [\hat{\mathbf{v}}_\mathbf{1}|\ldots|\hat{\mathbf{v}}_\mathbf{k}], \text{where } \hat{\mathbf{v}}_\mathbf{1}, \ldots, \hat{\mathbf{v}}_\mathbf{k} \text{ are the first } k \text{ columns of } \hat{V} = V\Lambda^{\frac{1}{2}}.$$

Specifically, the i^{th} row of the $n \times k$ matrix $\hat{V}_k$ gives the k-dimensional coordinates of the i^{th} vertex of the mesh.

An advantage of using this particular framework for shape characterization is that if the affinities in the matrix A are invariant to a particular transformation, then the resultant embeddings will also be invariant to that transformation. This property can be exploited to construct bending-invariant embeddings if we note that the geodesic distance between two points on a mesh remains constant when the shape undergoes bending. We now explain the construction of bending-invariant spectral embeddings using approximate geodesic distances.

3.2 Bending-Invariant Spectral Embedding

In order to achieve invariance to bending, we wish to define the affinities based on geodesic distances. However, conventional methods for geodesic estimation over a mesh depend largely on the mesh being connected. This limits the use of geodesic distances, as we have noticed that many shapes in all the well-known shape database [1,3] have disconnected parts (a small number of shapes are simply triangle soups), in which case, the geodesic distance estimation would fail. We thus turn to a heuristic as a work-around.

Construction of structural graph: We use shortest graph distances over a mesh graph to approximate geodesic distances. This not only leads to simpler implementation, but also removes the constraint that the shape be defined using a connected manifold mesh. However, disconnected meshes are still not handled properly. To this end, we add extra edges to the mesh graph composed of its original vertices and edges while making sure that the structure of the shape remains largely unchanged. Specifically, given a 3D mesh M, let $G_M = (V, E_M)$ be its connectivity graph and $C_1, C_2, \ldots$ be its disconnected components. We construct a p-connected graph $G_p = (V, E_p)$ over the mesh vertices such that the graph faithfully represents the shape. This is done using Yang's algorithm [25] for constructing p-connected graph over point clouds in Euclidean space that locally minimizes edge lengths by computing and combining p Euclidean minimum spanning trees of the given point cloud. As shown in [25] and verified by our experiments, the resulting graph G_p approximates the structure of the shape well. With G_M and G_p in hand, the final *structural graph* is defined as

$$G = (V, E), \text{where } E = E_M \cup \{(i, j) \mid (i, j) \in E_p, i \in C_s, j \in C_t, s \neq t\}.$$

Clearly, G is connected and it includes all edges of G_M and only those edges of G_p that join two disconnected components in G_M. This helps better preserving the structure of the mesh. Once the structural graph G has been constructed, the geodesic distance between two vertices can be approximated by the shortest path length in G computed using Dijkstra's shortest path algorithm.

In our implementation, we restrict p, in Yang's algorithm, to be 1 or 2 since higher values of p may result in edges between far away (hence, unrelated) components. This is illustrated in Fig. 1 where we plot the average error (as a percentage of the bounding box diagonal or BBD) in the estimation of geodesic distances using the approach described above, against degeneracy levels in the mesh. To add degeneracy in a mesh we randomly select a number of faces and disconnect them from the mesh and jitter the position of their vertices. The noise level plotted on the horizontal axis indicates the number of such faces. The meshes on which the experiment was carried out all contain 350 faces. Clearly, increasing p would result in non-robustness of geodesic distance estimation.

Gaussian affinities: Now that we have a way to robustly estimate geodesic distances, we can define the affinity matrix A, which is given by a *Gaussian*: $A_{ij} = \exp(-d_{ij}^2/2\sigma^2)$, where d_{ij} is the approximate geodesic distance between the i^{th} and j^{th} vertices of the mesh, and σ is the Gaussian width. We simply

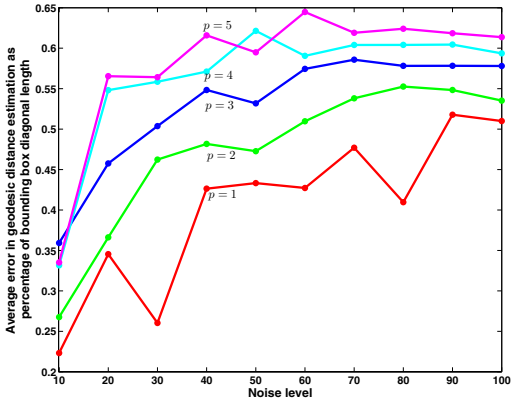


Fig. 1. Error (as % of BBD) in geodesic distance approximations with varying p

Fig. 2. Spectral embeddings (bottom row) of some articulated 3D shapes (top row) from the McGill shape database. Note that normalizations have been carried out.

set $\sigma = \max_{(i,j)}\{d_{ij}\}$. Defining σ in this data-dependent manner renders the embedding invariant to uniform scaling. We observe experimentally that the embeddings are relatively stable with respect to σ as long as it is sufficiently large. As a consequence of setting σ to a large number, the row-sums of the matrix A become almost constant. It follows that the first eigenvector $\mathbf{v_1}$ of A is very close to being a constant vector. Hence, we exclude the first eigenvector and consider only a $(k-1)$-dimensional embedding defined by $\mathbf{v_2}, \ldots, \mathbf{v_k}$.

For all the 3D shape retrieval results obtained based on spectral embeddings in this paper, we represent every shape with a 3D spectral embedding given by the 2^{nd}, 3^{rd} and 4^{th} eigenvectors, scaled by the square-root of the corresponding eigenvalues, of the affinity matrix associated with the shape. The 3D embeddings of some articulated shapes from the McGill database are shown in Fig. 2.

3.3 Nyström Approximation

The time complexity for constructing the full affinity matrix A for a mesh with n vertices is $O(n^2 \log n)$. Moreover, the eigen-decomposition of an $n \times n$ matrix takes $O(n^3)$ time; $O(kn^2)$ if only the first k eigenvectors are computed. This complexity does not affect the retrieval performance drastically, since the

spectral embeddings of all the shapes in the database can be precomputed. However, the query model needs to be processed at run-time. To speed things up, we use Nyström approximation [6] to efficiently approximate the eigenvectors of A. Nyström approximation is a sub-sampling technique that reduces the time complexity of affinity matrix construction and eigen-decomposition to $O(ln \log n + l^3)$, where l is the number of samples selected; typically, $l \ll n$. We adopt furthest point sampling, which at each step, chooses a sample which maximizes the minimum (approximated) geodesic distance from the new sample to the previously found samples; the first sample can be chosen randomly. Our extensive experiments confirm that for the purpose of shape retrieval, only 10 to 20 samples, from meshes with thousands of vertices, are sufficient.

3.4 Other Affinity Measures

Although geodesic-based affinities lead to bending invariance, it might cause adverse effects in some cases. For example, consider two chair models. Suppose that the arm-rest of one model is connected directly to its back-rest, however, on the other model, this connection is only through the seat. In the first case, the geodesic distance between a point on the arm-rest and a point on the back-rest is small, whereas in the second case this distance will be relatively large. Hence, the spectral embeddings of the two chairs could be radically different and the retrieval result will suffer. In general, geodesic distances are sensitive to topological noise in the shapes. If we define affinities based on Euclidean distances, the above problem would be resolved but the affinities can no longer be expected to be invariant (or even robust) to bending. Nevertheless, the current discussion reveals the flexibility of our approach, with the use of affinity matrices, in that they can be easily tuned to render the retrieval process invariant to a particular class of transformations depending on the database in question.

In Section 4, we compare retrieval results using different affinity measures. In addition to (approximated) geodesic-based affinities and affinities based on Euclidean distances, we also include *combined distance*, where a uniform combination of the above two measures is used. Since our target database is that of articulated shapes, it is not surprising that the geodesic-based affinities perform the best. Minor improvements over conventional shape descriptors can still be seen using other affinities, which strengthens our proposal of performing retrieval on spectral embeddings instead of on the original shapes.

4 Global Descriptors for Shape Retrieval

We now present a comparative study of two global shape descriptors, the spherical harmonics descriptor [5] (SHD) and the light field descriptor [2] (LFD), in the context of shape retrieval. Both of these descriptors have been shown to give excellent shape retrieval results in the Princeton shape benchmark [1]. In fact, the light field descriptor is the best among all the descriptors compared in [1]. We evaluate the performance of the descriptors when they are applied to the

original meshes as compared to when they are applied to the spectral embeddings of the meshes. We use the McGill database of articulated 3D shape [3] for our experiments. We also present two simple descriptors, easily obtained from the spectral embeddings, that perform comparably to the other descriptors.

The McGill shape database contains 255 models in 10 categories: Ants, Crabs, Hands, Humans, Octopuses, Pliers, Snakes, Spectacles, Spiders, and Teddybears. There are 20 to 30 models per category. Some shapes from the database are shown in Fig. 2 and 4. We now explain the descriptors we compare.

1. **Light Field Descriptor (LFD) [2]:** represents the model using histograms of 2D images of the model captured from a number of positions, uniformly placed on a sphere. The distance between two models is the distance between the two descriptors minimized over all rotations between the two models, hence attaining robustness to rotations. The main idea of this descriptor is to define shape similarity based on the visual similarity of the two shapes.
2. **Spherical Harmonics Descriptor (SHD) [5]:** is a geometry based representation of the shape which is invariant to rotations. It is obtained by recording the variation of the shape using spherical harmonic coefficients computed over concentric spherical shells.
3. **Spectral Shape Descriptors:** The following are two shape descriptors that can be easily obtained from the spectral embeddings. The EVD descriptor consists of only six eigenvalues and performs comparably as and sometimes better than SHD. This shows the effectiveness of the affinity matrix and spectral embeddings in encoding essential shape information.
   (a) **Eigenvalue Descriptor (EVD):** While the eigenvectors of the affinity matrix form the spectral embedding which is a normalized representation of the shape, the eigenvalues specify the variation of the shape along the axes given by the corresponding eigenvectors. Hence, as a simple shape descriptor, we use the square root of the first six eigenvalues of the affinity matrix. The reason for choosing only six eigenvalues is that the remaining terms in the spectrum are believed to encode high frequency shape information, which may render the descriptor too sensitive to shape noise. Also, the eigenvalues tend to decrease quickly, hence, only the largest eigenvalues shall encode significant shape information. With EVD, the distance between two meshes P and Q is given by the χ^2-distance between the square root of their first six eigenvalues:

$$Dist_{EVD}(P,Q) = \frac{1}{2}\sum_{i=1}^{6} \frac{[\,|\lambda_i^P|^{\frac{1}{2}} - |\lambda_i^Q|^{\frac{1}{2}}]^2}{|\lambda_i^P|^{\frac{1}{2}} + |\lambda_i^Q|^{\frac{1}{2}}}.$$

   It is worth noting however that the eigenvalues are affected by mesh sizes and there are shapes with different number of vertices in the shape database. Thus the eigenvalues of the original affinity matrices cannot be used for shape comparison directly. However, recall that we only compute a sampled affinity matrix, required by Nyström approximation. Thus with the same number of samples taken on each shape, the eigenvalues of the sampled affinity matrices can be used as is.

(b) **Correspondence Cost Descriptor (CCD):** The distance between two shapes in the CCD scheme is derived from the correspondence between the vertices of the two shapes. Given the respective k-dimensional spectral embeddings of two shapes P and Q in the form of an $n_P \times k$ matrix V_P and an $n_Q \times k$ matrix V_Q, the CCD distance given by:

$$Dist_{CCD}(P,Q) = \sum_{p \in P} \|V_P(p) - V_Q(match(p))\|,$$

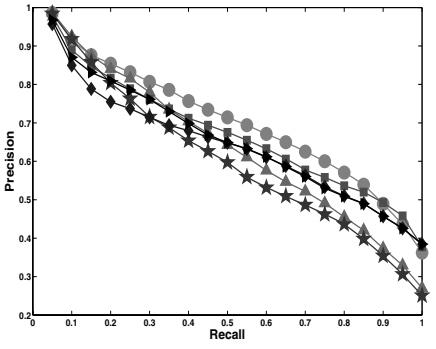
where $V_P(p)$ and $V_Q(q)$ are the p^{th} and q^{th} rows of V_P and V_Q, respectively, p represents a vertex of P, and $match()$ is some computed mapping between the vertices of P and the vertices of Q. This matching can be obtained using any correspondence algorithm, e.g., [18,19]. We have chosen to compute correspondence using the spectral embeddings obtained from the previous step. The correspondence algorithm uses best matching based on Euclidean distance in the embedding space [18],

$$match(p) = \text{ argmin }_{q \in Q} \|V_P(p) - V_Q(q)\|.$$

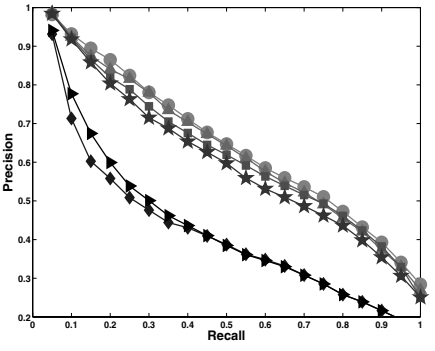
The intuition behind defining such a similarity cost is that if two shapes are similar (though they may differ by a bending transformation), their spectral embeddings would be similar, hence the Euclidean distance between a point and its match will be small, resulting in a smaller value of $Dist_{CCD}(P,Q)$. However, note that the time complexity of finding the distance between two shapes in the CCD scheme is $O(n^2)$, where n is the number of vertices. This is extremely slow and is not feasible to apply for comparing the query model with a large number of models in a database. Hence, we use CCD in conjunction with EVD. We first use EVD to filter out all poor matches via thresholding. Only the top few matches obtained from EVD are further refined using CCD.

5 Experimental Results

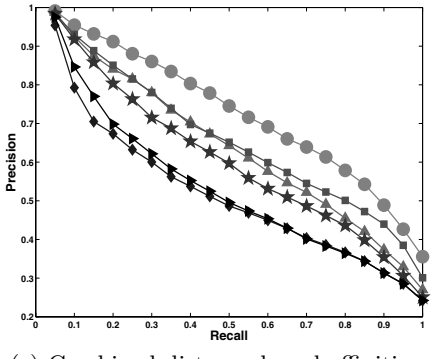
In this section, we present experimental results. First, we plot the precision-recall (PR) curves for the four descriptors, given in the previous section, when they are applied to the McGill database of articulated shapes, in Fig. 3. For (a), the approximate geodesic distances are used to construct the affinities. Clearly, the descriptors show significant improvements when applied to bending-invariant embeddings, compared to their spatial domain counterparts. In (b), we show the performance of the same set of descriptors. However, we construct embeddings based on Euclidean distances. Note that the performance of spectral descriptors degrades considerably. This is mainly because these are naive descriptors that rely on the ability of the affinity matrix to normalize proper transformations between the shapes. Since Euclidean distance based affinity matrix does not normalize the shapes against bending and the database in question is particularly that of articulated shapes, such poor performance is expected.



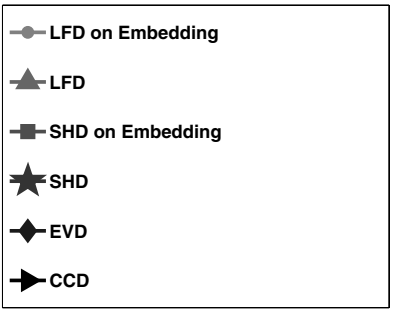
(a) Geodesic distance based affinities.



(b) Euclidean distance based affinities.



(c) Combined distance based affinities.

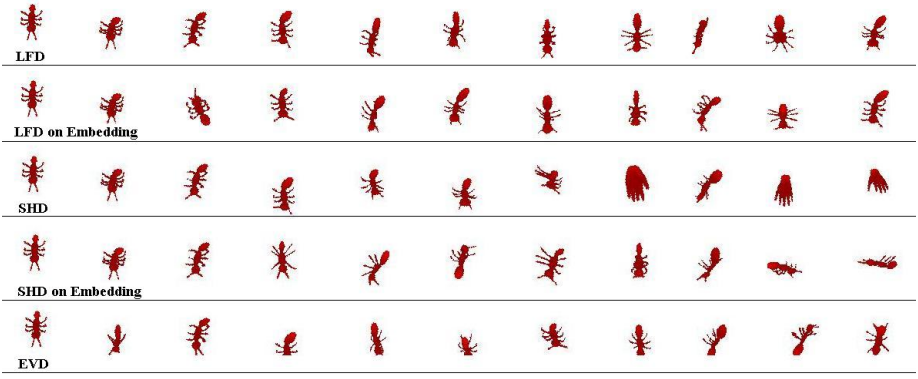


Legends.

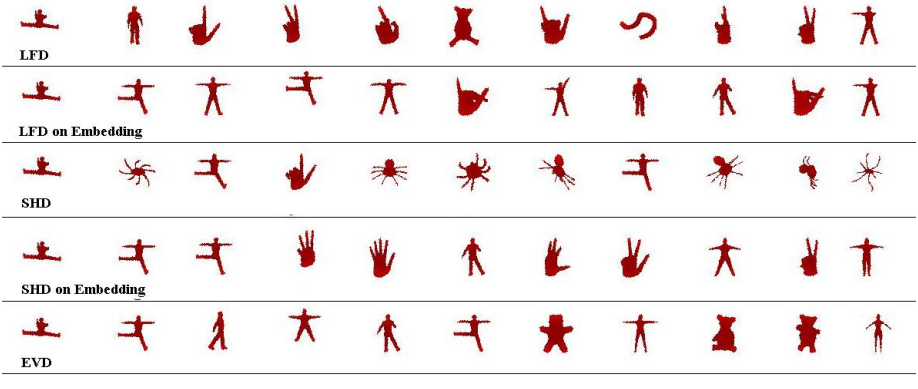
Fig. 3. Precision-recall (PC) plots for various global descriptors, derived from different distance measures, when applied to the McGill database of articulated shapes [3]

Fig. 3(c) shows the performance of the descriptors where the affinities are calculated using an average of geodesic and Euclidean distances. For LFD and SHD, both geodesic affinities and combined affinities give considerable improvements. Whereas, Euclidean affinities show only minor improvements since they fail to normalize the shapes against bending. EVD and CCD perform well only when the embeddings are normalized against bending for reasons mentioned above.

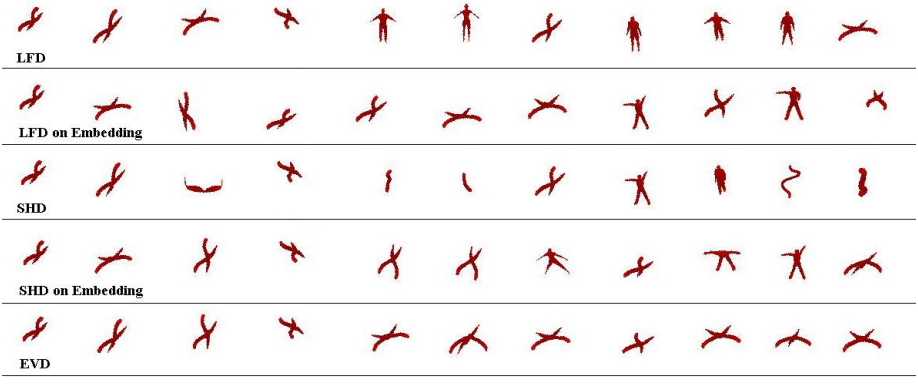
In terms of running times, the EVD's are quite efficient to compute due to subsampling. The time taken to compute the subsampled affinity matrix grows linearly with mesh size while the time required for computing the descriptor, an eigenvalue problem of size $k \times k$, is the same as we select $k = 10$ samples throughout. On an Intel Pentium M 1.7GHz machine with 1GB RAM, it takes between 1.4 and 2 milliseconds to compute the EVDs of meshes whose face counts range from 2,000 to 4,000. SHD and LFD computations are more expensive, taking an average of about 2 and 2.3 milliseconds for meshes in that range of face counts, respectively.



(a)



(b)



(c)

Fig. 4. Retrieval results using the McGill database of articulated 3D shapes [3]: First column in each row is the query shape. This is followed by the top ten matches retrieved using the shape descriptor as indicated.

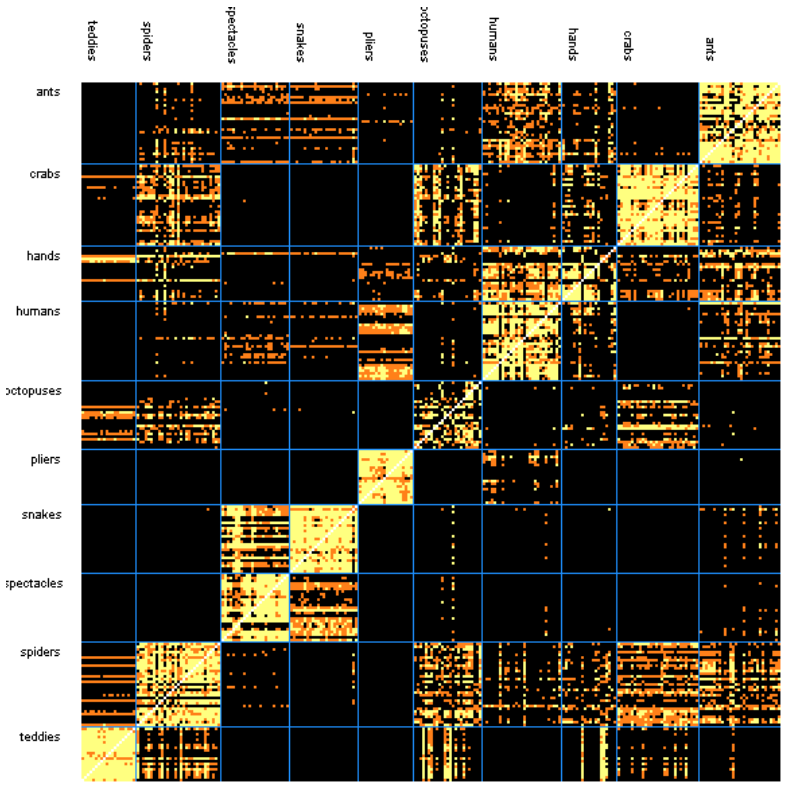


Fig. 5. Similarity matrix for shapes from the McGill database of articulated 3D shapes [3], computed using LFD on spectral embeddings

Next, we show some visual results which emphasize the need for bending-invariant spectral embeddings in order to obtain more robust retrieval of articulated shapes. These results are shown in Fig. 4. The spectral embeddings used in these results are all constructed using geodesic distance based affinities. Fig. 4(a) shows the results of retrieving an ant shape from the database. Note the poor performance of SHD even when the amount of bending is moderate. Fig. 4(b) and (c) show results for querying the database with a human and a plier shape, respectively, that have a relatively larger amount of bending. As we can see, LFD performs rather poorly on the original shapes.

It is quite evident from Fig. 4 that shape descriptors applied to spectral embeddings show clear and consistent improvement over their spatial domain based counterparts. It is also interesting to note from Fig. 4 that EVD, our simple shape descriptor based on eigenvalues, appears to work the best. We have indeed observed that most, if not all, incorrect retrieval results using EVD are caused by having parts of a shape incorrectly connected in our construction of the structural graph. Recovering the correct shape information from a soup of triangles or sparsely and nonuniformly sampled points (which occur often in the shape

databases) is not an easy problem, but any improvements in this regard will improve the performance of the EVD. Our current heuristic is quite primitive and we would like to look into this problem in our future work.

In Fig. 5, we show an image representation of the similarity matrix for all the shapes of the database. Here, a bright pixel represents greater similarity. The descriptor used here is LFD on spectral embedding instead of the original shape. The diagonal structure of the image matrix shows that similar shapes have greater similarity value.

6 Conclusion and Future Work

In this paper, we consider the problem of shape-based retrieval of 3D models from a database and our focus is on articulated shapes. We present a method which renders conventional shape descriptors invariant to shape articulation, with the use of spectral embeddings derived from an appropriately defined affinity matrix. The affinity matrix encodes pairwise relations between the data points and invariance to a particular type of transformation can be achieved through a judicious choice of a distance measure. When conventional shape descriptors, e.g., LFD and SHD, are applied to spectral embeddings derived from approximated geodesic distances, on the McGill database of articulated 3D shapes, absolute improvements are achieved for shape retrieval. Robustness of affinity matrices is also shown, as minor improvements for the LFD and SHD descriptors can be observed, on the same database, even with Euclidean distance based affinities that are not invariant to bending.

In the future, we would like to explore more ways to define affinities that are robust and/or invariant to other complex shape transformations such as non-uniform linear scaling and moderate stretching (note that allowing arbitrary stretching and bending would then only enable us to distinguish shapes having different topology). Another interesting study would be to find other shape descriptors based on spectral embeddings that can be used for retrieval; we have suggested two simple ones, EVD and CCD, in this paper and their performances are only about equivalent to that of the state of the art. Issues such as, the number of eigenvalues or eigenvectors chosen and the distance norm (other than Euclidean) used for computing correspondence costs, all require further investigation. Finally, spectral methods can be sensitive to the presence of outliers in the data. However, this issue is not of great concern for 3D model retrieval, as the 3D models are mostly free of outliers. Moreover, since most models define a surface, outliers are easy to detect and remove. Study into ways of making the spectral method robust to outliers is more interesting and necessary with respect to retrieval and recognition of more general form of data.

References

1. P. Shilane, P. Min, M.K., Funkhouser, T.: The princeton shape benchmark. In: Proc. of Shape Modelling International. (2004)
2. D.-Y. Chen, X.-P. Tian, Y.T.S., Ouhyoung, M.: On visual similarity based 3d model retrieval. In: Computer Graphics Forum. (2003) 223–232

3. McGill 3D shape benchmark: http://www.cim.mcgill.ca/ shape/benchMark/.
4. Tangelder, T., Veltkamp, R.: A survey of content based 3d shape retrieval methods. In: Proc. of Shape Modeling International. (2004) 145–156
5. M. Kazhdan, T.F., Rusinkiewicz, S.: Rotation invariant spherical harmonic representation os 3d shape descriptors. In: Symposium on Geometry Processing. (2003)
6. Fowlkes, C., Belongie, S., Chung, F., Malik, J.: Spectral grouping using the nyström method. IEEE Trans. on PAMI **26**(2) (2004) 214–225
7. Osada, R., Funkhouseri, T., Chazelle, B., Dobkin, D.: Matching 3d shapes with shape distribution. In: Proc. of Shape Modeling International. (2001) 154–166
8. Vranic, D.: An improvement of rotation invariant 3d shape descriptor based on functions on concentric spheres. In: Proc. of ICIP. (2003) 757–760
9. Shum, H.: On 3d shape similarity. In: Proc. of CVPR. (1996) 526–531
10. Kang, S., Ikeuchi, K.: Determining 3-d object pose using the complex extended gaussian image. In: Proc. of CVPR. (1991) 580–585
11. Kazhdan, M., Funkhouser, T., Rusinkiewicz, S.: Shape matching and anisotropy. ACM Trans. Graph. **23**(3) (2004) 623–629
12. Ohbuchi, R., Minamitani, T., Takei, T. In: Shape-similarity search of 3D models by using enhanced shape functions. (2003) 97–104
13. Hilaga, H., Shinagawa, Y., Kohmura, T., Kunii, T.K.: Topology matching for fully automatic similarity estimation of 3d shapes. In: SIGGRAPH. (2001) 203–212
14. Zhang, J., Siddiqi, K., Macrini, D., Shokoufandeh, A., Dickinson, A.: Retrieving articulated 3-d models using medial surfaces and their graph spectra. In: Int. Workshop On Energy Minimization Methods in CVPR. (2005)
15. Ng, A.Y., Jordan, M.I., Weiss, Y.: On spectral clustering: analysis and an algorithm. In: NIPS. (2002) 857–864
16. Zhang, H., Liu, R.: Mesh segmentation via recursive and visually salient spectral cuts. In: Proc. of Vision, Modeling, and Visualization. (2005) 429–436
17. Carcassoni, M., Hancock, E.R.: Spectral correspondence for point pattern matching. Pattern Recognition **36** (2003) 193–204
18. Shapiro, L.S., M., B.J.: Feature based correspondence: an eigenvector approach. Image and Vision Computing **10**(5) (1992) 283–288
19. Jain, V., Zhang, H.: Robust 3d shape correspondence in the spectral domain. In: Proc. of Shape Modeling International. (2006) to appear
20. Elad, A., Kimmel, R.: On bending invariant signature of surfaces. IEEE Trans. on PAMI **25**(10) (2003) 1285–1295
21. Umeyama, S.: An eigen decomposition approach to weighted graph matching problem. IEEE Trans. on PAMI **10** (1988) 695–703
22. Scott, G., Longuet-Higgins, H.: An algorithm for associating the features of two patterns. Royal Soc. London **B244** (1991)
23. C. Gotsman, X.G., Sheffer, A.: Fundamentals of spherical parameterization for 3d meshes. In: ACM Transactions on Graphics (Proceedings of SIGGRAPH). (2003)
24. Gotsman, C., Karni, Z.: Spectral compression of mesh geometry. In: Computer Graphics (Proceedings of SIGGRAPH). (2000) 279–286
25. Yang, L.: k-edge connected neighborhood graph for geodesic distance estimation and nonlinear data projection. In: Proc. of ICPR. (2004)

Separated Medial Surface Extraction from CT Data of Machine Parts

Tomoyuki Fujimori[1], Yohei Kobayashi[2], and Hiromasa Suzuki[1]

[1] Research Center for Advanced Science and Technology, The University of Tokyo
[2] CREED Coorpolation
{fujimori, yohei, suzuki}@den.rcast.u-tokyo.ac.jp

Abstract. This paper describes a new method for extracting separated medial surfaces from CT (Computed Tomography) data of machine parts. Plate structures are common mechanical structures such as car body shells. When designing such structures in CAD (Computer Aided Design) and CAE (Computer Aided Engineering) systems, their shapes are usually represented as surface models associated with their thickness values. In this research we are aiming at extracting medial surface models of a plate structure from its CT data so as to be used in CAD and CAE systems. However, such a structure consist of many components which are adjacent each other. For example, car body shells are consist of many welded plates. CT imaging technology has some weak points in the area. One of them is that, if there are two or more objects made of same material, CT scanner cannot make the distinction between them. The problem is caused by the principles of CT imaging technology. Because CT image represents the mass distribution within a cross section, we cannot separate the objects only from image information. However, there are many requests for scanning assembled parts and separating objects made of same material. Therefore, we propose a method to separate each components. CT data has not enough information amount as has been metinoned, so we adopt other knowledge about model shapes. We conclude with experiments on welded mechine parts for effectiveness of our method.

1 Introduction

In this paper, we propose a new method to extract medial surfaces from CT (Computed Tomography) data of machine parts. We attach importance to industrial applicabilities of the medial surfaces. CT is a powerful nondestructive evaluation technique to generate cross sectional X-ray images of objects, from which we can further produce three dimensional volumetric images.

In this study, we focused on the industrial application of CT to analyze plate structures. Because of the purpose, our medial surfaces are given slightly different definitions from ones commonly used.

Medial surfaces (or medial axes in general) have been important tools in the visualization, the feature analysis, the feature recognition and the finite element mesh generation. There are many basic researches and applied researches. The easily understandable simple definition is given in [1]: *the medial surface is described by the locus of the center of a maximal sphere as it rools around the object interior*. However, the exact medial surfaces extraction is quite difficult to realize. Therefore, approximated

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 313–324, 2006.
© Springer-Verlag Berlin Heidelberg 2006

medial surfaces are calculated in many cases. The approximated surfaces often involve a noise, and so many methods have been proposed to solve the problem.

The medial surfaces proposed in this paper are given different definitions in regard to treatment of areas in which two or more plates contact each other. We want to acquire one medial surface for each plate in the conjugated area.

Figure 1 shows an example of our problem. Two plates are contact each other in (a). This is a two dimensional analogy of welded plates which appear frequently in mechanical engineering. As far as we know, cars contain a lot of those features. However, when scanning those parts by CT, we obtain an image of one thick plate structure as shown in the center of (b). If we apply the existing algorithm to the data, it will extract only one medial surface corresponding to the pseudo thick plate like (c). The problem is caused from the principles of CT imaging technology and strictly speaking this may not be a defect of the medial surface techniques. However, we desire to obtain the separated medial surface for each original plate as shown in (d).

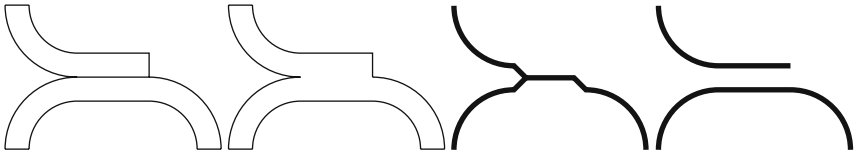


Fig. 1. An example of our problem. Two plates contact at the center of (a). (b) is its CT image. The existing medial algorithms extract medials surfaces as shown in (c). However, we expect to obtain the separated medial axes such as (d).

Therefore, we propose a new method to extract medial sufaces which satisfy the requirement. To avoid an ambiguity of the problem, we add two constraints to scanning objects. The first constraint is that the space is comprised of two or more plate structures and the second is that a thickness of each plate does not change drastically. In this research, we separate combined plates in discrete space under those constraints. Once we separate plates, we can easily extract each medial surface for the plate.

Concretely, first of all, we extract discrete medial surfaces in three dimensional grid space. In other words, we apply an image based algorithm to obtain medial surfaces. Our method is like to so called voxel thinning algorithms. Because extracted medial surfaces are represented by a set of cubes corresponding to voxels, we call them *medial cells*.

Our basic idea is to separate the medial cells per plate. In this paper, we focus on the medial cell separation. After extracting and separating medial cells, we create continuous medial surfaces as triangular polygonal meshes from the cells. We extend a method that we have already proposed [2] to create meshes. Final outputs of the process are these medial surfaces represented by polygonal meshes.

By the way, it is possible to create solid models from CT data by using Marching Cubes [3] or other contouring methods and extract medials by the existing algorithm.

There are three reasons we do not adopt the idea. The first reason is that topological informations (e.g. connectivities) are implicitly involved in adajacent relationships

of cells in three dimensional grid space. For example, if we represent a medial surface which has a complicated toplogy (e.g. T-junktion) by polygonal meshes, we must develop complicate toplogical operations to separate the medial.

At the same time, the image based noise reduction algorithm is more rapid than solid based one. This is the second reason. Furthermore, though two concepts output almost same results, the image based algorithm can process the data more robustly.

The third reason is that we assume CT data as inputs. Because CT data is three dimensional volumetric image, there is no need for applying rasterization algorithms unlike solid models. For the reasons stated above, we propose a cell based medial surface extraction method.

This paper consists of six sections. Section 2 surveys important related research works and Section 3 introduces a discrete medial surface extraction technique. Section 4 disucusses is our major contibution of this study. We propose a method to separate medial surfaces in this section. Section 5 shows some results obtained by prototype systems and Section 6 gives summary and future works.

2 Related Works

2.1 Image Based Thinning Algorithm

In this section, we survey image-based medial axes/surfaces extraction algorithms related our research.

Lam et al. [4] surveyed thinning algorithms which keep the topology of the object, but this study focused on medial edge but medial surface extraction. On the contrary, Pudney [5] proposed to combine thinning algorithms with a distance map based approach. This algorithm is faster and more robust than a naive implementation of a pure topological thinning algorithm that visits every voxel again and again.

If a voxel can be removed to preserve the homotopy of the skeleton, it may be required to conserve geometrically. How to distinguish these voxels is a big problem when designing a thinning algorithm. A distance map based approach has the advantage of allowing the overall geometry of the object to be considered, and can solve the problem by parameter controlling the thickness and size of the skeleton.

However parameter controlling made not easy to design algorithms that strictly maintain the object's topology. Gagvani et al. [6] proposed a measure, based on the distance map in the neighborhood of each voxel, allowing to identification of skeleton voxels. Malandain et al. [7] took the nearest boundary voxels into account. Prohaska et al. [8] followed the idea which they take more global view. Not only the nearest boundary voxels, but the geoedesic distance on the boundary surface is token into account. Prohaska's approach is challenging because it requires no parameter controlling but fast and robust. We mention later about the problems of the approach.

2.2 Contouring Methods

We first extract medial surfaces in the discrete space, and then extract triangular polygonal meshes from the discrete form by using the contouring method. We introduce the notable contouring methods below.

Cube-based methods such as the Marching Cubes algorithm [9] and its variants generate one or more triangles for each cube in the grid that intersects the contour of a signed field. Surface samples are computed by the intersection of the cube edges with the surface, and triangles connecting these samples are generated. Most cube-based techniques are conceptually derived from the Marching Cubes algorithm where a preprocessed triangulation is stored in a table for all possible configurations of edge intersections. Additionally, Lorensen's original Marching Cubes is unable to process non-manifold objects. Hege et al. [10] proposed the Generalized Marching Cubes.

Dual methods such as the SurfaceNets algorithm of [11] generate one vertex in a cube lying on or near the contour for each dual that intersects the contour. These methods generate a mesh which is the dual graph of a mesh generated by cube-based methods. Kobbelt et al. [12] proposed a method of representing sharp features on a surface in a volume-sampled distance field by applying a directed distance function. This method is a hybrid between a cube-based method and a dual method. Ju et al. [13] proposed a method for contouring a signed grid whose edges are tagged by Hermite data.

Hormann et al. [14] proposed a method to hierarchically extract isosurfaces. Esteve et al. [15] describes a method to obtain a closed surface that approximates a general data point set with non-uniform density.

In terms of combining volume thinning and Marching Cubes, we must refer to Ito et al. [16]. They proposed a method to use volume thinning for isosurface generation. They first extract a skeleton structure by volume thinning, and use this skeleton to efficiently search a seed to start Marching Cubes to generate an isosurface. So only a few seed cubes in the skeleton are used by Marching Cubes. But in our approach, we contour almost all the cubes contained in the skeleton. Lachaud et al. [17] proposed a local method to construct a continuous analog of a digital boundary. We also refer to a text book by Lohmann [18] for studying basic topics for operating 3D binary images described in its chapters 3 and 4.

We can apply a general contouring algorithm for the signed field to the cell. This paper discusses the Marching Cubes algorithm but other algorithms (Extended Marching Cubes algorithm [12], Dual Contouring algorithm [13] etc.) are applicable in principle. However, they require Hermite data or normal vectors at samples that cannot be accurately defined in our typical CT image examples. Thus we cannot guarantee the accuracy with those contouring methods.

And last, we have to refer again Prohaska et al. [8]. They also create surface from medial axes/surfaces voxels. However, their method did not assure connectivity of the surfaces. That is, it is one of our purpoises that assure the connectivity and the topology.

3 Discrete Medial Surfaces Extraction

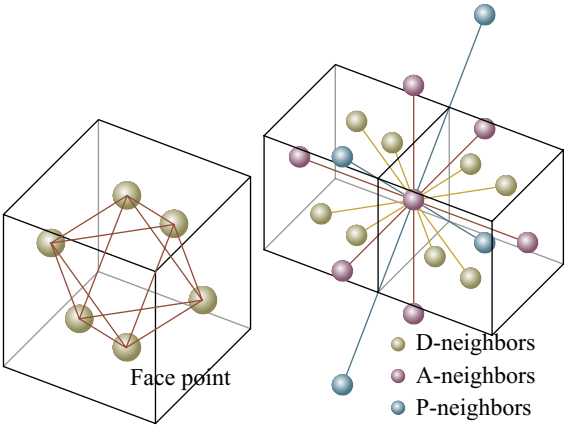
We introduce a discrete medial surface extraction technique in this section. The method was proposed in [2] and our intrinsic contributions appear in the next section.

We treat the three dimensional CT image as volumetric data. Elements of the volumetric data are called voxel while elements of two dimensional image are called pixel. A voxel is a sample of an object at a point of a three dimensional regular grid and has a scalar value (usually called CT value).

Image based medial surface extraction is generally done by thinning an object of voxels by removing extra voxels from the object's surface, and leaving voxels only on the medial axis (surface) of the object. This problem is generally known as *Medial Axis Transform* and there are many algrotihms as introduced in Section 2.

Considering the dual structure to a three dimensional regular grid, we define a cubic geometric element cell. A cell is surrounded by three kind of topological elements: its 8 corner voxels (vertices), 12 edges, and 6 faces.

Contouring methods such as Marching Cubes algorithm generate one or more triangles in each boundary cell. The triangles approximate a closed boundary surface and pass through boundary cells' faces. Therefore, each cell face can be classified to three classes: inside object, outside (background) or boundary. Also, we sample center points on cell faces (*f(ace)-points*) as shown in Figure 2(a), and classify f-points into *object* f-points, *background* f-points or *boundary* f-points corresponding to its parent face class.



(a) Face points are center of cell's faces.

(b) A face point has eight diagonal neighbors (D-neighbors), six axial neighbors (A-neighbors) and four plane diagonal neighbors (P-neighbors).

Fig. 2. Face points and their grid structure

3.1 Cell Skeletonization

The following intoduces an algrotithm of *Cell Skeletonization* which was proposed in [2]. For every object f-point, we find its nearest boundary f-point. A distance transform is defined by

$$\mathrm{DT}(p) = \min\left(\mathrm{d}(p,q) : q \in BP\right)$$

where $\mathrm{d}(p,q)$ is the distance between f-points p and q, and BP is a set of boundary f-points. d is the distance defined on our grid where we use $\langle 3,4,6\rangle$. The distance to

diagonal neighbors, axial neighbors and plane diagonal neighbors is 3, 4 and 6 respectively (Figure 2(b)).

Let us denote the nearest boundary f-point of p by $\mathrm{NBP}(p)$. After the distance transform, $\mathrm{NBP}(p)$ is defined for all object f-points and the boundary f-points. This information is use to extract medial cells. Each cell has six f-points. A pair of these f-points is made in each of the x, y and z directions. For example, we make a pair (x_1, x_2) by selecting f-points on the two faces perpendicular to the x axis. If the two f-points (x_1, x_2) in the pair are not both background f-points, we can defined a pair of $\mathrm{NBP}(x_1)$ and $\mathrm{NBP}(x_2)$. The geodesic distance can be calculated by propagation similar to the distance transfrom. In the same way, we calculate the other two geodesic distances by using the f-point pairs in the y and z direcitons. If the maximum of those three geodesic distances is greater than a threshold t, the cell is taken to be a medial cell as shown in Figure 3.

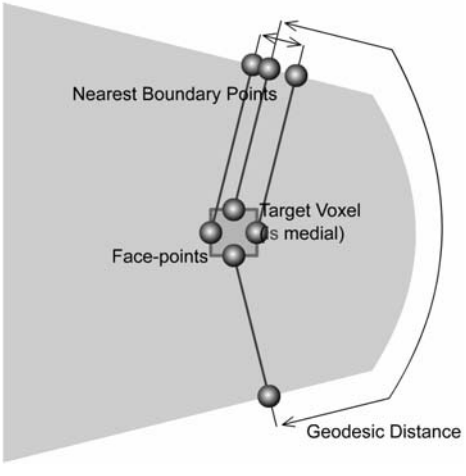


Fig. 3. Two dimensional example of Cell Skeletonization. We calculate the geodesic distances. If the maximum of them is ghreater than t, the cell is a medial.

4 Medial Cell Separation

In the previous section, we described the method to extract medial cells of a plate structure from CT data by skeletonization. We depict a Medial Cell Separation algorithm in this section.

At the beginning, we binarize the grid space into foreground and background as shown in Figure 4(a). In this time, contacted objects are recognized as one large object. We apply the algorithm described in Section 3 to the foreground cells. As a result, we acquire not only *the first medial cells* but also thickness information of the cells. Next, We use the thickness information to divide the conjugated forground cells as shown in Figure 4(b). At last, we re-calculate medial cells for each separated foreground cells as shown in (c). This is our main concept.

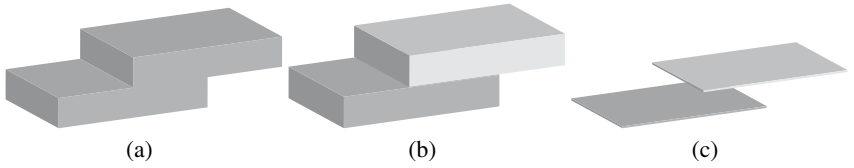


Fig. 4. Foreground cells of the grid (a), sets of forground cells separated for each original plate (b), and result medial surfaces (c)

4.1 Prerequisites

We would like to separate plate structures adjacent each other and reconstruct medial cells of original machine parts without contact. However, we cannot distinguish between one thick plate and welded two plates only from CT data in general. Because the CT principles cause the problem, we add the following constraints to solve it. Summarizing the principles simply, performance of X-ray radiation sources and detectors affect signal-to-noise ratio.

Machine parts (e.g. we plan to apply our method to process CT data of motorcars) are frequently made by welding of molded plates. Therefore, we can assume that thickness distribution of a plate is constant before welding process and the distribution after the process change little by bending or stretching. The assumption is reasonable when we attach importance to the industrial applicabilities.

Furthermore, we assume that whole of the grid space are comprised of plate structures or background. This is because we cannot harmonize the medial representation and so-called solid representation. For example, our method is not suitable for model creation from CT data of casting parts such as a car cylinder head.

We can obtain medial cells of the target space by applying the method mentioned in Section 3. We call them the first medial cells. Additionally, we can make use of thickness distribution by calculating the distance between an arbitrary cell and its nearest boundary points.

The standard deviation σ of thicknesses need to be known. Certain degree of variance is always observed because of CT's mechanical limits and quantization distortion, if we measure constant thick plates by CT. Therefore, we make a collection of calibration factors for CT scanners by measuring the plate which geometry is given.

4.2 The First Medial Cell Classification

We classify the first medial cells into three region types: *combined region*, *single region* and *transition region*. The combined region is a region in which two or more plates contact each other. Medial cells in the single region are influenced by boundary surfaces of the single plate. The transition region exists between the combined region and the single region. In this section, we propose an algorithm to judge which region type each medial cell is beglonging (Figure 5). Additionally, we use figures of the test piece which is consist of welded plates (Figure 6).

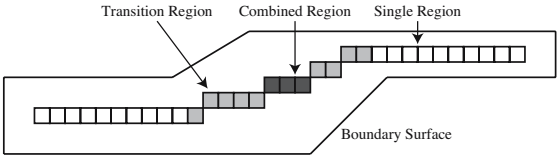


Fig. 5. We classify the first medial cells into three region types

Fig. 6. The left is a photogrpah of the simple test piece which is consist of welded plates. The right is its first medial cells.

At the start, we traverse with the first medial cells with the greedy algorithm and create clusters. The condition for a cluster c to merge a medial cell mc is defined by

$$|T_c - T_{mc}| < 3\sigma$$

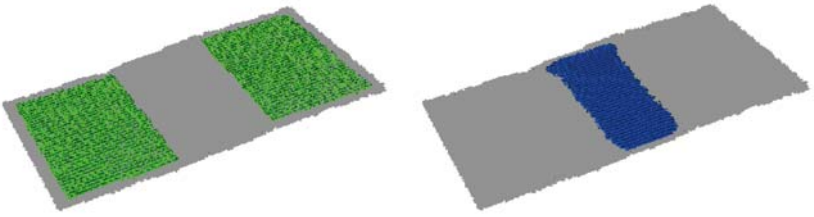
where T_c is the current average thickness of cluster c and T_{mc} is the thickness at mc. The merging process can extract a certain extent of the region, in which medial cells have almost same thicknesses. Additionally, we can obviously treat clusters as the single region, which have the smallest average thickness (Figure 7(a)).

And next, we find a cluster y in the vincinity of clusters $x_1, x_2, \ldots, x_n$ classified as the single regions. If cluster y satisfy 1, we classify cluster y into combined region as shown in Figure 7(b).

$$|T_y - \sum_{1}^{n} T_{x_i}| < 3\sigma \tag{1}$$

At this time, if unclassified clusters exist, we iterate the above process so that classify clusters into single region or combined region. The iteration is based on not strict theoritical background but heuristics. For example, the algorithm cannot properly judge stacked plates. However, we did not face the problem when applying to CT data of the real industrial parts.

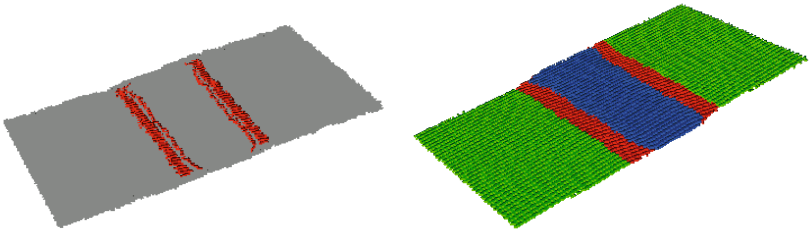
Finally, we treat a medial cell mc as transition region, which stands on between single region cluster x and combined region cluster y and satisfy condition $T_x < T_{mc} < T_y$ as shown in Figure 8(a).



(a) (b)

Fig. 7. Simple region clusters (a) and a combined region cluster (b) of the simple test piece

The classification method can classify roughly the first medial cells depending on thicknesses. However, the thicknesses is not very accurate because they are discretely calculated in the grid space. And also, there is a problem we cannot define a thickness at the edge of medials. Because of these problems, there remain not classified cells. To solve the problems, we propagate types of the first medial cells which have already classified so that all of the cells are finally classified into combined region, single region or transition region as shown in Figure 8(b).



(a) (b)

Fig. 8. Transion region clusters (a) and the final classification result (b) of the simple test piece

4.3 Foreground Separation and Mesh Generation

Note that we concentrate to introduce the case two plates contact each other in this section. However, if n plates are combined, we can separate a plate from $n-1$ plates and solve as the iterative problem.

First, we separate the combined region. We offsets the cells in the direction of the object surface by one half of the original thickness as shown in Figure 9(a).

The first medial cells belonging to the combined region have the nearest neighbor points information. That is, we have already computed the nearest points on the obverse side and reverse side of the surface. Therefore we can easily calculate offset vectors. The thicknesses of the original plates are led approximately from the average thicknesses of the single region.

And next, we connect the separated combined region and the single region as shown in Figure 9(b). At this point, because the transition regions have no geometrical contribution, we decimate them simply. We bridge the lacked spaces by use of linear interpo-

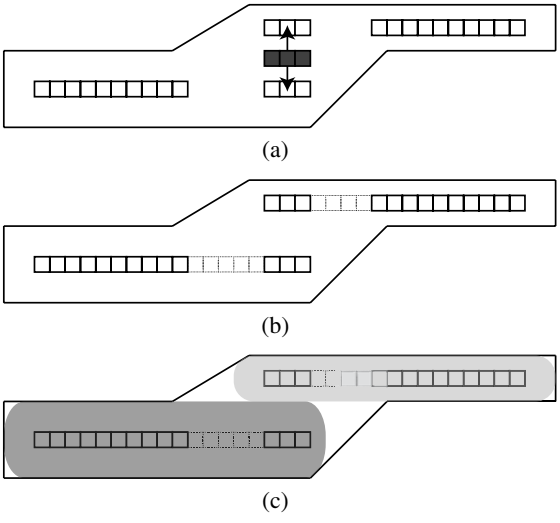


Fig. 9. We seprated the combined region and offset it in the direction of the object surface (a), connect the spartaed combined region and the single region (b), and inflate the pseudo mediall cells to separate the foreground cells

lation. This process is equivalent to match a discrete plane between an edge of separated combined regions and an edge of single regions.

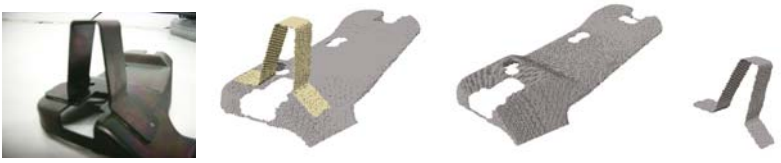
We offset the pseudo medial cells so that separate the forground cells. We use classical morphological operator to fatten the skelton. Herewith, we obtain sets of foreground cells corresponding to original plates.

Then, we apply the medial cell extraction method again to the sets of cells. We can obtain the separated medial cells for each plate. Now, the medials are represented by discrete form in this stage, so it is far difficult to put them into practical industrial use. Therefore, we use Local Marching Cubes [2] to extract triangular polygonal meshes from the medial cells.

We assume that a medial surface of the plate structure pass through the medial cells. If we can make a consistent sign assignment to the medial cells, the Marching Cubes will generate a medial surface. THe important concept here is the *Marching Cubeability* of the set of cells.

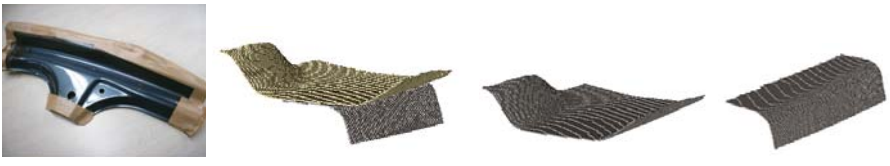
5 Result

We implemented the separated medial surface extraction algorithm on a standard PC equipped with AMD Athlon 64 X2 Dual 3800+ 2.0GHz and 2GB RAM. It took 5 minutes to extract the final medial surfaces for the CT data of an iron bracket and clamp ($270 \times 300 \times 150$) as shown in Figure 10(a). The figure also shows the result of application. As a result, we can successfully separated two welded parts. We also applied our method to the CT data of a car body shell (Figure 11). We can successfully separate the combined paltes and extract medial surfaces.



(a) (b) (c) (d)

Fig. 10. (a) is a photograph of an iron bracket and clamp. The bracket and the clamp are contact each other (b), and the separated bracket is (c) and the separated clamp is (d).



(a) (b) (c) (d)

Fig. 11. (a) is a photograph of an car body shell. Two alminum plates are combined by spot weldings. (b,c,d) show separated medial surfaces.

6 Conclusion and Future Works

We presented a new method to extract medial surfaces from CT data of machine parts. First, we extract the medial surfaces nad separated it in three dimensional grid space. And then, we create continuous medial surfaces as triangular polygonal meshes from the cells. We extend a method that we have already proposed to create meshes. Final outputs of the process are these medial surfaces represented by polygonal meshes.

Some of the problems to be resolved in future work included the issue of the accuracy of the separated medial surfaces (In the other area, we could easily estimate the accuracy). The accuracy of the surface has not been analyzed, and guaranteed accuracy is required for practical industrial applications. We also need to ease the constraints of our algorithms, and we are planning to solve the problem.

References

1. Sheehy, D.J., Armstrong, C.G., Robinson, D.J.: Computing the medial surface of a solid from a domain delaunay triangulation. In: Proceedings of the third ACM symposium on Solid modeling and applications, ACM Press (1995) 201–212
2. Fujimori, T., Suzuki, H., Kobayashi, Y., Kase, K.: Contouring medial surface of thin plate structure using local marching cubes. Journal of Computing and Information Science in Engineering (Short Paper) **5**(2) (2005) 111–115
3. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. In: Proceedings of the 14th annual conference on Computer graphics and interactive techniques, ACM Press (1987) 163–169

4. Lam, L., Lee, S.W., Suen, C.Y.: Thinning methodologies–a comprehensive survey. IEEE Transactions on Pattern Analysis and Machine Intelligence **14**(9) (1992) 869–885
5. Pudney, C.: Distance-ordered homotopic thinning: a skeletonization algorithm for 3d digital images. Comput. Vis. Image Underst. **72**(3) (1998) 404–413
6. Gagvani, N., Silver, D.: Parameter-controlled volume thinning. CVGIP: Graph. Models Image Process. **61**(3) (1999) 149–164
7. Malandain, G., Fernandez-Vidal, S.: Euclidean skeletons. IVC **16** (1998) 317–327
8. Prohaska, S., Hege, H.C.: Fast visualization of plane-like structures in voxel data. In: Proceedings of the conference on Visualization '02, IEEE Computer Society (2002)
9. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. In: Proceedings of the 14th annual conference on Computer graphics and interactive techniques, ACM Press (1987) 163–169
10. Hege, H.C., Seebaß, M., Stalling, D., Zöckler, M.: A generalized marching cubes algorithm based on non-binary classifications. Technical Report SC-97-05, Zuse Institute Berlin (1997)
11. Gibson, S.F.F.: Using distance maps for accurate surface representation in sampled volumes. In: Proceedings of the 1998 IEEE symposium on Volume visualization, ACM Press (1998) 23–30
12. Kobbelt, L.P., Botsch, M., Schwanecke, U., Seidel, H.P.: Feature sensitive surface extraction from volume data. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, ACM Press (2001) 57–66
13. Ju, T., Losasso, F., Schaefer, S., Warren, J.: Dual contouring of hermite data. In: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, ACM Press (2002) 339–346
14. Hormann, K., Labsik, U., Meister, M., Greiner, G.: Hierarchical extraction of iso-surfaces with semi-regular meshes. In: Proceedings of the seventh ACM symposium on Solid modeling and applications, ACM Press (2002) 53–58
15. Esteve, J., Brunet, P., Vinacua, A.: Approximation of a variable density cloud of points by shrinking a discrete membrane. Technical Report LSI-02-75-R, Universitat Politècnica de Catalunya (2002)
16. Itoh, T., Yamaguchi, Y., Koyamada, K.: Volume thinning for automatic isosurface propagation. In: Proceedings of the 7th conference on Visualization '96, IEEE Computer Society Press (1996) 303–310
17. Lachaud, J.O., Montanvert, A.: Continuous analogs of digital boundaries: A topological approach to iso-surfaces. Graphical Models and Image Processing **62** (2000) 129–164
18. Lohmann, G., ed.: Volumetric Image Analysis. John Wiley & Sons, Ltd. (1998)

Two-Dimensional Selections for Feature-Based Data Exchange

Ari Rappoport[1], Steven Spitz[2], and Michal Etzion[3]

[1] The Hebrew University
http://www.cs.huji.ac.il/~arir
[2] Proficiency Inc.
[3] Proficiency Ltd.

Abstract. Proper treatment of selections is essential in parametric feature-based design. Data exchange is one of the most important operators in any design paradigm. In this paper we address two-dimensional selections (faces and surfaces) in feature-based data exchange (FBDE). We define the problem formally and present algorithms to address it, in general and in various cases in which feature rewrites are necessary. The general algorithm operates at a geometric level and does not require solving the persistent naming problem, which is required for selection support inside a single CAD system. All algorithms are applicable to the Universal Product Representation (UPR) FBDE architecture, and the general algorithm is also applicable to the STEP parametrics specification.

1 Introduction

Parametric feature-based design is the dominant modeling paradigm in modern CAD systems [Hoffmann93, Shah95]. Data exchange is in general a problem of substantial theoretical and practical value. Consequently, feature-based data exchange (FBDE) is an attractive issue to address in Solid Modeling. Moreover, FBDE is a problem that is difficult and challenging technically, from both an architectural and an algorithmic point of view.

In this paper we present an algorithm that addresses one of the fundamental issues in FBDE: selections of two-dimensional entities. The problem of representing selections that serve as feature arguments inside a single CAD system is usually called the 'persistent naming' problem, and is known to be challenging [Kripac97]. The problem of representing selections in the context of data exchange is different, as will be discussed below.

In a previous paper [Rappoport05], we introduced the problem of handling selections in FBDE, and gave a detailed solution to the case where the selected entities are one-dimensional (edges and curves). In this paper we complete that work by presenting a solution to the case where the selected entities are two-dimensional (faces and surfaces). The 2-D algorithm is different from the 1-D one, and utilizes the latter where possible.

Our algorithm operates in the context of the Universal Product Representation (UPR) FBDE architecture [Rappoport03, Spitz04]. Nonetheless, the algorithm is applicable to a wider class of architectures [Mun03], including the extension of STEP to support FBDE [Pratt04]. Although the selections problem was not explicitly recognized by the

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 325–342, 2006.
© Springer-Verlag Berlin Heidelberg 2006

STEP effort, it should not be too difficult to integrate our general algorithm into a STEP implementation.

As far as we know, the present paper is the first one that identifies two-dimensional selections for FBDE as a non-trivial problem and offers a solution. The STEP specification only specifies the file format and does not recognize the problem explicitly, similarly to other related academic efforts such as the EREP project [Hoffmann93], while [Mun05] presents a method that helps users address some selections manually.

Sections 2 and 3 discuss 2-D selections, in feature-based design and in the context of feature-based DE respectively. Section 4 reviews the UPR architecture. Section 5 describes the problem formally, and Section 6 presents the first phase of the algorithm, the computation of a selection cover. The second phase, dealing with the more complex case of rewrites, is described in Section 7. Section 8 discusses our implementation, and we conclude with a discussion. We have made an effort to make the paper stand on its own; specifically, it can be read without reading the previous 1-D selections paper. Some of the presentation of the previous paper, mostly in Sections 4 and 5, is thus repeated here.

2 2-D Selections in Parametric Feature-Based Design

In parametric feature-based design, the model is represented by a directed a-cyclic graph of operations called features. Most features either create new geometry or modify a part's existing geometry (some features only insert or modify meta-data and other attributes). The graph can be 'evaluated' after each feature, generating an object that is represented using a boundary representation (Brep). The Brep contains two components: a graph (topology) of vertices, edges, faces and shells (*Brep entities*) and their interconnectivities, and concrete geometry corresponding to each of these entities.

During interactive design, the user defines new features or edits existing features, and sees a 3-D graphical view of the current Brep on the screen. However, the Brep is not used only for viewing; it is also used for defining the arguments of some of the subsequent features. This is one of the major differences between the parametric feature-based design paradigm and classic Constructive Solid Geometry. Enabling such argument selection constitutes a primary constraint on the nature of Brep representations, as discussed in [Rappoport96].

Every feature has arguments that define its semantics. Brep entities serve as feature arguments in most of the useful and commonly used features. This is done by letting the user select a set of Brep entities on the 3-D view and define them as arguments to the present feature. There may be several different such arguments for a single feature.

Brep entities can be 0-D (vertices, points), 1-D (edges, curves, loops), 2-D (faces, surfaces) or 3-D (shells). The interesting cases in which those entities serve as feature arguments are the 1-D and 2-D cases. In [Rappoport05] we have given many examples for the 1-D case, perhaps the most central of which is the edge round (or fillet) feature.

Here we focus on the 2-D case. In general, there are two types of 2-D selections: selections whose goal is to select a *surface*, and selections whose goal is to select a *face* (or a set of faces). The difference is that in the former case the feature only needs the carrier surface, while in the latter case a bounded subset of a carrier surface is needed, a subset which can be represented as the union of a set of bounded faces.

Following are some central examples for features whose arguments include user selected two-dimensional Brep entities:

- Extrude Until Face: the Extrude feature is the most common geometry defining feature. It takes a parametric 2-D sketch and extrudes it to create a 3-D shape. Extrude comes in many different variations. One useful variation is when it is defined to be *Until Face*, where the created 3-D shape is trimmed when it is blocked by a selected existing Brep face or by the carrier surface of that face. Other variants include *Until Next* and *Until Last*, which terminate at the next or last face (or carrier surface) encountered, respectively.
- Draft: this is a complex feature mostly used for plastic injection molding. A set of Brep faces (or subsets of Brep faces) is skewed at a specified angle, modifying prior geometry in a global manner. The user needs to select the set of faces to be skewed, and to optionally sketch curves on those faces to define face subsets.
- Shell: in this feature, the user selects a set of faces to be removed from the current Brep. The remaining faces are 'thickened' and then trimmed in order to create a valid 3-D solid.
- Offset: this feature creates a face (or a surface) defined as the offset (at a specified distance) of a selected face (or of a set of faces, or of a carrier surface.)
- Face Round (or Fillet, or Blend): this feature is very similar to the common Edge Round. It creates smooth surfaces between selected faces (or sets of selected faces), and removes from the Brep everything 'covered' or 'cut off' by those new surfaces such that the end result is a 3-D solid. Face Round is in some sense more powerful than Edge Round, because it can blend faces that are far from each other in the Brep topology.
- Sketch On Face: the user selects a face whose carrier surface is a plane embedded in 3-D space, and draws a parametric 2-D sketch on this plane. Usually, the sketch serves as one of the arguments of an Extrude feature.

Some CAD systems allow the user to select only a subset of selection required as the feature argument, completing the rest automatically. The main automatic completion method is to recursively add to the entities explicitly selected by the user all entities that are adjacent with smooth connectivity (e.g., G1 continuity).

In the following sections we will give examples for all of the above features except Sketch On Face, which for the problem in this paper is conceptually similar to Until Face.

3 Potential Problems with 2-D Selections in FBDE

CAD systems must represent selected Brep entities in a way that generalizes over the current geometry, because at any point in time the user may modify the parameters of any feature. When this happens, the system plays the feature history again, a process which usually results in a different geometry. A purely static geometric representation of the selections would hence not be valid. This problem is known as the persistent naming problem.

To tackle this problem, CAD systems abstract away some of the properties of the selected entities. Usually, usage is made of properties that are independent of the numerical values in the model and are functions of more intrinsic properties, such as Brep topology, identities of the features (or carrier surfaces) creating Brep entities, qualitative geometric properties (such as convexity), etc. In other words, they represent selections using generic names that are persistent under parameter changes (hence the term). For a general solution to generic naming of Brep entities see [Rappoport97], and for solutions in the context of CAD systems see e.g. [Kripac97].

When performing feature-based data exchange there are in principle no immediate parametric changes. On the contrary, the main goal is to construct a model in the target system that is as similar as possible to the model in the source system. Hence a full persistent naming solution is perhaps not needed. It is possible, as we do in this paper,

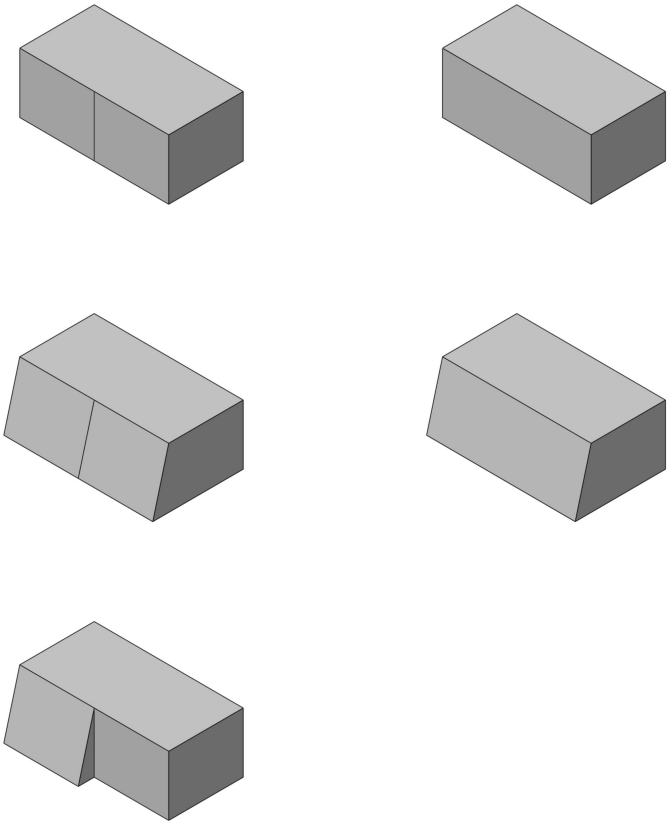


Fig. 1. Draft. Top: situation in CAD systems A (left) and B (right) before the definition of a Draft. Note that in system A there is a Brep edge that does not exist in system B. Middle: one possibility for the situation after the draft. Bottom: another possibility for the situation after the draft. The small drafted face in A does not correspond to any face in B, so the draft cannot be directly defined in B. We address this using rewrites.

to use representational methods that are different in character and are closer to Brep geometry.

In order to motivate the algorithm of this paper, we first discuss a naive solution and why it is not adequate. Consider the following algorithm for supporting selections for data exchange from system U to system W: (i) identify the selected face f in system U; (ii) locate the face f in system W; (iii) select it in W and use the selection as the argument of the desired feature (the one that the data exchange system defines presently.)

As it is phrased, this algorithm is wrong. Consider Figure 1, which shows a Draft feature. The top row shows the situation before defining the draft, in two different CAD systems. Note that the face drafted on the system on the right (B) does not exist as a single face on the system on the left (A). When exchanging from B to A, Step (ii) above would thus return a 'failure' answer, although it is clearly possible to complete the draft on the left by selecting two faces, as shown in the middle row (left).

The problem stems from the fact that the Breps on the two sides, although geometrically equivalent (as point sets), are topologically different. This situation may arise due to various reasons. For example, suppose that the box has been created by extruding the bottom face upwards. If the 2-D sketch defining the face contains the 'extra' vertex, the extrusion creates a corresponding vertical edge. The vertex may be there for the use of other features, for manufacturing information, for assembly operations, etc.

There is an even graver potential problem, when exchanging a draft from system A to system B. Suppose only one of the two smaller faces in A was to be drafted (the desired result is shown in the bottom row.) This face cannot be expressed as a union of B faces, because it is a proper subset of a B face. The result is thus impossible to exchange directly to system B, because it is impossible to define the desired selection. In Section 7 we will discuss our handling of such cases.

An example containing a cylinder and the Offset feature is shown in Figure 2. CAD systems differ in the way they represent cylinders in Breps. A cylinder can be represented in several ways: (i) using two circular edges and three faces, one cylindrical face and two disks (on the right); (ii) using an additional edge to split the cylindrical face

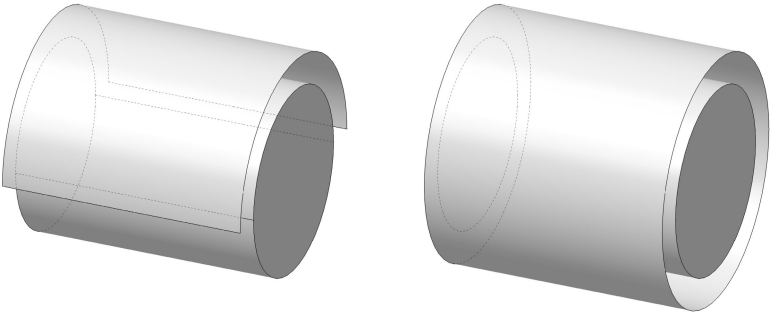


Fig. 2. Offset. A cylinder can be represented using one or two cylindrical faces (right and left respectively.) If the user selects a single face as the argument of the offset feature, the results are different in the two cases (assuming no automatic propagation of selections is done by the system on the left.)

(usually, this is done in order to establish a well-defined parameterization of the face or to avoid a single face having two bounding unconnected loops); and (iii) using two edges to split the cylinder (on the left. Usually, this is done in order to avoid the same edge appearing twice in the same loop.) Figure 2, left, shows the latter situation, where the user selected the top half-cylinder face and the offset is done on that face alone. Figure 2, right, shows the first situation, where the offset is done on the whole cylindrical face. Again, exchanging from right to left requires selection of two faces rather than a single one, and exchanging from left to right necessitates a wider rewrite.

4 Review of the UPR FBDE Solution

The solution presented in this paper for 2-D selections is applicable to a wide variety of FBDE architectures. However, our specific formal problem definition is done in the context of our UPR architecture. Therefore, in this section we describe it briefly, emphasizing those aspects that are relevant to this paper.

The UPR is a star architecture, like most other data exchange architectures. Export and import modules are responsible for communication with the source and target CAD systems respectively. Starness is not a requirement for our selection algorithms, which are applicable also to other FBDE architectures, for example direct source to target translations.

The UPR differs from all other data exchange approaches in that it recognizes that CAD systems differ from each other, both in terms of functional semantics and in terms of implementation of theoretically equivalent operations. Due to market forces and the richness of the feature-based design paradigm, it is not realistic to dictate an ultimate set of features that are supported by a CAD system. Each CAD system provides features and sub-features that are not directly provided by other systems. In addition, due to the complexity of the semantics of certain features, a feature's implementation in one CAD system might result in geometry that is somewhat different from the geometry that a different system generates from the 'same' feature. That is, the feature is the same at a certain level of abstraction (usually, overall function as perceived by the user), but different at a detailed geometric level of abstraction. Finally, we should expect the geometry generated by features to be different due to the fact that different systems utilize different tolerances for different operations, a phenomenon that plagues ordinary geometric data exchange [Qi04].

The UPR is thus explicitly designed to handle the following two cases: (i) a data item explicitly supported by one system and not by another, and (ii) incompatibilities between systems that can be identified only during run-time due to lack of formal specification of implementational differences.

The UPR representation of a feature makes use of two central concepts to address the above: rewrites and verifications. Each feature has an associated set of rewrites, which intuitively are different ways to import that feature into a system that had not succeeded importing it using other means. Instead of dictating a certain fixed representation, the UPR allows an unlimited number of representations that attempt to simulate the semantics of a feature. Rewrites are applied at decreasing levels of abstraction, starting from fully parametric and ending at fully geometric. In [Spitz04] we have described

an algorithm for implementing a 'Geometry Per Feature' rewrite, which replaces any feature by a piece of geometry identical to the feature's geometric effect. The Geometry Per Feature rewrite is the first geometric rewrite attempted when all parametric rewrites fail. There are additional geometric rewrites, e.g., replacing a *set* of features by their geometric effect.

In addition to rewrites, each feature stores a set of verification data, used to dynamically identify whether feature import has succeeded or not. Verification data is of three main types: volume and surface area, various higher order moments of inertia of the solid, and a cloud of points lying on the boundary of the solid or on the faces generated (or removed) by the feature alone. Verification data is computed at the source CAD system after invocation of the feature, then stored at the UPR. Verification data is computed at the target system during import and compared to the data stored at the UPR, to verify success of feature import.

When verification fails the system attempts a graceful recovery, e.g. by applying a different rewrite. Note that a source and target system feature may generate geometries that are slightly different but would still be regarded as equivalent for the sake of feature-based data exchange. For example, fillets are not necessarily required to produce the exact same geometry. Hence care must be taken when interpreting verification results.

All of those mechanisms are taken care of at the global architectural level and do not form a part of the selection algorithms, which operate at the feature internal level.

5 Assumptions

In this section we state and explain our assumptions on the context in which the selection problem occurs.

Our goal is to provide support for two-dimensional user selections that serve as feature arguments in feature-based data exchange systems. We assume the following assumptions, which do not pose any restriction on our algorithms because all parametric CAD systems obey them: (i) The FBDE system defines features one after the other in the target CAD system; (ii) When selections are specified the target CAD system holds a boundary representation (Brep) of the current model; (iii) The Brep conforms to the theoretical definition of a Brep, as embodied in the concept of a selective geometric complex (SGC) [Rossignac88]; (iv) The Brep is available for inspection and usage by the selections support algorithm. Selections can thus be specified to the target system in terms of identifiers of current Brep entities.

The model defined by the features up to F (the feature containing the selection that needs to be defined) is thus assumed to be correct in the sense that the pointset at the target system is identical up to a geometric tolerance to that at the source system. The geometric tolerance issue, resulting from floating point computation and inexact algorithms, is one of the major problems in solid modeling and here we do not attempt to deal with it completely.

We also assume here that identity of selection geometry is accessible through the API of the source CAD system. Virtually all modern CAD systems provide such access to Breps. They usually do not provide an interface to the persistent names, but they do provide Brep interfaces to selections.

We do not assume that feature histories or Brep topologies are identical at the source and target systems. Feature histories may be different owing to rewrites or different feature repertoires of the source and target systems. Brep topology (the boundary graph that represents the vertices, edges and faces and their interconnectivity relationships) varies from system to system, as discussed in Section 3.

For simplicity of exposition we assume in this paper that the Brep scheme in both source and target systems is 2-manifold. Extension to non-manifold geometry is beyond our scope here. Regarding terminology, 'face' is used for a single bounded connectivity component of a two-dimensional part of the boundary of the solid, lying on a single carrier surface.

6 The Selection Cover Algorithm

In this section we present the first phase of our 2-D selection algorithm. In this phase we export selection information from the source system, find a target model cover of the source selection, and classify its status with respect to further actions needed to complete a proper selection. We describe the general scheme (6.1), the export process and what data is stored in the UPR (6.2), and the import selection cover algorithm (6.3).

6.1 The General Scheme

We had already noted above that a generic selection representation in the style of persistent naming solutions is not essential in our case, because the geometries on both sides can be assumed to be identical up to a tolerance. It is thus of conceptual elegance to try to utilize only that static geometry. The general idea is then:

- Export the geometry of the selection into the UPR.
- When selections need to be defined during import, select a subset of the current Brep (in the target system) that covers as tightly as possible the selections stored in the UPR.
- If there are faces in the UPR selection that cannot be exactly covered by faces in the current Brep, create new faces or split existing faces in the current model so that an exact cover is obtained, or use other feature rewrites to preserve feature semantics as much as possible.

Section 3 showed that the second step is not trivial. It is not the case that every selected face in the source system corresponds to exactly one face in the target system. We cannot make assumptions regarding the topology of the two Breps, only about their geometry.

6.2 Export to UPR

The goal of the export process is to make sure that all data needed during the import phase is available in the UPR. Since the import algorithm only needs the geometric pointset defining the selection, it is straightforward to represent it in the UPR. Due to the philosophy behind the UPR, which is designed to support the *union* of object types generated by CAD systems, we use the same surface types used in the source

CAD system. In principle there should not be any degradation in the quality of the representation, and specifically no loss of tolerance. We refer to the selection as stored in the UPR as the 'source selection'.

In some situations we may export relevant symbolic model data along with the selection geometry. For example, when the model contains several parametric history graphs, the ID of the graph containing each part of the selection can be stored with the selection geometry, in order to make it easier to locate the target image of that body during import or in order to identify the correct body if several bodies overlap geometrically. The IDs of the owning features of each selection part can be used in the same manner (in the terminology of some CAD systems, an owning feature of a Brep entity is the first feature that had caused the entity to be added to the model's Brep). These kinds of techniques are obvious and we will not elaborate on them further in this paper.

6.3 Import: Selection Cover

Recall that the Brep generated by the features preceding the feature whose arguments are our selections is assumed to be present in the target system. What we seek in phase 1 of the import algorithm is a set of entities belonging to that Brep that are an exact cover of the selection geometric pointset. If there is such a set, we are done. We may need to compute the connectivity structure of the entities to be selected as well as the entity set, in case the target system does not allow an arbitrary selection order. If there is no exact cover, we need to know this and provide as much information as possible to phase 2 of the import process, the rewrite phase. For simplicity the algorithm is described for a single connectivity component of the selection set. We assume a preprocessing stage in which all such components have been identified. The algorithm should be invoked on those iteratively. The algorithm targets the case where the selection is comprised of a face or a set of faces; selection of a carrier surface is easily implemented using a point and the normal at the point.

There are many potential methods to compute an exact cover. Below we describe a method that relies on the power of the 1-D selection algorithm from [Rappoport05]. That algorithm computes a cover for each source selection edge separately, using point projection for an initial mapping plus a recursive search based on edge overlap tests. It is possible that the cover computed for an edge is not an exact cover, but the algorithm ensures that the union of the covers of all selection edges is an exact cover of the whole 1-D selection set.

The 2-D selection cover algorithm (Figure 3) is different from the 1-D algorithm in that it deals with all selection faces simultaneously, not separately. We start by invoking the 1-D selection algorithm for the 1-D boundary (denoted by c) of the 2-D selection set. The 1-D boundary c is always well defined; it could be empty when the selection set contains all of the boundary of the solid. The reader may be surprised to learn that this case does happen in practice, when users use the Copy Faces feature in order to copy the whole connectivity component of a solid (this is not good design practice when the CAD system provides a Copy Body feature, of course.)

If the 1-D selection algorithm has succeeded in finding an exact cover c' at the target system, all we have to do is collect the faces that its bounds. We can do that by finding an arbitrary point strictly inside the selection, locating a face f' on which it lies in the

```
Import: Selection Cover Algorithm
  Denote by c the 1-D boundary of the whole source selection set.
  Use the 1-D selection algorithm in order to identify c in the target
    system (call it c').
  If c was covered exactly by c'
    Find in the source system a single point p strictly inside the selection.
    In the target system, locate a face f' containing p.
    Recursively tag adjacent faces starting from f' and ending when
      reaching an edge of c'.
    Return successfully.
    // note: the above works even when c is empty.
  Else
    // c contains points that are not in c'
    // (happens when there are missing edges in the target system model)
    // or c' contains points that are not in c
    // (happens when the 1-D selection algorithm finds a cover that's too large)
    // or both
    compute c-c' and give it to a rewrite algorithm.
    // these are edges we want to insert explicitly into the model, if we can.
```

Fig. 3. The selection cover phase in the import part of the 2-D selections algorithm

target system (there may be more than one such face, but it doesn't matter), and performing a recursive search from f' that ends when reaching the 1-D selection boundary c'. Alternatively, we can start the search from any edge in c' in a direction determined by orientation considerations to be in the selection (we would need to synchronize orientations to do that reliably.) Note that this method works even if the 1-D boundary c is empty – it would simply select all faces in the connectivity component of the solid.

Note that the halting criterion is different from that used in the 1-D algorithm, where we halted when reaching an edge that does not overlap the selection pointset. The 2-D criterion is more efficient, because it does not require expensive face overlap tests. It utilizes the power of the 1-D selection algorithm. We could not have used a similar condition in the 1-D algorithm ('halt the edge propagation when reaching vertices that constitute the selection boundary'), because edge propagation can reach a specified vertex in many different and complex paths across the 2-D boundary of a 3-D 2-manifold solid.

If the 1-D selection algorithm has not succeeded in finding an exact cover, it says so and marks source edges that were exactly covered, source edges that were partially covered, source edges that were not covered at all, and target edges that are not fully covered by the source selection (note that here the cover is in the opposite direction.)

Source edges that are partially covered or not covered correspond to edges that are 'missing' (completely or partially) from the target system. Examples are the vertical edge of the drafted face in Figure 1, left, and the straight edge on the cylindrical face in Figure 2, left.

Target edges that are not fully covered by the source selection exist when a target face covers 'too much' of a source face. Examples are the horizontal edges of the drafted face in Figure 1, right, and the circular edges (at the top and bottom) of the cylinder in Figure 2, right.

Any connected two-dimensional subset of the boundary of a 3-D solid is completely determined by its one-dimensional boundary and a single point in it. As a result, if the 1-D selection algorithm is correct (proven in [Rappoport05]) then the algorithm above is correct as well.

7 Rewrites

The second and more complex phase of the import 2-D selection algorithm is when rewrites are necessary because an exact selection cover could not be found. In this section we classify the possible types of rewrites and give examples and algorithms for most of them.

7.1 Face/Carrier Rewrite

The simplest type of rewrite is when the feature semantics really needs only the carrier surface of a face, but for some reason it is not possible to define the carrier in exactly the same manner in both source and target systems.

As an example, take the 'Mirror By Plane' feature, which requires the selection of a datum plane (introduced to the model by some earlier operation), and unites the current model with its mirror across the plane. Suppose that in the source system there is no such feature but there is a 'Mirror By Face' feature, having exactly the same semantics but which requires the user to select a face, not a plane, and uses the face's plane. Suppose that the target system has only Mirror By Plane and does not have Mirror By Face. The selection algorithm from the previous section will succeed finding the face, but the feature import will fail as a whole because the target system doesn't have the corresponding feature.

A simple rewrite of the source Mirror By Face to a target Mirror By Plane will solve the problem. The rewrite should modify the name of the feature and also convert the selection of a plane to the selection of a face. The face is specified by 'any planar face lying on a given plane'. Implementing this selection is easy.

Note that in this case the selection algorithm from Section 6 is not entered at all. This rewrite is done at a higher level in the system, and is a feature rewrite, not a selection-only rewrite.

7.2 Face/Edge Round Rewrite

A situation that is somewhat similar to the previous one is with the Round (Fillet) features. In Figure 4, we see two CAD systems (left, right), before (top) and after (bottom) a Face Round feature (in this case it creates a fillet.) On the right, the fillet is around the full perimeter of the vertical cylinder, because the cylinder is represented as a single cylindrical face. On the left, the cylinder is broken into two faces, and the fillet does not go beyond the thin box.

When exchanging from left to right, we get a wrong result. In this case rewriting the Face Round feature to an ordinary (edge based) Round feature solves the problem. The rewrite should identify which edges between the two selected faces were covered (replaced) by the fillet, and provide those edges to the Round feature on the right.

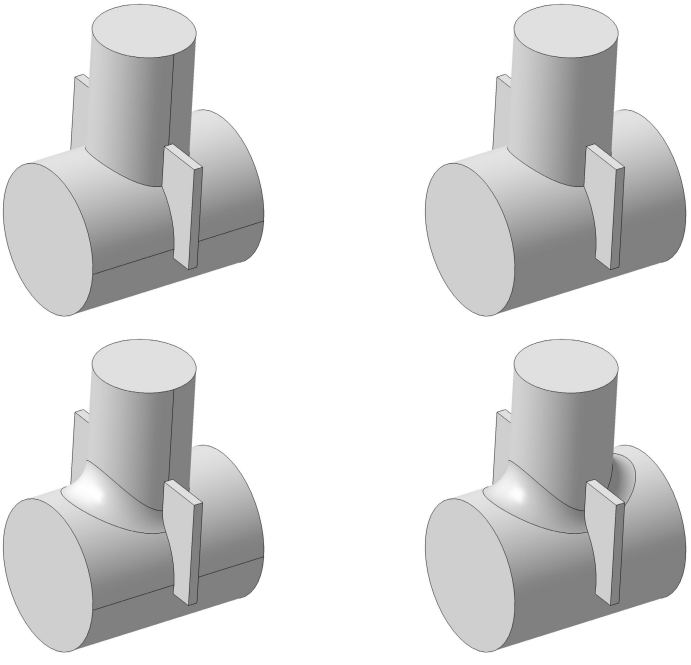


Fig. 4. Face Round. Two CAD systems (left, right), before (top) and after (bottom) of the Face Round feature.

Note that this solution is not always possible, because it is in principle possible that there are no intersection edges between the two faces of a Face Round feature; the power of this feature is in defining long distance fillets and rounds (imagine the intersection edge on the left as a small face completely obliterated by the fillet.) If this is the case there is no simple symbolic solution, and the situation can be addressed by the rewrite described next (Edge or Face Insertion), or by the general Geometry Per Feature rewrite [Spitz04].

7.3 Edge or Face Insertion Rewrite

The most common case in which the algorithm of Section 6 fails is when there are 'missing' edges in the target system. A direct rewrite for tackling this problem is to explicitly insert those edges into the model. Some CAD systems support a 'split face by adding a new edge' feature. In this case this feature is used for inserting the required edges.

This is a nice example of the power of the rewrite concept in the UPR architecture – as we see here, a rewrite can insert a wholly new feature if it is needed in order to enable the parametric import of another feature into the target system.

Unfortunately, not all CAD systems provide to the user a Split Face feature. It is still possible that this operations is available to CAD extension programmers through the CAD system's API, and can be added internally to the feature graph. In this case we proceed as before.

When the above is not possible, in many systems it can still be emulated. Many systems contain a Patch feature whose arguments are a solid Brep B and an open surface sheet S having a material side and boundary edges that are assumed to lie on the boundary of the solid B. The result of the feature is to glue S to the boundary of B and discard that part of B's boundary that lies on the non-material side of S. We had used the Patch feature in our algorithm for solving the Geometry Per Feature (GPF) problem [Spitz04].

To emulate Split Face using Patch, we first prepare surface sheets that coincide with part of the target face and whose boundary edges include the desired selection edges. We compute these sheets by traversing the edges returned by the algorithm in Figure 3, and piecing them together to form connectivity components (there may be several connectivity components, and each is patched into the model using a different Patch feature.) Once we have those edges, we intersect them with the carrier surface of the target system face, creating trimming curves that specify the sheet to be patched in a well defined manner. Equivalently, we can simply invoke a face-face intersection algorithm [Patrikalakis02] between the source and target faces. However, care should be taken in the implementation of this algorithm, because the faces are known to overlap substantially, possibly resulting in numerical problems for the intersection algorithm.

Having computed the desired sheets to be patched, the material side for each is specified to be identical to that of the face. We then use each of these sheets as the argument of a Patch feature. This process usually results in the desired selection edges being added to the current Brep.

Figure 5 shows an example (the full solid is shown at the top, and cross sections in the middle and bottom.) A circle is sketched on the top planar face, and an Extrude Cut Until Face feature is performed. The face that serves as the 'until face' is a cylindrical face. However, when the cylinder is represented using only a single face, it is not clear whether the cut should stop as in the middle row or as in the last row. Initially it seems that the latter is wrong, but the reader should note that the intersection of the initial box with the cylinder creates a loop in the top part of the cylinder, so the Extrude Cut passes through that loop. The problem occurs whenever the cylinder (or any other non-planar surface) is represented using a single cylindrical face (on the left, the face contains a single edge, and on the right it does not contain any such edges.)

The problem can be solved by preparing the desired target face and patching it to the model. The desired target face is represented as the geometry of the place where the Extrude Cut is supposed to stop, which is the intersection of the Extrude Cut section and the desired part of the cylinder. That face, after the Patch, is used as the Until Face of the Extrude Cut in the target system.

Unfortunately this solution is not guaranteed to work in every CAD system, because it depends on the specific implementation of the patch feature and on the target system's general Brep policy. Many systems actively unify faces by removing edges separating them when those faces are considered to be the 'same' face, e.g., when they are coplanar. The system may refuse to do the Patch simply because it detects that no portion of the previous Brep is removed and hence the feature is considered redundant. The problems here are the classic problems of what is considered by CAD systems to be a valid face [Mäntylä88].

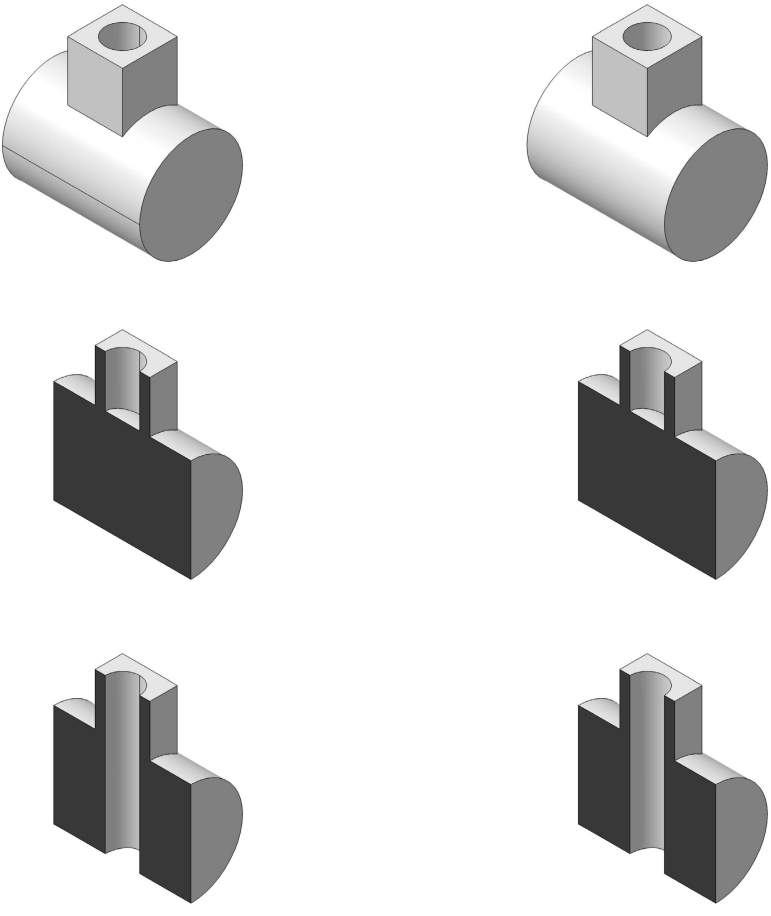


Fig. 5. Until Face. Top: solid view of an Extrude Cut of a circle sketched on the top face, defined to be Until Face of the cylinder. On the CAD system on the left, a cylinder is represented using three faces (two circles and one cylindrical face) and three edges (two circular edges and one straight edge). On the right, a cylinder is represented using a single cylindrical face, two circular faces, and two circular edges. Middle: cross section of one option for when the Extrude Cut stops. Bottom: cross section of another option for when the Extrude Cut stops. (On the left, cylinders are represented using an edge that cuts across the cylindrical face, and on the right there are no such edges.) An example such as this can occur with any non-planar surface, not necessarily a cylinder.

Figure 6 shows another example in which Patch solves the problem. However, in this case the problem stems from a totally different reason. Suppose that before the Shell, the object is defined as a box from which a slot is subtracted. In some systems, the associativity between the two top faces is retained to an extreme extent – the system remembers that they were originally the same face, and does not let users select each

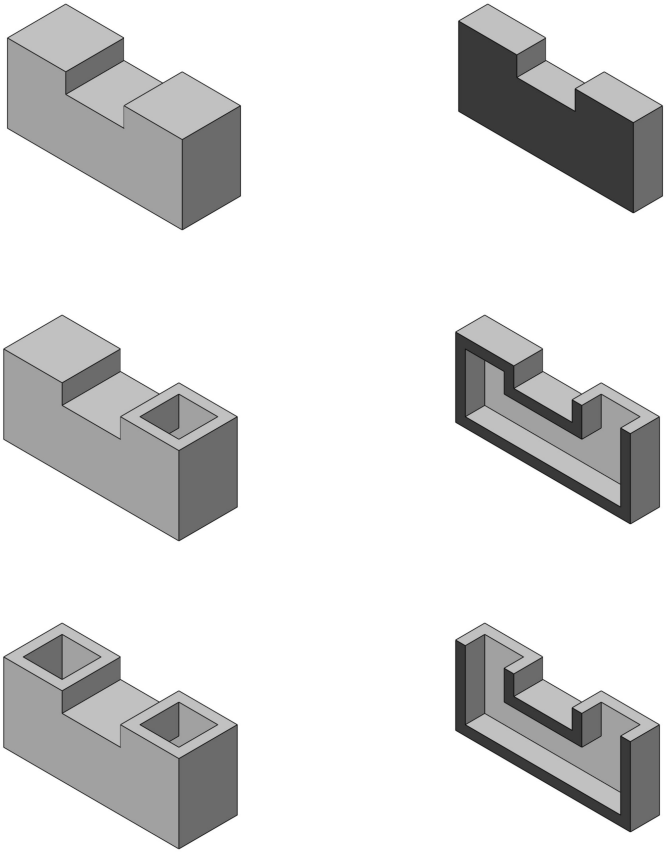


Fig. 6. Shell. Left: solid view. Right: cross section view. Top: before a Shell feature. Middle: the selection argument of the Shell feature is the top right face. Bottom: the selection argument of the Shell feature are both top faces. In some CAD systems, the middle option is not possible when the two top faces have originated from the same operation.

one individually. In such systems, exchanging the middle row is not possible, and the bottom row is the only possible outcome.

In such cases what we need to do is 'disconnect' the associativity between the problematic faces. In all of the systems that we handled, this can be done by patching into the model an orphan face identical to one of the top faces. Because the patched face can have an arbitrary geometry, the CAD system no longer assumes that the patched face lies on the same plane as the other top face, and breaks the associativity requirements between them. Now selecting only the patched face is possible, so Shell produces the desired results.

This example is of course related to the 'real' persistent naming problem, but implementing selections in terms of persistent naming (instead of in geometric terms like we do in this paper) would not solve the problem – in any case the two faces have the same

persistent name and this must be broken by applying an explicit operation that creates another name to one of the faces.

7.4 Orphan Face Insertion Rewrite

In some cases, it is sufficient to add the desired until face as an 'orphan' (or datum) face of the part, without actually patching it so that it becomes a part of the solid's Brep. For example, in Figure 5 this is possible. In Figure 6 it is not possible, because the argument of the Shell feature must be a Brep face of the solid.

Note that both options exhibit the same degree of loss of associativity, so the orphan option may be preferable due to performance and tolerancing considerations. Another consideration is which option is better from the point of view of user interaction in case people would need to look at the resulting model. In both options the feature graph at the target system does not look the same as that in the source system, and it is unclear which option is less confusing. We tend to feel that the orphan option is better because it uses fewer operations: it only uses one additional feature (inserting the orphan), while the Patch option uses two additional features (inserting the orphan and then patching it).

7.5 Reparameterization Rewrite

Consider again Figure 6. What if the target system does not allow patching a new face and calling it a different face? in other words, what if the associativity between the two top faces simply cannot be broken?

In this case we would need to rebuild the solid using different operations. For example, instead of defining a box and removing a slot from it, we could have sketched the vertical face as a 2-D sketch and use the Extrude feature in order to create the solid. In both cases the same pointset is created using two features, but the parameterization (feature history) is totally different.

General reparameterization of feature-based solids is an extremely difficult open problem in solid modeling. In our opinion it is one of the fundamental problems of the field, whose solution would throw light on many aspects of general shape modeling. It is certainly beyond the scope of the present paper. We have given the above example in order to complete the description of cases in which 2-D selections might need rewrites.

7.6 Summary

To summarize this section, in some cases rewrites can be completely symbolic. In most cases, our capability of creating selection faces explicitly where they did not previously exist is a function of the feature repertoire of the target system. When the target system does not allow a direct or emulated Split Face feature, it may still be possible to import the feature parametrically, by replacing it by a totally different feature combination that achieves the same geometric effect. This should be examined on an individual feature basis. The UPR architecture enables doing that through its support of feature rewrites, using a single feature or a number of features.

8 Implementation

The 2-D selection framework described in this paper has been implemented in the UPR architecture at Proficiency. The current UPR implementation supports the five high-end CAD systems in the market: Catia V4, Unigraphics, I-DEAS, ProEngineer, and Catia V5. Several versions and most of the design features of each system are supported. The data exchange process is controlled by a web server through a web-based user interface. The server locates export and import 'agents' over a network and distributes export and import jobs according to load parameters. The software is being used routinely in production. The number of real parts that have been successfully exchanged is in the hundreds of thousands.

In our implementation, the UPR file stores all relevant data, including the selection data, represented geometrically as described in this paper. Intermediate computations are done on the UPR data structure or using the software library of the target CAD system, according to implementational convenience.

We implemented the rewrites concepts of the first three types (face-carrier, adding edges, adding faces). Patching faces is done as part of the general Geometry Per Feature rewrite. Faces are added as separate bodies (orphans) when needed. Missing edges were added for import into I-DEAS. The fourth rewrite type was implemented for the very small number of cases that were encountered in practice.

9 Discussion

The issue of supporting two-dimensional selections is a crucial one in feature-based data exchange systems and algorithms. 2-D selections are used as feature arguments in important features such as Extrude, Draft, Offset, Shell and Face Round. Selections are what endows models with true associativity, and FBDE systems must support selections in a way that is as close as possible to the design intent as expressed in the source CAD system.

In this paper we have presented the first solution to this important problem, following our solution to the twin problem of 1-D selections [Rappoport05]. Our solution is applicable to a wide variety of FBDE architectures, among them the UPR and the STEP architectures, which are the only documented ones at present (note that the UPR has been fully implemented in practice, unlike STEP.) Our algorithms have been implemented in the UPR architecture, and are being used on a daily basis in real projects.

An interesting future work on our problem is to base the algorithm on the persistent names used by CAD systems rather than on geometric data alone. At present CAD systems do not expose those names, but the reliability and perhaps performance of selection exchange could be increased when utilizing persistent names.

A highly challenging topic arising from our problem (as well as from other problems) is that of reparameterization of parametric feature-based models. This topic is both very deep theoretically and has useful practical applications.

Acknowledgements. The Proficiency UPR implementation is a collective effort of the Proficiency development team, headed by Alex Tsechansky.

References

Hoffmann93 Hoffmann, C.M., Juan, R., Erep, an editable, high-level representation for geometric design and analysis. In: P. Wilson, M. Wozny, and M. Pratt, (Eds), Geometric and Product Modeling, pp. 129-164, North Holland, 1993.

Kripac97 Kripac, J., A mechanism for persistently naming topological entities in history-based parametric solid models. Computer-Aided Design, 29(2):113–122, 1997. Also: proceedings, Solid Modeling '95, pp. 21–30, ACM Press, 1995.

Mäntylä88 Mäntylä, M., An Introduction to Solid Modeling, Computer Science Press, Maryland, 1988.

Mun03 Mun, D., Han, S., Kim, J., Oh, Y., A set of standard modeling commands for the history-based parametric approach. Computer-Aided Design, 35:1171-1179, 2003.

Mun05 Mun, D., Han, S., Identification of topological entities and naming mapping for parametric CAD model exchange. Intl. J. of CAD/CAM, 5:69–82, Dec. 2005.

Patrikalakis02 Patrikalakis, N.M., Maekawa, T., Shape Interrogation for Computer-Aided Design and Manufacturing. Springer Verlag, 2002.

Pratt04 Pratt, M.J., Extension of ISO 10303, the STEP standard, for the exchange of procedural shape models. Proceedings, Shape Modeling International 2004 (SMI '04).

Qi04 Qi, J., Shapiro, V., Epsilon-solidity in geometric data translation, TR SAL 2002-4, Spatial Automation Laboratory, University of Wisconsin-Madison, June 2004.

Rappoport96 Rappoport, A., Breps as displayable-selectable models in interactive design of families of geometric objects. Geometric Modeling: Theory and Practice, Strasser, Klein, Rau, (Eds), Springer-Verlag, pp. 206-225, 1996.

Rappoport97 Rappoport, A., The Generic Geometric Complex (GGC): a modeling scheme for families of decomposed pointsets. Proceedings, Solid Modeling '97, May 1997, Atlanta, ACM Press.

Rappoport03 Rappoport, A., An architecture for universal CAD data exchange. Proceedings, Solid Modeling '03, June 2003, Seattle, Washington, ACM Press.

Rappoport05 Rappoport, A., Spitz, S., Etzion, M., One-dimensional selections for feature-based data exchange. Proceedings, Solid Modeling '05, June 2005, MIT, ACM Press.

Rossignac88 Rossignac, J.R., O'Connor, M.A., SGC: a dimension-independent model for pointsets with internal structures and incomplete boundaries. In: Wozny, M., Turner, J., Preiss, K. (eds), *Geometric Modeling for Product Engineering,* North-Holland, 1988. Proceedings of the 1988 IFIP/NSF Workshop on Geometric Modeling, Rensselaerville, NY, September 1988.

Shah95 Shah, J.J., Mantyla, M., Parametric and Feature-Based CAD/CAM, Wiley, 1995.

Spitz04 Spitz, S., Rappoport, A., Integrated feature-based and geometric CAD data exchange. Proceedings, Solid Modeling '04, June 2004, Genova, Italy, ACM Press.

Geometric Modeling of Nano Structures with Periodic Surfaces

Yan Wang

NSF Center for e-Design, University of Central Florida, Orlando, FL 32816-2993, U.S.A
wangyan@mail.ucf.edu

Abstract. Commonly used boundary-based solid and surface modeling methods in traditional computer aided design are not capable of constructing configurations with large numbers of particles or complex topology. In this paper, we propose a new geometric modeling scheme, periodic surface, for material design at atomic, molecular, and meso scales. At molecular scale, periodicity of the model allows thousands of particles to be built efficiently. At meso scale, inherent porosity of the model represents morphology of polymer and macromolecule naturally. Model construction and operation methods are developed to build crystal and molecular models based on periodic surfaces.

1 Introduction

To accelerate the development of nanotechnology, computer aided design tools are critical to solve the "lack of design" issue, meaning that no extensive and systematic design of nano systems is available compared to other traditional engineering domains such as mechanical mechanisms and electronic circuits. Computer aided nano design (CAND) is to enable engineering design, traditionally at the component and system levels, to be extended to nano scales. CAND helps to set functional objective, construct model, simulate and optimize design, and guide laboratory effort during physical property implementation.

Existing atomic and molecular simulation methods such as density function theory, molecular mechanics, Monte Carlo, and molecular dynamics and related tools enable scientists to visualize molecular structure and behavior, calculate properties such as electrical conductivity, elasticity, and thermodynamics, as well as to simulate reactions between molecules. This simulation-based approach saves time and resources of conducting real experiments to study material properties and interactions of molecules, which has started being used in drug and material design. Material properties can solely be calculated, and the results become the input of bulk-scale finite element analysis. Nevertheless, lack of efficient compound construct methods for large quantities of atoms and molecule becomes the bottle-neck of CAND. A good initial geometry is required to find optimal molecular configuration in simulation. Providing chemists good geometry conformation that is reasonably close to true minimum energy is highly desirable to save simulation time and lessen the risk of trapping into local minima.

Based on the observation that hyperbolic surfaces exist in nature ubiquitously, we propose a new geometric modeling scheme, periodic surface model to support

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 343 – 356, 2006.
© Springer-Verlag Berlin Heidelberg 2006

multi-scale modeling and simulation. This surface model is based on non-Euclidean geometry that allows for rapid model construction ranging from atoms to polymers. Model construction and operations are introduced to create compounds with thousands of elements or structures with complex topology. It takes a generic approach to explore symmetric tiling and packing of loci surfaces in 2D hyperbolic space and subsequently map into conventional 3D Euclidean space. 3D structures can also be built with foci searching based on surface envelops.

In the rest of the paper, Section 2 gives a background of molecular scale geometric modeling and minimal surface in nano structures. Section 3 introduces periodic surface modeling and associated operations. Section 4 describes the symmetric tiling methods to create mapping from 2D hyperbolic space to 3D Euclidean space, followed by the surface enclosure method of model creation in Section 5.

2 Background

2.1 Molecular Scale Geometric Modeling

At the molecular scale, atoms and particles are represented by geometry (coordinates of positions in Euclidean space) and topology (connection between atoms). Traditionally used visualization methods are space-filled, wireframe, stick, ball and stick, and ribbon models, as illustrated in Fig.1.

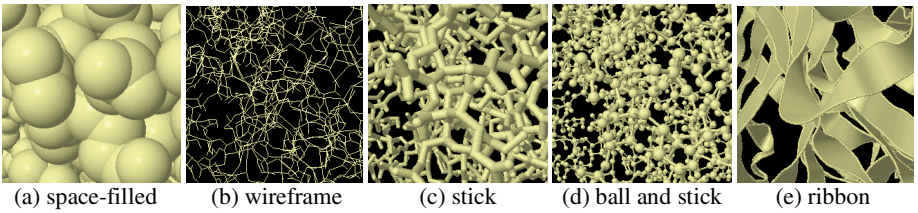


Fig. 1. Different types of visualization models for molecules

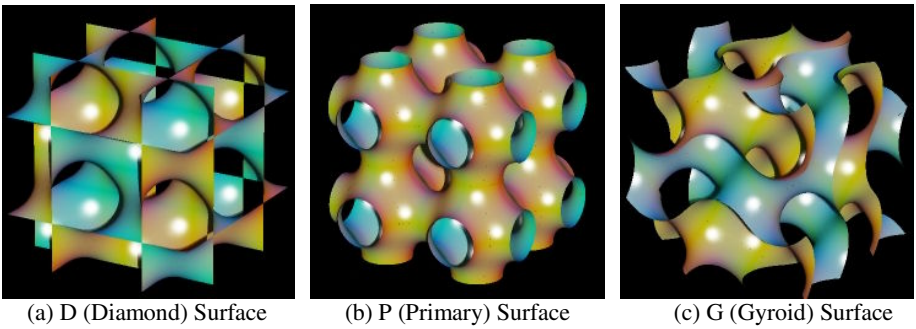
To reduce graphic processing time, there has been some research on molecular surface modeling [1]. Lee and Richards [2] first introduced solvent-accessible surface, the locus of a probe rolling over Van der Waals surface, to represent boundary of molecules. Connolly [3] presented an analytical method to calculate the surface. Recently, Bajaj et al. [4, 5] represent solvent accessible surface by NURBS (non-uniform rational B-spines). Carson [6] represent molecular surface with B-spline wavelet. These research efforts concentrate on boundary representation of molecules mainly for visualization, while model construction itself is not considered.

In order to support design and analysis from both material and engineering perspectives, computational models need to accommodate the need of model construction at multiple levels from atomic, to molecular, meso, and bulk scales. Traditional boundary representation of objects is not efficient for geometric modeling at nano scales. Creation of parametric model for multi-scale uses instead of simple "seamless zooming" is important.

Recently, hyperbolic surfaces attract attentions of physicists, chemists, and biologists. Hyperbolic geometry commonly exists in natural shapes and structures. The proposed periodic surface modeling is in the domain of hyperbolic geometry. Minimal surfaces are among the most studied hyperbolic surfaces.

2.2 Minimal Surface

The *mean curvature* of a surface at a point is defined as $H = (\kappa_1 + \kappa_2)/2$, where κ_1 and κ_2 are principle curvatures. Minimal surfaces are those with mean curvature of *zero*. If a minimal surface has space group symmetry, it is periodic in three independent directions. Triply Periodic Minimal Surfaces (TPMSs) are of special interests because they appear in a variety of structures such as silicates, bicontinuous mixtures, lyotropic colloids, detergent films, lipid bilayers, and biological formations. Three types of TPMSs are D, P, and G surfaces, as shown in Fig.2.



(a) D (Diamond) Surface (b) P (Primary) Surface (c) G (Gyroid) Surface

Fig. 2. Triply Periodic Minimal Surfaces [7]

2.3 Minimal Surface in Atomistic Scale

Minimal surfaces appear in atomistic scale in a very natural way. For example, TPMSs can be found in various natural or man-made crystal structures such as the zeolite sodalite and perovskite-type structure [8]. Consider an array of electrostatic point charges arranged in different crystallographic symmetries, the surfaces of zero electrostatic potential are very close to minimal surfaces such as P surface (e.g. CsCl), D surface (e.g. NaTl), and I-WP (e.g. $BaCuO_2$) [9]. The electron localization distribution functions also show the shape of G surfaces [10]. Based on a salient variable – curvature, over 20 years' scientific exploration of shapes in nature is summarized in a recent book of Hyde et al. [11].

Besides the beauty of periodic property, TPMSs are efficient partitioners of congruent spaces. They allow very high surface-to-volume ratios compared with other membrane packing, which is a desirable chemical property such as in enzyme design. In additions, they offer regular networks that define labyrinths with easily accessible positions, as shown in the meso scale.

2.4 Minimal Surface in Meso Scale

Minimal surface structures are found in macromolecule. Luzzati and Spegt [12] first discovered the intricate interconnected triply periodic network domain structure in strontium soap. Subsequently, they are revealed in various meso scale structures such as sea urchins [13], lyotropic liquid crystals [14], and lipid-protein-water systems [15]. In copolymer environment, a necessary condition for equilibrium is that the interfacial energy is minimized and the interface has constant mean curvature. TPMSs have been observed in systems of emulsions [16, 17] and biological structures [18, 19]. D surface bicontinuous tetrapod network is found in polystyrene-polyisoprene star polymers [20, 21] and diblock polymers [22]. Anderson and Thomas [23] modeled bicontinuous double-diamond structure using D surface. Hajduk et al. [24] studied gyroid morphology using G-surface. Wohlgemuth et al. [25] extended the structural investigation of morphologies to level surfaces with constant mean curvature. Matsushita et al. [26] showed that triblock copolymer exhibits the patterns of the G surface. TPMS's also appear in self-assembly of organic-inorganic composites [27] and the water-oil multi-continuous phases [28].

The ubiquity of hyperbolic surface structures appearing in both atomic and meso scales naturally distinguishes them as excellent candidates for multi-scale geometric modeling. It provides good analytical representation of geometry with complex topology, which is commonly seen in nano scales. Our belief behind multi-scale geometric modeling is that since nature is manifested by geometry, geometry is the nature. Compared to geometry in nature, regularly used engineering shapes are far too simple. Because it is easy to model those shapes in Euclidean space, and engineers are used to thinking in Euclidean world. Nevertheless, some seemingly complex shapes are easy to construct in non-Euclidean space. Here, we propose a new hyperbolic surface modeling scheme, called periodic surface, to represent nano-scale geometry.

3 Periodic Surface Model

We define a periodic surface as

$$\phi(\mathbf{r}) = \sum_k A_k \cos\left(2\pi(\mathbf{h}_k \cdot \mathbf{r})/\lambda_k + p_k\right) = \phi_0 \tag{1}$$

where $\mathbf{r}$ is the location vector in Euclidean space, $\mathbf{h}_k$ is the k^{th} lattice vector in reciprocal space, A_k is the magnitude factor, λ_k is the wavelength of periods, p_k is the phase shift, and ϕ_0 is a constant value. Specific periodic structures and phases can be constructed based on the implicit form.

With a generic and simpler form, some periodic surfaces can approximate TPMSs very well [29]. Compared to parametric TPMS representations, known as Weierstrass formula [30], periodic surfaces have much simpler forms. We can represent a periodic surface by a *periodic vector* $\left\langle \mathbf{A}^T, \mathbf{H}^T, \mathbf{P}^T, \mathbf{\Lambda}^T \right\rangle$ in the multi-dimensional configuration and phase space, where $\mathbf{A} = [A_k]$, $\mathbf{H} = [\mathbf{h}_k]$, $\mathbf{P} = [p_k]$, and

$\mathbf{\Lambda} = [\lambda_k]$ are concatenations of magnitudes, reciprocal lattice vectors, phases, and period lengths respectively. Table 1 lists some examples of periodic surface models that approximate TPMSs, including P-, D-, G-, and I-WP cubic morphologies that are frequently referred in chemistry and polymer literature. Besides the cubic phase, other mesophase structures such as spherical micelles, lamellar, rod-like hexagonal phases can also be modeled by periodic surfaces. In the rest of the paper, we refer to P, D, G, and I-WP surfaces as periodic surfaces, unless specified with the term of TPMS.

Periodic surface partitions 3D space into two congruent subspaces. As a special case, if we consider the surface $\phi(\mathbf{r}) = 0$, the periodic *zero surface*, two sides of the zero surface have opposite (+ and −) signs of function evaluation. Two types of operations on periodic zero surfaces can be used to construct complex shapes. One is volume-oriented, such as union, difference, and exclusive or, the other is surface-oriented, such as intersection, modulation, and convolution.

3.1 Volume-Oriented Operations

Given surfaces $\phi_1(\mathbf{r}) = 0$ and $\phi_2(\mathbf{r}) = 0$, the *union* operations are defined as $\phi_1 \cup^- \phi_2 := \min(\phi_1, \phi_2)$ and $\phi_1 \cup^+ \phi_2 := \max(\phi_1, \phi_2)$. The union operation $\cup^-$ merges two labyrinth volumes with − sign, while $\cup^+$ merges those with + sign. Union operation changes the volume ratio of two sides.

The *difference* operations are defined as $\phi_1 \setminus^- \phi_2 := \min(-\phi_1, \phi_2) = \max(\phi_1, -\phi_2)$ and $\phi_1 \setminus^+ \phi_2 := \min(\phi_1, -\phi_2) = \max(-\phi_1, \phi_2)$. The difference operation $\setminus^-$ gives the volume difference of two labyrinths with − sign, while $\setminus^+$ returns the difference with + sign.

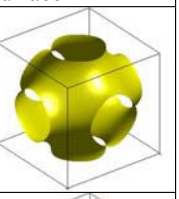
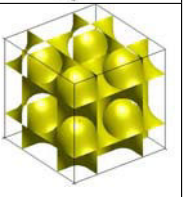
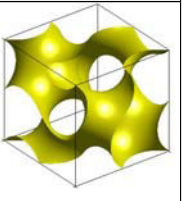
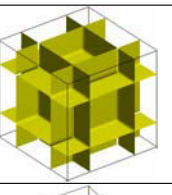
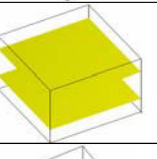
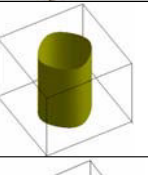
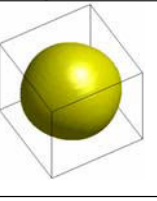
The *XOR* is defined as $\phi_1 \tilde{\vee} \phi_2 := \phi_1 \cdot \phi_2$. The XOR is an exclusive or operation on volumes with + and − signs. It can also be looked as surface union by which two periodic zero surfaces merge into one.

3.2 Surface-Oriented Operations

Given surfaces $\phi_1(\mathbf{r}) = 0$ and $\phi_2(\mathbf{r}) = 0$, the *intersection* is defined as $\phi_1 \wedge \phi_2 := (\phi_1)^2 + (\phi_2)^2$. Analytically, intersection of two periodic surfaces will be curves in E^3, and intersection of three periodic surfaces gives points.

The *modulate* operation is defined as $\phi_1 \oplus_m \phi_2 := \phi_1 + \phi_2 / m$, where ϕ_1 is the main surface, ϕ_2 is the modulating surface, and m is modulation index. The modulate operation adds fine features of the modulating surface onto the main surface. Fig. 3 illustrates the effect of modulate operation, where surface $\phi_A(\mathbf{r}) = 0$ is a P surface and surface $\phi_C(\mathbf{r}) = 0$ is a G surface. The modulated surface $(\phi_A \oplus_m \phi_C)(\mathbf{r}) = 0$ with different modulation indices m=1.0, 2.0, and 5.0 are shown in Fig.3-c, -d, and -e respectively.

Table 1. Periodic surface models represent TPMSs and some cubic phase nano structures

Morphology	Periodic Surface Model & Periodic Vector	Surface
P	$\mathbf{A}^T=\begin{bmatrix}1 & 1 & 1\end{bmatrix}\ \mathbf{H}^T=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$ $\mathbf{P}^T=\begin{bmatrix}0 & 0 & 0\end{bmatrix}\ \mathbf{\Lambda}^T=\begin{bmatrix}1 & 1 & 1\end{bmatrix}$	
D	$\mathbf{A}^T=\begin{bmatrix}1 & 1 & 1 & 1 & -1 & 1 & 1 & -1\end{bmatrix}$ $\mathbf{H}^T=\begin{bmatrix}1 & 1 & 1 & 1 & -1 & -1 & -1 & -1\\ -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1\\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1\end{bmatrix}$ $\mathbf{P}^T=\begin{bmatrix}0 & 0 & 0 & 0 & \pi/2 & \pi/2 & \pi/2 & \pi/2\end{bmatrix}$ $\mathbf{\Lambda}^T=\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\end{bmatrix}$	
G	$\mathbf{A}^T=\begin{bmatrix}1 & 1 & 1 & 1 & 1 & -1\end{bmatrix}$ $\mathbf{H}^T=\begin{bmatrix}-1 & -1 & 0 & 0 & -1 & -1\\ -1 & 1 & -1 & -1 & 0 & 0\\ 0 & 0 & -1 & 1 & -1 & 1\end{bmatrix}$ $\mathbf{P}^T=\begin{bmatrix}\pi/2 & \pi/2 & \pi/2 & \pi/2 & \pi/2 & \pi/2\end{bmatrix}$ $\Lambda^T=\begin{bmatrix}1 & 1 & 1 & 1 & 1 & -1\end{bmatrix}$	
Grid	$\mathbf{A}^T=\begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}\ \mathbf{H}^T=\begin{bmatrix}1 & 1 & 1 & 1\\ -1 & -1 & 1 & 1\\ -1 & 1 & -1 & 1\end{bmatrix}$ $\mathbf{P}^T=\begin{bmatrix}0 & 0 & 0 & 0\end{bmatrix}\ \mathbf{\Lambda}^T=\begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}$	
Lamellar	$\mathbf{A}^T=\begin{bmatrix}1\end{bmatrix}\ \mathbf{H}^T=\begin{bmatrix}0\\ 0\\ 1\end{bmatrix}\ \mathbf{P}^T=\begin{bmatrix}0\end{bmatrix}\ \mathbf{\Lambda}^T=\begin{bmatrix}1\end{bmatrix}$	
Rod	$\mathbf{A}^T=\begin{bmatrix}1 & 1 & 3 & 3 & 3\end{bmatrix}\ \mathbf{H}^T=\begin{bmatrix}1 & 0 & -1 & 0 & 0\\ 0 & 1 & 0 & -1 & 0\\ 0 & 0 & 0 & 0 & 0\end{bmatrix}$ $\mathbf{P}^T=\begin{bmatrix}0 & 0 & 0 & 0 & 0\end{bmatrix}\ \mathbf{\Lambda}^T=\begin{bmatrix}1 & 1 & 1 & 1 & 1\end{bmatrix}$	
Sphere	$\mathbf{A}^T=\begin{bmatrix}1 & 1 & 1 & 3 & 3 & 3 & 9\end{bmatrix}$ $\mathbf{H}^T=\begin{bmatrix}1 & 0 & 0 & -1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & -1 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & -1 & 0\end{bmatrix}$ $\mathbf{P}^T=\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$ $\mathbf{\Lambda}^T=\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1\end{bmatrix}$	

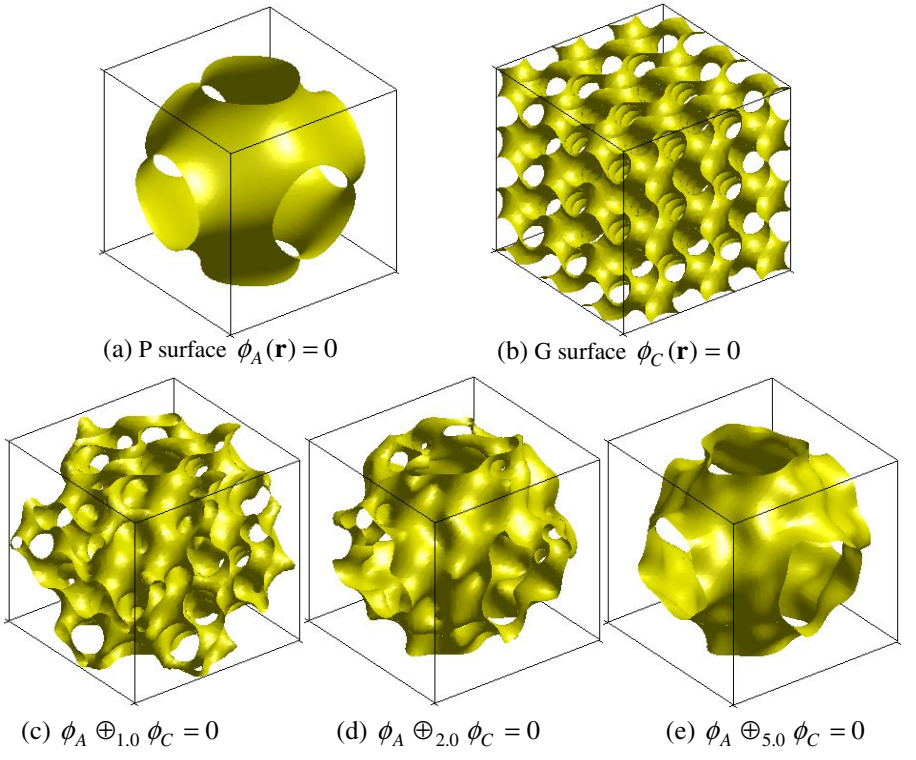


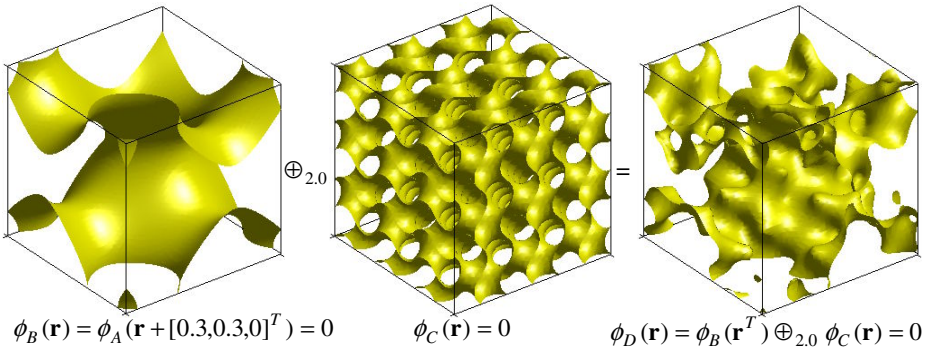
Fig. 3. Modulate operation of periodic surfaces

The *convolute* operation is defined as

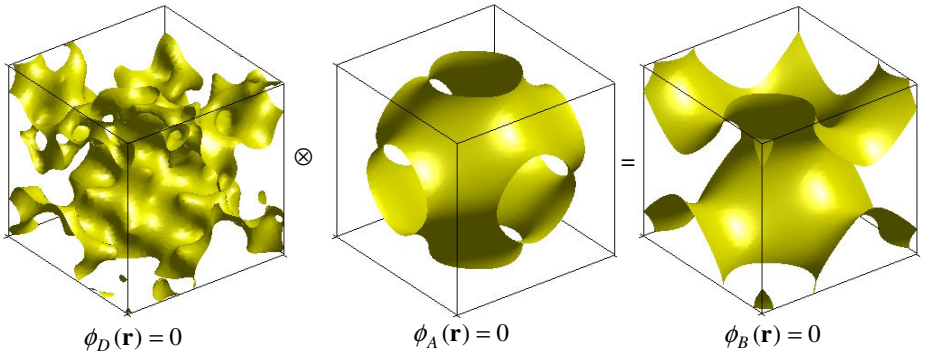
$$\phi_1 \otimes \phi_2(\mathbf{r}) := \iiint_V \phi_1(\mathbf{q})\phi_2(\mathbf{r}-\mathbf{q})dV \tag{2}$$

for the volume of interest V. The convolute operation can be looked as the inverse operation of modulate. Convolution has the effect of filtering and smoothing. As illustrated in Fig. 4, surface $\phi_B(\mathbf{r}) = \phi_A(\mathbf{r} + [0.3,0.3,0]^T) = 0$ is a P surface. Surface $\phi_C(\mathbf{r}) = 0$ is a G surface. Surface $\phi_D(\mathbf{r}) = 0$ is generated by modulation. If surface $\phi_D(\mathbf{r}) = 0$ is convoluted with surface $\phi_A(\mathbf{r}) = 0$, the fine features of $\phi_C(\mathbf{r}) = 0$ are filtered out and the original surface $\phi_B(\mathbf{r}) = 0$ is recovered.

Periodic surfaces exhibit infinitely periodic thus scalable properties. They inherently represent continuous labyrinth space and high surface-volume ratio geometry with porous features, which universally appear in natural structures. Given its simple mathematic representation, periodic surface can play an important role in CAND. With the basic periodic surface skeleton, atomistic structures can be constructed based on mapping from 2D hyperbolic space to 3D Euclidean space, which is a tiling process. In the next section, symmetric tiling methods for crystal construction are presented.



(a) Modulation



(b) Convolution

Fig. 4. Convolution as the inverse of modulation

4 Symmetric Tiling

Tiling is periodic subdivision of space into bounded and connected regions. The problem of constructing molecules and crystals is indeed tiling in Euclidean 3D space. Instead of filling 3D space directly, an indirect approach [31, 32] is to tile the 2D hyperbolic space first and then create mappings from 2D hyperbolic space H^2 to 3D Euclidean space E^3. Inorganic chemists have recognized that symmetric pattern is the centrality in understanding condensed atomic, molecular, and colloidal aggregates. A broad spectrum of crystal structures in Euclidean space can result from a single hyperbolic one with symmetric tiling.

To tile a periodic surface, two approaches can be taken. One is tiling by surface intersection, in which tiling is intersection by two or three periodic surfaces, the other is tiling by surface modulation, in which tiling is the intersection between the main surface and the modulated surface.

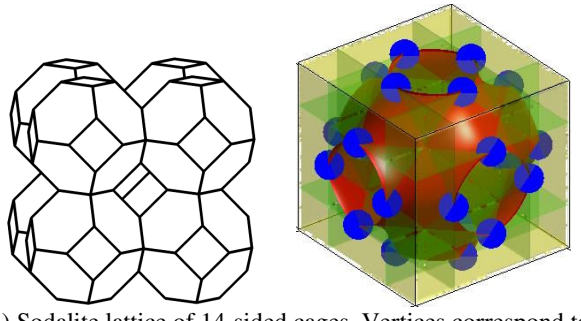
4.1 Surface Tiling by Intersection

The locations of atoms or particles in E^3 space can be determined by their simultaneous appearances in three or more H^2 spaces. Tiling surface $\phi_1(\mathbf{r}) = b_1$ with periodic surfaces $\phi_2(\mathbf{r}) = b_2$ and $\phi_3(\mathbf{r}) = b_3$ is to find solutions to

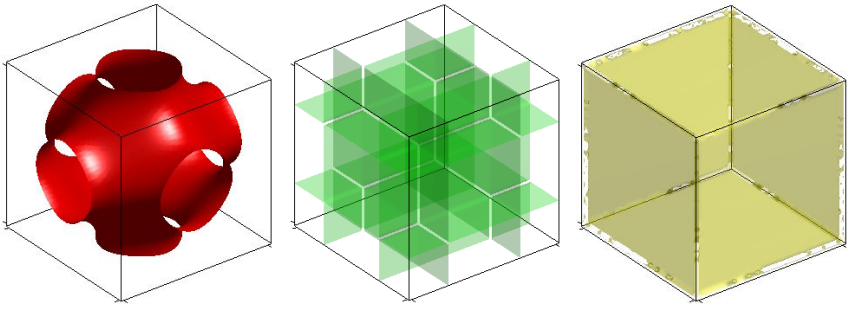
$$\phi(\mathbf{r}) = [\phi_1(\mathbf{r}) - b_1]^2 + [\phi_2(\mathbf{r}) - b_2]^2 + [\phi_3(\mathbf{r}) - b_3]^2 = 0 \tag{3}$$

For example, the periodic sodalite framework in Fig.5 can be naturally generated by the intersection of one P surface and two Grid surfaces with periodic vectors

$$\left\langle \begin{bmatrix}1\\1\\1\\1\end{bmatrix}\begin{bmatrix}1 & -1 & -1\\1 & -1 & 1\\1 & 1 & -1\\1 & 1 & 1\end{bmatrix}\begin{bmatrix}0\\0\\0\\0\end{bmatrix}\begin{bmatrix}1\\1\\1\\1\end{bmatrix} \right\rangle \text{ and } \left\langle \begin{bmatrix}1\\1\\1\\1\end{bmatrix}\begin{bmatrix}1 & -1 & -1\\1 & -1 & 1\\1 & 1 & -1\\1 & 1 & 1\end{bmatrix}\begin{bmatrix}\pi/2\\-\pi/2\\-\pi/2\\\pi/2\end{bmatrix}\begin{bmatrix}2\\2\\2\\2\end{bmatrix} \right\rangle .$$



(a) Sodalite lattice of 14-sided cages. Vertices correspond to Si (Al) and edges represent Si-O-Si (Si-O-Al) bonds

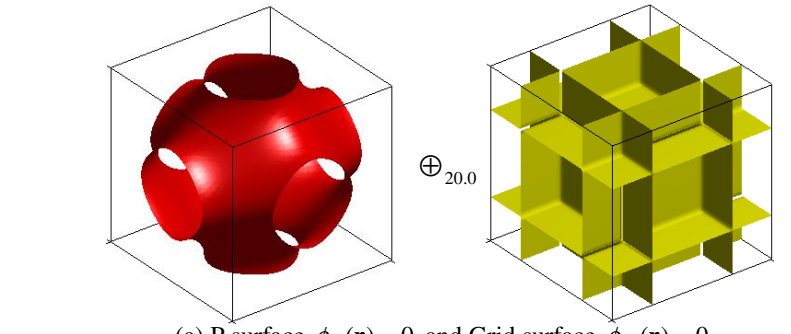


(b) P surface $\phi_A(\mathbf{r}) = 0$ (c) Grid surface $\phi_{X1}(\mathbf{r}) = 0$ (d) Grid surface $\phi_{X2}(\mathbf{r}) = 0$

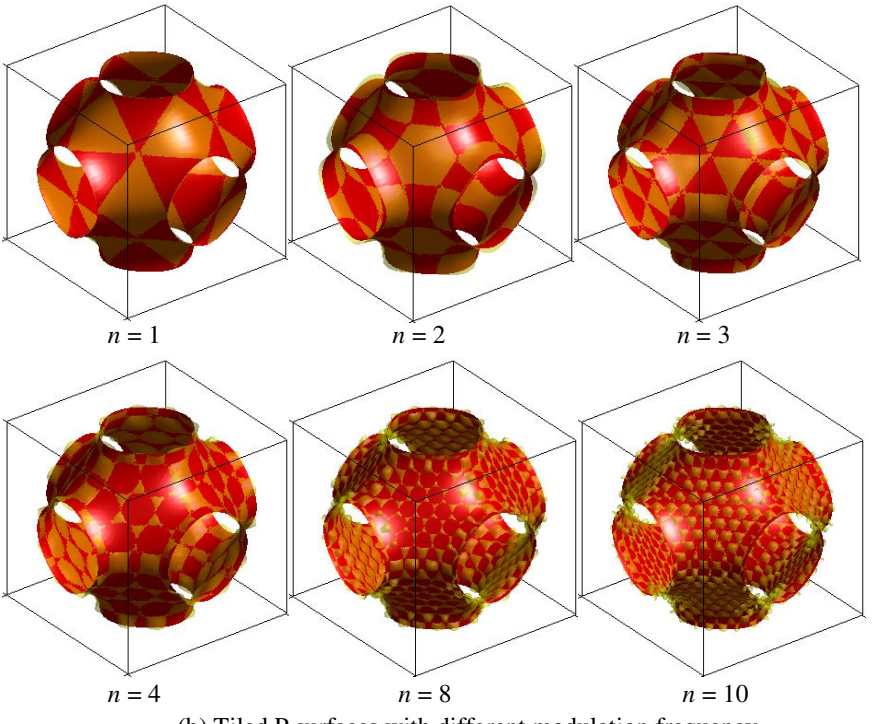
Fig. 5. Tiling by intersection of P surface with Grid surfaces to create sodalite framework

4.2 Surface Tiling by Modulation

In the modulation tiling approach, the main surface is modulated by high frequency surfaces. The intersection curves between the modulated surface and the original main



(a) P surface $\phi_A(\mathbf{r}) = 0$ and Grid surface $\phi_X(\mathbf{r}) = 0$



(b) Tiled P surfaces with different modulation frequency

Fig. 6. Tiling P surface with Grid surface modulation

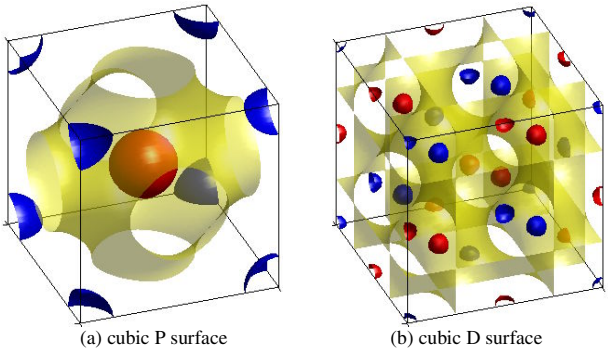
surface naturally divide the surface with symmetric patterns. For instance, if a P surface $\phi_A(\mathbf{r}) = 0$ is modulated by a Grid surface $\phi_X(\mathbf{r}) = 0$ with the periodic vector

$$\left\langle \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} n & -n & -n \\ n & -n & n \\ n & n & -n \\ n & n & n \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \right\rangle$$

and a modulation index m=20, where n ranges from 1 to 10, the tiled P surfaces are shown in Fig. 6.

5 Surface Enclosure

Instead of being looked as *loci* of atoms and particles as in tiling, periodic surfaces are regarded as isosurfaces of energy or potential in surface enclosure, such as in ref. [9] and [10]. Atoms or particles are enclosed by periodic surfaces, and the *foci* of surfaces determine positions of particles. Surfaces then are energy or potential envelops of particles. For instance, in Fig. 7, periodic P- and D-surface divides 3D space into two congruent labyrinth subspaces. The Body-Centered Cubic (BCC) crystal structure is easily constructed by P or D surfaces. Complex structures need careful selections of potential envelop and periodicity.



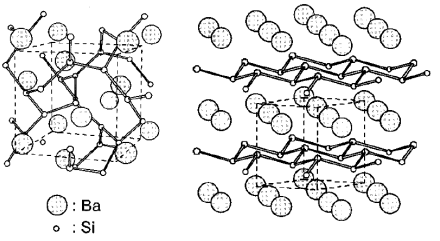
(a) cubic P surface (b) cubic D surface

Fig. 7. BCC crystal structures enclosed by periodic P and D surfaces

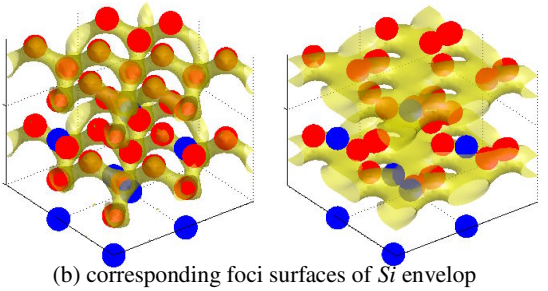
Given a periodic vector thus the potential envelop, the positions of surface foci can be determined. Instead of directly using maximum and minimum potentials in Euclidean space, we can search foci based on divergence of potential field intensity

$$\nabla^2\phi(\mathbf{r}) = \nabla \bullet \mathbf{F}(\mathbf{r}) = \rho(\mathbf{r}) \tag{4}$$

where $\mathbf{F}(\mathbf{r})$ is the potential field intensity and $\rho(\mathbf{r})$ can be looked as the charge density of particles. $\rho(\mathbf{r})$ is close to zero in inter-particle space and tends to increase or decrease as it approaches particle positions. Foci searching based on $\rho(\mathbf{r})$ also increases the robustness of searching with possible presence of noise compared to $\phi(\mathbf{r})$. Fig. 8 shows an example of foci enclosure to build the atomic structure of $BaSi_2$. The cubic and layer phases can be constructed based on periodic surface envelops, where positions of Ba are labeled by red dots and Si are in blue. The foci search method provides a good connection between physics and geometry in the form of force and field.



(a) $BaSi_2$ structures in cubic phase and layer phase



(b) corresponding foci surfaces of *Si* envelop

Fig. 8. Foci searching from surface enclosure of $BaSi_2$

6 Discussion

The periodic surface modeling enables rapid nano-scale geometry construction for simulation and design validation purpose. It provides flexible geometric models to represent natural shapes and man-made structures in atomistic, molecular, and meso scales, including crystals, polymers, and porous composites. This model provides simple yet versatile solution to lack of modeling and design methods of nano products. At the meso scale, the partition of space in organelle enables cells to control the concentrations of various molecules and their transport across bilayers. Also, the surfactant can be used as a template for polymerization reactions and produce nano products such as mesoporous silica molecular sieve, or hydro gels with well defined pore sizes and shapes for contact lenses, etc.

The simplicity of implementation is another advantage of the periodic surface modeling. A discrete approach can be taken to create the implicit surface models. Periodic surfaces are evaluated at lattice points within a periodic unit of interest, and isosurfaces are generated to approximate and visualize the surfaces. Periodic zero surface operations in Section 3 can be implemented based on the approximation of finite grids within a periodic volume unit. The volume integration of the convolute operation can be implemented based on Fast Fourier Transform (FFT). Because of the FFT's dimensional separability, the time complexity of the 3D convolution is $O(N^3 \log N)$ where N is the number of elements in one dimension within a periodic volume unit. Because of the periodicity, special attention is needed in calculating gradients and divergence in surface enclosure model construction.

Topology representation in periodic surface needs further study. So far only geometry of surface structure is discussed without consideration of topology, though

surface enclosure can be used to represent envelop of chemical bonds. There are some challenges in nano-scale topology capturing. For example, various intra- and inter-molecular forces exist with different strengths and ranges. The angle of a covalent bond plays an important role. Bond formation and breaking is a dynamic equilibrium in nature. Representation of structural defects also needs to be further investigated.

7 Conclusion

In this paper, we propose a new geometric modeling scheme, periodic surface, for computer aided nano design. Based on simple periodic vectors, geometry of thousands of particles can be built efficiently at the molecular scale. A generic approach is given to explore symmetric tiling and packing of loci surfaces in 2D hyperbolic space and subsequently map into conventional 3D Euclidean space. The surface enclosure method provides a physics-based crystal structure construction based on foci searching. At the meso scale, morphology of polymer and macromolecule can also be modeled. Associated implicit surface and volume operations are defined to support model creation. This new scheme enables a versatile system to model periodic and porous structures for material design at atomic, molecular, and meso scales.

References

1. Connolly, M.L. (1996) Molecular Surfaces: A Review. *Network Science*, Available at http://www.netsci.org/Science/Compchem/index.html
2. Lee, B. and Richards, F.M. (1971) The interpretation of protein structures: Estimation of static accessibility. *J. Mol. Biol.*, **55**: 379-400
3. Connolly, M.L. (1983) Solve-accessible surfaces of proteins and nucleic acids. *Science*, **221**(4612): 709-713
4. Bajaj, C., Lee, H.Y., Merkert, R., Pascucci, V. (1997) NURBS based B-rep models for macromolecules and their properties. in *Proc. 4th ACM Solid Modeling & Applications, Atlanta, GA*, pp.217-228
5. Bajaj, C., Pascucci, V., Shamir, A., Holt, R., and Netravali, A. (2003), Dynamic Maintenance and Visualization of Molecular Surfaces. *Discrete Applied Mathematics*, **127**: 23-51
6. Carson, M (1996) Wavelets and molecular structure. *J. Comp. Aided Mol. Des.*, **10**: 273-283
7. Hoffman, D.A. and Hoffman, J.T., Scientific Group Project, available at http://www.msri.org/about/sgp/jim/geom/index.html
8. Andersson, S. (1983) On the Description of Complex Inorganic Crystal Structures. *Angew. Chem. Int. Ed. Engl.*, **22**(2): 69-81
9. von Schnering, H.G. and Nesper, R. (1987) How Nature Adapts Chemical Structures to Curved Surfaces. *Angew. Chem. Int. Ed. Engl.*, **26**(11): 1059-1080
10. Savin, A., Jepsen, O., Flad, J., Andersen, O.-K., Preuss, H., and von Schnering, H.-G. (1992) Electron localization in solid-state structures of the elements: the diamond structure. *Angew. Chem. Int. Ed. Engl.*, **31** (2): 187-188
11. Hyde, S., Andersson, S., Larsson, K., Blum, Z., Landh, T., Lidin, S., and Ninham, B.W. (1997) *The Language of Shape*, Elsevier, Amsterdam

12. Luzzati, V. and Spegt, P.A. (1967) Polymorphism of Lipids. *Nature*, **215**(5102): 701-704
13. Donnay, G. and Pawson, D.L. (1969) X-ray Diffraction Studies of Echinoderm Plates. *Science*, **166**(3909): 1147-1150
14. Scriven, L.E. (1976) Equilibrium Bicontinuous Structure. *Nature*, **263**(5573): 123-125
15. Erricsson, B., Larsson, K., and Fontell, K., (1983) A Cubic Protein-Monoolein-Water Phase. *Biochim. Biophys. Acta*, **729**: 23-27
16. Ciach, A. and Holyst, R. (1999) Periodic surfaces and cubic phases in mixtures of oil, water, and surfactant. *J. Chem. Phys.*, **110**(6):3207-3214
17. Schwarz, U.S. and Gompper, G. (2000) Stability of bicontinuous cubic phases internary amphiphilic systems with spontaneous curvature. *J. Chem. Phys.*, 112(8):3792-3802
18. Mariani, P., Luzzti, V., and Delacroix, H. (1988) Cubic phases of lipid-containing systems: Structure analysis and biological implications. *J. Mol. Biol.*, **204**(1): 165-189
19. Luzzati, V., Varas, R., Mariani, P., Gulik, A., and Delacroix, H. (1993) Cubic phases of lipid-containing systems: Elements of a theory and biological connotations. *J. Mol. Biol.*, **229**(2):540-551
20. Aggarwal, S.L. (1976) Structure and Properties of Block Polymers and Multiphase Polymer Systems: An Overview of Present Status and Future Potential. *Polymer*, **19**(11): 938-956
21. Thomas, E.L., Alward, D.B., Kinning, D.J., Martin, D.C., Handlin, D.L., and Fetters, L.J. (1986) Ordered Bicontinuous Double-Diamond Structure of Start Block Copolymers: A New Equilibrium Microdomain Morphology. *Macromolecules*, **19**(8): 2197-2202
22. Hasegawa, H., Tanaka, H., Yamasaki, K., and Hashimoto, T. (1987) Bicontinuous Microdomain Morphology of Block Copolymers. 1. Tetrapod-Network Structure of Polystyrene-Polyisoprene Diblock Polymers. *Macromolecules*, **20**(7): 1651-1662
23. Anderson, D.M. and Thomas, E.L. (1988) Microdomain Morphology of Star Copolymers in the Strong-Segregation Limit. *Macromolecules*, **21**(11): 3221-3230
24. Hajduk, D.A., Harper, P.E., Gruner, S.M., Honeker, C.C., Kim, G., Thomas, E.L., and Fetters, L.J. (1994) The Gyroid: A New Equilibrium Morphology in Weakly Segregated Diblock Copolymers. *Macromolecules*, **27**(15): 4063-4075
25. Wohlgemuth, M., Yufa, N., Hoffman, J., and Thomas, E.L. (2001) Triply Periodic Bicontinuous Cubic Microdomain Morphologies by Symmetries. *Macromolecules*, **34**(17): 6083-6089
26. Matsushita, Y., Suzuki, J., and Seki, M. (1998) Surfaces of Tricontinuous Structure Formed by an ABC Triblock Copolymer in Bulk. *Physica B*, **248**: 238-242
27. Davis, M.E., "Organizing for Better Synthesis", *Nature*, Vol.364, No.6436 (July 29, 1993), pp.391-392
28. Gozdz, T.W. and Holyst, R., "Triply Periodic Surfaces and Multiply Continuous Structures from the Landau Model of Microemulsions", *Physical Review E*¸ Vol.54, No.5 (November 1996), pp.5012-5027
29. Gandy, P.J.F., Bardhan, S., Mackay, A.L., and Klinowski, J. (2001) Nodal surface approximations to the P, G, D and I-WP triply periodic minimal surfaces. *Chem. Phys. Lett.*, **336**: 187-195
30. Fogden, A. and Hyde, S.T. (1992) Parametrization of triply periodic minimal surfaces. I. Mathematical basis of the construction. *Acta Crys.*, **A48**: 442-451
31. Hyde, S.T. and Oguey, C. (2000) From 2D Hyperbolic forests to 3D Euclidean entangled thickets. *Eur. Phys. J. B*, **16**: 613-630
32. Nesper, R. and Leoni, S. (2001) On tilings and patterns on Hyperbolic surfaces and their relation to structural chemistry. *ChemPhysChem*, **2**: 413-422

Minimal Mean-Curvature-Variation Surfaces and Their Applications in Surface Modeling*

Guoliang Xu[1] and Qin Zhang[2]

[1,2] LSEC, Institute of Computational Mathematics, Academy of Mathematics and System Sciences, Chinese Academy of Sciences, Beijing 100080, China
[2] Department of Basic Courses, Beijing Information Science and Technology University, Beijing 100085, China

Abstract. Physical based and geometric based variational techniques for surface construction have been shown to be advanced methods for designing high quality surfaces in the fields of CAD and CAGD. In this paper, we derive a Euler-Lagrange equation from a geometric invariant curvature integral functional–the integral about the mean curvature gradient. Using this Euler-Lagrange equation, we construct a sixth-order geometric flow (named as minimal mean-curvature-variation flow), which is solved numerically by a divided-difference-like method. We apply our equation to solving several surface modeling problems, including surface blending, N-sided hole filling and point interpolating. The illustrative examples provided show that this sixth-order flow yields high quality surfaces.

Keywords: Euler-Lagrange equation, Minimal mean-curvature-variation flow, Energy functional, Discretization, Surface modeling.

1 Introduction

Problems such as surface fairing, free-from surface design, surface blending and N-sided hole filling have been important issues in the areas of CAD and CAGD. These problems can be efficiently solved by an energy-based variational approach (e. g. [2,6,14,15]). Roughly speaking, the variational approach is to pursue a curve or surface which minimizes certain type of energy simultaneously satisfying prerequisite boundary conditions. A problem one meets within this approach is the choice of energy models. Energy models previously used can be classified into the categories of physical based and geometric based. The class of physical models mainly encompasses membrane energy $\mathscr{E}_1$ and strain energy $\mathscr{E}_2$ of thin elastic plate (see [4,15]):

$$\mathscr{E}_1(f) := \int_\Omega (f_x^2 + f_y^2)\, dx\, dy, \qquad \mathscr{E}_2(f) := \int_\Omega (f_{xx}^2 + 2f_{xy}^2 + f_{yy}^2)\, dx\, dy,$$

* Project supported in part by NSFC grant 10371130 and National Key Basic Research Project of China (2004CB318000). The second author is also supported in part by the NSFC grant 10571012 and the Beijing Natural Science Foundation 1062005.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 357–370, 2006.
© Springer-Verlag Berlin Heidelberg 2006

where $f(x, y)$ and Ω are surface parametrization and its domain, respectively. Recently, energy functionals based on geometric invariants begin to lead in this field. As is well-known, area functional and total curvature functional (see [7])

$$\mathscr{E}_3(\mathcal{M}) := \int_{\mathcal{M}} dA, \qquad \mathscr{E}_4(\mathcal{M}) := \int_{\mathcal{M}} (k_1^2 + k_2^2)\, dA$$

are the most frequently used energies, where k_1 and k_2 are the principal curvatures. The energy

$$\mathscr{E}_5(\mathcal{M}) := \int_{\mathcal{M}} \left[\left(\frac{dk_1}{d\mathbf{e}_1} \right)^2 + \left(\frac{dk_2}{d\mathbf{e}_2} \right)^2 \right] dA$$

proposed by Moreton et al. in [11] punishes the variation of the principal curvatures, where $\mathbf{e}_1$ and $\mathbf{e}_2$ are principal directions corresponding to the principal curvatures k_1 and k_2. The advantage of utilizing physical based models is that the resulting equations are linear and therefore easy to solve. The disadvantage is that the resulting equations are parameter dependent. Energy models based on geometric invariants can overcome this shortcoming.

Another critical problem of the variational approach is how to find out those surfaces which minimize these energy functionals. Two approaches have been employed to solve this problem. One method is using the optimization approach (see [6,11,15]). The minimization problem can be discretized to arrive at finite dimensional linear or nonlinear systems. Approximate solutions are then obtained by solving the constructed systems. Another widely accepted method is based on variational calculus. The first step of this method is to calculate the Euler-Lagrange equations for the energy functionals, then solve these equations for the ultimate surface. This method is superior to the optimization technique in general because optimization is lack of local shape control and computationally expensive.

To solve the Euler-Lagrange equations, gradient descent flow method has been introduced and widely accepted. For instance, from the Euler-Lagrange equation $H = 0$ of $\mathscr{E}_3(\mathcal{M})$, the well-known mean curvature flow $\frac{\partial \mathbf{r}}{\partial t} = H\mathbf{n}$ is constructed. Here $\mathbf{n}$ is the normal vector field of the surface. When the steady state of the flow is achieved, we obtain $H = 0$. Similarly, *Willmore surfaces* (see [16]), the solution of the Euler-Lagrange equation $\Delta H + 2H(H^2 - K) = 0$ of the energy

$$\mathscr{E}_6(\mathcal{M}) := \int_{\mathcal{M}} H^2 dA,$$

can be constructed by this gradient descent flow method. For the purpose of volume-preserving for closed surfaces, surface diffusion flow (see [10]) $\frac{\partial \mathbf{r}}{\partial t} = \Delta H\mathbf{n}$ is sometimes employed.

It is well-known that the second-order flows, such as mean curvature flow or averaged mean curvature flow, yield G^0 continuous surfaces at the boundaries of the constructed surfaces. The fourth-order flows, such as surface diffusion flow (SDF) and Willmore flow (WF) ([8]), result in G^1 continuity. However, higher order continuity are sometimes required in the industrial and engineering

applications. For instance, in the shape design of the streamlined surfaces of aircraft, ships and cars, G^2 continuous surfaces are crucial. Therefore, higher order flow need to be considered. On this aspect, Xu et al. have utilized a sixth-order flow in [20] to achieve G^2 continuity and Zhang et al. have used another sixth-order PDE in [21,22] to obtain C^2 continuity. A sixth-order equation is also proposed in [3] by Botsch and Kobbelt to conduct real-time freeform medeling. But all these sixth-order flows and PDEs are neither physical based nor geometric based in the sense mentioned above.

In this paper, a sixth-order geometric based PDE is introduced. It is derived from the Euler-Lagrange equation of the energy functional

$$\mathscr{F}(\mathcal{M}) := \int_{\mathcal{M}} \|\nabla H\|^2 dA, \tag{1.1}$$

which punishes the variation of mean curvature. A surface which minimizes functional (1.1) is called *minimal mean-curvature-variation surface*. We expect that G^2 continuity can be achieved using this sixth-order flow in solving the surface modeling problems, such as surface blending, N-sided hole filling and scattered points interpolation. A semi-implicit divided-difference-like discretization scheme is proposed to solve the highly nonlinear PDE. The experimental and comparative results show that high quality surfaces are obtained.

The rest of this paper is organized as follows. In section 2, some used notations and preliminaries are introduced. One sixth-order flow is derived in section 3. The numerical solving of the flow is discussed in section 4. The application and examples are provided in section 5. Section 6 concludes this paper.

2 Notations and Preliminaries

In this section, we introduce some notations and several differential operators defined on surface used throughout this paper. Let $\mathcal{M}$ be a regular parametric surface represented as $\mathbf{r}(u,v) \in \mathbb{R}^3$, $(u,v) \in \Omega \subset \mathbb{R}^2$, whose unit normal vector is $\mathbf{n} = \frac{\mathbf{r}_u \times \mathbf{r}_v}{\|\mathbf{r}_u \times \mathbf{r}_v\|}$ after suitable orientation being chosen, where the subscript of $\mathbf{r}$ denotes partial derivative and $\|\mathbf{x}\| := \langle \mathbf{x}, \mathbf{x} \rangle^{\frac{1}{2}} := (\mathbf{x}^T \mathbf{x})^{\frac{1}{2}}$ is the usual Euclidean norm. Superscript T stands for the transpose operation. We assume at least $\mathbf{r} \in C^6(\overline{\Omega}, \mathbb{R}^3)$. The coefficients of the first fundamental form and the second fundamental form are

$$\begin{aligned} g_{11} &= \langle \mathbf{r}_u, \mathbf{r}_u \rangle, \quad g_{12} = \langle \mathbf{r}_u, \mathbf{r}_v \rangle, \quad g_{22} = \langle \mathbf{r}_v, \mathbf{r}_v \rangle, \\ b_{11} &= \langle \mathbf{n}, \mathbf{r}_{uu} \rangle, \quad b_{12} = \langle \mathbf{n}, \mathbf{r}_{uv} \rangle, \quad b_{22} = \langle \mathbf{n}, \mathbf{r}_{vv} \rangle. \end{aligned}$$

To simplify notation we sometimes write $w = (u,v)$ and $u^1 = u, u^2 = v$ and

$$[\, g^{\alpha\beta} \,] = [\, g_{\alpha\beta} \,]^{-1}, \quad g = \det[\, g_{\alpha\beta} \,], \quad [\, b^{\alpha\beta} \,] = [\, b_{\alpha\beta} \,]^{-1}, \quad b = \det[\, b_{\alpha\beta} \,].$$

To introduce the mean curvature and Gaussian curvature, let us first introduce the concept of *Weingarten map*. The Weingarten map or shape operator of

surface $\mathcal{M}$ is a self-adjoint linear map on the tangent space $T_{\mathbf{r}}\mathcal{M} := \text{span}\{\mathbf{r}_u, \mathbf{r}_v\}$ defined by (see [5])

$$S : T_{\mathbf{r}}\mathcal{M} \to T_{\mathbf{r}}\mathcal{M}, \quad S(\mathbf{v_r}) = -D_{\mathbf{v}}\mathbf{n},$$

where $\mathbf{v_r}$ is an arbitrary tangent vector of $\mathcal{M}$ at point $\mathbf{r}$ and $\mathbf{v}$ is a tangent vector field satisfying $\mathbf{v}(\mathbf{r}) = \mathbf{v_r}$, and $D_{\mathbf{v}}$ is directional derivative operator along direction $\mathbf{v}$. We can represent this linear map by a matrix as $S = [b_{\alpha\beta}][g^{\alpha\beta}]$. The trace divided by 2 and determinant of S, $H = \text{tr}(S)/2$, $K = \det(S)$ are the *mean curvature* and *Gaussian curvature*, respectively. Now let us introduce some used differential operators defined on surface $\mathcal{M}$.

Tangential gradient operator. Let f be a smooth function on $\mathcal{M}$. Then the *tangential gradient operator* ∇ acting on f is given by

$$\nabla f = [\mathbf{r}_u,\ \mathbf{r}_v][\, g^{\alpha\beta}\,][f_u,\ f_v]^T \in \mathbb{R}^3.$$

Second tangent operator. Let f be a smooth function on $\mathcal{M}$. Then the *second tangent operator* $\diamond$ acting on f is given by

$$\diamond f = [\mathbf{r}_u,\ \mathbf{r}_v][\, K\, b^{\alpha\beta}\,][f_u,\ f_v]^T \in \mathbb{R}^3.$$

Divergence operator. Let $\mathbf{v}$ be a C^1 smooth vector field on $\mathcal{M}$. Then the *divergence* of $\mathbf{v}$ is defined by

$$\text{div}(\mathbf{v}) = \frac{1}{\sqrt{g}}\left[\frac{\partial}{\partial u}, \frac{\partial}{\partial v}\right]\left[\,\sqrt{g}\,[\,g^{\alpha\beta}\,]\,[\mathbf{r}_u,\ \mathbf{r}_v]^T\mathbf{v}\,\right].$$

Note that if $\mathbf{v}$ is a normal vector field of $\mathcal{M}$, $\text{div}(\mathbf{v}) = 0$.

Laplace-Beltrami operator. Let $f \in C^2(\mathcal{M})$. Then ∇f is a smooth vector field on $\mathcal{M}$. The Laplace-Beltrami operator (LBO) Δ applying to f is defined by $\Delta f = \text{div}(\nabla f)$. From the definitions of ∇ and div, we can easily derive that

$$\Delta f = \frac{1}{\sqrt{g}}\left[\frac{\partial}{\partial u}, \frac{\partial}{\partial v}\right]\left[\,\sqrt{g}\,[\,g^{\alpha\beta}\,]\,[f_u,\ f_v]^T\,\right].$$

It is easy to see that Δ is a second-order differential operator which relates closely to the *mean curvature normal* $\mathbf{H} := H\mathbf{n}$ by the relation

$$\Delta\mathbf{r} = 2\mathbf{H}. \tag{2.1}$$

It should be emphasized that these differential operators are all geometric intrinsic, though they are defined by the local parametrization of surface .

3 A Sixth-Order Geometric Flow

In this section, we first derive a Euler-Lagrange equation for the functional (1.1) and then construct a sixth-order geometric flow.

Theorem 3.1. *Let $\mathscr{F}(\mathcal{M})$ be defined as* (1.1)*. Then the Euler-Lagrange equation of $\mathscr{F}(\mathcal{M})$ is*

$$\Delta^2 H + 2(2H^2 - K)\Delta H + 2\langle \nabla H, \diamond H\rangle - 2H\|\nabla H\|^2 = 0. \tag{3.1}$$

Proof. At first, we can rewrite the functional (1.1) as

$$\mathscr{F}(\mathcal{M}) = \int_\Omega \|\nabla H\|^2 \sqrt{g} du^1 du^2, \tag{3.2}$$

which is parameter-invariant. Consider now an extremal $\mathcal{M}$ of functional (3.2) and a family of normal variation $\underline{\mathbf{r}}(w,\varepsilon)$ of $\mathcal{M}$ defined by

$$\underline{\mathbf{r}}(w,\varepsilon) = \mathbf{r}(w) + \varepsilon\varphi(w)\mathbf{n}(w), \quad w \in \overline{\Omega}, \quad |\varepsilon| \ll 1,$$

where $\varphi \in C_c^\infty(\Omega) := \{\phi \in C^\infty(\Omega, \mathbb{R}); \mathrm{supp}\phi \subset \Omega\}$. Then we obtain

$$0 = \frac{d}{d\varepsilon}\mathscr{F}(\underline{\mathcal{M}}(\cdot,\varepsilon))\big|_{\varepsilon=0} =: \delta\mathscr{F}(\mathcal{M},\varphi), \tag{3.3}$$

where

$$\delta\mathscr{F}(\mathcal{M},\varphi) = \int_\Omega \left[\delta(\|\nabla H\|^2) + \|\nabla H\|^2(\delta\sqrt{g})/\sqrt{g}\right]\sqrt{g} du^1 du^2. \tag{3.4}$$

From

$$\delta(g_{\alpha\beta}) = -2\varphi b_{\alpha\beta}, \qquad \delta(g) = -4gH\varphi,$$
$$\delta(\sqrt{g}) = -2H\varphi\sqrt{g}, \qquad \delta(H) = (2H^2 - K)\varphi + \tfrac{1}{2}\Delta\varphi,$$

we can deduce that

$$\delta(\|\nabla H\|^2) = 4H\varphi\|\nabla H\|^2 - 2\langle\nabla H, \diamond H\rangle\varphi + 2\langle\nabla H, \nabla[(2H^2 - K)\varphi + \frac{1}{2}\Delta\varphi]\rangle. \tag{3.5}$$

Substituting (3.5) into (3.4), we arrive at

$$\begin{aligned}\delta\mathscr{F}(\mathcal{M},\varphi) &= \int_\Omega \Big[2H\|\nabla H\|^2\varphi - 2\langle\nabla H, \diamond H\rangle\varphi \\ &\quad + 2\langle\nabla H, \nabla[(2H^2 - K)\varphi + \frac{1}{2}\Delta\varphi]\rangle\Big]\sqrt{g} du^1 du^2 \\ &= \int_\Omega \Big[2H\|\nabla H\|^2\varphi - 2\langle\nabla H, \diamond H\rangle\varphi \\ &\quad - 2(2H^2 - K)\Delta H\varphi - \Delta H\Delta\varphi\Big]\sqrt{g} du^1 du^2.\end{aligned}$$

Using the Green's formula, we finally write (3.3) as

$$\int_\Omega \left[2H\|\nabla H\|^2 - 2\langle\nabla H, \diamond H\rangle - 2(2H^2 - K)\Delta H - \Delta^2 H\right]\varphi\sqrt{g} du^1 du^2 = 0,$$

for any $\varphi \in C_c^\infty(\Omega)$. In the end, the Euler-Lagrange equation of functional (1.1) is (3.1) and the theorem is proved.

Obviously, equation (3.1) is of sixth-order. It is easy to see that surfaces with constant mean curvature, such as Delaunay surfaces (see [12], pp. 144-148) (include unduloid and nodoid), sphere, cylinder, and minimal surfaces, are the solutions of the equation. But tori and cone are not the solution surfaces of the equation. It is not difficult to derive that

Theorem 3.2. *Equation* (3.1) *is invariant under the transforms of rotation, translation and scaling.*

Here the *invariant* means that a solution surface of (3.1) is still a solution under the three transforms mentioned.

Now let us introduce the sixth-order flow used in this paper. Let $\mathcal{M}_0$ be a compact immersed orientable surface in $\mathbb{R}^3$. A curvature driven geometric evolution consists of finding a family $\{\mathcal{M}(t) : t \geq 0\}$ of smooth immersed orientable surfaces in $\mathbb{R}^3$ which evolve according to the flow equation

$$\frac{\partial \mathbf{r}(t)}{\partial t} = \mathbf{n}V, \quad \mathcal{M}(0) = \mathcal{M}_0. \tag{3.6}$$

Here $\mathbf{r}(t)$ is a surface point on $\mathcal{M}(t)$, V denotes the normal velocity of $\mathcal{M}(t)$. Let $\mathcal{M}(t)$ be a closed surface with outward normal. Then it has been shown that (see [9], Theorem 4)

$$\frac{dA(t)}{dt} = -2\int_{\mathcal{M}(t)} VH\, dA, \quad \frac{dV(t)}{dt} = \int_{\mathcal{M}(t)} V\, dA, \tag{3.7}$$

where $A(t)$ denotes the area of surface $\mathcal{M}(t)$ and $V(t)$ denotes the volume of the region enclosed by $\mathcal{M}(t)$. If $\frac{dA(t)}{dt} = 0$, we say the flow is area-preserving. Similarly, the flow is volume-preserving if $\frac{dV(t)}{dt} = 0$.

Let $\mathcal{M}_0$ be a compact orientable surface in $\mathbb{R}^3$ with boundary Γ. Then the sixth-order flow constructed from the Euler-Lagrange equation (3.1) is

$$\begin{cases} \dfrac{\partial \mathbf{r}}{\partial t} = \left[\Delta^2 H + 2(2H^2 - K)\Delta H + 2\langle \nabla H, \diamond H\rangle - 2H\|\nabla H\|^2\right]\mathbf{n}, \quad \mathbf{r} \in \mathcal{M}(t), \\ \mathcal{M}(0) = \mathcal{M}_0, \quad \partial\mathcal{M}(t) = \Gamma. \end{cases} \tag{3.8}$$

If $\mathcal{M}(t)$ is a closed constant mean curvature surface, (3.7) implies that $\frac{dA(t)}{dt} = 0$ and $\frac{dV(t)}{dt} = 0$ for the flow (3.8). In general, this area-preserving or volume-preserving properties are not valid.

In this paper we name this newly introduced flow as *minimal mean-curvature-variation flow* (abbreviated as MMCVF). Though the problems of the existence and the uniqueness of the solutions of this flow are currently left open, the numerical solving of the equation could be conducted by either the divided-difference-like (generalized divided difference) method or the finite element approach. For simplicity, we solve it in this paper by the divided-difference-like method.

4 Numerical Solving of the GPDE

Discretizations of curvatures and geometric differential operators. To solve the geometric PDE (3.8) over a triangular surface mesh M with vertex set $\{\mathbf{r}_i\}$ using a divided-difference-like method, discrete approximations of the mean curvature, Gaussian curvature and various differential operators are required. In order to use a semi-implicit scheme, we require the approximations of the differential operators mentioned above at $\mathbf{r}_i$ to have the following form

$$\Theta f(\mathbf{r}_i) = \sum_{j \in N_1(i)} w_{ij}^{\Theta} f(\mathbf{r}_j),$$

where Θ represents one of above mentioned differential operators and $w_{ij}^{\Theta} \in \mathbb{R}$ or $w_{ij}^{\Theta} \in \mathbb{R}^3$, $N_k(i)$ is the index set of the k-ring neighbor vertices of $\mathbf{r}_i$. Although there are several discretization schemes of Laplace-Beltrami operator and Gaussian curvature (see [17,19] for a review), the discretizations of Gaussian curvature are not in the required form and may be not consistent in the following sense.

Definition 4.1. *A set of approximations of differential geometric operators is said consistent if there exists a smooth surface $\mathcal{S}$, such that the approximate operators coincide with the exact counterparts of $\mathcal{S}$.*

Here we use a biquadratic fitting of the surface data and function data to calculate the approximate differential operators. The algorithm we adopted is from [18]. Let $\mathbf{r}_i$ be a vertex of M with valence n, $\mathbf{r}_j$ be its neighbor vertices for $j \in N_1(i)$. Then approximations of the used differential operators are represented as (see [18] for detail)

$$\nabla f(\mathbf{r}_i) \approx \textstyle\sum_{j \in N_1(i)} w_{ij}^{\nabla} f(\mathbf{r}_j), \qquad \diamond f(\mathbf{r}_i) \approx \textstyle\sum_{j \in N_1(i)} w_{ij}^{\diamond} f(\mathbf{r}_j),$$
$$\Delta f(\mathbf{r}_i) \approx \textstyle\sum_{j \in N_1(i)} w_{ij}^{\triangle} f(\mathbf{r}_j), \qquad K(\mathbf{r}_i) \approx \textstyle\sum_{j \in N_1(i)} (w_{ij}^{K})^T \mathbf{r}_j,$$

where w_{ij}^{∇}, $w_{ij}^{\diamond}$, $w_{ij}^{K} \in \mathbb{R}^3$ and $w_{ij}^{\triangle} \in \mathbb{R}$. Using the relation (2.1), we have

$$\mathbf{H}(\mathbf{r}_i) \approx \frac{1}{2} \sum_{j \in N_1(i)} w_{ij}^{\triangle} \mathbf{r}_j, \quad H(\mathbf{r}_i) \approx \frac{1}{2} \sum_{j \in N_1(i)} w_{ij}^{\triangle} \mathbf{n}(\mathbf{r}_i)^T \mathbf{r}_j.$$

Remark 4.1. The reasons why we approximate the used differential operators basing on the parameter fitting have been given in [18]. In a word, the scheme we adopted leads to convergent, consistent and required form approximations.

Semi-Implicit discretization of the GPDE. Let us now consider the discretization of (3.8). An explicit scheme for solving the equation (3.8) in general is unstable, therefore requires a small time step-size. To make the evolution process more efficient, an implicit scheme is more desirable. However, since the used PDE is highly nonlinear, a complete implicit scheme is hard to solve. In the following

we present a semi-implicit scheme, which leads to a linear system of equations. The basic idea for forming the linear equations is to decompose each of the terms of (3.1) as a product of a linear term and a remaining term. The linear term is discretized using the discretized differential operator. The remaining term is computed from previous approximation of the surface. Specifically, the terms of the equation (3.8) are approximated as follows:

$$
\begin{aligned}
&\frac{\partial \mathbf{r}}{\partial t} \approx \frac{\mathbf{r}_i^{(k+1)} - \mathbf{r}_i^{(k)}}{\tau},\\
&\Delta^2 H \approx \Delta(\Delta H_i^{(k+1)}),\\
&2(2H^2 - K)\Delta H \approx (2H_i^{(k+1)} H_i^{(k)} - K_i^{(k+1)})\Delta H_i^{(k)} + [2(H_i^{(k)})^2 - K_i^{(k)}]\Delta H_i^{(k+1)},\\
&2\langle \nabla H, \diamond H\rangle \approx \langle \nabla H_i^{(k+1)}, \diamond H_i^{(k)}\rangle + \langle \nabla H_i^{(k)}, \diamond H_i^{(k+1)}\rangle,\\
&2H\langle \nabla H, \nabla H\rangle \approx H_i^{(k+1)} \langle \nabla H_i^{(k)}, \nabla H_i^{(k)}\rangle + H_i^{(k)} \langle \nabla H_i^{(k+1)}, \nabla H_i^{(k)}\rangle,
\end{aligned}
$$

where τ is the time step-size, the subscript i denotes the corresponding quantity is evaluated at the vertex $\mathbf{r}_i$, the superscript (k) denotes the quantity is at the time $k\tau$, the superscript $(k+1)$ denotes the quantity is at the time $(k+1)\tau$. The quantities at $(k+1)\tau$ are unknowns. Using these approximations, we can discretize the equation(3.8) recursively, and derive a linear system with $\mathbf{r}_i^{(k+1)}$ as unknowns. For instance,

$$
\begin{aligned}
\mathbf{n}_i \Delta(\Delta H_i^{(k+1)}) &\approx \mathbf{n}_i \sum_{j\in N_1(i)} w_{ij}^{\Delta} \Delta H_j^{(k+1)}\\
&\approx \sum_{j\in N_1(i)} w_{ij}^{\Delta} \sum_{l\in N_1(j)} w_{jl}^{\Delta} (\mathbf{n}_i \mathbf{n}_l^T) \mathbf{H}_l^{(k+1)}\\
&\approx \frac{1}{2} \sum_{j\in N_1(i)} w_{ij}^{\Delta} \sum_{l\in N_1(j)} w_{jl}^{\Delta} (\mathbf{n}_i \mathbf{n}_l^T) \sum_{m\in N_1(l)} w_{lm}^{\Delta} \mathbf{r}_m^{(k+1)}.
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
\mathbf{n}_i \langle \nabla H_i^{(k+1)}, \diamond H_i^{(k)}\rangle &\approx \sum_{j\in N_1(i)} \langle w_{ij}^{\nabla}, \diamond H_i^{(k)}\rangle \mathbf{n}_i H_j^{(k+1)}\\
&\approx \sum_{j\in N_1(i)} \langle w_{ij}^{\nabla}, \diamond H_i^{(k)}\rangle (\mathbf{n}_i \mathbf{n}_j^T) \mathbf{H}_j^{(k+1)}\\
&\approx \frac{1}{2} \sum_{j\in N_1(i)} \langle w_{ij}^{\nabla}, \diamond H_i^{(k)}\rangle (\mathbf{n}_i \mathbf{n}_j^T) \sum_{l\in N_1(j)} w_{jl}^{\Delta} \mathbf{r}_l^{(k+1)},
\end{aligned}
$$

where $\mathbf{n}_i := \mathbf{n}(\mathbf{r}_i^{(k)})$ is the surface normal at $\mathbf{r}_i^{(k)}$. Note that the discretized equation at the vertex $\mathbf{r}_i$ involves three-ring neighbor vertices.

Boundary condition. Suppose we are given a triangular surface mesh M with certain vertices are tagged as interior. The interior vertices are under the change. The remaining vertices are fixed. Using the above mentioned approximation of differential operators, we can discretize recursively the GPDE for each interior

vertex $\mathbf{r}_i$ and finally derive a linear equation. This equation is a linear combination of the three-ring neighbor vertices of $\mathbf{r}_i$.

$$\mathbf{r}_i^{(k+1)} + \tau \sum_{j \in N_3(i)} w_{ij} \mathbf{r}_j^{(k+1)} = \mathbf{r}_i^{(k)}, \quad w_{ij} \in \mathbb{R}^{3\times 3}.$$

If an involved vertex $\mathbf{r}_j^{(k+1)}$ is not an interior one, $\mathbf{r}_j^{(k+1)} = \mathbf{r}_j$ is fixed and the term $\tau w_{ij} \mathbf{r}_j^{(k+1)}$ is moved to the right hand side of the equation. Such a treatment of the boundary condition leads to a system of n equations with n unknowns. Here n is the number of interior vertices. The idea of this boundary treatment is adopted from [20].

Solving the linear system. The result system is highly sparse. An iterative approach for solving the system is desirable. We employ Saad's iterative method (see [13]), named GMRES to solve the system. The experiments show that this iterative method works very well.

Remark 4.2. The experiments show that the semi-implicit discretization scheme proposed equipped with Saad's solver of the linear system is very stable. The time step-size could be oftentimes chosen fairly large (see Table 5.1).

5 Illustrative Examples

Recover property to some surfaces. We have mentioned that constant mean curvature surfaces are the solutions of equation (3.1). Fig. 5.1 is used to illustrate that constant mean curvature surfaces can be recovered from their perturbed counterparts by MMCVF. The test is performed as follows. We first replace

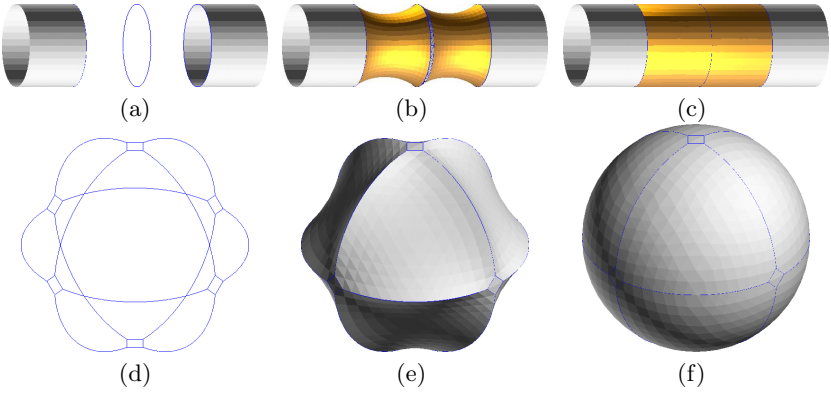


Fig. 5.1. (a) is a cylinder with certain parts are removed. (b) shows the minimal surface filling of the removed parts. (c) shows the evolution results. (d) is a wire-frame of a sphere with eight openings. These openings are filled with minimal surfaces as shown in (e). (f) shows the evolution result.

certain parts of a given constant mean curvature surface with another surface, and then we use our geometric flow to evolve the surface.

The first row of Fig. 5.1 shows that a cylinder is recovered, where (a) is a cylinder with certain parts missing. Figure (b) shows the minimal surface filling of the missing parts. This minimal surface acts as an initial surface $\mathcal{M}_0$ for the geometric flow. (c) shows the evolution result. It can be seen that the cylinder is correctly recovered. The second row of Fig. 5.1 shows that a sphere is recovered, where (d) is a wire-frame of a sphere with eight openings. These openings are filled with minimal surfaces as shown in (e). These minimal surfaces act as an initial surface $\mathcal{M}_0$ of the geometric flow. (f) shows the evolution result. It is easy to see that the sphere is perfectly recovered.

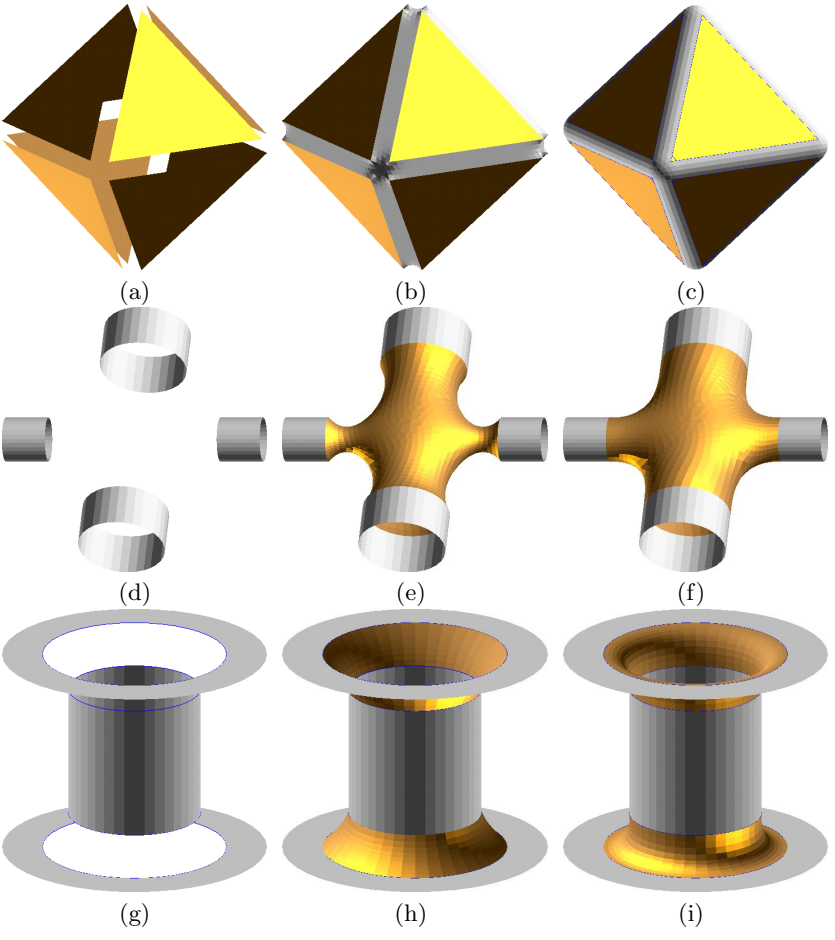
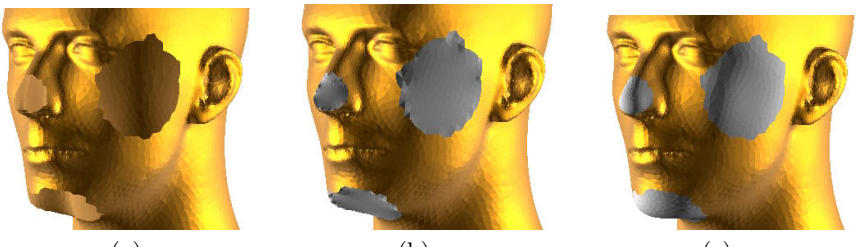


Fig. 5.2. Figure (a), (d) and (g) show surfaces to be blended with initial minimal surface constructions (figure (b), (e) and (h)). The surfaces (c), (f) and (i) are the blending meshes generated using MMCVF.

Smooth blending of surfaces. Given a collection of surface meshes with boundaries, we construct a fair surface to blend smoothly the meshes at the boundaries. Fig 5.2 shows the case, where surfaces to be blended are given (figure (a), (d) and (g)) with initial minimal surface constructions (figure (b), (e) and (h)) using [1] and then mean curvature flow. The surfaces (c), (f) and (i) are the blending meshes generated using our sixth-order flow.

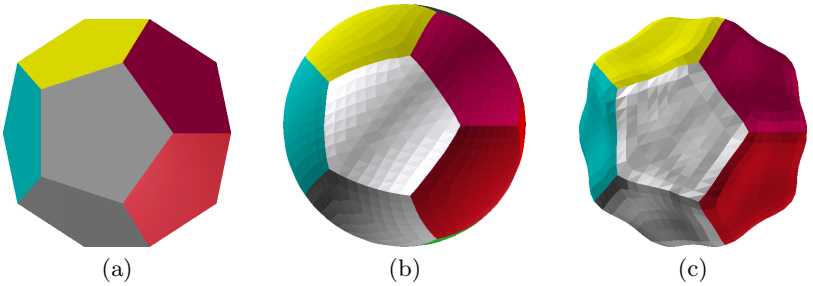
N-sided hole filling. Given a surface mesh with holes, we construct a fair surface to fill smoothly the holes with G^2 continuity on the boundary. Fig 5.3 shows such an example, where a head mesh with several holes in the nose, face and jaw subregions is given as input (figure (a)). An initial G^0 filler of the holes are shown in (b) using [1] and then evolved with the mean curvature flow. The blending surface (c) is generated using flow (3.8).

Point interpolation. For the point interpolation problem, we are given some points as the input data, and we wish to construct a fair surface mesh to interpolate this multi-dimensional data. Fig. 5.4 shows this surface construction approach, where a dodecahedron is served as input as shown in figure (a). The constructed surface is required to interpolate the vertices of the input polygon. Each face of the input polygon is triangulated by subdividing the 5-sided face



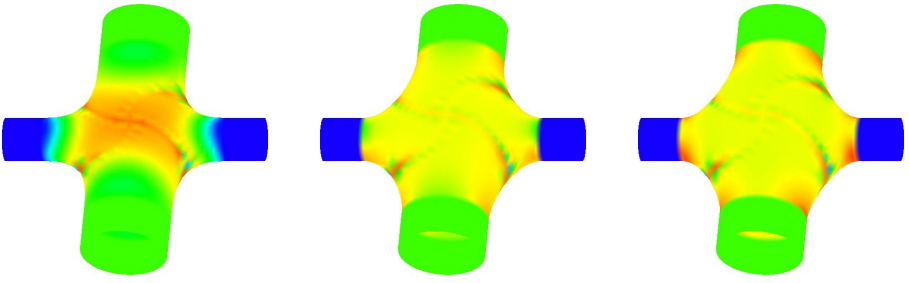
(a) (b) (c)

Fig. 5.3. (a) shows a head mesh with several holes. (b) shows an initial filler construction. (c) is the smooth filling surface, after 50 iterations, generated by using equation (3.8).



(a) (b) (c)

Fig. 5.4. (a) shows the input dodecahedron. (b) is the evolution result of MMCVF. The surface is required to interpolate the vertices of the input polygon. (c) shows an intermediate result of the evolution process.

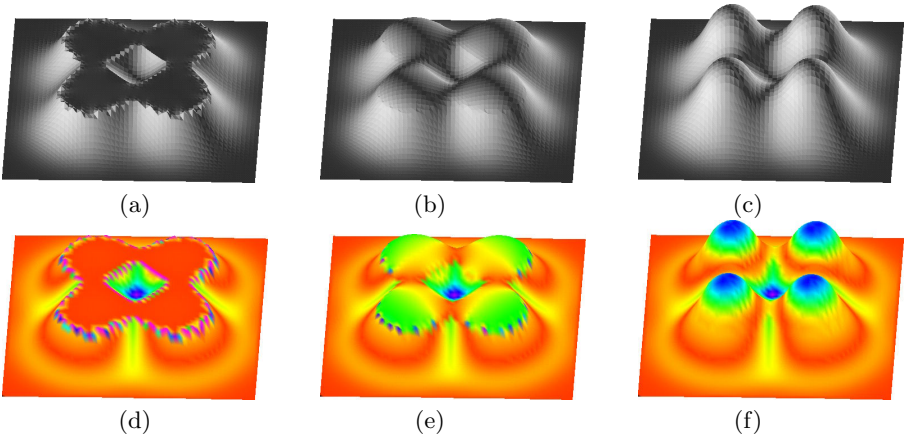


(a) (b) (c)

Fig. 5.5. (a), (b) and (c) are the mean curvature plots of the evolution results of MMCVF, SDF and WF, respectively.

into three triangles. Then each triangle is subdivided into 64 sub-triangles. The GPDE is applied to the triangulated polygon with the input vertices fixed. (b) is the evolution result of MMCVF. (c) shows an intermediate result of the evolution.

Comparison with lower order flows. Now we compare the used sixth-order flow MMCVF with three well-known lower order flows (see [20]): mean curvature flow (MCF), surface diffusion flow (SDF) and Willmore flow (WF). From the definition of MMCVF, we know that the main difference of MMCVF from the lower order flows is that the former yields G^2 and mean curvature uniformly distributed surfaces. Fig. 5.5 shows the evolution results of the sixth- and fourth-order flows for the input (d) of Fig. 5.2, where (a), (b) and (c) show the mean curvature plots of the evolution results of the MMCVF, SDF and WF, respectively. From these figures, we can observe that the surface produced by MMCVF is mean-curvature continuous at the blending boundaries, while the surfaces produced by SDF and WF are not.



(a) (b) (c)

(d) (e) (f)

Fig. 5.6. (a), (b) and (c) are the evolution results of the MCF, WF and MMCVF. (d), (e) and (f) are the mean curvature plots of (a), (b) and (c), respectively.

Table 5.1. Running Times

Examples	# unknowns	Time step-size	Form matrix	# steps	Solving Time
Fig 5.1(c)	3480	1.0	0.12	100	2.11
Fig 5.1(f)	11160	1.0	0.42	350	6.72
Fig 5.2(c)	3222	1.0	0.12	5	1.62
Fig 5.2(f)	4410	1.0	0.17	50	2.75
Fig 5.2(i)	2400	0.01	0.09	10	1.43
Fig 5.3(c)	1296	0.5	0.05	10	1.26
Fig 5.6(c)	2520	2.0	0.09	5	1.50

In Fig. 5.6, the surface to be evolved is defined as a graph of a function g: $\mathbf{x}(u,v) = [u, v, g(u,v)]^T$, $g(u,v) = e(u,v)+e(u+1,v)+e(u,v+1)+e(u+1,v+1)$, with $(u,v) \in \Omega := [-1,1]^2$ and $e(u,v) = \exp[-\frac{81}{16}(u-0.5)^2 + (v-0.5)^2]$. This surface is uniformly triangulated using a 60×60 grid over the domain Ω. We evolve a part of the surface, where $g > 1.5$. Figures (a), (b) and (c) show the results of MCF, WF and MMCVF, respectively. Figures (d), (e) and (f) are the mean curvature plots of (a), (b) and (c), respectively. It is easy to see that the second and the fourth-order flows are not curvature continuous at the boundaries of the evolved surface patch.

Running Times. We summarize in Table 5.1 the computation time needed by some of our examples. The algorithm was implemented in C++ running on a Dell PC with a 3.0GHz Intel CPU. All the examples presented in this section are the approximate steady solution ($t \to \infty$). Hence, the total time costs depend greatly on how far we go in the time direction, which in turn depend on how far the initial surface away from the final solution. In Table 5.1, we list the costs for a single iteration. The second column in Table 5.1 is the number of unknowns. These numbers are counted as $3n_0$ (each vertex has x, y, z variables). Here n_0 is the number of interior vertices. The third column is the used time step-size. The fourth column in the table is the time (in seconds) for forming the coefficient matrix. The fifth column is the number of evolution steps. The last column is the time for solving the linear systems for one time step.

6 Conclusions

We have derived a sixth-order nonlinear geometric flow from the functional $\mathscr{F} = \int_{\mathcal{M}} \|\nabla H\|^2 dA$. We name it as minimal mean-curvature-variation flow. This flow can be used to solve several surface modeling problems, such as surface denoising, surface blending, N-sided hole filling and free-form surface design, when G^2 continuity on the boundary is required. The experimental results show that the semi-implicit discretization using the divided-difference-like method equipped with Saad's solver of the linear system is efficient and stable for solving this sixth-order nonlinear equation. Comparative results also show that the proposed sixth-order flow yields high quality and high order continuity surfaces.

References

1. C. Bajaj and I. Ihm. Algebraic surface design with Hermite interpolation. *ACM Transactions on Graphics*, 11(1):61–91, 1992.
2. G. P. Bonneau, H. Hagen, and St. Hahmann. Variational surface design and surface interrogation. *Computer Graphics Forum*, 12(3):447–459, 1993.
3. M. Botsch and L. Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Transaction on Graphics*, 23(3):630–634, 2004. Proceedings of the 2004 SIGGRAPH Conference.
4. H. Du and H. Qin. Dynamic PDE-based surface design using geometric and physical constraint. *Graphical Models*, 67(1):43–71, 2005.
5. A. Gray. *Modern Differential Geometry of Curves and Surfaces with Mathematica*. CRC Press, second edition, 1998.
6. G. Greiner. Variational design and fairing of spline surface. *Computer Graphics Forum*, 13:143–154, 1994.
7. M. Kallay. Constrained optimization in surface design. In B. Falcidieno and T. L. Kunii, editors, *Modeling in Computer Graphics*, pages 85–93. Springer-Verlag, Berlin, 1993.
8. E. Kuwert and R. Schätzle. The Willmore flow with small initial energy. *J. Differential Geom.*, 57(3):409–441, 2001.
9. H. B. Lawson. *Lectures on Minimal Submanifolds*. Publish or Perish, Berkeley, CA, 1980.
10. U. F. Mayer. Numerical solutions for the surface diffusion flow in three space dimensions. *Comput. Appl. Math.*, 20(3):361–379, 2001.
11. H. P. Moreton and C. H. Séquin. Functional optimization for fair surface design. *SIGGRAPH'92 Conference Proceedings*, pages 167–176, 1992.
12. J. Oprea. *Differential Geometry and Its Applications*. Pearson Education, Inc., second edition, 2004.
13. Y. Saad. *Iterative Methods for Sparse Linear Systems*. Second Edition with corrections, 2000.
14. R. Schneider and L. Kobbelt. Geometric fair meshes with G^1 boundary conditions. In *Geometric Modeling and Processing*, pages 251–261, 2000. Hongkong, China.
15. W. Welch and A. Witkin. Variational surface modeling. *Computer Graphics*, 26:157–166, 1992.
16. T. J. Willmore. *Riemannian Geometry*. Clarendon Press, Oxford, England, 1993.
17. G. Xu. Discrete Laplace-Beltrami operators and their convergence. *Computer Aided Geometric Design*, 21(8):767–784, 2004.
18. G. Xu. Consistent approximation of some geometric differential operators. Research Report No. ICM-06-01, Institute of Computational Mathematics, Chinese Academy of Sciences, 2006.
19. G. Xu. Convergence analysis of a discretization scheme for Gaussian curvature over triangular surfaces. *Computer Aided Geometric Design*, 23(2):193–207, 2006.
20. G. Xu, Q. Pan, and C. L. Bajaj. Discrete surface modelling using partial differential equations. *Computer Aided Geometric Design*, 23(2):125–145, 2006.
21. L. H. You, P. Comninos, and J. J. Zhang. PDE blending surfaces with C^2 continuity. *Computers and Graphics*, 28(6):895–906, 2004.
22. J. J. Zhang and L. H. You. Fast surface modelling using a 6th order PDE. *Comput. Graph. Forum*, 23(3):311–320, 2004.

Parametric Design Method for Shapes with Aesthetic Free-Form Surfaces

Tetsuo Oya[1], Takenori Mikami[1], Takanobu Kaneko[2], and Masatake Higashi[1]

[1] Toyota Technological Institute, Hisakata 2-12, Tenpaku, Nagoya, Japan
[2] AISIN Seiki Co., Ltd., Asashi-2-1, Kariya, Aichi, Japan

Abstract. In this paper, a parametric design method for aesthetic shapes which provides a direct and interactive modeling is presented. As used in an actual automobile design, our method utilizes a curve-ruler to create a free-form surface. In order to construct a surface, a curve-ruler is defined by a function like a polynomial at first. And then, two guide curves are generated by Bézier curves. Moving a curve-ruler on guide curves, a free-form surface is obtained as its locus. Designer only designates a type parameter of the curve-ruler and the parameters to allocate the curve-ruler on the guide curves. To construct a whole shape of a product, we introduce another set of parameters to control an outer shape. In this way, it is possible to control the local surface and the global shape independently. Moreover, the methods to generate a fillet, trimmed surface and features are presented. We have implemented a prototype system and will demonstrate its application on an automobile.

1 Introduction

The importance of styling in industrial design has been increasing. In addition to the product's price and quality, its aesthetic aspect influences the value of the product. Especially, in the automotive industry, its appearance is a decisive factor to catch the customer's attention. Aesthetic design is a realization of designer's idea, as well as a process to produce a product which satisfies the regulations. In other words, the design process consists of several stages where many requirements, that are styling, producibility, safety, quality and so on, should be considered simultaneously.

This paper focuses on the phase of styling. In order to create an aesthetic shape, many supporting systems have been studied. Igarashi et al. [1] presented a sketching interface for freeform design. This system enables users to draw a freeform shapes on screens by hand and its 3D model is constructed from the 2D sketches. With this system designers can draw his intension on screen, though, it is not enough to evaluate its quality with curvature variation. Aoyama et al. [2] presented an aesthetic design system that enables users to form a shape with a sketch and design language. By this system, a rough shape is drawn by sketching and a 3-D model is constructed in a computer. Then, its shape is modified using design language. Finally, characteristic lines are input for adding details to finish the aesthetic design. Although this method provides an effective

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 371–384, 2006.
© Springer-Verlag Berlin Heidelberg 2006

interface to industrial designers, there remains some limitations. This system can form and modify the whole shape of the object, while it is impossible to handle individual surfaces.

There have been many studies which adopt the *feature* concept to construct a design method. The term feature means any perceived geometric or functional element of an object [3] and has been used in mechanical engineering offering meaningful elements consist of lower-level entities [4]. Although feature based approaches have been adopted by many CAD systems and studied in literature [5], features in these studies are usually for solid modeles and restricted in its degree of freedom in the aspect of modification. To be a more flexible and designer-friendly interface, Cheutet et al. presented a method which combined 3D sketching and free form feature modeling [6]. This method has a three level of design processes from semantic to geometric level in which NURBS [7] based free-form feature modeling is described, which enables users to create and change surfaces. However their method is based on NURBS technique, each feature would consist of many control points. Therefore shaping and modifying would be a time-comsuming task. Besides, their method is not connected with the total design of a product.

In order to be an effective and easy-to-use tool for industrial designers, a design method which has flexibility in surface manipulation and compactness of data, namely fewer input points or a restricted number of parameters is required.

In an actual automobile design field, a "curve-ruler" is used to form the shape of a clay model which is included from the designer's sketch or mock-up data. The curve-ruler has several patterns with different curvature distribution to draw various types of monotone and smooth curves. Moving this ruler along guide curves, an aesthetic free-form surface is created as its locus [8]. We consider the concept of the curve-ruler is suitable for the design of an automobile because it is used in the actual design, therefore, the movement of a curve-ruler is formulated in our method to express surfaces with parameters.

In order to express a surface with a curve-ruler as its locus, we planned to construct a mathematical expression with a few parameters instead of using Bézier curves or other spline techniques. The reason is that using Bézier or spline will lead to the huge number of points to be input by designers, and this is not suitable for interactive and intuitive design method.

There are many candidates for the expression of a curve-ruler, we adopt a polynomial of degree two in this paper based on Harada 's study [9] [10] [11]. Harada introduced the concept of *rhythm* and *volume* to evaluate the characteristics of curves. Harada studied about 2-D curves whose curvature changes monotonically. The definition of *volume* is an area surrounded by the curves itself and the chord. The term *volume* expresses thickness or thinness of the curve. On the other hand, *rhythm* is evaluated the frequency of length $\bar{s}_j = \log\left(s_j/S_{\text{all}}\right)$ where S_{all} is the total length of the curve and s_j means an infinitesimal difference of the curve. Slope β of $\bar{s}_j$ in a logarithm distribution is considered to affect the impression of the curve. Harada classified the type of curves by the sign of the slope β. Based on this concept, studies of an aesthetic curves have

been presented to find out its expression [12]. We chose a simplest polynomial, $y = ax^2$, as the function of the curve-ruler. The polynomial of degree two has a positive sign of β and it has a monotonous characteristics in curvature variation without the extreme value.

This paper also presents a method to handle the whole shape of a product and each local surfaces independently. We introduce schematic parameters which control the whole shape and surface parameters which control local surfaces. Surface parameters are the parameters which is already mentioned above embedded in the expression of local surfaces. Schematic parameters are used to determined the outer shape of a product and it affects the end points of the guide curves. In this paper, we propose a method for an aesthetic design to generate a free-form surface. Surfaces are obtained by moving the curve-ruler along two guide curves as its locus, and it is expressed with a few parameters. With these parameters, namely the surface parameters, designers are able to create and modify each surfaces. In addition, schematic parameters are introduced to control the whole shape of a product. Using these two types of parameters, design process will be conducted intuitively. Section 2 describes the surface parameters and the expression of the ruler surface. Then, section 3 explains the concept of schematic parameters with an example. Furthermore, we presents some features which also consists of a few parameters. In section 4, classification of features is done and the examples of each type of features are presented. Section 5 shows results of the generation and modification of a model of an automobile by changing both schematic parameters and surface parameters. Our method's effectivity is revealed through this application results.

2 Surface Generation by a Curve Ruler

2.1 Concept of Our Method

A curve-ruler which is a ruler expressing various kind of curves is used in an actual design field of automobiles. Curve-rulers are made based on various mathematical expressions so as to generate smooth curves whose curvature variation is monotonous. Figure 1 shows a picture of curve-rulers. In order to generate a free-form surface, a designer move a curve-ruler and a surface is obtained as its locus. Using a spline technique, it would be difficult to keep smoothness of a surface and designers should handle numerous control points to modify a surface. On the other hand, using a curve-ruler compensate the smoothness of a surface and it is easy to modify a surface shape because designers are only required to choice the type of a curve-ruler and constraint conditions. We have constructed a free-form surface generation method based on this design process. At first, a curve-ruler is expressed by a polynomial of degree two. Although various functions can be used as an expression of a curve-ruler, we adopt a polynomial of degree two because of its simplicity and smoothness in curvature variation. Next, two guide curves are introduced as a "rail" of a curve-ruler. Then, a surface is generated by moving a curve-ruler along these guide curves. Here some constraint condition is necessary to allocate a curve-ruler on guide curves. In this paper we use

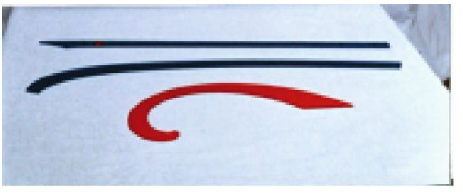


Fig. 1. Actual curve-rulers

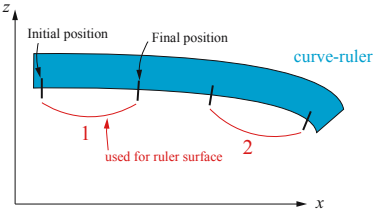


Fig. 2. Different region in a curve-ruler

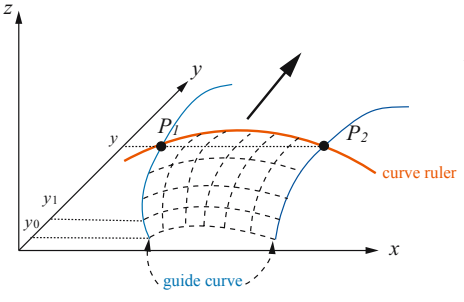


Fig. 3. Surface generation concept

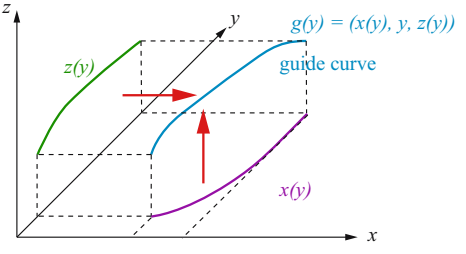


Fig. 4. Construction of a guide curve by projection

a tangent condition where an angle of the tangent of a curve-ruler on a guide curve is designated. While a curve-ruler is moving this constraint condition is kept. Parameters are included in the expression of a surface to specify the type of a curve-ruler and the constraint condition. By changing these parameters, modification of a surface is easily performed.

2.2 Surface Generation Process

As the function of a curve-ruler, we adopt the following polynomial of degree two:

$$z(x) = ax^2 + bx + c. \tag{1}$$

The value a is specified by a designer and this value designates the type of the curve-ruler. Coefficients b and c are obtained in later process. Next, guide curves are introduced as a space curve. Two guide curves are builed in the coordinate system (x, y, z) and the curve-ruler on them moves in the y direction as depicted in Fig. 3. Each guide curves are generated by two Bézier curves of degree two, which exist in a two planes of projection, that is, x-y plane and y-z plane. By composing these two Bézier curves, we have a guide curve $g(y) = (x(y), y, z(y))$ as shown in Fig.4. Two guide curves $g_1(y)$ and $g_2(y)$ are required to generate a surface and these curves are represented in the rotated coordinate system (x', y, z') to introduce a tangential angle θ of the curve-ruler. By using the rotation matrix, that is,

$$\begin{Bmatrix} x_1'(y) \\ z_1'(y) \end{Bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{Bmatrix} x_1(y) \\ z_1(y) \end{Bmatrix}, \tag{2}$$

$$\begin{Bmatrix} x_2'(y) \\ z_2'(y) \end{Bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{Bmatrix} x_2(y) \\ z_2(y) \end{Bmatrix} \tag{3}$$

, we have two guide curves $g_1(y) = (x_1'(y), y, z_1'(y))$ and $g_2(y) = (x_2'(y), y, z_2'(y))$. In z'-x' plane, points P_1 and P_2 on these guide curves are also the points on a curve-ruler as shown in Fig. 5, thus we have

$$z_1'(y) = ax_1'(y)^2 + bx_1'(y) + c, \tag{4}$$
$$z_2'(y) = ax_2'(y)^2 + bx_2'(y) + c. \tag{5}$$

By specifying the parameter a, remaining parameters b and c are obtained by solving the simultaneous equations (4) and (5) :

$$b(y) = \frac{a\left(x_2'(y)^2 - x_1'(y)^2\right) + z_1'(y) - z_2'(y)}{x_1'(y) - x_2'(y)}, \tag{6}$$

$$c(y) = \frac{a\left(x_1'(y)x_2'(y)^2 - x_1'(y)^2x_2'(y)\right) + z_1'(y)x_2'(y) - x_1'(y)z_2'(y)}{x_2'(y) - x_1'(y)}. \tag{7}$$

When the value a is a function of y, the equation of the curve-ruler becomes $z'(y) = a(y)x'^2 + b(y)x' + c(y)$ where $x' = x'(x, y) = (1 - x)x_1'(y) + xx_2'(y)$. The angle θ is still undetermined and we apply a constraint condition to fix the tangential direction of the curve-ruler. Differentiating the curve-ruler at a point on the guide curve, it equals to the rotation angle of the curve ruler as depicted in Fig.6 and we have

$$2a(y)x_1'(y) + b(y) = \tan(\theta + \alpha) \tag{8}$$

where α denotes the angle shown in the same figure. This α is the second parameter to be input by users to specify the tangential direction of the curve-ruler.

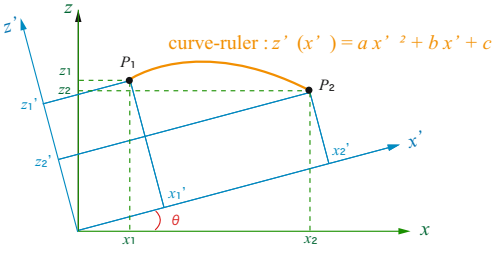


Fig. 5. View of x-z plane in the coordinate system where the curve-ruler surface exists

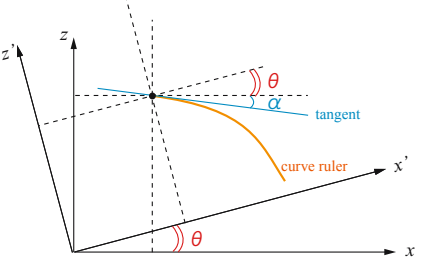


Fig. 6. Definition of θ and α

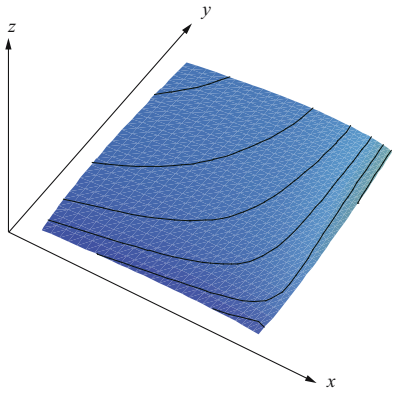


Fig. 7. Surface with highlight lines

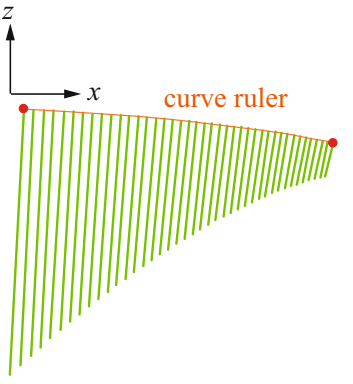


Fig. 8. Curvature profile of the curve ruler

Therefore, θ can be obtained by solving Eq. (8) at any y as $\theta(y)$. Note that one α is used for the constraint condition at one end of the curve-ruler. The choice which end is used is dependent on the designer's intension. In this case, P_1's end is used. Then, in our implementation, $\theta(y)$ is expressed as an interpolation function of several points that satisfy the constraint condition on the guide curve. In addition, designer can choose the region of the curve-ruler to be used by the constraint condition as shown in Fig.2. Designer will choose region 1 or 2 dependent on his intention. Region 1 has slower variation of curvature change, on the other hand region 2 has sharp change. In order to obtain the expression of a surface, the coordinate system (x, y, z) should be used instead of (x', y, z'), therefore, rotation matrices with angle $-\theta$ is applied and the final form is obtained:

$$S(x,y) = a(y)\cos\theta(y)x'(x,y)^2 + \{b\,(y)\cos\theta(y) - \sin\theta(y)\}\,x'(x,y) + c\,(y)\cos\theta(y). \tag{9}$$

Example of surface generation by our method is presented. In the case that $a = 0.1$, $\alpha = 0.05°$, the curve-ruler surface is depicted in Fig.7 with highlight lines and the curvature profile of the curve-ruler is illustrated in Fig.8. Here curvature profile means a distribution of curvature along a curve. In an aesthetic design field it is required to evaluate a surface with its curvature variation. As shown by these pictures, the presented method enables the designers to evaluate the surface in terms of curvature because it is differentiable at any position. This leads to the generation of high quality surface as the designers want.

3 Parametric Model of an Aesthetic Shape

Previous section described the method to generate a surface which is controlled by two parameters. This set of parameters is called surface parameters. In order

to form a whole shape of a product, however, other kind of parameters are required. This section introduces these parameters, namely schematic parameters, to determine the product's outer shape. Using these two types of parameters, designers can create and modify both the local and the global shape independently. As a demonstration, an automobile is adopted to present the concept.

In order to complete the whole styling of an automobile, it is necessary to determine the values that prescribe the outer form. Actually, such parameters that specify the length, width and height of an automobile and so on are required. Aoyama et al. [2] presented the method that connects the parameters that control the dimension of an automobile with semantics. In their method, inputing some word such as "cute" or "sporty" changes the parameters to achieve the desired shape. Ujiie et al. [13] studied how the change of the curvature affects human recognition using the automobile modeled by Bézire curves. In these two methods, it is not enough to modify surfaces locally for an aesthetic design. The automobile model is placed in coordinate system (x, y, z) as depicted in Fig.9. The front of the automobile is in the direction of negative x, and the width is parallel to y axis whose origin is a mid point of the width. Axis z is in the direction of the height. When changing the outer shape, a designer input three parameters : L, W and H. Here L means length of a model expressing the displacement of control points in the direction of x, W and H are width and height respectively. These are schematic parameters. In order to modify the outer shape, designers move control points of Bézier curves for modifying guide curves. L, W and H means displacement values that are input by the designer to change the position of the control points.

This concept is demonstrated by the following example. By moving two control points, the outer shape of an automobile is changed. The control point to be moved is an end point of a guide curve. In this model, symmetry to the x-z plane is assumed. Inputing schematic parameters to these points, the position of the point is moved and this change influences the whole shape because all of the guide curves are shared by the adjacent surfaces. For example, point P_1 which is shared by the bonnet surface and the front glass surface and point P_2 which connects the rear glass surface and the trunk surface are moved as shown in Fig. 10. Figures 11 and 12 depict the before and after shape of an automobile expressed by guide curves. These figures show that our concept is realized. As for the continuity, C^2 continuity is maintained in surfaces that are used for bonnet or roof at the central line because of the symmetry. Then, fillet is connected with

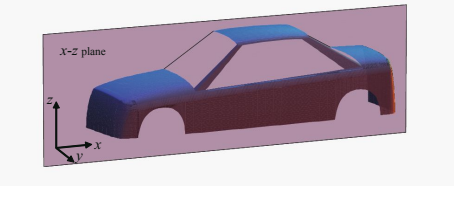


Fig. 9. Whole model of an automobile

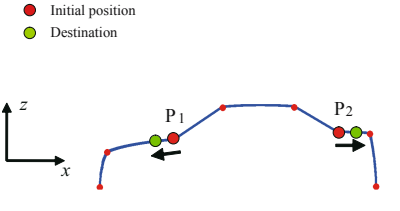


Fig. 10. Changes of schematic parameters

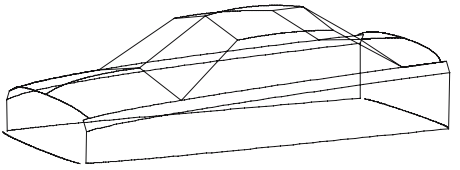


Fig. 11. The model expressed by guide curves

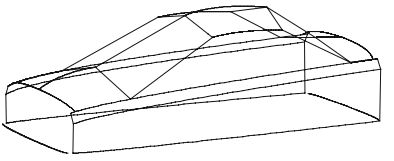


Fig. 12. The result of the changes of the schematic parameters

C^2 to other surfaces. However, C^2 is not guaranteed between fillets. In other cases, individual surfaces are connected with C^0. Finally, it should be stated that in our implementation only a sedan type is utilized for the outer shape now. Namely, the number of guide curves and surfaces and its topology is fixed beforehand.

4 Parametric Expression for Features

Using the method that has been described in the previous section of this paper, designers can create an aesthetic shape with surfaces. And trimmed surface and fillet can be generated. However, the actual shape has more detailed shapes like a side molding or wheel arch. Therefore, we present in this section a method to construct general expressions for these parts as features.

4.1 Classification of Features

The region where a feature is generated is defined as the definition region. We classify features into a few types. Feature is distinguished by its external characteristics. If some shape is added to the definition region it is an additive feature. If some shape is eliminated from a surface, it is a subtractive feature. As an additive feature, we define three features: boss feature, step feature and pocket feature. The definition region of boss feature and pocket feature exists within a surface. In other words, these are convex and concave appearances of the same feature. When the definition region of an additive feature crosses the boundary of a surface, it is step feature. As a subtractive feature, we define two features : inlet feature and hole feature. When the eliminated region is completely included within a surface it is hole feature. And in the other case of a subtractive feature, it is inlet feature.

These classification is useful for an automobile design. Namely, boss feature expresses a side molding or character line and step feature represents any stepped shape like a bumper. Hole feature can be used to express intakes and inlet feature is useful to express wheel arches. Examples of these features are presented below.

4.2 Boss Feature

Boss feature consists of two parts, that is, a main shape and two end shapes. And its shape is modified by changing three parameters included in an expression of

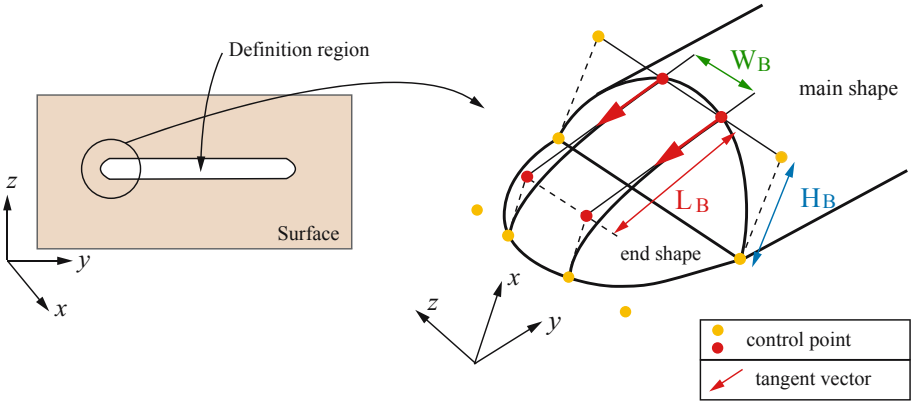


Fig. 13. The concept and details of a boss feature

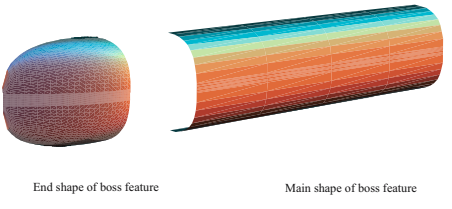


Fig. 14. An end shape(left) and a main shape(right) of a boss feature

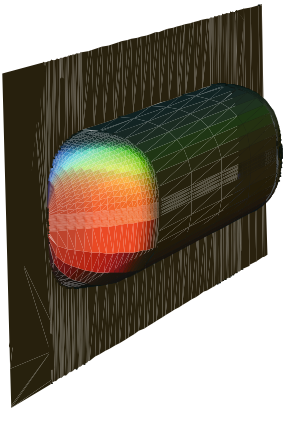


Fig. 15. Side molding of an automobile created by the boss feature

boss feature. Figure 13 shows the process to generate a boss feature. First, the definition region is trimmed with two curves as shown in the left of the figure. Next, at the point where a curve ruler begin sweeping, the section of the main shape is generated by Bézier curve of degree five within x-z plane and this is a section curve of boss feature. Here the position of control points of the Bézier curve are determined and moved by two parameters, that is, W_B and H_B as shown in the same figure. Then, the end shape is generated by using this curve. In order to connect two shapes smoothly, tangent vectors are calculated to satisfy C^1 continuous condition in the x-y plane at the central two control points that are expressed by red points in the figure. On the tangents, two new points are generated with distance L_B from the section curve of boss feature. And this is the third parameter to be input. Using these points as middle control points, two Bézier curves of degree two are generated. Curve-ruler moves along these curves,

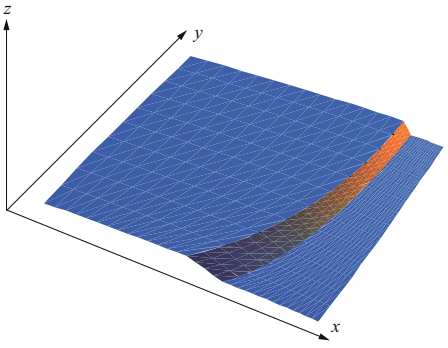


Fig. 16. Step feature

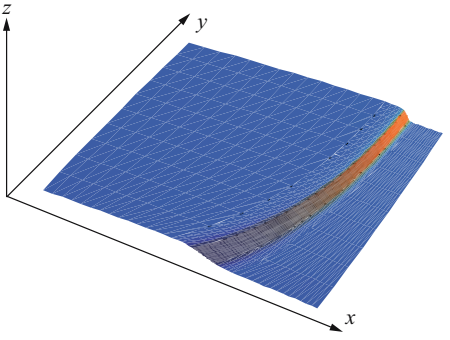


Fig. 17. Step feature with fillet

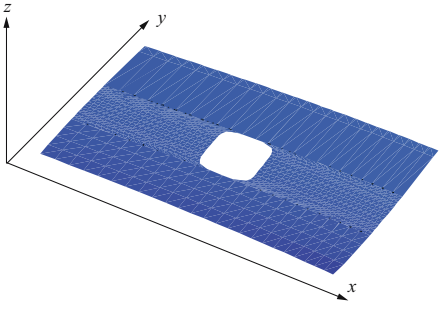


Fig. 18. Hole feature

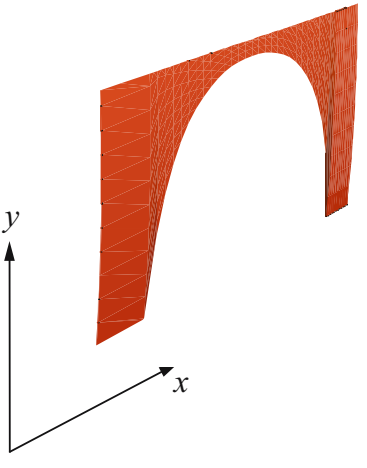


Fig. 19. Inlet feature

the end shape is obtained as shown in Fig. 15. This feature can be modified by three parameters mentioned above. Changing the parameter H_B from positive to negative, a pocket feature is obtained.

4.3 Step Feature

By trimming, a surface is divided into two parts. The surface that is indicated to be lifted is extruded with parameter L_S. The gap between two surfaces is connected by a ruled surface. In order to generate more complex stepping shapes, two trimming curve is applied to cut a surface into three parts. Then, the central part is eliminated and the others are connected with a ruled surface. Example is presented in Fig. 16. To avoid discontinuity on the connecting lines, fillet is used to produce a smooth feature shape. Figure 17 represents an example where the connecting lines are rounded by fillet.

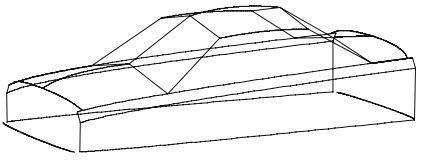


Fig. 20. First stage: A wire-framed body by guide curves

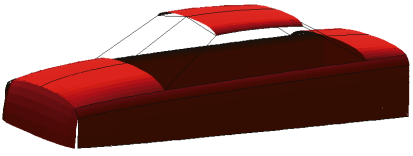


Fig. 21. Second stage: Surfaces are generated between guide curves

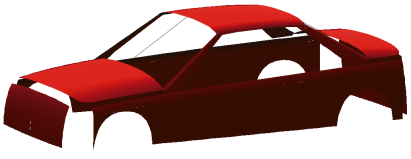


Fig. 22. Third stage: Trimming is concudted

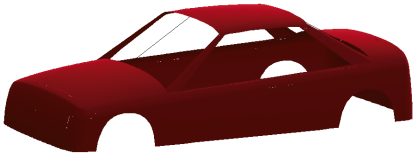


Fig. 23. Final stage: Each surfaces are connected by fillets

4.4 Hole Feature

Projecting a curve onto a surface, a hole or an inlet is generated by trimming. Example of a hole feature is shown in Fig. 18 and that of an inlet is presented in Fig. 19.

5 Application

5.1 Process

This section presents application examples of our method using a process of automobile construction. We used MATHEMATICA to implement our algorithm. At first, guide curves including schematic parameters express character lines and an outer shape of an automobile as shown in Fig.20 . Next, inputting surface parameters a whole shape of the automobile roughly appears in Fig.21 . Then, trimming is conducted on this model as shown in Fig.22. Finally, using fillet, the construction process of the model of an automobile is completed as depicted in Fig. 23.

5.2 Changing Schematic Parameters

We demonstrate changes of schematic parameters to modify the model's outer shape. Such parameters those of bonnet and trunk in the direction of y axis, those of roof in z are changed. The result of this change is shown in Fig.24 and 25.

5.3 Changing Surface Parameters

Here we provide an example which shows how surface parameters affect the whole shape and impression of an automobile. The parameters a that determines a type

Table 1. Value of parameter a

	Bonnet	Roof	Trunk
Initial values	0.21	0.2	0.2
Changed to	0.05	0.1	0.05

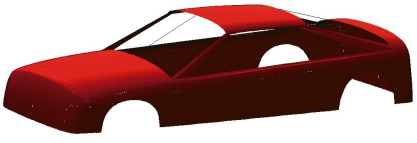


Fig. 24. The body before changing

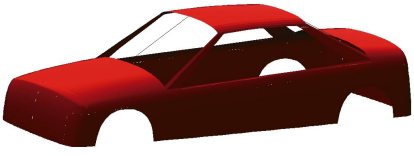


Fig. 25. Result of changing of schematic parameters

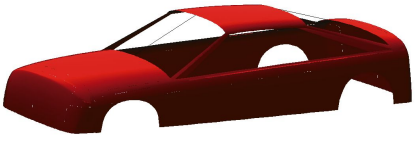


Fig. 26. The initial body

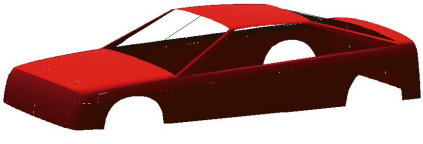


Fig. 27. Result of changing of surface parameters

of a curve ruler in bonnet, roof and trunk are changed as in Table 1. The results are presented in Fig.26 and 27 .

5.4 Addition of features

Features that are described in the previous section are added to the model to be a more realistic one. Boss features as side molding, step features as a front and a rear bumper are introduced. These three pictures, Fig. 28, 29, 30, show the result of this demonstration.

Fig. 28. Front view of a sedan with features

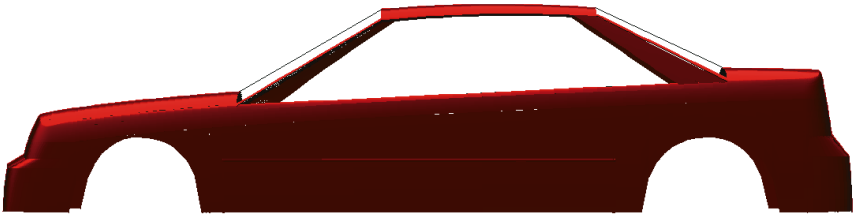


Fig. 29. Side view of a sedan with features

Fig. 30. Back view of a sedan with features

6 Conclusion

We proposed a method to generate an aesthetic surface for designers. Through an application constructing an automobile, the method show its effectiveness in both surface creation and modification. The following is a summary of our proposal:

1. Surface generating method using a curve-ruler and guide curves is presented. This surface is parametrically formulated, therefore, surface modification is conducted only by changing the surface parameters.
2. Defining schematic parameters, an outer shape of the product can be independently controlled by these parameters.
3. Classification of features that are useful for industrial design is presented. Features are expressed by a few parameters, thus, addition and modification of features are easily conducted.

Demonstrations on an automobile conclude that a model is easily created with a small number of parameters and modified locally and globally. The examples also show that the features defined in this paper add details on the body of an automobile. Features are generated by a few parameters and its shape can be easily changed as well.

There are some obstacles to be conquered. Other types of a curve-ruler are required to be more flexible design method. In the early stage of design by our method, various basic shapes should be available. By transferring the computation

environment from MATHEMATICA, fast computing time that is enough for interactive design should be accomplished. And GUI for designers is important to be a practical tool.

Acknowledgements

This study was financially supported by the High-tech Research Center for Space Robotics from the Ministry of Education, Sports, Culture, Science and Technology, Japan.

References

1. Igarashi, T., Matsuoka, S., Tanaka, H., 1999. A Sketching Interface for 3D Freeform Design. SIGGRAPH'99, 409.
2. Aoyama, H., Urabe, Y., Ohta, M., Kusunoki, T., 2003. Aesthetic Design System Based on Sketch, Design Language, and Characteristic Lines. CIRP Journal of Manufacturing Systems, Vol 32, No 2.
3. McMahon, C., Browne, J., 1998. CAD CAM 2nd Edition. Prentice Hall.
4. Shah, J. J., Mantyla, M., 1995. Parametric and feature-based CAD/CAM. Wiley-Interscience Publication. John Wiley & Sons, Inc..
5. Fontana, M., Giannini, F., Meirana, M., 1999. A Free Form Feature Taxonomy. EUROGRAPHICS'99, Vol. 18.
6. Cheuter, V., Catalano, C. E., 2004. 3D Sketching with Fully Free Form Deformation Features(σ-F) for Aesthetic Design. EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling.
7. Farin, G., 2002. Curves and Surfaces for CAGD 5th Edition. Morgan Kaufmann.
8. Higashi, M., Kohzen, I. and Nagasaka, J., 1983. An interactive CAD system for construction of shapes with high-quality surface. Proc. of the First International IFIP conference on Computer Applications in Production and Engineering CAPE'83, North-Holland Publishing Company, pp371–390.
9. Harada, T., 2004. Study on Automobile Design Assistance by Example of Curves(JPN). The 18th Annual Conference of the Japanese Society for Artificial Intelligence. 1E3-02.
10. Harada, T., 1999. An aesthetic curve in the field of industrial design. Proc. of IEEE symposium on VL'99.
11. Harada, T., 2001. Automatic Curve Fairing System Using Visual Languages. Proc. of IV2001, pp53–62.
12. Yoshida, N., Saito, T., 2005. Aesthetic Curve Segment(JPN). IPSJ SIG Technical Report. CG-121(17).
13. Ujiie, Y., Matsuoka, Y., 2001. Shape-Generation Method Using Macroscopic Shape-Information. Transactions of the Japan Society of Mechanical Engineers(C). Vol.67, No.664, pp3930–3937.

Control Point Removal Algorithm for T-Spline Surfaces

Yimin Wang and Jianmin Zheng

School of Computer Engineering
Nanyang Technological University
50 Nanyang Avenue, Singapore 639798
{wang0066, asjmzheng}@ntu.edu.sg

Abstract. This paper discusses the problem of removing control points from a T-spline control grid while keeping the surface unchanged. An algorithm is proposed to detect whether a specified control point can be removed or not and to compute the new control points if the point is removable. The algorithm can be viewed as a reverse process of the T-spline local knot insertion algorithm. The extension of the algorithm to remove more control points is also discussed.

1 Introduction

In the areas of geometric modeling and computer graphics, a popular mathematical representation for free form surfaces is B-splines (or NURBS) [1]. B-spline basis functions can be refined by linear transformation and this important property enables the operation of B-spline knot insertion [2,3]. By knot insertion, the number of the knots in a B-spline surface is increased and the shape of the surface can thus be modeled at a finer detail level. A reverse process of B-spline knot insertion is B-spline knot removal [4,5], which aims to eliminate redundant knots from a B-spline surface without altering its shape. While knot insertion can always be performed without introducing errors, removing a knot without changing the surface is possible only under certain circumstances. Therefore, in general, approximation algorithms will be used for B-spline knot removal [6,7].

One drawback of B-spline surface knot insertion and knot removal is that, due to the restriction on the topology of B-spline surfaces, knots can only be added or removed in a row-wise or column-wise fashion in order to make the B-spline control mesh a regular grid. To overcome this inflexibility, a new surface representation called T-splines [8] was recently developed, which is actually a generalization of B-splines. In a T-spline surface a row or column of control points is allowed to terminate and the final control point of the partial row or column is called a T-junction. One important advantage of T-splines is that T-splines allow local refinement.

In this paper, we study the reverse process of inserting control point(s) into a T-spline surface, i.e., T-spline control point removal. Two questions are tackled: the first one is to detect whether a specified T-spline control point is able to be removed; and the second one is to compute the updated topology and geometry

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 385–396, 2006.
© Springer-Verlag Berlin Heidelberg 2006

of the T-spline surface after a removable control point is removed. Compared to the B-spline knot removal in which a whole row (or column) of control points needs to be removed, our control point removal for T-splines focuses on the removal of a single control point, which usually causes only local change to the T-spline control grid.

Previous work of T-spline control point removal was reported in [9] where the problem of T-spline surface simplification was considered. The method starts with a simple B-spline surface defined by a 4×4 control grid, and then adaptively refines the grid until the least squares T-spline surface defined over the refined grid approximates the original T-spline surface within the given tolerance. If the tolerance is chosen to be zero, then the control point removal can be achieved. The method is global in nature and is useful for eliminating as many control points as possible. In this paper, however, we seek local knot removal and try to eliminate a single point or a few points which is/are specified by a user. This is required in some applications (especially in some interactive environment).

The rest of this paper is organized as follows. In Section 2, T-splines are briefly overviewed. In Section 3, an algorithm for removing one control point from a T-spline surface is presented. The possible extension of the algorithm for removing more control points is given in Section 4. Section 5 draws the conclusion.

2 T-Splines

A T-spline surface is defined by a control grid called T-mesh. The T-mesh is similar to a NURBS control mesh except that in a T-mesh a partial row or column of control points is permitted. The permission of existence of partial rows or columns makes it possible to add a single control point to a T-mesh without propagating an entire row or column of control points and without altering the surface. The knot information of a T-spline is expressed using knot intervals indicating the difference between two knots and assigned to the edges of the T-mesh. Fig. 1 shows an example of a T-spline. The left figure is the pre-image of the T-mesh in the parameter domain, the middle one shows the T-mesh, and the right one shows the T-spline surface.

The equation for a T-spline surface in homogeneous representation is

$$P(s,t) = \sum_{i=1}^{n} P_i B_i(s,t) \tag{1}$$

where the $P_i = (w_i x_i, w_i y_i, w_i z_i, w_i)$ are homogeneous control points and the w_i are control point weights. The T-spline blending function corresponding to control point P_i is $B_i(s,t)$:

$$B_i(s,t) = N[\mathbf{s}_i](s) N[\mathbf{t}_i](t) \tag{2}$$

where $N[\mathbf{s}_i](s), N[\mathbf{t}_i](t)$ are the cubic B-spline basis functions associated with the knot quintuples

$$\mathbf{s}_i = [s_{i0}, s_{i1}, s_{i2}, s_{i3}, s_{i4}] \tag{3}$$

and

$$\mathbf{t}_i = [t_{i0}, t_{i1}, t_{i2}, t_{i3}, t_{i4}] \tag{4}$$

respectively. For example,

$$N[\mathbf{s}_i](s) = \begin{cases} \dfrac{(s-s_{i0})^3}{(s_{i1}-s_{i0})(s_{i3}-s_{i0})(s_{i2}-s_{i0})}, & s_{i0} < s \le s_{i1} \\ \dfrac{(s-s_{i0})^2(s_{i2}-s)}{(s_{i2}-s_{i1})(s_{i3}-s_{i0})(s_{i2}-s_{i0})} + \dfrac{(s_{i3}-s)(s-s_{i0})(s-s_{i1})}{(s_{i2}-s_{i1})(s_{i3}-s_{i1})(s_{i3}-s_{i0})} \\ \quad + \dfrac{(s_{i4}-s)(s-s_{i1})^2}{(s_{i2}-s_{i1})(s_{i4}-s_{i1})(s_{i3}-s_{i1})}, & s_{i1} < s \le s_{i2} \\ \dfrac{(s-s_{i0})(s_{i3}-s)^2}{(s_{i3}-s_{i2})(s_{i3}-s_{i1})(s_{i3}-s_{i0})} + \dfrac{(s_{i4}-s)(s_{i3}-s)(s-s_{i1})}{(s_{i3}-s_{i2})(s_{i4}-s_{i1})(s_{i3}-s_{i1})} \\ \quad + \dfrac{(s_{i4}-s)^2(s-s_{i2})}{(s_{i3}-s_{i2})(s_{i4}-s_{i2})(s_{i4}-s_{i1})}, & s_{i2} < s \le s_{i3} \\ \dfrac{(s_{i4}-s)^3}{(s_{i4}-s_{i3})(s_{i4}-s_{i2})(s_{i4}-s_{i1})}, & s_{i3} < s \le s_{i4} \\ 0, & s \le s_{i0} \ \text{ or } \ s > s_{i4} \end{cases}$$

The knot quintuples $\mathbf{s}_i$ and $\mathbf{t}_i$ are extracted from the T-mesh neighborhood of P_i. The details on T-splines can be found in [8,9].

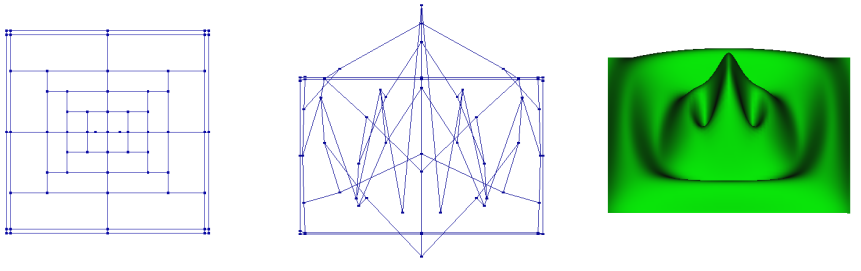


Fig. 1. An example of a T-spline: the pre-image, the T-mesh and the surface

T-splines support local refinement, which means adding a new control point into the T-mesh usually would not cause the insertion of too many extra points. It is essential that the geometry of a T-spline surface is not changed during the refinement of the T-mesh. Therefore, the process of inserting a control point should be treated with care. T-spline local knot insertion algorithm was first proposed in [8]. An improved algorithm was presented in [9] where the number of extra control points needed is significantly reduced. The main idea of the improved knot insertion algorithm is to maintain the validity of the T-mesh and to make all the blending functions be properly associated with the control points. The fundamental operation there is the blending function refinement which involves re-expressing a blending function by a linear combination of several new blending functions defined over finer knot sequences. Refer to [9] for the formulae of the blending function refinement.

3 Remove One Control Point from a T-Spline Surface

In this section, we will derive an algorithm for T-spline control point (knot) removal. The algorithm is based on two fundamental operations. One is the blending function refinement that has already been used in T-spline control point insertion [9]. The other one is the reverse blending function transformation. In the following, the reverse blending function transformation will be given first, and then follows the T-spline control point removal algorithm.

3.1 Reverse Blending Function Transformation

While the blending function refinement is used to split a basis function into two new ones with finer knot quintuples, the reverse blending function transformation presented here works in an opposite way.

Let $\mathbf{s} = [s_0, s_1, s_2, s_3, s_4]$ denote a knot vector (quintuple) in which s_2 is the center knot. $N[\mathbf{s}](s) = N[s_0, s_1, s_2, s_3, s_4](s)$ is the associated B-spline basis function defined on $\mathbf{s}$. Now suppose that a new knot quintuple $\mathbf{s}'$ is constructed from $\mathbf{s}$ by eliminating a knot s_i ($i = 0, 1, 3$,or 4) that is other than the center knot in $\mathbf{s}$, inserting another knot s_{add} which satisfies $s_{add} \leq s_0$ or $s_{add} \geq s_4$, and meanwhile keeping the center knot of $\mathbf{s}'$ still to be s_2. Let the B-spline basis function corresponding to $\mathbf{s}'$ be denoted as $N[\mathbf{s}'](s)$. $N[\mathbf{s}](s)$ can be re-expressed in the form of $N[\mathbf{s}'](s)$ plus another term. Since the knot span of $\mathbf{s}'$ is larger than that of $\mathbf{s}$, such an operation is called the reverse basis function transformation which is essentially derived from the equation of the basis function refinement. There are four different types of reverse basis function transformation, depending on which knot in $\mathbf{s}$ is replaced.

If $s_{add} \leq s_0$ and $\mathbf{s}' = [s_{add}, s_1, s_2, s_3, s_4]$, then

$$N[s_0, s_1, s_2, s_3, s_4](s) = c_0 N[s_{add}, s_1, s_2, s_3, s_4](s) + d_0 N[s_{add}, s_0, s_1, s_2, s_3](s) \tag{5}$$

where $c_0 = 1$ and $d_0 = \frac{s_{add} - s_0}{s_3 - s_{add}}$.

If $s_{add} \leq s_0$ and $\mathbf{s}' = [s_{add}, s_0, s_2, s_3, s_4]$, then

$$N[s_0, s_1, s_2, s_3, s_4](s) = c_1 N[s_{add}, s_0, s_2, s_3, s_4](s) + d_1 N[s_{add}, s_0, s_1, s_2, s_3](s) \tag{6}$$

where $c_1 = \frac{s_4 - s_1}{s_4 - s_0}$ and $d_1 = \frac{(s_{add} - s_1)(s_4 - s_1)}{(s_3 - s_{add})(s_4 - s_0)}$.

If $s_{add} \geq s_4$ and $\mathbf{s}' = [s_0, s_1, s_2, s_4, s_{add}]$, then

$$N[s_0, s_1, s_2, s_3, s_4](s) = c_2 N[s_0, s_1, s_2, s_4, s_{add}](s) + d_2 N[s_1, s_2, s_3, s_4, s_{add}](s) \tag{7}$$

where $c_2 = \frac{s_4 - s_0}{s_3 - s_0}$ and $d_2 = \frac{(s_{add} - s_3)(s_4 - s_0)}{(s_1 - s_{add})(s_3 - s_0)}$.

If $s_{add} \geq s_4$ and $\mathbf{s}' = [s_0, s_1, s_2, s_3, s_{add}]$, then

$$N[s_0, s_1, s_2, s_3, s_4](s) = c_3 N[s_0, s_1, s_2, s_3, s_{add}](s) + d_3 N[s_1, s_2, s_3, s_4, s_{add}](s) \tag{8}$$

where $c_3 = 1$ and $d_3 = \frac{s_{add} - s_4}{s_1 - s_{add}}$.

The reverse blending function transformation for a T-spline surface blending function $B(s,t)$ can be easily derived from the above four equations. In general, if both $N[\mathbf{s}](s)$ and $N[\mathbf{t}](t)$ are decomposed, we can rewrite $B(s,t)$ by

$$B(s,t) = N[\mathbf{s}](s)N[\mathbf{t}](t) = \sum_i c_i N[\mathbf{s_i}](s) \cdot \sum_j c_j N[\mathbf{t_j}](t) = \sum_k r_k B_k(s,t), \quad (9)$$

where $B_k(s,t)$ stand for the refined T-spline surface blending functions that are the product of the two univariate B-spline basis functions.

3.2 T-Spline Control Point Removal

Now let us look at how to eliminate a specified control point from the T-mesh without altering the geometry of the surface. This process can also be called T-spline knot removal due to the fact that removing a control point causes the corresponding knot to be removed from the T-spline pre-image in the parameter domain as well.

An immediate result of removing a control point is the change of the topology of the T-mesh. Such change includes the disappearance of the control point, and possible removing or adding of some edge(s) due to the removal of that point. Fig. 2 shows three examples of the topology change. Fig. 2 (d), (e), (f) are the results of removing P_r from an T-mesh shown in Fig. 2 (a), (b), (c), respectively. Sometimes the topology for the new T-mesh is not unique. Refer to Fig. 3 for a more complicated example, where (b) and (c) are two possible topological structures when the control point P_r is removed from a T-mesh shown in Fig. 3(a). In such a case, both situations could be checked or user's recommendation may be needed.

Another important component of the T-spline control point removal algorithm is to update the geometry of the control points so as to keep the shape of the T-spline surface unchanged. Assume we want to eliminate the control point P_r which is associated with knot (s_r, t_r). Our approach begins with the given T-spline surface. The T-spline surface equation $P(s,t) = \sum_{i=1}^{n} P_i B_i(s,t)$ is split into two parts: $\sum_{i \neq r} P_i B_i(s,t)$ and $P_r B_r(s,t)$. We call the second part a residue. The first part defines a new T-spline surface whose control points one-to-one correspond to those of the new T-mesh. However, the knot quintuples for the blending functions in $\sum_{i \neq r} P_i B_i(s,t)$ do not necessarily match those derived from the new T-mesh. It is important to keep in mind that the blending functions and the T-mesh are tightly coupled in a valid T-spline surface [8,9]. Therefore the main process of our algorithm is to use the reverse blending function transformation and the blending function refinement to update both $\sum_{i \neq r} P_i B_i(s,t)$ and $P_r B_r(s,t)$ such that their blending functions gradually match the new T-mesh except that $B_r(s,t)$ has (s_r, t_r) as its center knots in the knot quintuples. During this process, local knot insertion may also be required (see a discussion in the end of this section). As a result, $\sum_{i=1}^{n} P_i B_i(s,t)$ will eventually be decomposed into a T-spline surface defined over the new T-mesh and a residue term whose blending function has knot quintuples centered at (s_r, t_r). If the residue

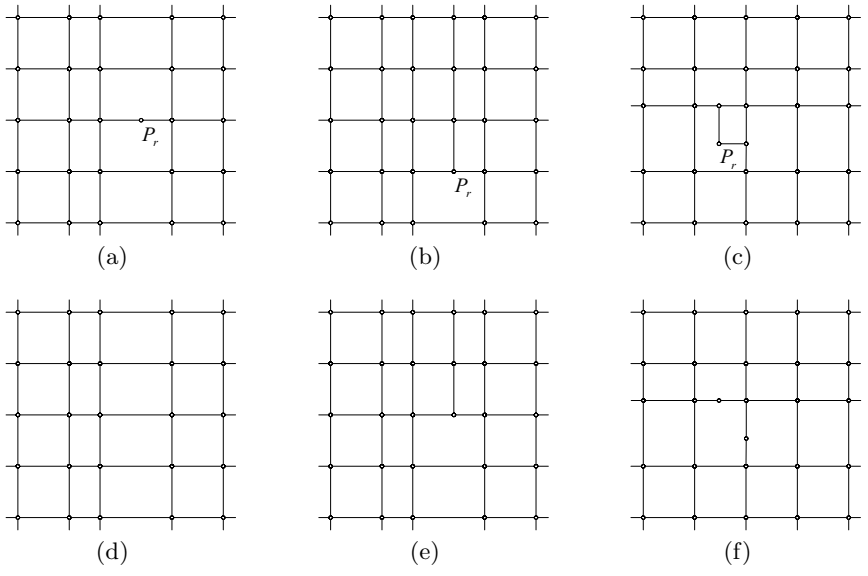


Fig. 2. T-mesh topology change after removing control point P_r

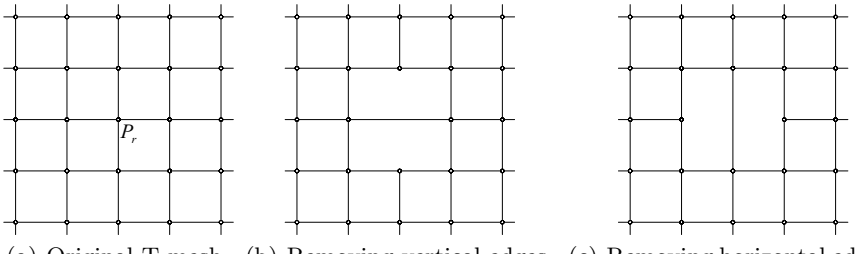


Fig. 3. Another T-mesh topology change example

term becomes zero, a valid new T-spline surface without the control point P_r has been found. Otherwise, the point P_r cannot be removed.

The T-spline control point removal algorithm is thus given as follows.

1) Remove a control point P_r with a knot (s_r, t_r) from the T-mesh and update the topology of the T-mesh.
2) Set the current T-spline surface to be $\sum_{i \neq r} P_i B_i(s,t)$ and the residue to be $P_r B_r(s,t)$.
3) **for** each blending function from the current T-spline surface
   3.1) **if** the blending function has the same knot quintuples as the residue's blending function, move it to the residue.
   3.2) **else if** the blending function contains the knot (s_r, t_r) such that at least one of s_r and t_r is not the center in the respective knot quintuple, perform a proper reverse blending function transformation.

3.3) **else if** the blending function is missing a knot inferred from the current T-mesh, perform a proper blending function refinement.

3.4) **else if** the blending function has a knot other than (s_r, t_r), which is not indicated in the current T-mesh, add an appropriate control point into the T-mesh.

4) **if** the blending function of the residue is missing a knot inferred from the current T-mesh, perform a proper blending function refinement and move the new generated term whose corresponding knot quintuples are not centered at (s_r, t_r) to the current T-spline surface.

5) Goto step 3) until there is no new operation in steps 3.2)-3.4) and step 4). Now all the blending functions are properly associated with the control points in the T-mesh.

6) If the final residue equals zero, the control point P_r is successfully removed; else, the control point P_r cannot be removed.

Note that this algorithm is in the similar fashion of the T-spline knot insertion algorithm proposed in [9]. The main different step is step 3.2) which invokes the operation of reverse blending function transformation. Here we use an example to illustrate this step topologically. Fig. 4(a) shows a T-mesh from which we want to remove the point P_r. After removing P_r, the T-mesh becomes Fig. 4(b). However, the blending function corresponding to (s_2, t_2) is $N[s_0, s_1, s_2, s_3, s_4](s)N[t_0, t_1, t_2, t_3, t_4](t)$. It has a knot (s_3, t_2) that corresponds to the removed control point P_r. Therefore, according to step 3.2), a reverse blending function transformation is performed and we obtain two new blending functions: $N[s_0, s_1, s_2, s_4, s_5](s)N[t_0, t_1, t_2, t_3, t_4](t)$ and $N[s_1, s_2, s_3, s_4, s_5](s)$ $N[t_0, t_1, t_2, t_3, t_4](t)$. The former conforms with the current T-mesh, and the latter has the same knot quintuple as the residue (see Fig. 4(c)) and thus is moved to the residue.

Validity of the Algorithm. For an algorithm described in a recursive manner, it is important that the algorithm terminates after a finite number of steps. We examine the two basic operations in this T-spline control point removal algorithm. Since the knot values involved in this procedure are those that initially exist in the T-mesh, the blending function refinement would be called for only a limited number of times if such a process is needed [9]. For the reverse blending

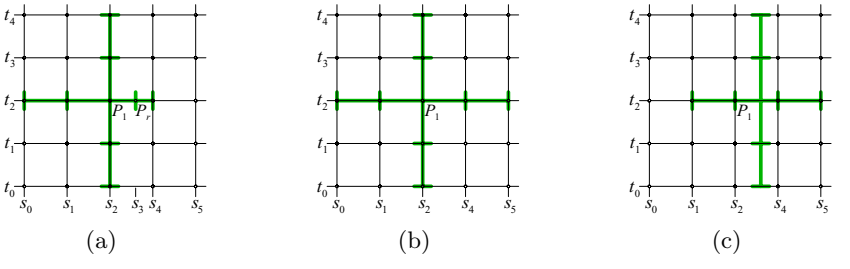


Fig. 4. Control point removal example

function transformation, it can be seen that each of those four reverse blending function transformations replaces a blending function by two new functions. One of the new blending functions corresponds to a knot quintuple which does not contain the removed knot, and the other one corresponds to a knot quintuple which is closer to the quintuple of the removed control point. Once the center knot of the quintuple becomes the knot to be removed, the reverse blending function transformation is completed. In this way, after a finite number of steps of performing reverse blending function transformation and blending function refinement, the T-spline surface is decomposed into a new T-spline surface which is defined by the new T-mesh without the removed control point plus a residue term whose blending function has knot quintuples centered at (s_r, t_r). If the coefficient of the residue term is zero, then the removal algorithm succeeds and the new T-spline surface is the result. Otherwise, the algorithm returns that the specified point cannot be removed. Therefore, the algorithm for T-spline control point removal is always guaranteed to terminate.

Discussion. In the process of removing a control point, sometimes the algorithm will introduce a few new control points into the T-mesh. The insertion of these control points is to make the blending functions be properly associated with the control points. Fig. 5 illustrates such a situation. If the point P_1 in a T-mesh shown in Fig. 5(a) is removed, then a new control point P_4 will automatically be added into the T-mesh by our algorithm (see Fig. 5(b)), which ensures that the blending function corresponding to P_3 is compatible with the T-mesh.

It should be pointed out that in the situation where removing a control point causes the insertion of extra point(s), the total number of the control points will not be reduced, and thus the user may choose not to remove that point for applications such as surface simplification. However, if the user's concern is whether a specified point is removable and how to remove it, our algorithm is attractive because the topology of the new T-mesh is automatically determined by the algorithm. Some other possible approaches for control point removal such as setting up a system of linear equation describing the relationship between the blending functions (or control points) before the removal and after the removal need to know the topology of the new T-mesh in advance.

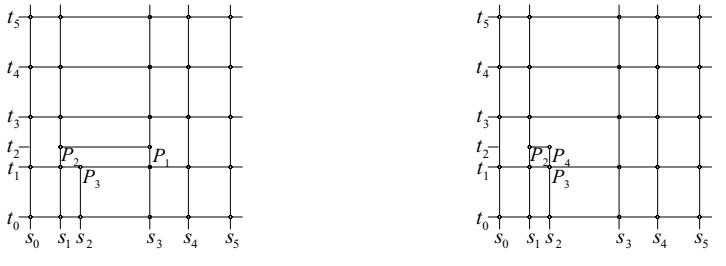


Fig. 5. Extra control point insertion in the removal process

4 Remove More Control Points

This section extends the algorithm developed in Section 3 to remove more control points.

If a user specifies n control points in a T-mesh, we may extend the algorithm to detect whether these n control points can be removed simultaneously and to compute the new T-mesh if they are removable. The possible modifications include: 1) n control points should be removed in updating the topology of the new T-mesh; and 2) the residue should consist of n terms. However, the topology of the resulting T-mesh after removing several control points could generally have many possibilities. This increases the complexity of the algorithm. In addition, it is unlikely that arbitrarily specified n control points can be removed simultaneously. Therefore, if we want to remove many control points (especially those generated by knot insertion), it is not practical to identify them first and then to apply the extended algorithm.

An alternative approach could be based on the single control point removal algorithm. An unsophisticated attempt is described as follows: for every control point in the current T-mesh, check its removability; if it is removable, remove it. This method is quite simple. However, the following example shows that this method may fail to remove some control points although they are generated by knot insertion.

Consider a T-mesh shown in Fig.6(b), which is the result of inserting a point P_1 into a T-mesh shown in Fig.6(a). Point P_2 is a control point automatically introduced by the knot insertion algorithm [9]. Suppose no further geometrical change is made to these control points. Obviously, P_1 and P_2 are two redundant control points in the T-mesh and should be removable. However, if we apply the single control point removal algorithm to point P_2, it is *surprising* to find that P_2 cannot be removed from the T-mesh in Fig.6(b) by carefully checking the removal algorithm!

Fortunately, in the above situation, point P_1 can be removed by the single control point removal algorithm, and furthermore after that, point P_2 becomes removable for the single control point removal algorithm (see Fig.6(c)).

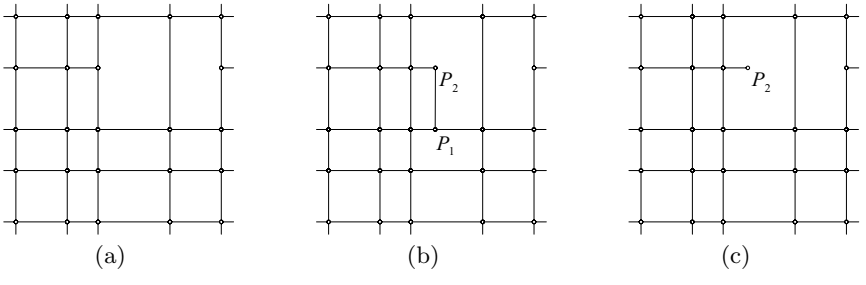


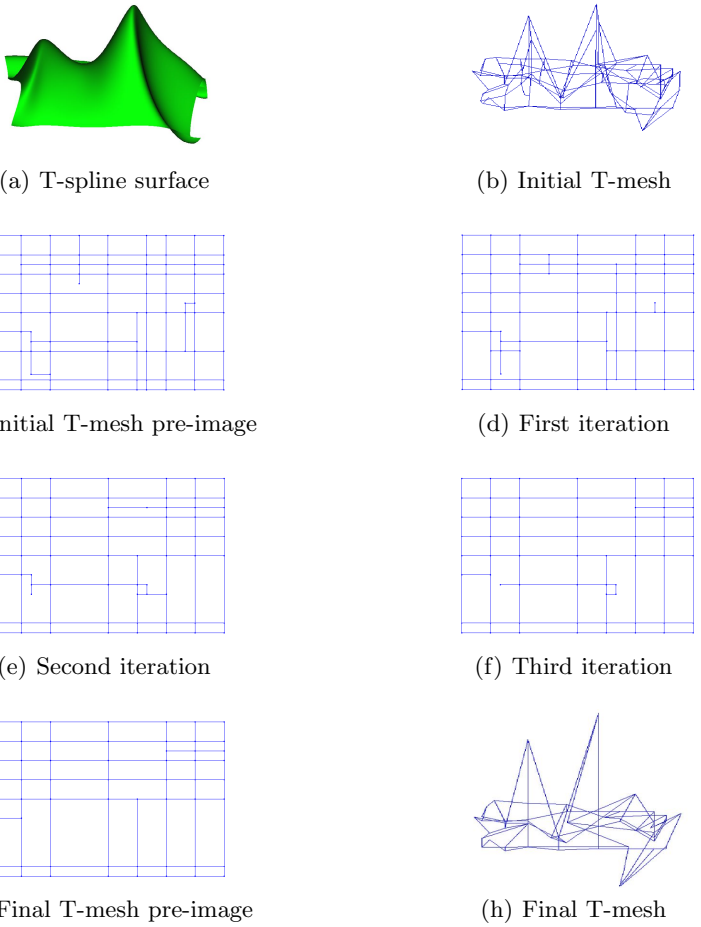
Fig. 6. Example for identifying the removable control points

The above example indicates that one control point may not be removed until some other control points are removed. This observation motivates the following removal strategy for removing as many control points in a T-mesh as possible:

1) Check each control point in the T-mesh. If it is removable, remove it and update the T-mesh.
2) If at least one control point has been removed, execute step 1) again.

4.1 An Example

An example of removing many control points from a T-spline surface is provided here. Fig. 7(a) shows an T-spline surface, and its associated T-mesh and pre-image are displayed in Fig. 7(b) and (c), respectively. (Fig. 7(b) is uniformly



(a) T-spline surface (b) Initial T-mesh

(c) Initial T-mesh pre-image (d) First iteration

(e) Second iteration (f) Third iteration

(g) Final T-mesh pre-image (h) Final T-mesh

Fig. 7. An example of removing many control points

scaled down in order to be properly fit into the page.) The T-mesh contains 94 control points and the algorithm is then invoked to eliminate the removable control points among them. Fig. 7(d) is the pre-image of the resulting T-mesh after we apply the single control point removal algorithm to all the points of the T-mesh once. We call this process one iteration. 11 control points are eliminated during the first iteration. As indicated by the algorithm, more control points might now become removable and we should continue this process to the new T-mesh. Fig. 7(e) and (f) are the pre-images of the T-mesh after the second and third iterations. It can be seen that the number of control points in the T-spline surface is gradually reduced. The final result is displayed in Fig. 7(g) and (h). According to the algorithm, no more control point can be removed at this stage and the whole process is then terminated. During this removal process, there are totally 37 control points that are removed. Thus the T-mesh is simplified while the T-spline surface remains the same.

5 Conclusion

This paper investigates the problem of removing control points from a T-spline surface. The T-spline control point removal is found to be much more complicated than the B-spline knot removal, since the T-spline control point removal could lead to different result and sometimes the control point removal could cause the insertion of extra control points. A single control point removal algorithm is developed, which is in the style of the T-spline knot insertion algorithm [9]. The algorithm can be used to detect whether a user-specified control point can be removed or not. If the control point is found to be removable, the algorithm returns the new T-mesh with the control point removed. The algorithm may have applications in interactive design.

The extension of the algorithm to remove more control points is also proposed. In many situations, the control points that are added by knot insertion can be completely removed by this extended algorithm. However, there still exist some situations, in which some inserted control points cannot be removed. Therefore developing algorithms that are able to remove all those control points added by knot insertion warrants further investigation. Besides, a method of checking the removability of a control point directly from the topological structure of the T-mesh in its neighborhood would also be an enhancement to the current algorithm.

Acknowledgement

The work is supported by the URC-SUG8/04 of Nanyang Technological University.

References

1. L. Piegl and W. Tiller, *The NURBS Book*. Springer-Verlag, 1997.
2. W. Boehm, "Inserting new knots into B-spline curves," *Computer Aided Design*, vol. 12, pp. 199–201, 1980.

3. E. Cohen, T. Lyche, and R. Riesenfeld, "Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics," *Computer Graphics and Image Processing*, vol. 14, pp. 87–111, 1980.
4. D. Handscomb, "Knot elimination: reversal of the oslo algorithm," *International Series of Numerical Mathematics*, vol. 81, pp. 103–111, 1987.
5. R. Goldman and T. Lyche, *Knot Insertion and Deletion Algorithms for B-Spline Curves and Surfaces.* SIAM, 1993.
6. T. Lyche and K. Mørken, "Knot removal for parametric B-spline curves and surfaces." *Computer Aided Geometric Design*, vol. 4, no. 3, pp. 217–230, 1987.
7. T. Lyche, *Knot Removal for Spline Curves and Surfaces.* Academic Press, New York, 1992, ch. Approximation Theory VII, pp. 207–226.
8. T. Sederberg, J. Zheng, A. Bakenov, and A. Nasri, "T-splines and T-NURCCs," *ACM Transactions on Graphics (SIGGRAPH 2003)*, vol. 22, no. 3, pp. 477–484, 2003.
9. T. Sederberg, D.Cardon, G.Finnigan, N.North, J. Zheng, and T. Lyche, "T-spline simplification and local refinement," *ACM Transactions on Graphics (SIGGRAPH 2004)*, vol. 23, no. 3, pp. 276–283, 2004.

Shape Representations with Blossoms and Buds

L. Yohanes Stefanus

University of Indonesia, Faculty of Computer Science,
Depok 16424, Indonesia
yohanes@cs.ui.ac.id

Abstract. Polynomials, either on their own or as components of splines, play a fundamental role for shape representations in computer-aided geometric design (CAGD) and computer graphics. This paper shows that any polynomial $p(t)$ of degree $d \leq n$ can be represented in the form of a blossom of another polynomial $b(t)$ of degree d evaluated off the diagonal at the linear functions $X_j(t)$, $j = 1, \ldots, n$, chosen under some conditions expressed in terms of the elementary symmetric functions. The polynomial $b(t)$ is called a *bud* of the polynomial $p(t)$. An algorithm for finding a bud $b(t)$ of a given polynomial $p(t)$ is presented. Successively, a bud of $b(t)$ can be computed and so on, to form a sequence of representations. The information represented by the original polynomial is preserved in its buds. This scheme can be used for encoding/decoding geometric design information.

1 Introduction

In computer-aided geometric design (CAGD) and computer graphics, polynomials either on their own or as components of splines, play a fundamental role for shape representations [1] [2] [4] [5] . The technique of blossoming-and-evaluating-off-the-diagonal has been applied to generalize the Bézier and B-spline schemes [7] [8] [9]. A related technique is used in this paper for constructing a sequence of successive shape representations in terms of polynomials.

The outline of this paper is as follows. In Section 2 we present some useful notations and mathematical background, including the technique of blossoming and the convolution basis functions. In Section 3 we introduce the concept of a *bud* and show that under some mild conditions, a bud of a given polynomial can be found. We then establish an algorithm for computing a bud of a polynomial and provide some illuminating examples involving cubic polynomials and Bézier curves. We also remark how the scheme can be used for encoding/decoding shape representations. Finally, in Section 4, we provide a brief summary of the main results.

2 Symmetric Functions and Blossoming

This section introduces some background mathematics and useful notation that will be applied in later sections.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 397–408, 2006.
© Springer-Verlag Berlin Heidelberg 2006

2.1 Elementary Symmetric Functions

The elementary symmetric functions $s_j(a_1, \ldots, a_n)$, for $j = 0, 1, \ldots, n$, are defined by the identity

$$Q(t) \equiv \prod_{i=1}^{n}(a_i t + 1) = \sum_{j=0}^{n} s_j(a_1, \ldots, a_n)t^j. \tag{1}$$

Each $s_j(a_1, \ldots, a_n)$ is symmetric with respect to $a_1, \ldots, a_n$ since any permutation of $a_1, \ldots, a_n$ leaves the generating function $Q(t)$ unchanged. Explicitly, $s_j(a_1, \ldots, a_n)$ is a sum of $\binom{n}{j}$ terms, i.e.,

$$s_j(a_1, \ldots, a_n) = \sum_{\{\sigma_1, \ldots, \sigma_j\}} a_{\sigma_1} \cdots a_{\sigma_j}, \tag{2}$$

where the summation notation means summing over all j-subsets $\{\sigma_1, \ldots, \sigma_j\}$ of the set $\{1, 2, \ldots, n\}$. For instance, the 2-subsets of $\{1, 2, 3, 4\}$ are $\{1, 2\}$, $\{1, 3\}$, $\{1, 4\}$, $\{2, 3\}$, $\{2, 4\}$ and $\{3, 4\}$.

Example 1. $s_0(a_1, a_2, a_3) = 1$, $s_1(a_1, a_2, a_3) = a_1 + a_2 + a_3$, $s_2(a_1, a_2, a_3) = a_1a_2 + a_2a_3 + a_1a_3$, $s_3(a_1, a_2, a_3) = a_1a_2a_3$. □

2.2 Composite Symmetric Functions

More general than the elementary symmetric functions are the composite symmetric functions

$$\mathcal{S}_j([a_1, b_1], \ldots, [a_n, b_n]), \quad j = 0, 1, \ldots, n,$$

defined by the identity

$$R(t) \equiv \prod_{i=1}^{n}(a_i t + b_i) = \sum_{j=0}^{n} \mathcal{S}_j([a_1, b_1], \ldots, [a_n, b_n])t^j. \tag{3}$$

Each $\mathcal{S}_j([a_1, b_1], \ldots, [a_n, b_n])$ is a symmetric function with respect to the pairs $[a_1, b_1], \ldots, [a_n, b_n]$, since any permutation of these pairs does not change the generating function $R(t)$.

Like the elementary symmetric functions, $\mathcal{S}_j([a_1, b_1], \ldots, [a_n, b_n])$ is also a sum of $\binom{n}{j}$ terms. That is,

$$\mathcal{S}_j([a_1, b_1], \ldots, [a_n, b_n]) = \sum_{\{\sigma_1, \ldots, \sigma_j\}} a_{\sigma_1} \cdots a_{\sigma_j} b_{\sigma_{j+1}} \cdots b_{\sigma_n}. \tag{4}$$

The set $\{\sigma_{j+1}, \ldots, \sigma_n\}$ is the complement of $\{\sigma_1, \ldots, \sigma_j\}$ with respect to $\{1, 2, \ldots, n\}$, i.e., $\{\sigma_{j+1}, \ldots, \sigma_n\} = \{1, 2, \ldots, n\} \setminus \{\sigma_1, \ldots, \sigma_j\}$.

Example 2. $\mathcal{S}_0([a_1, b_1], [a_2, b_2], [a_3, b_3]) = b_1b_2b_3$, $\mathcal{S}_1([a_1, b_1], [a_2, b_2], [a_3, b_3]) = a_1b_2b_3 + b_1a_2b_3 + b_1b_2a_3$, $\mathcal{S}_2([a_1, b_1], [a_2, b_2], [a_3, b_3]) = a_1a_2b_3 + a_1b_2a_3 + b_1a_2a_3$, $\mathcal{S}_3([a_1, b_1], [a_2, b_2], [a_3, b_3]) = a_1a_2a_3$. □

Some properties of the composite symmetric functions are as follows:

1) $\mathcal{S}_j([a_1,1],\ldots,[a_n,1]) = s_j(a_1,\ldots,a_n)$.
2) $\mathcal{S}_n([a_1,b_1],\ldots,[a_n,b_n]) = s_n(a_1,\ldots,a_n)$.
3) $\mathcal{S}_0([a_1,b_1],\ldots,[a_n,b_n]) = s_n(b_1,\ldots,b_n)$.
4) $\mathcal{S}_j([a_1,b_1],\ldots,[a_n,b_n]) =$
   $a_n\,\mathcal{S}_{j-1}([a_1,b_1],\ldots,[a_{n-1},b_{n-1}]) + b_n\,\mathcal{S}_j([a_1,b_1],\ldots,[a_{n-1},b_{n-1}])$.

2.3 Multiaffine Functions

A univariate function f is called **affine** if it preserves affine combinations, i.e., if f satisfies:

$$f(\sum_i c_i u_i) = \sum_i c_i f(u_i)$$

for all real numbers $c_1,\ldots,c_n,u_1,\ldots,u_n$, where $\sum_i c_i = 1$. A multivariate function f is called **multiaffine** if it is affine with respect to each of its arguments when the others are held fixed. A multivariate polynomial each of whose arguments appears to at most the first power is multiaffine. Since each argument a_i of $s_j(a_1,\ldots,a_n)$ appears only to the first power, each $s_j(a_1,\ldots,a_n)$ is multiaffine with respect to $a_1,\ldots,a_n$.

2.4 Blossoming

The blossoming approach to B-spline theory was proposed by **Ramshaw** [6] and, in a different form, by **de Casteljau** [3]. The **multiaffine blossom** of a degree n polynomial $p(t)$ is the unique symmetric multiaffine polynomial $\mathcal{B}[p](u_1,\ldots,u_n)$ for which $\mathcal{B}[p](t,\ldots,t) = p(t)$. *Multiaffine blossoms* will be referred to simply as *blossoms*.

The basic formula for blossoming a polynomial $p(t) = \sum_{k=0}^{n} a_k t^k$ is

$$\mathcal{B}[p](u_1,\ldots,u_n) = \sum_{k=0}^{n} a_k \frac{s_k(u_1,\ldots,u_n)}{\binom{n}{k}}.$$

Note that

$$\mathcal{B}[p](\underbrace{t,\ldots,t}_{n}) = p(t).$$

We say that p is $\mathcal{B}[p]$ evaluated along the diagonal (i.e., when all arguments are equal).

Example 3. The blossom of the cubic polynomial

$$p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

is the symmetric 3-affine polynomial

$$\mathcal{B}[p](u_1,u_2,u_3) = a_0 + \frac{a_1}{3}(u_1+u_2+u_3) + \frac{a_2}{3}(u_1u_2+u_2u_3+u_3u_1) + a_3(u_1u_2u_3).$$

□

The elegance and power of the blossoming approach to Bézier and B-spline curves are a consequence of the dual functional property of blossoms [6]:
If $\mathcal{B}[b](u_1, \ldots, u_n)$ is the blossom of a B-spline segment $b(t)$ of degree n, $t \in [t_\ell, t_{\ell+1})$, with de Boor control points $P_{\ell-n}, \ldots, P_\ell$ and knots $t_{\ell-n+1}, \ldots, t_{\ell+n}$, then

$$P_k = \mathcal{B}[b](t_{k+1}, \ldots, t_{k+n}), \qquad k = \ell-n, \ldots, \ell.$$

Equivalently,

$$b(t) = \sum_{k=\ell-n}^{\ell} \mathcal{B}[b](t_{k+1}, \ldots, t_{k+n})\, N_k^n(t), \qquad t \in [t_\ell, t_{\ell+1}) \tag{5}$$

where $N_k^n(t)$ is the B-spline with support $[t_k, t_{k+n+1}]$.

As a Bézier curve is a special case of a B-spline segment, we have a special case of the above property:
If $\mathcal{B}[p](u_1, \ldots, u_n)$ is the blossom of a polynomial curve $p(t)$ of degree n, then $\mathcal{B}[p](\underbrace{0, \ldots, 0}_{n-k}, \underbrace{1, \ldots, 1}_{k})$ is the k-th Bézier point. Equivalently,

$$p(t) = \sum_{k=0}^{n} \mathcal{B}[p](\underbrace{0, \ldots, 0}_{n-k}, \underbrace{1, \ldots, 1}_{k})\, B_k^n(t) \tag{6}$$

where $B_k^n(t)$ is the Bernstein basis function $\frac{n!}{k!(n-k)!} t^k (1-t)^{n-k}$.

2.5 Convolution Basis Functions

Convolution basis functions have been studied in [7] [8] [9]. In this section, some relevant results that will be used in the next section are reviewed.

The convolution basis functions $C_k^n(t), k = 0, 1, \ldots, n$ can be obtained by blossoming the Bernstein basis functions $B_k^n(t)$ respectively and evaluating the blossoms off the diagonal at the linear functions $X_j(t) = a_j t + b_j$, $j = 1, \ldots, n$ where a_j and b_j are constants and $a_j \neq 0$, i.e.,

$$C_k^n(t) \;=\; \mathcal{B}[B_k^n(x)](X_1, \ldots, X_n). \tag{7}$$

Unless explicitly stated otherwise, X_j is assumed to denote $X_j(t) = a_j t + b_j$.

Example 4.

$C_0^3(t) = (1 - X_1)(1 - X_2)(1 - X_3)$,
$C_1^3(t) = (1 - X_1)(1 - X_2)X_3 \;+\; (1 - X_1)X_2(1 - X_3) \;+\; X_1(1 - X_2)(1 - X_3)$,
$C_2^3(t) = (1 - X_1)X_2X_3 \;+\; X_1(1 - X_2)X_3 \;+\; X_1X_2(1 - X_3)$,
$C_3^3(t) = X_1X_2X_3$. □

Lemma 1. *A recursion formula for expressing the monomial t^j in terms of the convolution basis functions of degree $n \geq j$ is*

$$t^j = \frac{\sum_{k=j}^{n} \binom{k}{j} C_k^n(t) - \sum_{r=0}^{j-1} \left[\sum_{\{\sigma_1,\ldots,\sigma_j\}} \mathcal{S}_r([a_{\sigma_1}, b_{\sigma_1}], \ldots, [a_{\sigma_j}, b_{\sigma_j}]) \right] t^r}{s_j(a_1, \ldots, a_n)},$$

where the elementary symmetric functions $s_i(a_1, \ldots, a_n) \neq 0$ for $i = 0, \ldots, j$. The basis of the recursion is

$$t^0 = \sum_{k=0}^{n} C_k^n(t) = 1.$$

Proof. By the dual functional property of the blossom for Bernstein basis functions (Equation 6),

$$\sum_{k=j}^{n} \binom{k}{j} B_k^n(t) = \binom{n}{j} t^j.$$

Blossoming both sides with respect to t, evaluating at $X_1, \ldots, X_n$, and applying Equation 7, we obtain

$$\sum_{k=j}^{n} \binom{k}{j} C_k^n(t) = s_j(X_1, \ldots, X_n).$$

By Equation 2 and Equation 3,

$$\begin{aligned} s_j(X_1, \ldots, X_n) &= \sum_{\{\sigma_1,\ldots,\sigma_j\}} X_{\sigma_1} \cdots X_{\sigma_j} \\ &= \sum_{\{\sigma_1,\ldots,\sigma_j\}} (a_{\sigma_1} t + b_{\sigma_1}) \cdots (a_{\sigma_j} t + b_{\sigma_j}) \\ &= \sum_{\{\sigma_1,\ldots,\sigma_j\}} \left[\sum_{r=0}^{j} \mathcal{S}_r([a_{\sigma_1}, b_{\sigma_1}], \ldots, [a_{\sigma_j}, b_{\sigma_j}]) t^r \right] \\ &= \sum_{r=0}^{j} \left[\sum_{\{\sigma_1,\ldots,\sigma_j\}} \mathcal{S}_r([a_{\sigma_1}, b_{\sigma_1}], \ldots, [a_{\sigma_j}, b_{\sigma_j}]) \right] t^r. \end{aligned} \tag{8}$$

Therefore, we get:

$$\sum_{k=j}^{n} \binom{k}{j} C_k^n(t) = \sum_{r=0}^{j} \left[\sum_{\{\sigma_1,\ldots,\sigma_j\}} \mathcal{S}_r([a_{\sigma_1}, b_{\sigma_1}], \ldots, [a_{\sigma_j}, b_{\sigma_j}]) \right] t^r. \tag{9}$$

We can solve for t^j to obtain the desired formula provided that the coefficient of t^j, which by the properties of the composite symmetric functions is equal to $s_j(a_1, \ldots, a_n)$, does not vanish. Note that $s_0(a_1, \ldots, a_n) = 1$. □

Lemma 2. *The set of convolution polynomials* $\{C_k^n(t) \mid k = 0, 1, \ldots, n\}$ *is a basis for the space of polynomials of degree* $\leq n$ *if and only if for* $j = 1, \ldots, n$, *the elementary symmetric functions* $s_j(a_1, \ldots, a_n) \neq 0$.

Proof. Lemma 1 provides a proof that if $s_j(a_1, \ldots, a_n) \neq 0$ for $j = 1, \ldots, n$, then the convolution polynomials $C_k^n(t)$ form a basis for the space of polynomials of degree $\leq n$. This result holds since any polynomial is a linear combination of monomials and Lemma 1 provides a way for expressing each monomial as a linear combination of convolution polynomials.

Conversely, suppose that $s_p(a_1, \ldots, a_n) = 0$, for some p where $1 \leq p \leq n$. Then for $j = 0, \ldots, p$ the right hand side of Equation 9 provides $p + 1$ polynomials in t which are linearly dependent since t^p is missing. Let the j-th polynomial be denoted by $P_j(t)$. Then there exist constants K_j not all equal to zero such that

$$\sum_{j=0}^{p} K_j P_j(t) = 0.$$

By Equation 9, this corresponds to

$$(C_0^n \ \cdots \ C_n^n) \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & \binom{1}{1} & \cdots & 0 \\ 1 & \binom{2}{1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \binom{n}{1} & \cdots & \binom{n}{p} \end{pmatrix} \begin{pmatrix} K_0 \\ K_1 \\ \vdots \\ K_p \end{pmatrix} = 0.$$

But

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & \binom{1}{1} & \cdots & 0 \\ 1 & \binom{2}{1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \binom{n}{1} & \cdots & \binom{n}{p} \end{pmatrix} \begin{pmatrix} K_0 \\ K_1 \\ \vdots \\ K_p \end{pmatrix} = \begin{pmatrix} K_0 \\ K_0 + \binom{1}{1} K_1 \\ K_0 + \binom{2}{1} K_1 + \binom{2}{2} K_2 \\ \vdots \\ K_0 + \binom{n}{1} K_1 + \cdots + \binom{n}{p} K_p \end{pmatrix}$$

is not zero, because if this vector is zero then all the K_j are zero, which is not the case. Thus when $s_p(a_1, \ldots, a_n) = 0$, $C_0^n(t), \ldots, C_n^n(t)$ are linearly dependent and hence do not form a basis. □

3 Buds of a Polynomial

The following theorem states that under some mild conditions on the $X_j, j = 1, \ldots, n$, any polynomial can be rewritten in the form of a blossom of another polynomial evaluated off the diagonal.

Theorem 1. *Given any polynomial* $p(t)$ *of degree* $d \leq n$, *we can find a polynomial* $b(u)$ *of degree* d *such that*

$$\mathcal{B}[b(u)](X_1, \ldots, X_n) \ = \ p(t),$$

provided that for $j = 1, \ldots, d$, *the elementary symmetric functions* $s_j(a_1, \ldots, a_n) \neq 0$.

Proof. By Lemma 1 and Lemma 2, $p(t)$ can be expanded in terms of the convolution basis functions, i.e.,

$$p(t) = \sum_{k=0}^{n} K_k C_k^n(t),$$

where K_k are constants. Since

$$\mathcal{B}[\sum_{k=0}^{n} K_k B_k^n(u)](X_1, \ldots, X_n) = \sum_{k=0}^{n} K_k C_k^n(t),$$

we have

$$b(u) = \sum_{k=0}^{n} K_k B_k^n(u),$$

which is a polynomial in u expressed in terms of the Bernstein basis functions. The polynomial $b(u)$ is of degree d because $p(t)$ is of degree d and blossoming does not change the degree. □

Definition 1. *The polynomial $b(u)$ in Theorem 1 is called a* **bud** *of $p(t)$.*

Algorithm 1

Input: a polynomial $p(t)$ as in Theorem 1, represented in the monomial basis; a set of constants $a_1, \ldots, a_n$, $b_1, \ldots, b_n$ which satisfy the condition that for $j = 1, \ldots, d$, $s_j(a_1, \ldots, a_n) \neq 0$.
Output: a polynomial $b(u)$ which is a bud of $p(t)$.
Steps:

1) For $j = 1, \ldots, d$, **rewrite** t^j in terms of $X_1, \ldots, X_n$ by using the formula

$$t^j = \frac{s_j(X_1, \ldots, X_n) - \sum_{r=0}^{j-1} \left[\sum_{\{\sigma_1, \ldots, \sigma_j\}} \mathcal{S}_r([a_{\sigma_1}, b_{\sigma_1}], \ldots, [a_{\sigma_j}, b_{\sigma_j}])\right] t^r}{s_j(a_1, \ldots, a_n)} \tag{10}$$

recursively. The basis of the recursion is

$$t = \frac{s_1(X_1, \ldots, X_n) - \sum_{k=1}^{n} b_k}{s_1(a_1, \ldots, a_n)}. \tag{11}$$

2) $b(u)$ is obtained from the **rewritten** $p(t)$ by substituting u for each X_k, $k = 1, \ldots, n$.

Proof. The idea is to rewrite $p(t)$ in terms of $X_1, \ldots, X_n$, so that it is in the form of a blossom. Moreover, we can rewrite any polynomial $p(t) = \sum_{k=0}^{d} c_k t^k$ if we can rewrite any monomial $t^j, j = 1, \ldots, d$.

By Equation 8, we can solve for t^j in terms of $t, \ldots, t^{j-1}$ provided that the coefficient of t^j, which is equal to $s_j(a_1, \ldots, a_n)$, does not vanish. Recursively, we solve for t^{j-1} in terms of $t, \ldots, t^{j-2}$. The base case for this recursion is

$$t = \frac{s_1(X_1, \ldots, X_n) - \sum_{k=1}^{n} b_k}{s_1(a_1, \ldots, a_n)}.$$

Since the blossom of t^j evaluated off the diagonal at $X_1, \ldots, X_n$ is

$$\binom{n}{j}^{-1} s_j(X_1, \ldots, X_n),$$

any monomial t^j can be rewritten in terms of $a_1, \ldots, a_n$, $b_1, \ldots, b_n$, and $\mathcal{B}[x^m](X_1, \ldots, X_n), m = 1, \ldots, j$. Thus $b(u)$ can be obtained from the rewritten $p(t)$ by substituting u for each X_k, $k = 1, \ldots, n$. □

Notice that the right-hand sides of (10) and (11) are symmetric with respect to the pairs $[a_1, b_1], \ldots, [a_n, b_n]$.

Example 5. For the cubic case, step 1) of Algorithm 1 gives

$$\begin{aligned}
t &= \frac{(X_1 + X_2 + X_3) - (b_1 + b_2 + b_3)}{a_1 + a_2 + a_3}, \\
t^2 &= ((X_1X_2 + X_2X_3 + X_3X_1) - (b_1b_2 + b_2b_3 + b_3b_1) \\
&\quad -t\,(a_1b_2 + b_1a_2 + a_2b_3 + b_2a_3 + a_3b_1 + b_3a_1))/(a_1a_2 + a_2a_3 + a_3a_1), \\
t^3 &= (X_1X_2X_3 - b_1b_2b_3 - t\,(a_1b_2b_3 + b_1a_2b_3 + b_1b_2a_3) \\
&\quad -t^2\,(a_1a_2b_3 + a_1b_2a_3 + b_1a_2a_3))/(a_1a_2a_3),
\end{aligned}$$

which can be used for rewriting any cubic polynomial in t in terms of a_1, a_2, a_3, b_1, b_2, b_3, X_1, X_2, and X_3. □

Example 6. Given a polynomial $p(t) = 1 + 6\,t + 3\,t^2 - 9\,t^3$, we want to find a bud $b(t)$ of $p(t)$ with the constants $a_1 = 2, a_2 = 4, a_3 = 3, b_1 = -1, b_2 = 2, b_3 = 1$. The condition required by Theorem 1 is satisfied since $s_1(a_1, a_2, a_3) \neq 0$, $s_2(a_1, a_2, a_3) \neq 0$, and $s_3(a_1, a_2, a_3) \neq 0$. As step 1) of Algorithm 1, we rewrite $p(t)$ using the equations in Example 5 to get:

$$p(t) = \frac{17}{156} + \frac{29}{156}(X_1 + X_2 + X_3) + \frac{3}{13}(X_1X_2 + X_2X_3 + X_3X_1) - \frac{3}{8}X_1X_2X_3.$$

As step 2) of the algorithm, we obtain $b(t)$ by substituting t for X_1, X_2, and X_3 in the rewritten $p(t)$:

$$b(t) = \frac{17}{156} + \frac{29}{52}\,t + \frac{9}{13}\,t^2 - \frac{3}{8}\,t^3.$$

Next, we repeat the algorithm to compute a bud $c(t)$ of $b(t)$. We could choose a different set of values for the constants $a_1, a_2, a_3, b_1, b_2, b_3$, but let's just use the same values for simplicity. The result is

$$c(t) = \frac{3355}{48672} + \frac{979}{16224}\,t + \frac{255}{2704}\,t^2 - \frac{1}{64}\,t^3.$$

After this, we could repeat the algorithm again to find a next bud if we wish. A plot of $p(t)$, $b(t)$, and $c(t)$ for $t \in [0, 1]$ is given in Figure 1.

Notice that we can retrieve $b(t)$ from $c(t)$ and $p(t)$ from $b(t)$ by the blossoming-and-evaluating-off-the-diagonal operation:

$$\begin{aligned}
b(t) &= \mathcal{B}[c(t)](2t - 1, 4t + 2, 3t + 1), \\
p(t) &= \mathcal{B}[b(t)](2t - 1, 4t + 2, 3t + 1).
\end{aligned}$$

□

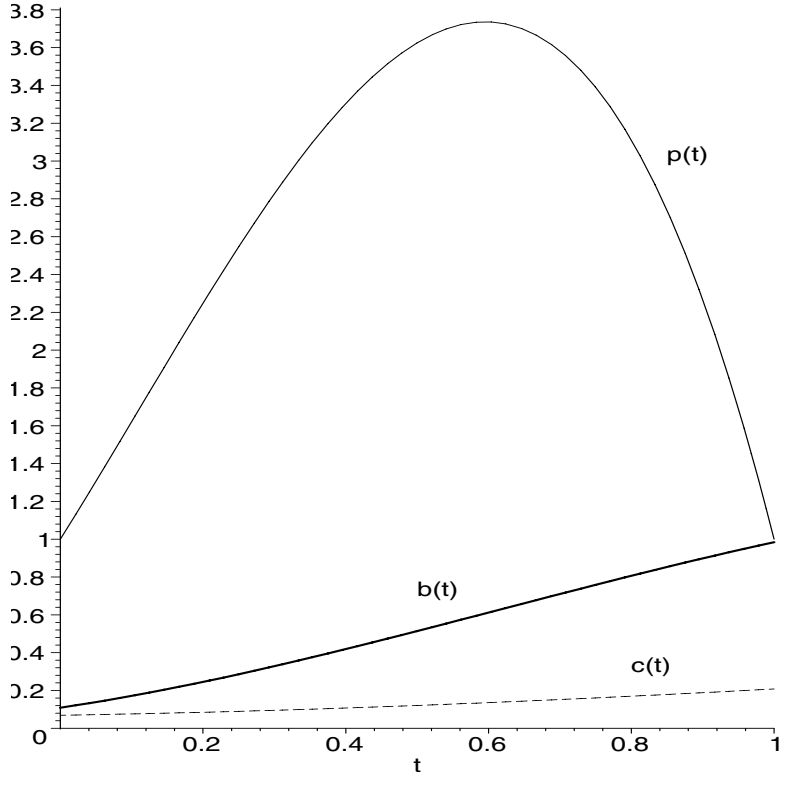


Fig. 1. $c(t)$ is a bud of $b(t)$ which is a bud of $p(t)$

Remark 1. Algorithm 1 computes algebraically. Given a polynomial $p(t)$ of degree $d \leq n$, we can compute one of its buds, $b(t)$, by using a suitable set of constants $a_1, \ldots, a_n$, $b_1, \ldots, b_n$ which satisfy the condition that for $j = 1, \ldots, d$, $s_j(a_1, \ldots, a_n) \neq 0$. After that, we can repeat the process to compute a bud of $b(t)$, and repeat again as long as we wish. In the end, we obtain a sequence of buds $\langle p(t), b(t), c(t), \ldots \rangle$. (The polynomial $p(t)$ can be viewed as a bud of itself because we can choose the constants $a_1 = \cdots = a_n = 1$ and $b_1 = \cdots = b_n = 0$ to obtain a bud which is identical to $p(t)$.) Certainly we can also go in the opposite direction, i.e. from buds to blossoms: we can recover $b(t)$ by blossoming $c(t)$ and evaluating off the diagonal at $X_1, \ldots, X_n$ with the same constants $a_1, \ldots, a_n$, $b_1, \ldots, b_n$ used in the process of going from $b(t)$ to $c(t)$. Likewise, we can go from $b(t)$ to $p(t)$ and continue to apply the process of blossoming and evaluating off the diagonal to go from $p(t)$ to get a new polynomial, and so forth. The information represented by the original polynomial is retained.

Now we extend the concept of a bud to curves.

Definition 2. *A* **bud** *of a curve* $\mathbf{C}(t) = (x_1(t), \ldots, x_d(t))$ *is a curve* $\mathbf{B}(t) = (x_1^b(t), \ldots, x_d^b(t))$ *where* $x_i^b(t)$ *is a bud of* $x_i(t)$ *for* $i = 1, \ldots, d$.

Example 7. Given a cubic Bézier curve $\mathbf{C}(t) = \mathbf{P}_0\, B_0^3(t) + \mathbf{P}_1\, B_1^3(t) + \mathbf{P}_2\, B_2^3(t) + \mathbf{P}_3\, B_3^3(t)$ with $t \in [0,1]$, control points $\mathbf{P}_0 = (1,1), \mathbf{P}_1 = (3,7), \mathbf{P}_2 = (6,2), \mathbf{P}_3 = (1,4)$ and Bernstein basis functions $B_0^3(t) = (1-t)^3, B_1^3(t) = 3t(1-t)^2, B_2^3(t) = 3t^2(1-t), B_3^3(t) = t^3$. A plot of this Bézier curve is given in Figure 2. We apply Algorithm 1 to the constituents of $\mathbf{C}(t)$:

$$x(t) = 1 + 6\,t + 3\,t^2 - 9\,t^3, \qquad y(t) = 1 + 18\,t - 33\,t^2 + 18\,t^3,$$

to compute a bud of $\mathbf{C}(t)$, which is $\mathbf{B}(t) = (x^b(t), y^b(t))$, written as follows:

$$x^b(t) = \frac{17}{156} + \frac{29}{52}\,t + \frac{9}{13}\,t^2 - \frac{3}{8}\,t^3, \qquad y^b(t) = -7 + 12\,t - \frac{9}{2}\,t^2 + \frac{3}{4}\,t^3.$$

We use the same set of constants $a_1, a_2, a_3, b_1, b_2, b_3$ as in Example 6. A plot of $\mathbf{B}(t)$ for $t \in [0,1]$ is also given in Figure 2. If we want, we can continue with computing a bud of $\mathbf{B}(t)$ et cetera. □

Figure 1 and Figure 2 show that the shape of a bud is more straightened-out than the shape of the original polynomial or curve. Indeed, this observation agrees

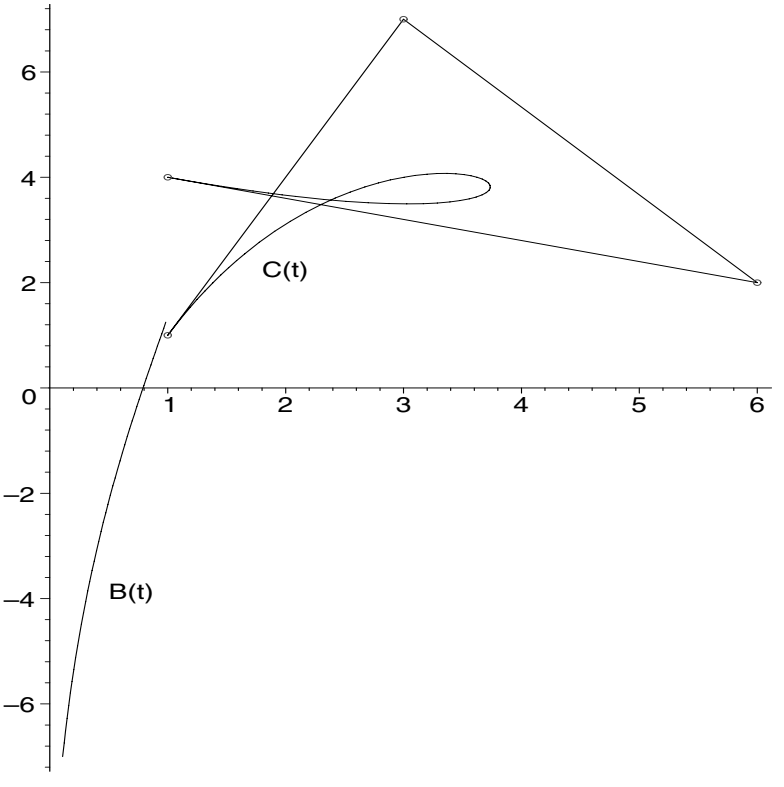


Fig. 2. A Bézier curve $\mathbf{C}(t)$ with its control polygon; $\mathbf{B}(t)$ is a bud of $\mathbf{C}(t)$

with the extrinsic curvature of the curves. Using the formula for curvature of a curve $(x(t), y(t))$ [4]:

$$\kappa = \left| \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{(\dot{x}^2 + \dot{y}^2)^{3/2}} \right|$$

where each dot denotes a differentiation with respect to t, and calculating up to two decimal places, we obtain the results: the maximum curvatures of the curves $(t, p(t))$, $(t, b(t))$, and $(t, c(t))$ are 26.18, 2.18, and 0.28 respectively, and the maximum curvatures of the Bézier curves $\mathbf{C}(t)$ and $\mathbf{B}(t)$ for $t \in [0, 1]$ are 5.64 and 0.02 respectively.

Precisely what are the geometric effects of varying the parameters a_k, b_k of the linear functions $X_k(t)$, $k = 1, \ldots, n$, in general? That needs further investigation.

3.1 Encoding/Decoding Shape Representations in Polynomials

We can take the process of computing a bud as an encoding process and the process of blossoming-and-evaluating-off-the-diagonal as a decoding process, or vice versa. We can consider the original polynomial $p(t)$ being encoded into a bud $b(t)$. Therefore, we have a scheme of encoding/decoding, and we could keep the constants secret if we wish. The curve $\mathbf{B}(t)$ in Example 7 is an encoded representation of the Bézier curve $\mathbf{C}(t)$.

3.2 Further Work

The concept of a bud can be extended to bivariate polynomials and surfaces. The details are reported in [10].

4 Summary

Any polynomial $p(t)$ of degree $d \leq n$ can be represented in the form of a blossom of another polynomial $b(t)$ of degree d evaluated off the diagonal at the linear functions $X_j(t) = a_j t + b_j$, $j = 1, \ldots, n$, chosen under the mild condition that the elementary symmetric functions $s_i(a_1, \ldots, a_n) \neq 0$ for $i = 1, \ldots, d$. The polynomial $b(t)$ is called a *bud* of the polynomial $p(t)$. An algorithm for finding a bud $b(t)$ of a given polynomial $p(t)$ is provided. Successively, a bud of $b(t)$ can be computed and so on, to form a sequence of shape representations. The information represented by the original polynomial is preserved in its buds. When the process of computing a bud is considered as an encoding process and the process of blossoming-and-evaluating-off-the-diagonal as a decoding process, or vice versa, we have a scheme for encoding/decoding shape representations which embody geometric design information.

Acknowledgement

I am grateful to the reviewers of this paper for their constructive comments and suggestions.

References

1. Bartels, R.H., Beatty, J.C., Barsky, B.A.: An Introduction to Splines for Use in Computer Graphics and Geometric Modeling. Morgan Kaufmann (1987)
2. Boehm, W., Farin, G., Kahmann, J.: A Survey of Curve and Surface Methods in CAGD. Computer Aided Geometric Design 1 (1984)1–60
3. de Casteljau, P.: Shape Mathematics and CAD. Kogan Page, London (1986)
4. Farin, G.: Curves and Surfaces for CAGD: A Practical Guide. Fifth edition. Academic Press (2002)
5. Goldman, R.N., Lyche, T. (eds.): Knot Insertion and Deletion Algorithms for B-Spline Curves and Surfaces. SIAM (1993)
6. Ramshaw, L.: Blossoming: A Connect-the-Dots Approach to Splines. Digital Equipment Corporation, SRC Report, June 21 (1987)
7. Stefanus, L.Y.: Blossoming off the Diagonal. Ph.D. Thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada (1991)
8. Stefanus, L.Y., Goldman, R.N.: Discrete Convolution Schemes. In: Lyche, T., Schumaker, L.L. (eds.): Mathematical Methods in Computer Aided Geometric Design II, Academic Press (1992)
9. Stefanus, L.Y., Goldman, R.N.: On the Linear Independence of the Bivariate Discrete Convolution Blending Functions. Australian Computer Science Communications 20(3)(1998) 231-244
10. Stefanus, L.Y.: Surface Representations using Blossoms and Buds. In preparation.

Manifold T-Spline

Ying He, Kexiang Wang, Hongyu Wang, Xianfeng Gu, and Hong Qin

Center for Visual Computing (CVC) and Department of Computer Science
Stony Brook University, Stony Brook, NY, 11794-4400, USA
{yhe, kwang, wanghy, gu, qin}@cs.sunysb.edu

Abstract. This paper develops the manifold T-splines, which naturally extend the concept and the currently available algorithms/techniques of the popular planar tensor-product NURBS and T-splines to arbitrary manifold domain of any topological type. The key idea is the global conformal parameterization that intuitively induces a tensor-product structure with a finite number of zero points, and hence offering a natural mechanism for generalizing the tensor-product splines throughout the entire manifold. In our shape modeling framework, the manifold T-splines are globally well-defined except at a finite number of extraordinary points, without the need of any tedious trimming and patching work. We present an efficient algorithm to convert triangular meshes to manifold T-splines. Because of the natural, built-in hierarchy of T-splines, we can easily reconstruct a manifold T-spline surface of high-quality with LOD control and hierarchical structure.

1 Introduction

Despite many new shape representations proposed in recent years, to date, NURBS remain the prevailing industrial standard for surface modeling in CAD/CAM primarily because of their many attractive geometric properties and their dominant use in modeling and design software industry. Nevertheless, they exhibit two major shortcomings: (1) NURBS control points must always align themselves in a rectangular grid. As a result, localized details and sharp features can not be easily accommodated without introducing many more control points via knot insertion. Moreover, level-of-detail (LOD) control and hierarchical structure facilitating multiresolution analysis are impossible using a single-level NURBS; (2) due to the nature of its rectangular structure, a single NURBS surface can only represent very simple shapes such as open surfaces or tori. In practice, in order to modeling surfaces of complicated topology, one must define a network of tensor-product *B*-spline or NURBS patches and maintain certain continuity requirement between adjacent patches [1,2]. Furthermore, surface trimming and abutting are oftentimes unavoidable.

To combat the above deficiencies of tensor-product NURBS, two recently developed techniques, T-spline [3] and manifold spline [4], have been introduced in shape modeling community. T-splines, developed by Sederberg, Zheng, Bakenov, and Nasri [3], are a generalization of NURBS surfaces that are capable of significantly reducing the number of superfluous control points by using the T-junction mechanism. The main difference between a T-spline control mesh and a NURBS control mesh is that T-splines allow a row or column of control points to terminate at anywhere without strictly enforcing the rectangular grid structure throughout the parametric domain. Consequently,

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 409–422, 2006.
© Springer-Verlag Berlin Heidelberg 2006

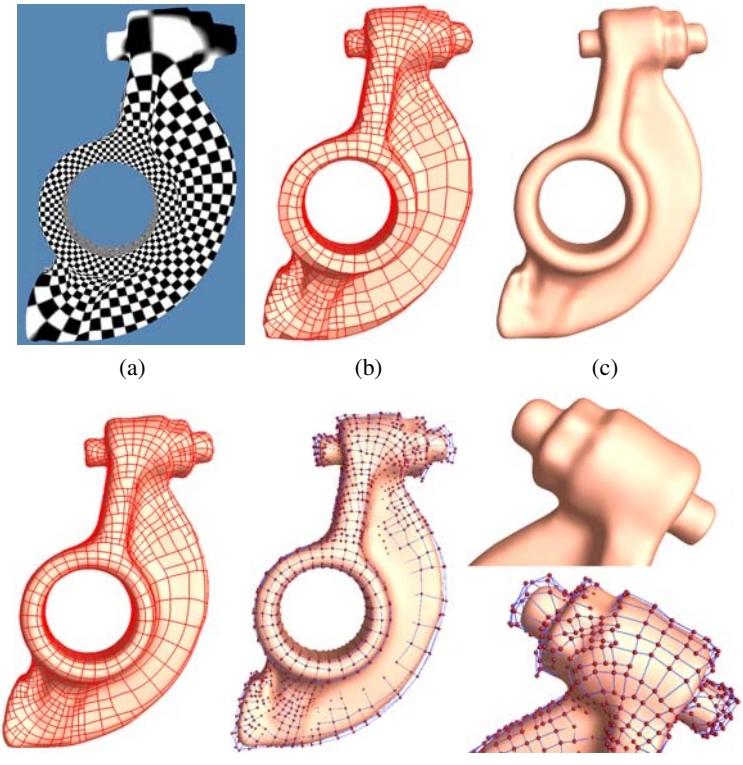


Fig. 1. Modeling the genus-one Rocker Arm model by manifold T-spline. (a) The conformal structure induces a natural curvilinear coordinate on the manifold domain. (b) Construct the domain manifold by tracing the iso-curves of the global conformal parameterization. Note that the domain manifold M contains only quadrilaterals and T-junctions. (c) A cubic (C^2-continuous) manifold T-spline surface. (d) The red curves on the manifold T-spline surface are the images of the edges on the domain manifold along the u and v directions. (e) $2,121$ control points are highlighted. (f) The close-up view of the details.

T-splines enable much better local refinement capabilities than NURBS. Furthermore, using the techniques presented in [3], it is possible to merge adjoining T-spline surfaces into a single T-spline without adding new control points. However, this patching process requires that the knot intervals of the to-be-merged edges must establish an one-to-one correspondence between the two surfaces.

Manifold spline, presented by Gu, He, and Qin [4], is a general theoretical framework, in which the existing spline schemes defined over planar domains can be systematically generalized to any manifold domain of arbitrary topology (with or without boundaries) using affine structures. They demonstrated the idea of manifold spline only using triangular B-splines because of the attractive properties of triangular B-splines, such as arbitrary triangulation, parametric affine invariance, and piecewise polynomial reproduction. Despite the generality of triangular B-splines, they have not been used in an industrial setting due to their modeling complexity in evaluation, differential

property computation, and data management. In practice, 2D-array-like control point layout facilitates the effective computation, shape analysis, and perhaps above all, the simplicity of data structure. In spite of all the potential modeling power associated with our manifold spline, its has not gained a widespread popularity mainly due to the fact that its constituent is triangular *B*-spline. To further promote its utility in real-world applications, we must bring tensor-product splines such as NURBS into our manifold spline framework and demonstrate its efficacy. Our current research reported here aims to serve this need.

In particular, this paper presents the manifold T-splines, a natural and necessary integration of T-splines and manifold splines, with a goal to retain all the desirable properties while overcoming the aforementioned modeling drawbacks at the same time. Manifold T-splines can be directly defined over the manifold of arbitrary topology to accurately represent various shapes with complicated geometry and topology. Manifold T-splines naturally inherit all the attractive properties from T-splines defined over a planar domain, including the powerful local refinement capabilities and the hierarchical organization for LOD control. Definitely worth mentioning here is that its building block comes from tensor-product NURBS, an industrial standard in all CAD/CAM software systems with a large variety of algorithmic routines available. The systematic development of our manifold T-splines streamlines the entire process of our manifold splines by demonstrating the intrinsic connection between manifold splines and popular tensor-product NURBS. As a result, our manifold T-splines are suitable for both expert users and novice users. Users, who are familiar with NURBS, can easily embrace our manifold T-splines without extra difficulties, as all the software routines and existing algorithms for tensor-product NURBS remain unchanged in our new modeling framework. Figure 1 shows the manifold T-spline of genus-one Rocker Arm model. This manifold T-spline is a single spline representation without any trimming, cutting and patching work.

2 Previous Work

2.1 Hierarchical Splines and B-Splines/Bézier Splines Based Modeling Techniques

Forsey and Bartels presented the hierarchical *B*-spline [5], in which a single control point can be inserted without propagating an entire row or column of control points. Gonzalez-Ochoa and Peters [6] presented the localized-hierarchy surface splines which extended the hierarchical spline paradigm to surfaces of arbitrary topology. Yvart *et al* presented the G^1 hierarchical triangular spline which works on any 2-manifold triangular mesh of arbitrary genus and has no restriction on the connectivity of the vertices. They demonstrated hierarchical triangular splines in smooth adaptive fitting of 3D models in [7]. In [3], Sederberg *et al.* presented the T-spline, a generalization of the non-uniform B-spline surfaces. T-spline control grids need not to be totally regular. In particular, they allow T-junctions, and lines of control points need not to traverse the entire control grid. Therefore, T-splines enable true local refinement without introducing additional, unnecessary control point in nearby regions. Sederberg *et al.* also

developed an algorithm to convert NURBS surfaces into T-spline surfaces, in which a large percentage of superfluous control points are eliminated [8].

There also exist large number of literatures in modeling 3D shapes of complicated topology using *B*-splines and Bézier patches. Due to the space limitation, we just name a few of them. Peters constructed C^1 surfaces of arbitrary topology using biquadratic and bicubic spliens [9]. This method generalizes the standard biquadratic tensor-product *B*-spline representation to irregular meshes, i.e., there are no regularity restrictions on the input meshes. Hahmann and Bonneau [10] presented a method for interpolating 2-manifold triangular meshes with a parametric surface composed of Béizer patches of degree 5. This method can generate visually pleasing shapes without the unwanted undulations, even if the interpolated mesh has irregular features. Loop and DeRose presented a method for constructing surfaces from control meshes of arbitrary topological type [11]. This method is based on S-patches which generalize biquadratic and bicubic B-splines. The above *B*-splines and Bézier spline based methods share one common property: they require the control points along the boundaries of adjacent spline patches satisfying certain constraints to reach G^1, C^1 or C^2 continuity. Therefore, only part of the control points serve the geometric modeling purpose.

2.2 Manifold Construction

There are some related work on defining functions over manifold. In essence, manifold construction is different from the above work on splines of arbitrary topology. The shape (2-manifold) is covered by several charts. One builds functions on each chart. Due to certain continuity requirement of the transition functions between overlapping charts, the smoothness properties of the manifold functions are *automatically* guaranteed. Therefore, there are no restrictions/constraints on the control points. All the control points are free variables in the entire modeling process. Furthermore, manifold constructions can generate C^k smooth surfaces.

Grimm and Hugues [12] pioneered a generic method to extend B-splines to surfaces of arbitrary topology, based on the concept of overlapping charts. Cotrina *et al.* proposed a C^k construction on manifold [13,14]. Ying and Zorin [15] presented a manifold-based smooth surface construction method which has C^∞-continuous with explicit non-singular parameterizations only in the vicinity of regions of interest.

More recently, Gu *et al.* [4] developed a general theoretical framework of manifold splines in which spline surfaces defined over planar domains can be systematically generalized to any manifold domain of arbitrary topology (with or without boundaries). Manifold spline is different from the above manifold construction methods in the following aspects: 1) The transition functions of manifold spline must be affine. Therefore, the requirements of manifold spline is much stronger than previous work. That is why topological obstruction plays an important role in the construction. 2) Manifold spline produces either polynomial or rational polynomials. On any chart, the basis functions are always polynomials or rational polynomials, and represented as *B*-splines or rational *B*-splines.

To further improve our manifold spline results, in this paper we develop the manifold T-spline, which combine the benefits of our manifold spline and T-spline towards a more practical solution on surface modeling and simulation.

3 Manifold T-Spline

As pointed out in [4], if a particular planar spline scheme is invariant under the parametric affine transformation, it can be generalized to manifold domain of arbitrary topology with no more than Euler number of singular points. For example, triangular *B*-splines and Powell-Sabin splines have been generalized from the planar domain to manifold of arbitrary topology [4,16].

T-splines [3] are a generalization of NURBS surfaces that are capable of significantly reducing the number of superfluous control points. T-splines are parametric affine invariant, and therefore, they can be generalized to manifold domain without theoretical difficulties. The overview of the construction algorithm is as follows:

Algorithm. Construction of manifold T-spline
Input: A polygonal mesh P, maximal fitting tolerance ε
Output: A manifold T-spline F which approximates P

1. Compute the global conformal parameterization of P.
2. Construct the domain manifold M (a coarse T-mesh) according to the conformal structure of P.
3. Assign the knot interval for each edge of M to get the initial T-spline F.
4. Compute the control points of F by minimizing a linear combination of the interpolation and fairness functional.
5. Locally refine the T-spline F if the fitting error is bigger than the user specified fitting tolerance ε and repeat step 4. Otherwise, output F.

3.1 Global Conformal Parameterization

Suppose P is a surface with handles, either open or closed. A global conformal parameterization is a map $\phi : P \to \mathbb{R}^2$, such that each point p on M is mapped to a point on the planar parametric domain $\phi(p) = (u(p), v(p))$. Furthermore, the map ϕ is angle preserving, which is equivalent to the following fact: suppose we arbitrarily draw two intersecting curves γ_1, γ_2 on M, the intersection angle is α, then the intersection angle of their images $\phi(\gamma_1)$ and $\phi(\gamma_2)$ is also α. Mathematically, the conformality of the parameterization is formulated in the following way: the first fundamental form of M under conformal parameterization (u, v) is represented as $ds^2 = \lambda^2(u, v)(du^2 + dv^2)$, where λ is called the conformal factor, which indicates the area ratio between the area on M and that on the plane.

In practice, it is more convenient to compute the gradient fields of ϕ, namely $(\bigtriangledown u, \bigtriangledown v)$. If ϕ is conformal, then it satisfies the following criteria:

$$\bigtriangledown v(p) = \mathbf{n}(p) \times \bigtriangledown u(p),$$

where $\mathbf{n}(p)$ is the normal at the point p, also

$$\bigtriangledown \times \bigtriangledown u = \bigtriangledown \times \bigtriangledown v = 0,$$

because the gradient fields are curl-free. Formally, a pair of vector fields satisfying the above conditions is a *holomorphic 1-form*. There exists an infinite number of this kind

of vector fields. They form a $2g$ dimensional real linear space, where g is the number of handles of P.The integration curves $\bigtriangledown u$ and $\bigtriangledown v$ are called *horizontal and vertical trajectories*, respectively. It is obvious that the horizontal and vertical trajectories are orthogonal everywhere and two horizontal (vertical) trajectories do not intersect each other in general. There are special points on P, where two horizontal trajectories intersect (two vertical trajectories also intersect). It can be proven that, at those points, the conformal factors are zero, therefore, such kind of points are called *zero points* of the holomorphic 1-form. By the Poincaré-Hopf theorem, every vector field on a closed surface of genus $g \neq 1$ must have zero points. The holomorphic 1-form has the unique property that it has the minimal number of zero points, i.e., $|2g-2|$ zero points.

The following theorem reveals the relationship between the conformal structure and the affine structure.

Theorem 1. ([4]) *Given a closed genus g surface M, and a holomorphic 1-form* ω. *Denote by* $Z = \{zeros\ of\ \omega\}$ *the zero points of* ω. *Then the size of Z is no more than* $2g-2$, *and there exists an affine atlas on* M/Z *deduced by* ω.

Essentially, Theorem 3 indicates that an affine atlas of a manifold M can be deduced from its conformal structure in a straightforward fashion.

3.2 Domain Manifold Construction

Unlike the manifold triangular B-spline which does not have any restriction on the domain manifold [4], manifold T-splines require that the domain manifold has mainly rectangular structure possibly with T-junctions. The global conformal parameterization induces the natural tensor-product structures on the domain manifold with Euler number of zero points, which furthermore induces the affine structure of the domain manifold. In the subsection, we present the method to construct the domain manifold (quad mesh with T-junctions). The method varies different types of surfaces. We explain the details for each case: genus zero closed surfaces, genus one closed surfaces, high genus closed surfaces and surfaces with boundaries.

Genus zero closed surfaces. Every genus zero closed surface P can be conformally mapped to a sphere. Practical algorithms for computing such maps are given in [17,18]. The idea used in [17] is that, for genus zero closed surfaces, conformal maps are equivalent to harmonic maps, which can be computed using heat flow method. Denote by $f : P \to \mathbb{S}^2$ the conformal map and (θ, ϕ) the spherical coordinates. The horizontal trajectories on P are the curves $f^{-1}(\phi = const.)$, and the vertical trajectories are $f^{-1}(\theta = const.)$. The preimages of the north and south poles are the zero points. The trajectories are orthogonal everywhere except at the zero points and form the conformal net. Figure 8 shows the conformal parameterization and the domain manifold of the genus zero Iphegenia model.

Genus one closed surfaces. The holomorphic 1-form ω on a genus one closed surface P is nonsingular everywhere, i.e., there are no zero points. Thus, the construction of domain manifold is straightforward. By integrating ω on P, the whole surface can be conformally mapped to a parallelogram on the plane, called the *fundamental period* of P. In general, this is not a rectangle, but a skewed parallelogram whose shape is determined by the conformal structure of P. If the fundamental period is a rectangle, then

all the horizontal and vertical trajectories forming the conformal net on the surfaces are closed circles. Otherwise, two families of curves parallel to the sides of the parallelogram are used as the trajectories. Figure 1 shows the conformal parameterization and domain manifold of the Rocker Arm model.

High genus closed surfaces. The global structure of conformal nets on high genus closed surfaces is more complicated than the above cases due to the existence of zero points.

Once the differential form is obtained, we locate all its zero points and all the horizontal trajectories passing through them, namely, the *critical horizontal trajectories.* The critical horizontal trajectories partition the surface into several patches. Each patch is either a cylinder or a disk. All patches can be conformally mapped to a planar rectangle. Therefore, we can build the conformal net for each patch, and glue them together. Note that, the T-junctions are allowed along the boundaries of the patches. The zero points, the critical horizontal trajectories, and the patches form a graph, the so called *critical graph*.

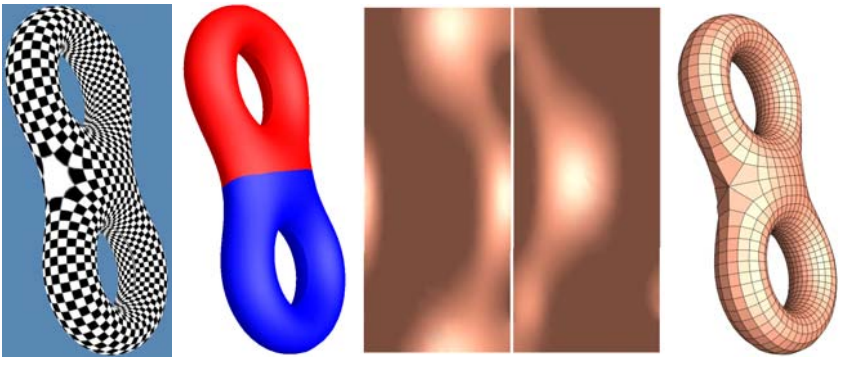


Fig. 2. Critical graph of the two-hole torus model. (a) Global conformal parameterization. (b) The critical horizontal trajectories partition the surface into two patches. Each patch is a cylinder. (c) Map each patch to a planar rectangle. (d) We build the quad mesh for each patch and then glue them together.

Surfaces with boundaries. For surface with boundaries, we need to double cover the original surfaces to make it become a closed surface (see [19] for the details of double covering technique). Generally, if P is of genus g and has b boundaries, then the double-covered surface $\bar{P}$ is a closed surface with genus $2g+b-1$. We compute the holomorphic 1-form basis of $\bar{P}$ and then find a special holomorphic 1-form $\omega = (\omega_u, \omega_v)$ on it such that ω_u is orthogonal to ∂P everywhere. This ω induces a conformal net on P for which all curves in ∂P are vertical trajectories. Figure 6 illustrates the critical graph of the Stanford Bunny. In order to get the uniform global conformal parameterization, three cuts are introduced in the original model, two are at the tips of ears, one is at the bottom. Therefore, it is topologically equivalent to a 2-hole disk. The double covered surface is of genus 2. The zero point is between the roots of the two ears. The critical

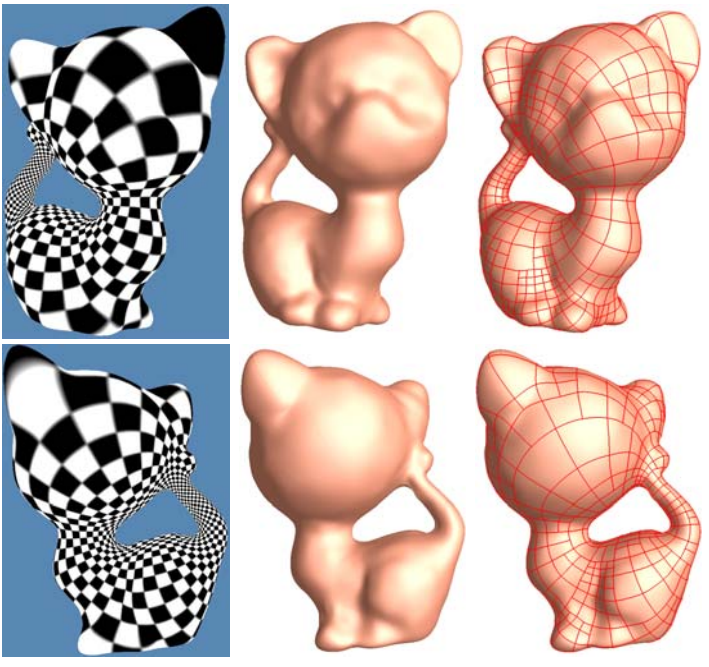


Fig. 3. Modeling the Kitten model using manifold T-spline with 765 control points

horizontal trajectories partition the surface into 2 connected components, each component is a topological disk which can be conformally mapped to a rectangle in the plane by integrating the holomorphic 1-form ω. Then the domain manifold can be constructed by remeshing each component.

3.3 Hierarchical Surface Reconstruction

Given the domain manifold M with conformal structure $\phi : M \to \mathbb{R}^2$, the manifold T-spline can be formulated as follows:

$$\mathbf{F}(\mathbf{u}) = \sum_{i=1}^{n} \mathbf{C}_i B_i(\phi(\mathbf{u})),\ \mathbf{u} \in M, \tag{1}$$

where B_is are basis functions and $\mathbf{C}_i = (x_i, y_i, z_i, w_i)$ are control points in $\mathbb{P}^4$ whose weights are w_i, and whose Cartesian coordinates are $\frac{1}{w_i}(x_i, y_i, z_i)$. The cartesian coordinates of points on the surface are given by

$$\frac{\sum_{i=1}^{n}(x_i, y_i, z_i)B_i(\phi(\mathbf{u}))}{\sum_{i=1}^{n} w_i B_i(\phi(\mathbf{u}))}. \tag{2}$$

Given a parameter $\mathbf{u} \in M$, the evaluation can be carried out on arbitrary charts covering $\mathbf{u}$.

We now discuss the problem of finding a good approximation of a given polygonal mesh P with vertices $\{\mathbf{p}_i\}_{i=1}^{m}$ by a manifold T-spline.

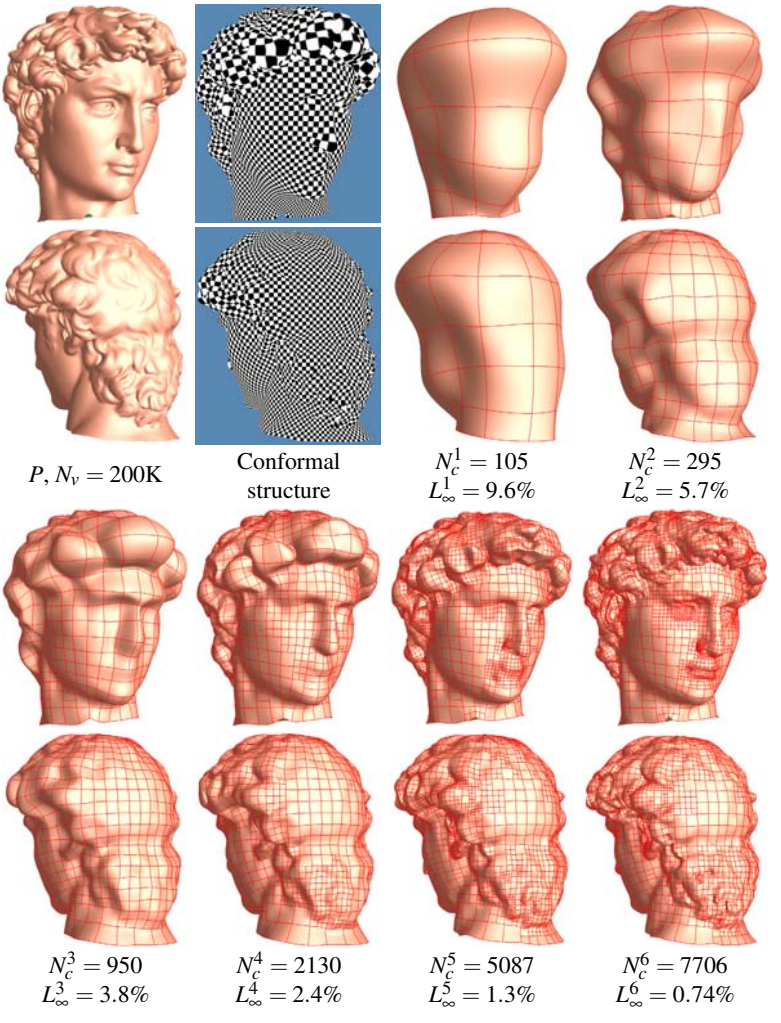


Fig. 4. Hierarchical surface reconstruction. N_c^i and L_∞^i are the number of control points and maximal fitting error in iteration i. N_v is the number vertices in the input polygonal mesh P. The input data is normalized to a unit cube.

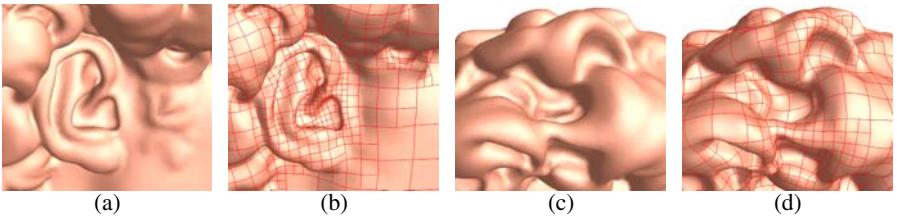
A commonly-used technology is to minimize a linear combination of interpolation and fairness functionals, i.e.,

$$\min E = E_{dist} + \lambda E_{fair}. \tag{3}$$

The first part is

$$E_{dist} = \sum_{i=1}^{m} \|\mathbf{F}(\mathbf{u}_i) - \mathbf{p}_i\|^2$$

where $\mathbf{u}_i \in M$ is the parameter for $\mathbf{p}_i$, $i = 1, \ldots, m$.



(a) (b) (c) (d)

Fig. 5. Close-up of the reconstructed details. (a),(c) The original polygonal mesh. (b),(d) Manifold T-spline where the red curves highlight the T-junctions.

The second part E_{fair} in (3) is a smoothing term. A frequently-used example is the thin-plate energy,

$$E_{fair} = \iint_M (\mathbf{F}_{uu}^2 + 2\mathbf{F}_{uv}^2 + \mathbf{F}_{vv}^2) du dv.$$

Note that both parts are quadratic functions of the unknown control points.

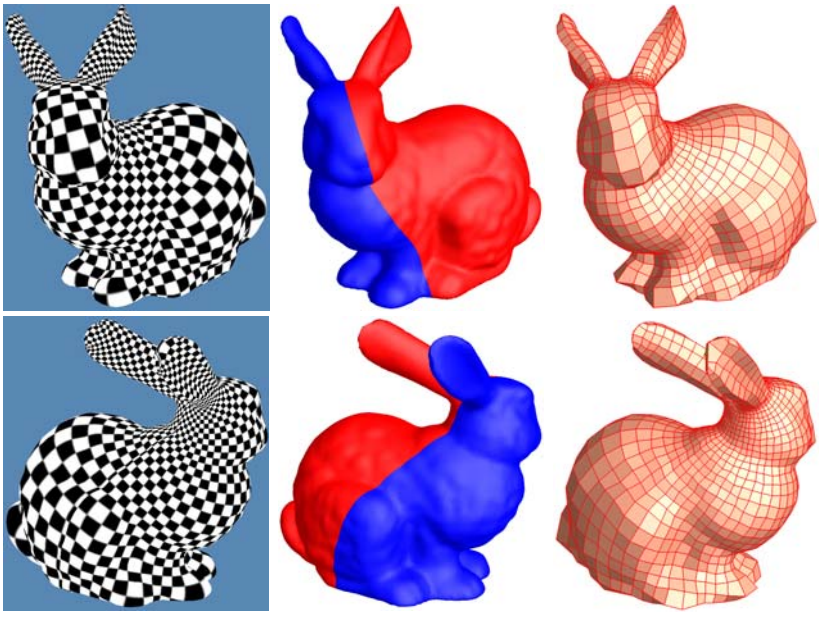
We solve Equation 3 for unknown control points using Conjugate Gradient method. The value and gradient of the interpolation functional and fairness functional can be computed straightforwardly.

In our method, we control the quality of the manifold T-spline spline by specifying the maximal fitting tolerance $L_\infty = \max \|\mathbf{F}(\mathbf{u}_i) - \mathbf{p}_i\|$, $i = 1, \ldots, m$. If the current surface does not satisfy this criterion, we employ adaptive refinement to introduce new degrees of freedom into the surface representation to improve the fitting quality. Because of the natural and elegant hierarchial structure of T-splines, this step can be done easily. Suppose a domain rectangle I violates the criterion and denote L_∞^I the L_∞ error on rectangle I. If the $L_\infty^I > 2\varepsilon$, split the rectangle I using 1-to-4 scheme; Otherwise, we divide I into two rectangles by splitting the longest edge.

After adaptive refinement, we then re-calculate the control points until the maximal fitting tolerance is satisfied. Figure 3.3 shows the whole procedure of hierarchical fitting of the David's head model. The initial spline contains only 105 control points and the maximal error $L_\infty = 8.6\%$. Through six iterations, we can obtain a much more refined spline with 7706 control points. The maximal fitting error reduces to 0.74%. As shown in the close-up view (Figure 5), our hierarchical data fitting procedure can produce high quality manifold T-splines with high-fidelity recovered details.

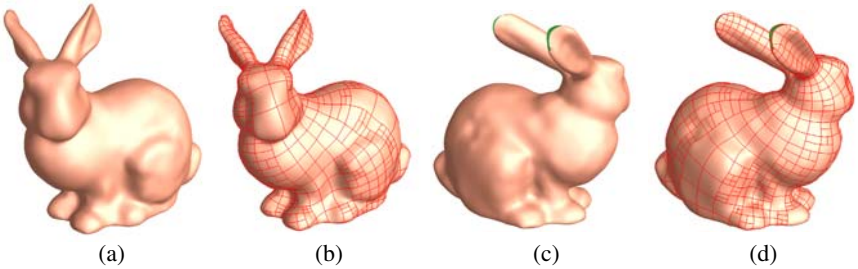
Table 1. Statistics of test cases. N_p, # of points in the polygonal mesh; N_c, # of control points; *rms*, root-mean-square error; L_∞, maximal error. The execution time measures in minutes.

Object	N_p	N_c	*rms*	L_∞	Time
David	200,000	7,706	0.08%	0.74%	39m
Bunny	34,000	1,304	0.09%	0.81%	18m
Iphegenia	150,000	9,907	0.06%	0.46%	53m
Rocker Arm	50,000	2,121	0.04%	0.36%	26m
Kitten	40,000	765	0.05%	0.44%	12m



(a) (b) (c)

Fig. 6. Critical graph and domain manifold. (a) shows the global conformal parameterization. (b) shows the critical horizontal trajectories partition the whole surface into two components. Each component can be conformally mapped to a rectangle. (c) Construct the domain manifold (quad mesh with T-junctions) by remeshing each component. T-junctions are allowed along the critical trajectories.



(a) (b) (c) (d)

Fig. 7. Converting Stanford Bunny into a manifold T-spline. (a)&(b) The front view. (c)&(d) The back view. The red curves illustrate the T-junctions on the spline surface.

3.4 Experimental Results

We have implemented a prototype system on a 3GHz Pentium IV PC with 1GB RAM. We perform experiments on various real-world surfaces. In order to compare the fitting quality across different models, we uniformly scale the models to fit within a unit cube. Table 1 summarizes the spline complexities and performance. The execution time includes the global conformal parameterization, domain manifold construction and

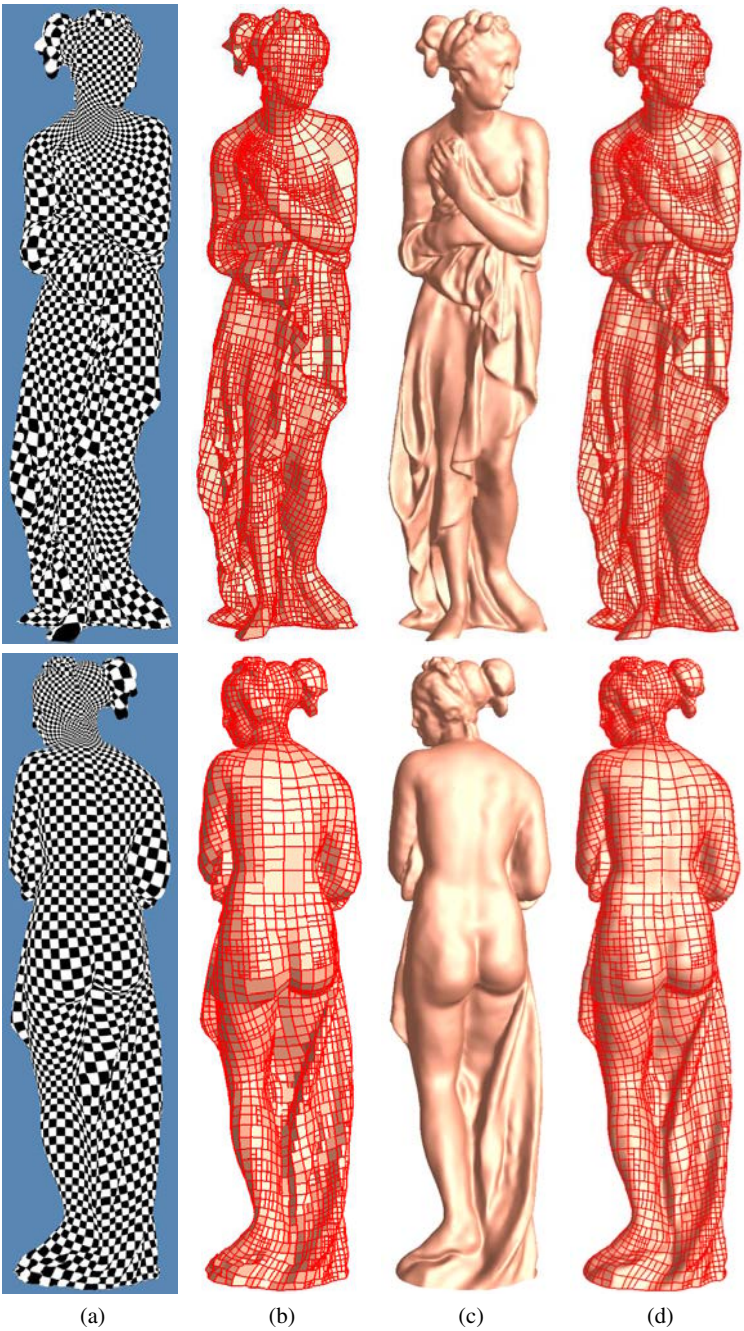


Fig. 8. Modeling the Iphegenia model using manifold T-spline. (a) Global conformal parameterization; (b) The domain manifold; (c) A C^2 manifold T-spline with $9,907$ control points; (d) The red curves are the images of the edges of the rectangles in the domain manifold.

hierarchical spline fitting. Figure 8 shows the manifold T-spline of Iphegenia model. Note that the details can be reconstructed easily with an appropriate number of control points.

4 Conclusions

In this paper, we have presented the manifold T-splines as a novel shape modeling paradigm for complicated geometry and topology. Built upon our previous work, the manifold T-splines integrate the algorithms and techniques of the widely-used, tensor-product NURBS and recently-proposed T-splines towards the effective shape modeling for arbitrary manifold. Our motivations come from two frontiers: (1) extending NURBS and T-splines to the manifold setting; and (2) promoting the widespread acceptance of manifold splines in real-world, shape modeling applications. The central idea is the global conformal parameterization that naturally induces a tensor-product structure over arbitrarily complicated manifold. In our shape modeling framework, the manifold T-splines are globally well-defined except at a finite number of extraordinary points without the need of any tedious and counter-intuitive trimming and patching work. Driven by the theoretical advances, we have developed an efficient algorithm automatically construct manifold T-splines from input data points. The salient features of our manifold T-splines include: natural hierarchical structure, local refinement, LOD control, tensor-product splines as building blocks, etc. Our new techniques are poised to be effective in shape modeling, and interactive design.

Acknowledgements

This work was supported in part by the NSF grants IIS-0082035 and IIS-0097646, and an Alfred P. Sloan Fellowship to H. Qin and the NSF CAREER Award CCF-0448339 to X. Gu. The models are courtesy of Leif Kobbelt, Cyberware, Stanford University.

References

1. Eck, M., Hoppe, H.: Automatic reconstruction of b-spline surfaces of arbitrary topological type. In: Proceedings of SIGGRAPH 96. (1996) 325–334
2. Krishnamurthy, V., Levoy, M.: Fitting smooth surfaces to dense polygon meshes. In: Proceedings of SIGGRAPH 96. (1996) 313–324
3. Sederberg, T.W., Zheng, J., Bakenov, A., Nasri, A.H.: T-splines and t-nurccs. ACM Trans. Graph. **22** (2003) 477–484
4. Gu, X., He, Y., Qin, H.: Manifold splines. In: Proceedings of ACM Symposium on Solid and Physical Modeling (SPM '05). (2005) 27–38
5. Forsey, D.R., Bartels, R.H.: Hierarchical b-spline refinement. In: SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press (1988) 205–212
6. Gonzalez-Ochoa, C., Peters, J.: Localized-hierarchy surface splines (less). In: SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics, New York, NY, USA, ACM Press (1999) 7–15

7. Yvart, A., Hahmann, S., Bonneau, G.P.: Smooth adaptive fitting of 3d models using hierarchical triangular splines. In: SMI '05: Proceedings of the International Conference on Shape Modeling and Applications 2005 (SMI' 05). (2005) 13–22
8. Sederberg, T.W., Cardon, D.L., Finnigan, G.T., North, N.S., Zheng, J., Lyche, T.: T-spline simplification and local refinement. ACM Trans. Graph. **23** (2004) 276–283
9. Peters, J.: Surfaces of arbitrary topology constructed from biquadratics and bicubics. In: Designing fair curves and surfaces. SIAM (1994) 277–293
10. Hahmann, S., Bonneau, G.P.: Polynomial surfaces interpolating arbitrary triangulations. IEEE Transactions on Visualization and Computer Graphics **9** (2003) 99–109
11. Loop, C.: Smooth spline surfaces over irregular meshes. In: SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques. (1994) 303–310
12. Grimm, C., Hughes, J.F.: Modeling surfaces of arbitrary topology using manifolds. In: SIGGRAPH. (1995) 359–368
13. Cotrina, J., Pla, N.: Modeling surfaces from meshes of arbitrary topology. Computer Aided Geometric Design **17** (2000) 643–671
14. Cotrina, J., Pla, N., Vigo, M.: A generic approach to free form surface generation. In: SMA '02: Proceedings of the seventh ACM symposium on Solid modeling and applications. (2002) 35–44
15. Ying, L., Zorin, D.: A simple manifold-based construction of surfaces of arbitrary smoothness. ACM Trans. Graph. **23** (2004) 271–275
16. He, Y., Jin, M., Gu, X., Qin, H.: A C^1 globally interpolatory spline of arbitrary topology. In: Proceedings of the 3rd IEEE Workshop on Variational, Geometric and Level Set Methods in Computer Vision. Volume 3752 of Lecture Notes in Computer Science. (2005) 295–306
17. Gu, X., Wang, Y., Chan, T., Thompson, P., Yau, S.T.: Genus zero surface conformal mapping and its application to brain surface mapping. IEEE Transactions on Medical Imaging **23** (2004) 949–958
18. Gotsman, C., Gu, X., Sheffer, A.: Fundamentals of spherical parameterization for 3d meshes. Proceedings of SIGGRAPH 03 (2003) 358–363
19. Gu, X., Yau, S.T.: Global conformal surface parameterization. In: Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing (SGP '03). (2003) 127–137

Composite $\sqrt{2}$ Subdivision Surfaces

Guiqing Li[1] and Weiyin Ma[2]

[1] School of Computer Science and Engineering,
South China University of Technology, Guangzhou, 510641, China
ligq@scut.edu.cn
[2] Department of Manufacturing Engineering and Engineering Management,
City University of Hong Kong, Hong Kong (SAR), China
mewma@cityu.edu.hk

Abstract. This paper presents a new unified framework for subdivisions based on a $\sqrt{2}$ splitting operator, the so-called composite $\sqrt{2}$ subdivision. The composite subdivision scheme generalizes 4-direction box spline surfaces for processing irregular quadrilateral meshes and is realized through various atomic operators. Several well-known subdivisions based on both the $\sqrt{2}$ splitting operator and 1-4 splitting for quadrilateral meshes are properly included in the newly proposed unified scheme. Typical examples include the midedge and 4-8 subdivisions based on the $\sqrt{2}$ splitting operator that are now special cases of the unified scheme as the simplest dual and primal subdivisions, respectively. Variants of Catmull-Clark and Doo-Sabin subdivisions based on the 1-4 splitting operator also fall in the proposed unified framework. Furthermore, unified subdivisions as extension of tensor-product B-spline surfaces also become a subset of the proposed unified subdivision scheme. In addition, Kobbelt interpolatory subdivision can also be included into the unified framework using VV-type (vertex to vertex type) averaging operators.

1 Introduction

In literature, most of previous subdivision schemes were proposed independently by different groups of researchers. The methods for constructing such subdivision schemes are also quite different. Motivated by techniques for the discrete generation of B-spline curves initiated by Cohen et al. [1], various universal subdivision frameworks or schemes have been developed. Such frameworks/schemes substantially extend previous subdivisions. The significance to establish unified subdivisions is at least threefold. Firstly, many new schemes of higher order can be explored. Secondly, the atomic decomposition potentially supports wavelet construction based on a lifting operator [2, 3], an important extension of subdivisions for modeling objects with fine details. The development of unified subdivisions also leads to simplicity for implementation and future standardization of subdivisions for engineering applications.

Some early work on unified subdivisions includes primal/dual subdivisions by Zorin and Schröder [4] and independently by Stam [5]. Variants of the Doo-Sabin [6] and Catmull-Clark [7] subdivisions can be derived, respectively, as the first dual and the first primal subdivisions from the framework. Apart from the generalization of B-spline

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 423 – 440, 2006.
© Springer-Verlag Berlin Heidelberg 2006

surfaces, Stam further applied the idea to triangular meshes and derived a unified framework of subdivisions to generalize triangular box spline surfaces of total degree $3m+1$. As a special case, it leads to a variant of Loop subdivision when $m=1$. Oswald and Schröder also applied the idea to establish a composite $\sqrt{3}$ subdivision [8] based on a $\sqrt{3}$ splitting operator [9]. As an extension, elementary averaging rules were defined to include weighted averaging operations from a geometric element (vertex, edge and face) to another. In a recent work, Oswald adapted the technique to a composite $\sqrt{7}$ subdivision [10]. Maillot and Stam extended the averaging unified framework to n-ary subdivisions [11]. Warren and Schaefer [12] derived a variant of the tri/quad subdivision of Stam and Loop [13] by decomposing the mask into a splitting operator and a sequence of weighted average operators. The introduction of weighted average further extends the family of atomic operators. Generation of sharp features was also addressed in [12]. Though only the tri/quad subdivision was discussed, it is straightforward to establish a unified framework for tri-quad meshes. Table 1 provides a summary of known unified subdivisions.

In addition, one may also find efforts to unify previous subdivisions based on other techniques. Observing that quadrilateral meshes can be viewed as tri-quad meshes, Velho implemented the Catmull-Clark and the Doo-Sabin subdivisions in the context of semi-4-8 meshes so as to apply the flexible multiresolution structure of 4-8 meshes to both schemes [14]. However, the geometric rules are not "atomic" enough and hence it does not lead to a unified framework/scheme. In another paper [15], Velho also decomposed subdivisions from the viewpoint of mesh splitting using a graph grammar formalism.

Table 1. Unified Subdivisions (In this table, N/A stands for not available)

Subdivisions	Splitting	Covered subdivisions	Parametric surfaces
Ref. [5]	1-4 Tri.	Loop subdivision	3-directional box splines
Refs. [4, 5]	1-4 Quad	Catmull-Clark, Doo-Sabin	B-splines surfaces
Comp. $\sqrt{3}$ [8]	$\sqrt{3}$ Tri.	$\sqrt{3}$ subdivision	N/A
Ref. [12]	1-4 quad/Tri.	Quad/Tri. subdivision	Hybrid parametric form
Comp. $\sqrt{7}$ [10]	$\sqrt{7}$ Tri.	N/A	N/A
Comp. $\sqrt{2}$ (This paper)	$\sqrt{2}$ Quad	4-8, $\sqrt{2}$, mid-edge, Catmull-Clark, Doo-Sabin	4-directional box splines, B-splines surfaces

In spite of the above work, however, it seems that there is no standard formulation up to now for the establishment of unified/composite subdivisions. For example, what do "atomic operators" or "elementary operators" indicate? What geometric rules can be viewed as atomic operators? In the scenarios of Oswald et al. [8, 10], many averaging operators were developed in order to enhance the versatility of the approach, but do we really need so many operators? Could we achieve the same ability using a small set of atomic operators? Is there any smallest set of operators?

This paper tries to address these problems to some extent by firstly introducing a formal treatment of atomic operators. We also demonstrate the power of the formula-

tion by involving the development of a new unified framework based on the $\sqrt{2}$ splitting operator for quadrilateral meshes. The proposed composite subdivision can properly reformulate almost all quadrilateral subdivisions into the same framework, which not only covers subdivisions based on the $\sqrt{2}$ splitting operator such as the mid-edge [16, 17], 4-8 [18], $\sqrt{2}$ [19, 20] schemes, but also includes 1-4 splitting based subdivisions such as Doo-Sabin and Catmull-Clark subdivisions. Furthermore, the unification generalizes a class of 4-direction box spline surfaces and the tensor-product spline surfaces. Kobbelt's interpolatory subdivision [21] can also be integrated into the new framework if allowing VV-type operators [22]. As a result of free conversion between quad-based $\sqrt{2}$ splitting and triangle-based 4-8 splitting, the unified framework can thus also be viewed as a unified 4-8 subdivision.

In the rest of the paper, section 2 presents a formal treatment of unified subdivision schemes and a new composite $\sqrt{2}$ subdivision scheme. Section 3 elaborates how the proposed composite $\sqrt{2}$ subdivision represents subdivisions that generalize 4-direction box-spline surfaces. Section 4 shows that the proposed composite $\sqrt{2}$ subdivision scheme also covers subdivisions that generalize B-spline surfaces. Section 5 provides some examples followed by some conclusions in section 6.

2 The Proposed Composite/Unified Subdivision

In literature on unified/composite subdivisions, atomic or elementary operators refer to averaging rules with small stencils, but there is little explanation why some rules are atomic or elementary and others are not. Here the meaning of "atomic" is twofold. On the one hand, it depends on the data structures used such that an atomic rule can be performed without introducing intermediate storage and extra data structure. On the other hand, an atomic rule should be the most elementary operator that can not be further decomposed into other atomic operators. In addition, the atomic rules should be independent of each other in a unified framework. Based on these considerations, we first introduce a basic data structure followed by the definition of basic types of atomic rules and the proposed composite subdivision in this section.

2.1 Data Structures

The data structures used for unified subdivisions should provide enough space for storing vertices of both primal and dual subdivisions, and sufficient adjacency information for answering proximity queries. The most popular structure, the half-edge structure, does not meet our requirements. For example, to perform a VF-type averaging operation, which averages all vertices of a face into its dual vertex (a point attached to the face), one has to retrieve all vertices of the face from the half-edge structure. As the operation may repeat multiple times in a round of refinement for composite subdivisions of higher order, this will considerably increase the computing time.

As summarized in Table 2, we devise a minimum set of data structures that consists of three arrays, i.e. vertex, face and edge arrays.

- *Vertex array*: Each element of the vertex array records the position of a vertex v, the type t of the vertex and the number of neighboring faces $|N_f(v)|$, where $N_f(v)$ stands for the set of neighboring faces of v (see Table 2 (I)).
- *Face array*: The element of the face array saves an ordered set of vertices f of a face and its corresponding dual point c_f (see Table 2 (II)).
- *Edge array*: The element of the edge array records two incident vertices $e_v[2]$ and faces $e_f[2]$ of an edge (see Table 2 (III)).

In the following, we will not strictly discriminate "set" and "array" for the convenience of description and, for an arbitrary set Ψ, we always denote the element number of Ψ by $|\Psi|$.

Table 2. Basic data structures

(I) Vertex array	(II) Face array	(III) Edge array
STRUCT _VERTEX	STRUCT _FACE	STRUCT _EDGE
Vector3D v;	VertexSet f;	integer $e_v[2]$;
integer t_v, $\|N_f(v)\|$;	Vector3D c_f;	integer $e_f[2]$;

2.2 Topological Operators $T_{\sqrt{2}}$ and T_D

A subdivision scheme consists of two classes of rules, i.e. topological rules and geometric rules. Topological rules specify how the vertices are connected to form the refined mesh after each level of subdivision. Geometric rules are responsible for computing the coordinates of refined mesh vertices. As described in [19], the $\sqrt{2}$ operator is the quadrilateral version of the 4-8 refinement [18]. Li et al. separately established an approximatory subdivision [19], which can be regarded as a variant of the 4-8 subdivision, and an interpolatory subdivision [20] based on the $\sqrt{2}$ operator in the setting of quadrilateral meshes. The operator was also devised to convert an arbitrary polygonal mesh into a quadrilateral one by Kobbelt [21] and to produce tri-quad meshes by Velho and Zorin [18]. The proposed unified framework will be established with respect to the $\sqrt{2}$ operator. In the following, we summarize the $\sqrt{2}$ operator and a dual operator. The latter is introduced for switching primal and dual meshes.

$\sqrt{2}$ splitting operator. Given a mesh M, the operator refines M by inserting a new vertex into each face (called F-vertex). For each edge, a quad face is generated by respectively connecting its two endpoints to two F-vertices of its neighboring faces. After splitting, each new F-vertex is assigned the position of the dual point of the face as shown in Fig. 1. Denote the operator by $T_{\sqrt{2}}$ and the refined mesh by $T_{\sqrt{2}} \circ M$.

Dual mesh operator. To unify both primal and dual subdivision into the same framework we need another topological operator that transforms a given mesh into its dual one. Let M be a closed mesh and $\hat{M} = T_D \circ M$ be the dual of M, where T_D is

the so-called dual mesh operator. The vertex set of dual mesh $\hat{M}$ consists of the dual points of the faces of M and each vertex of M contributes a face of $\hat{M}$ which consists of the dual points of all adjacent faces of the vertex. Fig. 2 illustrates an example of the dual transformation. It should be noticed that the dual counterpart of a general quad mesh might be an arbitrary polygonal mesh.

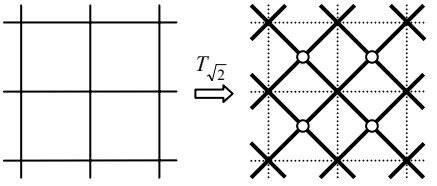


Fig. 1. Left: Initial mesh; Right: the result after one step of $\sqrt{2}$ refinement (black lines) in which new vertices are the dual points of the initial mesh (dotted lines)

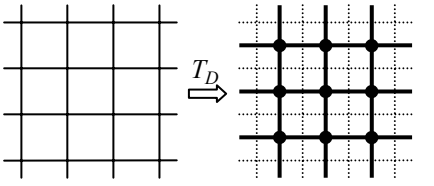


Fig. 2. Dual operation. Left: Initial mesh; Right: the dual mesh (black lines) generated by performing dual operator T_D on the initial mesh (dotted lines)

2.3 Atomic Geometric Rules

In the setting of unified or composite subdivisions, geometric rules generally comprise of a sequence of weighted averaging operators, which are performed after splitting and thus do not alter the connectivity of the mesh. Now we write a mesh as triplet $M =< V, F, E >$ where V, F and E are respectively a _VERTEX array, a _FACE array and an _EDGE set. In the following, Definition 1 relates an atomic geometric rule to the representing structures of meshes while Definitions 2~4 present three types of geometric operators, namely one scaling operator (Definition 2) and two averaging operators (Definitions 3-4). In these definitions, $| f |$ stands for the number of vertices of a face f and $\left|N_f(v)\right|$ stands for the number of faces incident to vertex v.

Definition 1. A weighted averaging rule is called *atomic* if it can be performed without introducing extra data structure other than the basic data structures of (I) and (II) in Table 1.

Definition 2. *A V-type scaling operator* $S_V(\lambda)$ maps M to $S_V(\lambda) \circ M = < S_V(\lambda) \circ V, F, E >$, where $S_V(\lambda) \circ V = \{\lambda v : v \in V\}$ and λ is a real parameter. Also, *an F-type scaling operator* is defined as $S_F(\lambda) \circ M =< V, S_F(\lambda) \circ F, E >$, where $S_F(\lambda) \circ F = \{\lambda c_f : f \in F\}$ and c_f is the dual point of face f.

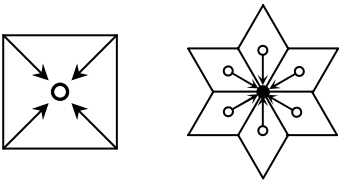
Definition 3. *A VF-type averaging operator* maps M to $A_{VF} \circ M = < V, A_{VF} \circ F, E >$, where

$$A_{VF} \circ F = \left\{ \frac{1}{| f |} \sum_{v \in f} v : f \in F \right\}.$$

Definition 4. *An FV-type averaging operator* maps M to $A_{FV} \circ M = < A_{FV} \circ V, F, E >$, where

$$A_{FV} \circ V = \left\{ \frac{1}{\left|N_f(v)\right|} \sum_{f \in N_f(v)} c_f : v \in V \right\}.$$

The scaling operator magnifies each vertex of the mesh by a factor λ. A VF-type operator updates the face centroid with the current position of its vertices (see Fig. 3a) while an FV-type operator replaces the vertices of the mesh with the average of the dual points of their adjacent faces (see Fig. 3b). Obviously, all these operators are atomic in the sense of definition 1.



(a) (b)

Fig. 3. Illustration of two atomic geometric rules: (a) operator A_{VF} and (b) operator A_{FV}

2.4 Outline of the Composite $\sqrt{2}$ Subdivision

Employing the topological and geometric operators introduced in section 2.4, we can now establish the framework for the proposed composite $\sqrt{2}$ subdivision.

Definition 5. *A round of primal composite* $\sqrt{2}$ *subdivision* is defined as a combination of one topological operation using the $T_{\sqrt{2}}$ operator followed by a sequence of geometric operations G_i, for $i = 1,2,\cdots,k$, i.e.,

$$S = G_k \circ G_{k-1} \circ G_{k-2} \cdots \circ G_1 \circ T_{\sqrt{2}}, \tag{1}$$

where k is an arbitrary positive integer and each of G_i, for $i = 1,2,\cdots,k$, can be one of the (partial) scaling, averaging, and blending operators. Appending the dual operator to S we get *a round of dual composite* $\sqrt{2}$ *subdivision:*

$$\hat{S} = T_D \circ G_k \circ G_{k-1} \circ G_{k-2} \cdots \circ G_1 \circ T_{\sqrt{2}}. \tag{2}$$

Given control mesh M^0, let $M^{j+1} =< V^{j+1}, F^{j+1}, E^{j+1} >$ be the result generated by a round of primal composite subdivision S^j on $M^j =< V^j, F^j, E^j >$. Then we get a dynamic subdivision scheme $\mathbb{S} = \cdots \circ S^j \circ \cdots \circ S^1 \circ S^0$ with each step defined by

$$M^{j+1} = S^j \circ M^j. \tag{3}$$

If all S^j use the same sequence of atomic operators, the subdivision reduces to a stationary scheme. Dual dynamic and stationary ones are defined in a similar way.

3 Generalization of 4-Direction Box Spline Surfaces

The composite subdivision given in Definition 5 is universal. This section shows that the proposed composite/unified subdivision properly covers a class of 4-direction box

spline surfaces as a special case (see Eqs. (1-2)). We firstly introduce some preliminaries on bivarite box spline surfaces and their subdivision setting in Sections 3.1 and 3.2, respectively. A unified framework is explored in Section 3.3 as generalization of box splines with total degree 4*n*-2, which is a special case of Definition 5.

3.1 Bivariate Box Spline Surfaces

Referring to the formulation in [23, 24], we denote Z and R the sets of integers and real numbers, respectively. For k-direction sequence $\Gamma = \left(\mathtt{d}_i = (a_i, b_i) \in Z^2, i = 1,2,\cdots,k\right)$ holding $\det(\mathtt{d}_1\ \mathtt{d}_2) = 1$, let β_k be the hyper-volume determined by Γ. The cross-section determined by hyper-plane $\sum_{1\le i\le k} \mathtt{d}_i t_i = \mathtt{x}$ can be expressed as follows [24]:

$$\beta_k(\mathtt{x}) = \left\{ \{t_1, t_2, \cdots, t_k\} \in \beta_k \mid \sum_{1\le i\le k} \mathtt{d}_i t_i = \mathtt{x} \right\}.$$

A bivariate box spline $B_k(\mathtt{x}) = B_k(\mathtt{x} \mid \mathtt{d}_1, \mathtt{d}_2, \cdots, \mathtt{d}_k)$ associated with the set of vectors Γ is then defined as the normalized cross-sectional volume [23]:

$$B_\kappa(\mathtt{x}) = \frac{1}{vol_k \beta_k} vol_{k-2} \beta_k(\mathtt{x}).$$

Box spline surfaces associated with Γ are further expressed as the linear combination of vertices of given regular control nets in terms of the above box splines:

$$C(\mathtt{x}) = \sum_{\tau \in Z^2} v_\tau B(\mathtt{x} - \tau), \tag{4}$$

where vertices $v_\tau \in R^3$ are control points. As box splines are normalized functions, box spline surfaces hold the affine invariant property.

3.2 Subdivision for Box Spline Surfaces

Based on the above definition of box splines, it is possible to establish the scaling relation between box splines of different resolutions by uniformly subdividing the polytopes into small ones. This leads to subdivisions of box spline surfaces [23, 25].

The box spline surfaces defined in (4) can also be represented in fine resolution

$$C(\mathtt{x}) = \sum_{\tau \in Z^2} v_\tau^2 B(2\mathtt{x} - \tau),$$

where vertices of finer mesh $\{v_\tau^2 : \tau \in Z^2\}$ are evaluated by the following process [23]:
(i) Firstly, initialize the new control net as follows

$$w_\tau^0 = \begin{cases} 0 & \tau/2 \notin Z^2 \\ v_{\tau/2} & \tau/2 \in Z^2 \end{cases} \tag{5}$$

(ii) Then recursively average along given directions as

$$w_\tau^r = \frac{1}{2}(w_\tau^{r-1} + w_{\tau - \mathtt{d}_r}^{r-1}),\ r = 1,2,\cdots,k, \tau \in Z^2 \tag{6}$$

(iii) Finally obtain

$$v_\tau^2 = 4w_\tau^k, \text{ for all } \tau \in Z^2 \tag{7}$$

To help to understand the above process and to briefly explain the subdivision procedure, let us see a special case with $\Gamma = (\mathtt{d}_1\mathtt{d}_2\mathtt{d}_3\mathtt{d}_4) = ((1,0),(0,1),(1,1),(1,-1))$. As also shown in Fig. 4a, it follows from Eq. (5) that:

$$w_{2i,2j}^0 = v_{i,j}, w_{2i-1,2j}^0 = w_{2i,2j-1}^0 = w_{2i-1,2j-1}^0 = 0, (i,j) \in Z^2.$$

Noticing the above result and iteratively performing Eq. (6), we deduce

$$\begin{aligned} v_{2i+1,2j}^2 &= 4w_{2i+1,2j}^4 = 2\left(w_{2i+1,2j}^3 + w_{(2i+1,2j)-\mathtt{d}_4}^3\right) \\ &= \left(w_{2i+1,2j}^2 + w_{(2i+1,2j)-\mathtt{d}_3}^2 + w_{(2i,2j+1)}^2 + w_{(2i,2j+1)-\mathtt{d}_3}^2\right) \\ &= \tfrac{1}{2}\left(w_{2i+1,2j}^1 + w_{(2i+1,2j)-\mathtt{d}_2}^1 + w_{(2i,2j-1)}^1 + w_{(2i,2j-1)-\mathtt{d}_2}^1 + \right. \\ &\qquad \left. w_{(2i,2j+1)}^1 + w_{(2i,2j+1)-\mathtt{d}_2}^1 + w_{(2i-1,2j)}^1 + w_{(2i-1,2j)-\mathtt{d}_2}^1\right) \\ &= \tfrac{1}{4}\left(2w_{(2i,2j)}^0 + w_{(2i,2j-2)}^0 + w_{(2i-2,2j)}^0\right) \\ &= \tfrac{1}{4}\left(2v_{i,j} + v_{i,j-1} + v_{i-1,j}\right). \end{aligned} \tag{8}$$

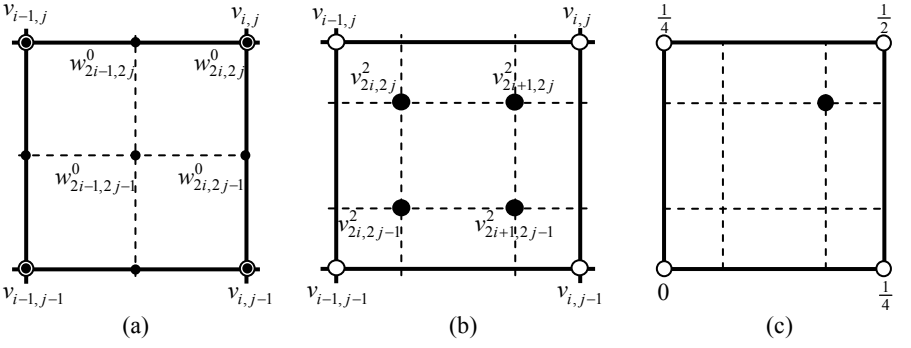


Fig. 4. Illustration of the subdivision: (a) indices after initialization ($v_{\tau/2} \Rightarrow w_\tau^0$); (b) vertices before and after subdivision; (c) the resulting subdivision mask

Similar to Eq. (8), the following can also be obtained (see Fig. 4b for their layout)

$$v_{2i,2j}^2 = \frac{2v_{i-1,j}+v_{i-1,j-1}+v_{i,j}}{4},\ v_{2i,2j-1}^2 = \frac{2v_{i-1,j-1}+v_{i-1,j}+v_{i,j-1}}{4},\ v_{2i+1,2j-1}^2 = \frac{2v_{i,j-1}+v_{i-1,j-1}+v_{i,j}}{4}.$$

It follows the same regular mask (see Fig. 4c) as that of mid-edge subdivision [17].

3.3 Composite Subdivision as Extension of 4-Direction Box Spline Surfaces

Now we are able to show that a special case of the composite subdivision of definition 5 produces subdivisions as extensions of the 4-directional box spline surfaces with direction sequences $\Gamma_n = (\mathtt{d}_1\mathtt{d}_2\mathtt{d}_3\mathtt{d}_4)^n = ((1,0),(0,1),(1,1),(1,-1))^n$. The task is

accomplished by demonstrating that the framework produces the same result as that of the subdivision process of the 4-directional box spline surfaces described in Eqs. (5-7).

First of all, the above 4-directional box splines are reduced to the Zwart-Powell functions for $n=1$, which are globally C^1 continuous and are piecewise polynomials of degree 2. Its subdivision generalization was addressed in [16, 17]. Consider composite operator

$$\overline{R}_{\sqrt{2}}(1)=T_D\circ A_{VF}\circ S_V(2)\circ T_{\sqrt{2}}$$

which consists of several phases: the $\sqrt{2}$ splitting operator $T_{\sqrt{2}}$ is first applied over the given control net; a scaling operator with factor 2 is further applied; a VF-type averaging operator is performed along vertical and horizontal directions; a dual topological operator follows. Fig. 5 illustrates the geometric meaning for each of the atomic operations involved. For arbitrary triangular meshes, $\overline{R}_{\sqrt{2}}(1)$ induces the simplest subdivision initiated in [16]. It is easy to show that performing $\overline{R}_{\sqrt{2}}(1)$ twice leads to one step of subdivision using a scheme described in [17].

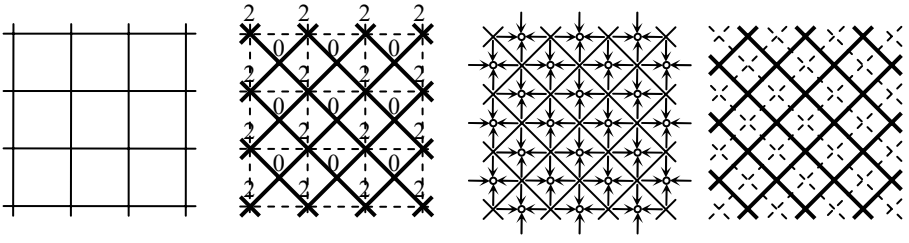


Fig. 5. Illustration of the composite operation: (a) The coarse mesh; (b) the refined mesh (heavy lines) using the composite operator $S_V(2)\circ T_{\sqrt{2}}$, in which newly inserted vertices and old ones are set to zero and scaled by 2, respectively; (c) perform VF averaging over the refined mesh of (b); (d) the dual mesh (drawn with heavy lines) of (c) (dotted lines)

Secondly, the degree of box spline surfaces for $n=2$ is six. The associated subdivisions include the 4-8 subdivision [18] as well as its variant for quadrilateral meshes, $\sqrt{2}$ subdivision [19]. In this case, the following composite operation yields the same mask as those employed in 4-8 subdivision for both regular and irregular cases:

$$R_{\sqrt{2}}(1)=A_{FV}\circ A_{VF}\circ S_V(2)\circ T_{\sqrt{2}}.$$

The above conclusions can be verified as follows. It is easy to show that $R_{\sqrt{2}}(1)$ transforms the 1-neighborhood of vertex v_0 of valence n with adjacent quads $f_i=(v_0,v_{2i-1},v_{2i},v_{2i+1}),i=1,2,\cdots,n$ to a new 1-neighborhood of v_0^1 with adjacent quads $f_i^1=(v_0^1,v_{2i-1}^1,v_{2i}^1,v_{2i+1}^1),i=1,2,\cdots,n$ (see Fig. 6) such that

$$v_0^1=\frac{1}{n}\sum_{1\le i\le n}v_i^{d1}=\frac{1}{2}v_0+\frac{1}{2n}\sum_{1\le i\le n}v_{2i-1}\ ,\quad v_i^1=\frac{1}{4}\left(v_0+v_{2i-1}+v_{2i}+v_{2i+1}\right),$$

which are exactly identical to the V-vertex and F-vertex subdivision masks of the 4-8 subdivision [18], respectively.

Now, let us consider the general case $\Gamma_n = (\mathbf{d}_1\mathbf{d}_2\mathbf{d}_3\mathbf{d}_4)^n$ for arbitrary integer $n>0$. Notice that the final result in Eq. (7) is independent of the ordering of directions $\mathtt{d}_r$. We may therefore reorder the directional sequence as $\Gamma_n == \left((\mathbf{d}_3\mathbf{d}_4)^n(\mathbf{d}_1\mathbf{d}_2)^n\right)$ to perform the subdivision procedure defined in Eqs. (5)-(7). Namely, we first use the sequence $(\mathtt{d}_3\mathtt{d}_4)^n$ to carry out Eq. (6) for calculating $w_\tau^r(r=1,2,\cdots,2n)$, and then employ the sequence $(\mathtt{d}_1\mathtt{d}_2)^n$ for evaluating $w_\tau^{2n+r}(r=1,2,\cdots,2n)$. This yields the following results.

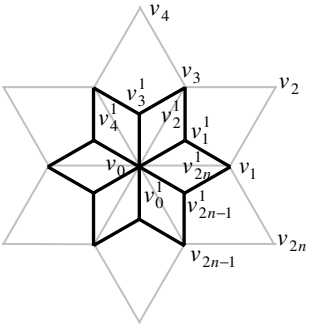


Fig. 6. Performing $R_{\sqrt{2}}(1)$ on the coarse (gray) mesh yields the refined (black) mesh.

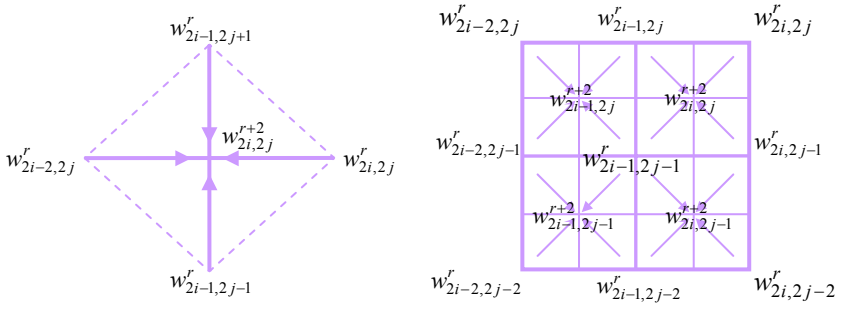


Fig. 7. Averaging configurations for lemma 1 (left) and lemma 2 (right) in which thick and thin lines stand for grid edges before and after average respectively while arrows are used to indicate the average styles

Lemma 1. The following is true for $r+2 \le 2n$ (See Fig. 7 (left))

$$
\begin{aligned}
w^{r+2}_{2i,2j} &= \tfrac{1}{4}\left(w^r_{2i,2j} + w^r_{2i-1,2j-1} + w^r_{2i-1,2j+1} + w^r_{2i-2,2j}\right),\\
w^{r+2}_{2i-1,2j} &= \tfrac{1}{4}\left(w^r_{2i-1,2j} + w^r_{2i-2,2j-1} + w^r_{2i-2,2j+1} + w^r_{2i-3,2j}\right),\\
w^{r+2}_{2i-1,2j-1} &= \tfrac{1}{4}\left(w^r_{2i-1,2j-1} + w^r_{2i-2,2j-2} + w^r_{2i-2,2j} + w^r_{2i-3,2j-1}\right),\\
w^{r+2}_{2i,2j-1} &= \tfrac{1}{4}\left(w^r_{2i,2j-1} + w^r_{2i-1,2j-2} + w^r_{2i-1,2j} + w^r_{2i-2,2j-1}\right).
\end{aligned}
$$

Proof. The proof is similar to the derivation of Eq. (8). ∎

Lemma 2. For $r+2 \le 2n$ we have (See Fig. 7 (right))

$$w_{2i,2j}^{r+2} = \tfrac{1}{4}\left(w_{2i,2j}^{r} + w_{2i-1,2j}^{r} + w_{2i,2j-1}^{r} + w_{2i-1,2j-1}^{r}\right),$$
$$w_{2i-1,2j}^{r+2} = \tfrac{1}{4}\left(w_{2i-1,2j}^{r} + w_{2i-2,2j}^{r} + w_{2i-1,2j-1}^{r} + w_{2i-2,2j-1}^{r}\right),$$
$$w_{2i-1,2j-1}^{r+2} = \tfrac{1}{4}\left(w_{2i-1,2j-1}^{r} + w_{2i-2,2j-1}^{r} + w_{2i-1,2j-2}^{r} + w_{2i-2,2j-2}^{r}\right),$$
$$w_{2i,2j-1}^{r+2} = \tfrac{1}{4}\left(w_{2i,2j-1}^{r} + w_{2i-1,2j-1}^{r} + w_{2i,2j-2}^{r} + w_{2i-1,2j-2}^{r}\right).$$

Proof. The proof is also similar to the derivation of Eq. (8). ■

Lemma 3 For $r \le n$, we have $w_{2i,2j}^{2r} \neq 0$, $w_{2i-1,2j-1}^{2r} \neq 0$, $w_{2i-1,2j}^{2r} = w_{2i,2j-1}^{2r} = 0$ for $(i, j) \in Z^2$.

Proof Carry out induction on r. Obviously, the lemma is true for $r = 0$ according to Eq. 5. Suppose that the lemma is true for integers not greater than r. Following Lemma 1, the lemma is also true for $r+1$. ■

Now we can introduce the proposed unified $\sqrt{2}$ subdivision as follows:

(i) **Primal unified $\sqrt{2}$ subdivision**

$$R_{\sqrt{2}}(m) = (A_{FV} \circ A_{VF})^m \circ S_V(2) \circ T_{\sqrt{2}}, m \ge 1 \tag{9}$$

(ii) **Dual unified $\sqrt{2}$ subdivision**

$$\overline{R}_{\sqrt{2}}(m) = T_D \circ A_{VF} \circ (A_{FV} \circ A_{VF})^{m-1} \circ S_V(2) \circ T_{\sqrt{2}}, m \ge 1 \tag{10}$$

Noticing that after the first $T_{\sqrt{2}}$ operation, faces of the refined mesh have the configuration as shown in Fig. 7 (left), we know that $A_{FV} \circ A_{VF}$ is equivalent to perform the average of Lemma 1 (or Eq. (6)) twice and hence $(A_{FV} \circ A_{VF})^m$ is equivalent to perform the average of Lemma 1 (or Eq. (6)) $2m$ times. Applying $T_{\sqrt{2}}$ again will initialize vertices like $w_{2i-1,2j}^{2r}, w_{2i,2j-1}^{2r}$ with zero but this does not update the position of the vertices of the grids due to Lemma 3. Also, considering that after the second $T_{\sqrt{2}}$ faces of the new refined mesh have the configuration as shown in Fig. 7 (right), $A_{FV} \circ A_{VF}$ is equivalent to perform the average of Lemma 2 (or Eq. (6)) twice and hence $(A_{FV} \circ A_{VF})^m$ is equivalent to perform the average of Lemma 2 (or Eq. (6)) $2m$ times. On the other hand, running $S_V(2)$ twice has the same effect as scaling with a factor 4 in Eq. (7). This results in the following conclusion.

Theorem 1 $\left(\overline{R}_{\sqrt{2}}(m)\right)^2$ and $\left(R_{\sqrt{2}}(m)\right)^2$ are respectively equivalent to the subdivision processes defined by Eqs. (5)~(7) for $[(1,0)\ (0,1)\ (1,1)\ (1,-1)]^{2m}$ and $[(1,0)\ (0,1)\ (1,1)\ (1,-1)]^{2m-1}$.

Theorem 1 indicates that the proposed framework defined in (9) and (10) are extension of 4-direction box spline surfaces to irregular meshes for direction sequence $[(1,0)\ (0,1)\ (1,1)\ (1,-1)]^{2m}$ and $[(1,0)\ (0,1)\ (1,1)\ (1,-1)]^{2m-1}$ respectively.

4 Composite Subdivision as Extension of B-Spline Surfaces

This section shows how to bring some 1-4 splitting based schemes for quadrilateral meshes, such as Doo-Sabin [6] and Catmull-Clark [7] subdivisions, into the proposed framework of composite $\sqrt{2}$ subdivision. While geometric operators described in section 2.3 are atomic operators, the VV-, VE-, EV-, EF-, FE-, FF- and EE- types averaging operators introduced in [8] are not atomic in the sense of Definition 1 due to the fact that all these operators require extra memory to save intermediate or final position data. In this section, we slightly relax the constraint of Definition 1 to include VV-type operations into the potential atomic setting.

Before defining new atomic operators, we assign a type flag to each vertex of the meshes produced in the splitting process. We distinguish odd and even number of subdivision with a different strategy for assigning the vertex type flags. In odd steps of subdivision, all old vertices are assigned with a flag of 0, while all newly inserted vertices (F-vertices) after $T_{\sqrt{2}}$ carry a flag of 1. In even steps of subdivision, the flag of all old vertices remain unchanged, while the flag of all newly inserted vertices (F-vertices) after $T_{\sqrt{2}}$ is set to 2. Using the type flags of mesh vertices, we are then able to describe our relaxation to the atomic geometric operators.

Definition 6. A geometric operator for modifying vertices is called *partial* if it is only applied to a subset of vertices of the mesh.

For example, performing the V-type scaling operator, FV-type averaging operator and VFV-type blending operator on vertices with type t and keeping other vertices unchanged will yield the partial V-type scaling operator $S_V(\lambda,t)$, partial FV-type averaging operator $A_{FV}(t)$ and partial VFV-type blending operator $B_{VFV}(\alpha,t)$, respectively. Obviously, these partial operators are also atomic because no extra storage is required for performing them. Here we only present the definition of $A_{FV}(t)$ which will be used later.

Definition 7. *A partial FV-type averaging operator associated with a vertex type t* maps mesh $M =< V, F, E >$ to mesh $A_{FV}(t) \circ M =< A_{FV}(t) \circ V, F, E >$, where

$$A_{FV}(t) \circ V = \left\{ \frac{1}{\left|N_f(v)\right|} \sum_{f \in N_f(v)} c_f : v \in V \text{ and } t_v = t \right\} \cup \{v : v \in V, t_v \neq t\}.$$

Consider the following composite operator

$$O = A_{FV}(1) \circ S_F(2) \circ A_{VF} \circ T_{\sqrt{2}}.$$

It is easy to verify that performing O on mesh M produces mesh $O \circ M$ in which F-vertices are computed as the average of vertices of the corresponding faces, while V-vertices remain unchanged. In addition, the dual point of each face in $O \circ M$ is equal to the midpoint of the corresponding edge in M. Fig. 8 illustrates the behavior of operator O, in which $v = \frac{1}{4}(v_0 + v_1 + v_2 + v_3)$ while the dual point of face f in $O \circ M$ is $c_f = \frac{1}{2}(v_0 + v_3)$. Therefore composite operator $T_{\sqrt{2}} \circ O$ leads to the linear 1-4 splitting for quadrilateral meshes as shown in Fig. 8. Denote $\overline{R}_{1\text{–}4}(1) = T_D \circ A_{VF} \circ T_{\sqrt{2}} \circ O$. It is easy to show that $\overline{R}_{1\text{–}4}(1)$ is equivalent to one step of Doo-Sabin subdivision for regular meshes. Furthermore, it is also true that $R_{1\text{–}4}(1) = A_{FV} \circ A_{VF} \circ T_{\sqrt{2}} \circ O$ is a composite operator that leads to a variant of the Catmull-Clark subdivision. Generally, we have the following conclusions.

1) Primal unified 1-4 subdivision based on $\sqrt{2}$ splitting

$$R_{1\text{–}4}(m) = (A_{FV} \circ A_{VF})^m \circ T_{\sqrt{2}} \circ A_{FV}(1) \circ S_F(2) \circ A_{VF} \circ T_{\sqrt{2}},\ m = 1,2,\cdots \tag{11}$$

2) Dual unified 1-4 subdivision based on $\sqrt{2}$ splitting

$$\overline{R}_{1\text{–}4}(m) = A_{VF} \circ (A_{FV} \circ A_{VF})^{m-1} \circ T_{\sqrt{2}} \circ A_{FV}(1) \circ S_F(2) \circ A_{VF} \circ T_{\sqrt{2}},\ m = 1,2,\cdots \tag{12}$$

Obviously, the unified framework (operators defined in Eqs. (11) and (12) as one step of subdivision) is equivalent to that described in [4, 5]. For regular control meshes, they produce B-spline surfaces of bi-degree $(2m+1)\times(2m+1)$ and $2m \times 2m$, respectively. Furthermore, with the introduction of VV-type operators, Li and Ma [22] derived a composite representation of Kobbelt interpolatory subdivision [21]. This composite interpolatory representation is also a special case of the proposed framework of unified subdivisions. Combining results of Sections 3 and 4, Table 3 summarizes previous subdivisions that are included in the proposed unified subdivision with corresponding operators.

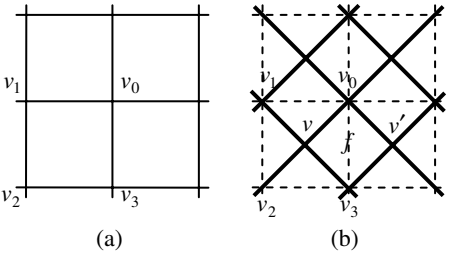


Fig. 8. Composite operator O : (a) The coarse mesh; (b) the result (mesh drawn with heavy lines) generated by O in which old vertices remain unchanged while newly inserted vertices are set to the corresponding face centroid

Table 3. Existing subdivisions covered by the proposed composite framework.

The covered subdivisions	Operators
Mid-edge [16,17]	$\overline{R}_{\sqrt{2}}(1)$
4-8[18],variant of $\sqrt{2}$ [19]	$R_{\sqrt{2}}(1)$
Variant of Doo-Sabin [6]	$\overline{R}_{1-4}(1)$
Variant of CC [7]	$R_{1-4}(1)$
Unified framework in [4]	$\overline{R}_{1-4}(m)$, $R_{1-4}(m)$

5 Results and Discussions

It is straightforward to implement the proposed composite subdivision. The program coded using VC++ runs on a PC computer with Windows XP operating system. The key ingredients of the program include topolicical operators $T_{\sqrt{2}}$ and T_D, and atomic geometric operators introduced in sections 2 and 4. A variety of known and new subdivisions are constructed through various compositions of these basic operations. Table 4 list the subdivisions employed for producing the examples presented in this section.

Table 4. Composite operators used to generate subdivision surfaces given in this section

Figures	(b)	(c)	(d)	(e)	(f)	(g)
Fig. 9	$\overline{R}_{\sqrt{2}}(1)$	$R_{\sqrt{2}}(1)$	$\overline{R}_{1-4}(1)$	$R_{1-4}(1)$		
Fig. 10	$\overline{R}_{\sqrt{2}}(1)$	$R_{\sqrt{2}}(1)$	$\overline{R}_{\sqrt{2}}(2)$	$R_{\sqrt{2}}(2)$	$\overline{R}_{\sqrt{2}}(3)$	$R_{\sqrt{2}}(3)$
Fig. 11	$\overline{R}_{1-4}(1)$	$R_{1-4}(1)$	$\overline{R}_{1-4}(2)$	$R_{1-4}(2)$	$\overline{R}_{1-4}(3)$	$R_{1-4}(3)$
Fig. 12	$R_{\sqrt{2}}(1)$	$R_{\sqrt{2}}(2)$	$R_{\sqrt{2}}(3)$			

The initial control mesh shown in the image of Fig. 9a has 20 vertices and 18 quads with extraordinary vertices. The four subdivisions illustrated in the figure are the mid-edge subdivision ($\overline{R}_{\sqrt{2}}(1)$, Fig. 9b) [16], the $\sqrt{2}$ version of the 4-8 subdivision ($R_{\sqrt{2}}(1)$, Fig. 9c)[18], variants of Doo-Sabin ($\overline{R}_{1-4}(1)$, Fig. 9d)and Catmull-Clark ($R_{1-4}(1)$, Fig. 9e) [4] subdivisions. Results corresponding to the first four $\sqrt{2}$ splitting operations are depicted for each case. From the figure, one can clearly see the difference of the geometry and connectivity of the resulting meshes with the increase of splitting levels. Figs. 10-12 show some further results of the proposed composite scheme on models of some quad-dominant tri/quad meshes. Fig. 10 illustrates the modeling of a bishop model with an initial control mesh of 250 vertices and 496 faces. All results are produced after six levels of $\sqrt{2}$ refinement. From the picture we can obviously observe that dual subdivisions suffer much shrinkage in contrast to primal subdivisions for the same number m of repeated averages. Similarly, Fig. 11

shows the modeling of a queen model (282 vertices and 560 faces). All results are generated after 6 levels of $\sqrt{2}$ refinement (or 3 levels of 1-4 refinement). Though we did not discuss the issue for boundary modeling in the present paper due to space limitations, it is quite important for the proposed scheme to support boundary and

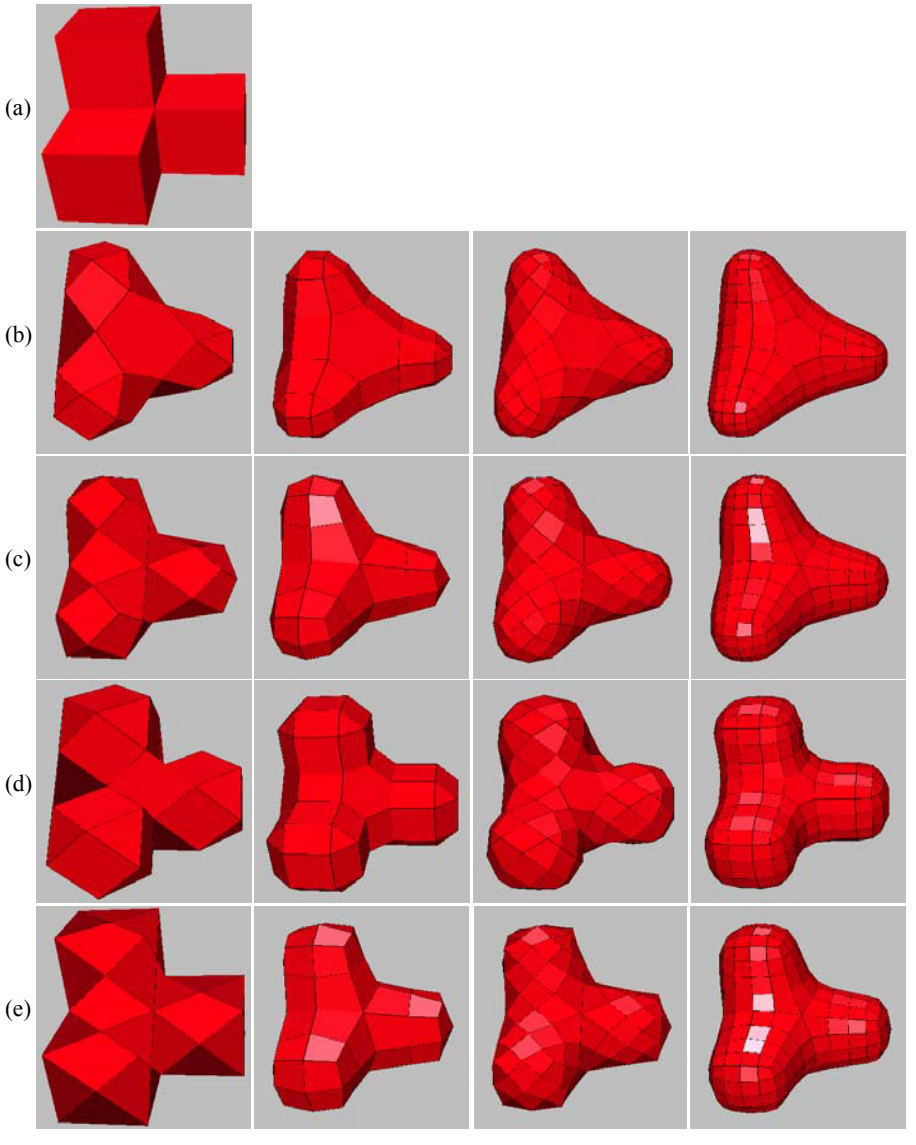
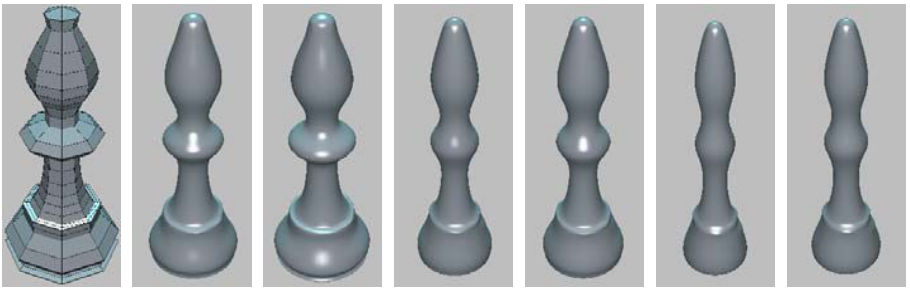


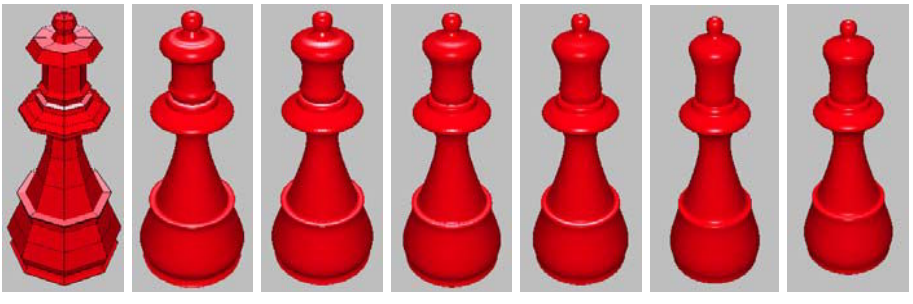
Fig. 9. An example illustrating four subdivisions produced by different compositions of atomic operations: (a) is the initial control net for all the four subdivisions (b)-(e)

crease features. We therefore give an example here to show that the proposed framework can also deal with open models as shown in Fig. 12. The initial banana model in this example has 272 vertices and 512 faces. The subdivision levels are also six.



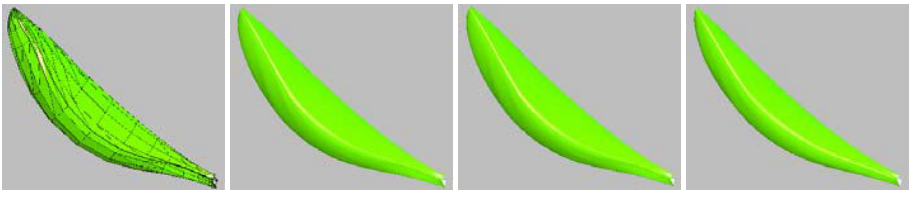
(a) (b) (c) (d) (e) (f) (g)

Fig. 10. Bishop model: (a) the initial control mesh; and (b)-(g) subdivision results generated by different composite subdivisions



(a) (b) (c) (d) (e) (f) (g)

Fig. 11. Queen model: (a) the control mesh; and (b)-(g) results generated by different composite subdivisions



(a) (b) (c) (d)

Fig. 12. Banana model: (a) the control mesh; and (b)-(d) are results produced by primal subdivisions $R_{\sqrt{2}}(1)$, $R_{\sqrt{2}}(2)$ and $R_{\sqrt{2}}(3)$, respectively

6 Conclusions

This article presents a new unified subdivision, the so-called composite $\sqrt{2}$ subdivision. The approach generalizes box spline surfaces of degree $4m+2$ associated with a set of vectors of 4 directions to arbitrary polygonal meshes. The composite $\sqrt{2}$ subdivision is established based on a formal treatment of unified subdivision schemes in section 2 of this paper such that the newly developed unified scheme covers as many subdivisions as possible with the minimum number of atomic rules. Noting the fact that two steps of $\sqrt{2}$ refinement lead to one step of 1-4 splitting, we also bring Doo-Sabin, Catmull-Clark subdivisions as well as their associated unified schemes described in [4, 5] into the proposed framework with the introduction of partial atomic geometric operators. Furthermore, Kobbelt interpolatory subdivision can also be included into the framework by introducing VV-type averaging operators. Considering the special relationship between $\sqrt{2}$ splitting operator and 4-8 splitting operator, the composite $\sqrt{2}$ subdivision can also be regarded as a composite 4-8 subdivision.

The work needs further improvement in the following aspects. Considering that we only described the framework for closed meshes, it is imperative for practical use to extend the framework such that it can deal with open meshes or meshes with sharp features in a unified style. Also, a unified adaptive strategy [26, 27] is also interesting for applications such as fast rendering, collision detection, surfaces intersections, and many other applications. In addition, the unified generalization of general box-spline surfaces remains open since only unified subdivisions corresponding to 2-direction (B-spline), 3-direction and 4-direction box spline surfaces have been addressed so far.

Acknowledgement. The work described in this paper is supported by National Science Foundation of China (60373034), City University of Hong Kong (SRG grant No. 7001798), the Research Grants Council of Hong Kong SAR (CERG grant No. CityU 1188/05E) and Natural Science Foundation of Guangdong Province (05006540).

References

[1] E. Cohen, T. Lyche, and R. Riesenfeld. Discrete b-splines and subdivision techniques in computer aided geometric design and computer graphics. *Computer Graphics and Image Processing*, 1980, 14 (2):87–111.

[2] W. Sweldens. The lifting scheme: a custom-design construction of biorthogonal wavelets. *Applied and Computational Harmonic Analysis*, 1996, 3: 186–200.

[3] M. Bertram. Biorthogonal Loop-Subdivision Wavelets. *Computing*, 2004, 72: 29-39.

[4] D. Zorin and P. Schröder. A unified framework for primal/dual quadrilateral subdivision scheme. *Computer Aided Geometric Design*, 2001, 18(5):429-454.

[5] J. Stam. On subdivision schemes generalizing uniform B-spline surfaces of arbitrary degree. *Computer Aided Geometric Design*, 18(5): 383-396.

[6] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356-360.

[7] E. Catmull and J. Clark Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 1978, 10(6):350-355.

[8] P. Oswald, and P. Schröder. Composite Primal/Dual $\sqrt{3}$ -Subdivision Schemes, *Computer Aided Geometric Design,* 2003, 20(2):135-164.
[9] L. Kobbelt. $\sqrt{3}$ -Subdivision. *SIGGRAPH 2000*, 103-112.
[10] P. Oswald. Designing composite triangular subdivision schemes. *Computer Aided Geometric Design*, 2005, 22(7): 659-679.
[11] J. Maillot, and J. Stam. A unified subdivision scheme for polygonal modeling. *Computer Graphics Forum(Eurographics 2001)*, 2001, 20(3): 471-479.
[12] Warren J. and Schaefer S. A factored approach to subdivision surfaces. *Computer Graphics & Applications*, 2004, 24(3): 74-81.
[13] J. Stam and C. Loop, quad/triangle subdivision, *Computer Graphics Forum*, 2003, 22(1): 79-85.
[14] L. Velho. Using semi-regular 4–8 meshes for subdivision surfaces. *Journal of Graphics Tool*, 2000, 5(3): 35-47.
[15] L. Velho. Stellar subdivision grammars. *Eurographics Symposium on Geometry Processing*, 2003: 188-199.
[16] J. Peters and U. Reif. The simplest subdivision scheme for smoothing polyhedra. *ACM Transactions on Graphics*, 1997, 16(4):420-431.
[17] A. Habib and J. Warren. Edge and vertex insertion for a class of C^1 subdivision surfaces. *Computer Aided Geometric Design*, 1999, 16(4): 223-247.
[18] L. Velho and D. Zorin. 4-8 Subdivision. *Computer Aided Geometric Design*, 2001, 18(5): 397-427.
[19] G. Li, W. Ma and H. Bao. $\sqrt{2}$ Subdivision for Quadrilateral meshes. *The Visual Computer*, 2004, 20(2-3): 180-198.
[20] G. Li, W. Ma and H. Bao. Interpolatory $\sqrt{2}$ -Subdivision surfaces. In Proceedings of Geometric Modeling and Processing 2004, 180-189.
[21] L. Kobbelt. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 1996)*, 15(3): 409-410.
[22] G. Li and W. Ma. A method for constructing interpolatory subdivision schemes and blending subdivisions. Submitted for publication, 2005.
[23] H. Prautzsch, W. Boehm and M. Paluszny. Bézier and B-Spline Techniques. Berlin, Springer, 2002.
[24] Warren J. and Weimer. Subdivision Methods for Geometric Design: A Constructive Approach. San Francisco, Morgan Kaufmann Publisher, 2002.
[25] C. de Boor, K. Hollig and S. Riemenschneiger. Box Splines. Springer, New York, 1993.
[26] A. Sovakar and L. Kobbelt. API design for adaptive subdivision schemes. *Computer & Graphics*, 2004, 28(1): 67-72.
[27] G. Li and W. Ma. Adaptive unified subdivisions with sharp features. Preprint, 2006.

Tuned Ternary Quad Subdivision

Tianyun Ni[1] and Ahmad H. Nasri[2]

[1] Dept. CISE, University of Florida
[2] Dept. of Computer Science American University of Beirut

Abstract. A well-documented problem of Catmull and Clark subdivision is that, in the neighborhood of extraordinary point, the curvature is unbounded and fluctuates. In fact, since one of the eigenvalues that determines elliptic shape is too small, the limit surface can have a saddle point when the designer's input mesh suggests a convex shape. Here, we replace, near the extraordinary point, Catmull-Clark subdivision by another set of rules derived by refining each bi-cubic B-spline into nine. This provides many localized degrees of freedom for special rules so that we need not reach out to, possibly irregular, neighbor vertices when trying to improve, or tune the behavior. We illustrate a strategy how to sensibly set such degrees of freedom and exhibit tuned ternary quad subdivision that yields surfaces with bounded curvature, nonnegative weights and full contribution of elliptic and hyperbolic shape components.

Keywords: Subdivision, ternary, bounded curvature, convex hull.

1 Introduction

Tuning a subdivision scheme means adjusting the subdivision rules or stencils to obtain a refined mesh and surface with prescribed properties. To date, there exists no subdivision algorithm on quadrilateral meshes that yields bounded curvature while guaranteeing the convex hull property at the extraordinary nodes. This paper proposes ternary refinement to obtain such a scheme. Ternary subdivision offers more parameters in a close vicinity of each extraordinary node to tune the subdivision than binary subdivision does and thereby localizes the tuning. Ternary quad subdivision generalizes the splitting of each quad into nine. If all nodes of a quad are of valence 4, the rules for bi-cubic B-splines shown in Figure 3 are applied and we have C^2 continuity. The challenge is with nodes of valence other than 4, called extraordinary nodes. In particular, we can devise rules at the extraordinary nodes and their newly created direct and diagonal neighbors to achieve

- eigenvalues in order of magnitude as $1, \lambda, \lambda, \lambda^2, \lambda^2, \lambda^2$, $\lambda_i, \ldots$ and $0 \leq |\lambda_i| < \frac{1}{9}$ for $i = 7, .., 2n + 1$; and
- nonnegative weights.

It is possible to get, in addition, $\lambda = 1/3$. Yet, Figure 2 shows that the macroscopic shape of the new scheme and Catmull-Clark are very similar (Figure 2), but, of course, the mesh is denser.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 441–450, 2006.
© Springer-Verlag Berlin Heidelberg 2006

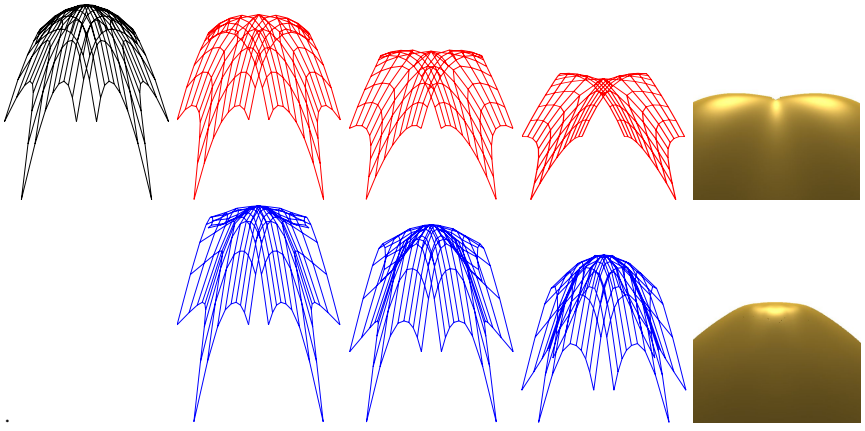


Fig. 1. (*left*) Control net: x, y the characteristic map of Catmull-Clark subdivision and $z = 50\,(1 - x^2 - 5y^2)$; subdivision steps 3,4,5 and surface for Catmull-Clark subdivision (*top*) and ternary subdivision (*bottom*)

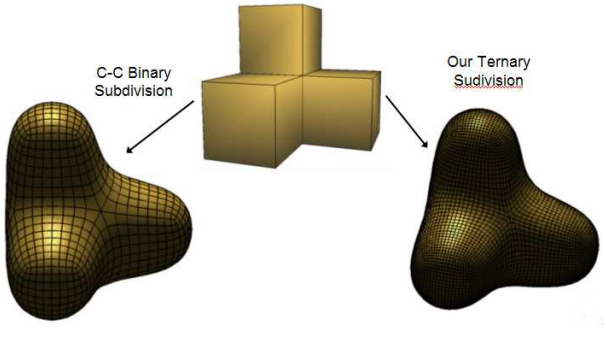


Fig. 2. Three steps of Catmull-Clark subdivision (*left*) and ternary subdivision (*right*)

1.1 Background

Subdivision is a widely adopted tool in computer graphics and is also making inroads into geometric modeling, if only for conceptual modeling. However, [5] pointed out that Catmull-Clark subdivision in particular, is lacking the full set of elliptic and hyperbolic subsubdominant eigenfunctions and therefore *typically* generates saddle shapes in the limit at vertices with valence greater than four (see Figure 1 *top*). This implies that any high-quality (standard, symmetric) scheme needs to have a spectrum $1, \lambda, \lambda, \lambda^2, \lambda^2, \lambda^2$ followed by smaller terms. With the strategy explained in the following, it is possible to achieve such a spectrum by making subdivision stencils near the extraordinary point depend on the valence (Figure 1 *bottom*). While such localized improvement cannot be expected to produce high-quality surfaces in all cases, it is worth seeing whether such improvements can lead to improved rules that are easy to substitute for the known unsatisfactory rules. A major challenge is technical: there are either too few or far too many

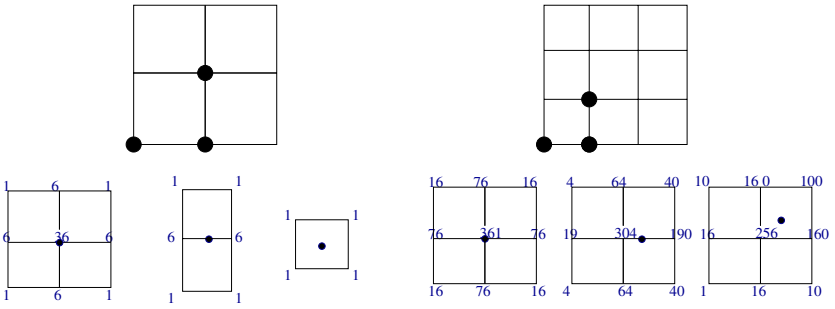


Fig. 3. The regular (bicubic B-spline refinement) stencils of Catmull-Clark (*left*) and ternary quad subdivision (*right*)

free parameters occuring in nonlinear inequality constraints. This makes any selection by general optimization [1] impractical. In [3,4], Loop proposed improvements to his well-known triangular scheme on triangular meshes to achieve both bounded curvature for a binary and ternary schemes. To derive weights, he used fixed-degree interpolation of known weights to set *many* unknown weights and reduce the number of free parameters. But quad mesh schemes are far more difficult than triangular mesh schemes since the leading eigenvalues come from a 2 by 2 diagonal block matrix rather than from a 1 by 1 block. They are therefore the roots of quadratic polynomials while, in the case of triangular meshes, the eigenvalue is simple. For n-sided facets, the problem is even more complex due to 4×4 blocks but can be avoided by applying one initial Catmull-Clark subdivision step.

2 Problem Statement

We want to derive a ternary, quad subdivision scheme that is stationary, (rotational and p-mirror) symmetric and affine invariant. Then the leading eigenvalues are, in order of magnitude, $1, \lambda, \lambda, \mu_1, \mu_2, \mu_3$. To achieve bounded curvature, the convex hull property, the eigenvalues and the weights α_i, β_i, γ_i, δ_i and v_i of the ternary quad scheme (see Figure 4) have to satisfy the following constraints:

(i)	C^1 scheme	$1 > \lambda > \mu_i$, and the characteristic map is regular and injective.
(ii)	Bounded Curvature	$\lvert\mu_1\rvert = \lvert\mu_2\rvert = \lvert\mu_3\rvert = \lambda^2$.
(iii)	Convex Hull	$\alpha_i \geq 0, \beta_i \geq 0, \gamma_i \geq 0, \delta_i \geq 0, i = 0..n-1, \quad v_1, v_2 \geq 0,$ $e_0 := 1 - \sum_{i=0}^{n-1}(\alpha_i + \beta_i) \geq 0, f_0 := 1 - \sum_{i=0}^{n-1}(\gamma_i + \delta_i) \geq 0,$ $v_0 := 1 - \sum_{i=1}^{2}(v_i) \geq 0.$
(iv)	Symmetry	$\alpha_l = \alpha_{n-l}, \beta_l = \beta_{n-1-l}, \gamma_l = \gamma_{(n+1-l) \mod n}$, and $\delta_l = \delta_{n-l}$

We focus on the leading eigenvalues and hence the 1-ring of control mesh around extraordinary point and its eigenstructure. As shown in Figure 4, each refinement step generates three types of points corresponding to vertices, edges and faces respectively. Let n be the valence of a generic extraordinary point. We have, offhand, $4n+2$ weights to determine. Symmetry reduces this to $2n + 4$ free parameters.

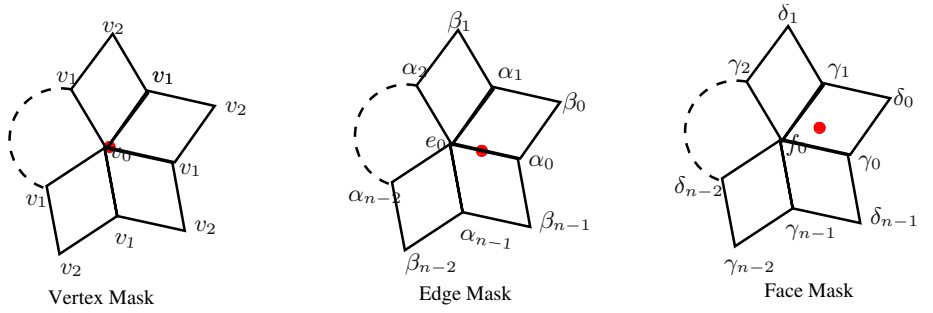


Fig. 4. Refinement stencils used at an extraordinary point

3 Spectral Analysis

The 1-ring subdivision matrix S with the control vertices shown in Figure 4 is:

$$S := \left(\begin{array}{c|cccccccc} v_0 & v_1 & v_2 & v_1 & v_2 & \cdots & v_1 & v_2 \\ \hline e_0 & \alpha_0 & \beta_0 & \alpha_1 & \beta_1 & \cdots & \alpha_{n-1} & \beta_{n-1} \\ f_0 & \gamma_0 & \delta_0 & \gamma_1 & \delta_1 & \cdots & \gamma_{n-1} & \delta_{n-1} \\ e_0 & \alpha_{n-1} & \beta_{n-1} & \alpha_0 & \beta_0 & \cdots & \alpha_{n-2} & \beta_{n-2} \\ f_0 & \gamma_{n-1} & \delta_{n-1} & \gamma_0 & \delta_0 & \cdots & \gamma_{n-2} & \delta_{n-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ e_0 & \alpha_1 & \beta_1 & \alpha_2 & \beta_2 & \cdots & \alpha_0 & \beta_0 \\ f_0 & \gamma_1 & \delta_1 & \gamma_2 & \delta_2 & \cdots & \gamma_0 & \delta_0 \end{array}\right) \in \mathbb{R}^{(2n+1)\times(2n+1)}.$$

Now denote the discrete Fourier transform of a cyclic sequence $\{\phi_i\}$ by

$$\hat{\phi}_i := \sum_{j=0}^{n-1} \phi_j \omega^{i,j}, \qquad w^{i,j} := e^{\sqrt{-1}\frac{2\pi}{n}ij}.$$

Lemma 1. *The spectrum of S is* $\Lambda = diag(1, \lambda_0^+, \lambda_0^-, \lambda_i^+, \lambda_i^-, \cdots, \lambda_{n-1}^+, \lambda_{n-1}^-)$ *where*

$$\lambda_i^{\pm} := \frac{(\hat{\alpha}_i + \hat{\delta}_i) \pm \sqrt{(\hat{\alpha}_i - \hat{\delta}_i)^2 + 4\hat{\beta}_i\hat{\gamma}_i}}{2}, \quad i = 1 \ldots n-1. \tag{1}$$

Proof. Since the 2n × 2n lower block matrix of S is cyclic, we can apply the discrete Fourier transform

$$F := \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & I_2 \bigotimes(\omega^{i,j}) \end{array}\right)$$

to obtain

$$Q := F^{-1}SF = \left(\begin{array}{c|cccccc} v_0 & nv_1 & nv_2 & 0 & 0 & \cdots & 0 & 0 \\ \hline e_0 & \hat{\alpha}_0 & \hat{\beta}_0 & 0 & 0 & \cdots & 0 & 0 \\ f_0 & \hat{\gamma}_0 & \hat{\delta}_0 & 0 & 0 & \cdots & 0 & 0 \\ e_0 & 0 & 0 & \hat{\alpha}_1 & \hat{\beta}_1 & \cdots & 0 & 0 \\ f_0 & 0 & 0 & \hat{\gamma}_1 & \hat{\delta}_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ e_0 & 0 & 0 & 0 & 0 & \cdots & \hat{\alpha}_{n-1} & \hat{\beta}_{n-1} \\ f_0 & 0 & 0 & 0 & 0 & \cdots & \hat{\gamma}_{n-1} & \hat{\delta}_{n-1} \end{array}\right).$$

The eigenvalues $1, \lambda_0^+$, and λ_0^- come from the first 3×3 block. Each 2×2 diagonal block

$$\begin{pmatrix} \hat{\alpha}_i & \hat{\beta}_i \\ \hat{\gamma}_i & \hat{\delta}_i \end{pmatrix}, \qquad i = 1 \ldots n-1,$$

contributes the pair of eigenvalues (1). □

By (iv) and since λ_i^+ and λ_{n-i}^+ are complex conjugate, we anticipate the following lemma.

Lemma 2. $\lambda_i^+ = \lambda_{n-i}^+, \quad \lambda_i^- = \lambda_{n-i}^-, \quad$ *and* $\quad \lambda_i^+, \lambda_i^- \in \mathbb{R}$.

Proof. By symmetry (iv),

$$\hat{\alpha}_i = \alpha_0 + \begin{cases} 2\sum_{j=1}^{\frac{n-1}{2}} \alpha_j \cos\frac{2\pi ij}{n} & \text{if n is odd,} \\ 2\sum_{j=1}^{\frac{n}{2}-1} \alpha_j \cos\frac{2\pi ij}{n} + \alpha_{\frac{n}{2}} \cos\pi i & \text{if n is even.} \end{cases} \tag{2}$$

Since $\cos\frac{2\pi ij}{n} = \cos\frac{2\pi(n-i)j}{n}$, $\hat{\alpha}_i = \hat{\alpha}_{n-i} \in \mathbb{R}$. The same reasoning applies to $\hat{\delta}_i$, $\hat{\beta}_i e^{\frac{\pi}{n}i}$ and $\hat{\gamma}_i e^{-\frac{\pi}{n}i}$ since, e.g.

$$\hat{\beta}_i e^{\frac{\pi}{n}i} = \begin{cases} 2\sum_{j=0}^{\frac{n-1}{2}-1} \beta_j \cos\frac{(2\pi j+\pi)i}{n} + \beta_{\frac{n-1}{2}} \cos\pi i & \text{if n is odd,} \\ 2\sum_{j=0}^{\frac{n}{2}-1} \beta_j \cos\frac{(2\pi j+\pi)i}{n} & \text{if n is even.} \end{cases} \tag{3}$$

Therefore, $\lambda_i^{\pm} = \lambda_{n-i}^{\pm +} \in \mathbb{R}$ as claimed. □

For $n = 4$ the subdivision stencils are shown in Figure 3. Based on this subdivision matrix, we can compute the matrix Q in Section 3 and therefore compute explicitly the expansions collected in Lemma 3.

Lemma 3. *In the regular setting ($n = 4$),*

$$\begin{aligned} [\hat{\alpha}_0^4, \hat{\alpha}_1^4, \hat{\alpha}_2^4, \hat{\alpha}_3^4] &= [\frac{337}{729}, \frac{19}{81}, \frac{1}{9}, \frac{19}{81}], \\ [\hat{\beta}_0'^4, \hat{\beta}_1'^4, \hat{\beta}_2'^4, \hat{\beta}_3'^4] &= [\frac{88}{729}, \frac{4\sqrt{2}}{81}, 0, -\frac{4\sqrt{2}}{81}], \\ [\hat{\gamma}_0'^4, \hat{\gamma}_1'^4, \hat{\gamma}_2'^4, \hat{\gamma}_3'^4] &= [\frac{352}{729}, \frac{16\sqrt{2}}{81}, 0, -\frac{16\sqrt{2}}{81}], \\ [\hat{\delta}_0^4, \hat{\delta}_1^4, \hat{\delta}_2^4, \hat{\delta}_3^4] &= [\frac{121}{729}, \frac{11}{81}, \frac{1}{9}, \frac{11}{81}]. \end{aligned} \tag{4}$$

4 Deriving Weights by Interpolating the Regular Case

The idea of deriving extraordinary rules is inspired by Loop's approach [4]. First, we interpolate the regular stencil by a polynomial. Such a polynomial, say β^n, will be evaluated to define the coefficients β_i^n. For $n = 4$, due to symmetry, the constraints on the polynomial β^n are $\beta^n(\cos\frac{\pi}{n}) = \frac{40}{729}$ and $\beta^n(\cos\frac{3\pi}{n}) = \frac{4}{729}$. The linear interpolant to these values is negative on some subinterval of $[-1, 1]$ and therefore cannot sei to give suitable coefficients $\beta_i^n \geq 0$ for $n > 4$. Adding the constraint $\beta^n(-1) = 0$ yields the polynomial

$$\beta^4(t) := \frac{1}{729}\Big((44 - 18\sqrt{2})(1-t)^2 + (44 - 9\sqrt{2})2t(1-t) + (36\sqrt{2})t^2\Big)$$

which is nonnegative on $[-1, 1]$. We conjecture the general formulae for arbitrary n based on the polynomials of the same low degree and choose

$$\begin{aligned}\alpha^n(t) &:= a_{n,1}\Big((t + b_{n,1})^2 + c_{n,1}\Big), \quad \beta^n := \frac{4}{n}\beta^4, \quad \gamma^n := \frac{4}{n}\beta^4,\\ \delta^n(t) &:= a_{n,4}\Big((t + b_{n,4})^2 + c_{n,4}\Big).\end{aligned} \tag{5}$$

Now, we predict the general formulae for any $n > 4$ by evaluating the polynomials at the x-component of points equally distributed over the unit circle. This yields two of the three types of stencils at the extraordinary point and leaves, as degrees of freedom, the coefficients depending on n in (5).

Lemma 4. *The subdivision rules, for valence n and $j = 0 \ldots n-1$ can be chosen as*

$$\begin{aligned}&\alpha_j^n := \alpha^n(u), \quad \delta_j^n := \delta^n(u), \quad u := \cos\frac{2\pi j}{n}, \\ &\beta_j^n := \beta^n(v_+), \quad v_+ := \cos\frac{2\pi j + \pi}{n}, \qquad \gamma_j^n := \gamma^n(v_-), \quad v_- = \cos\frac{2\pi j - \pi}{n}\end{aligned} \tag{6}$$

where

$$a_{n,1} := \frac{4}{n}\hat{\alpha}_2, \quad b_{n,1} := \frac{\hat{\alpha}_1}{n a_1}, \quad a_{n,4} := \frac{4}{n}\hat{\delta}_2, \quad b_{n,4} := \frac{\hat{\delta}_1}{n a_4}. \tag{7}$$

Note that, together with β and γ, equations (7) yield explicit expressions for $\lambda_1^{\pm}$ and $\lambda_2^{\pm}$ in terms of the the coefficients a_1, b_1, c_1, a_4, b_4, c_4.

Proof. By symmetry, the weights are a discrete cosine transformation of their Fourier coefficients:

$$\begin{aligned}\alpha_i^n &= \frac{1}{n}\sum_{j=0}^{n-1}\hat{\alpha}_j^n \cos\frac{2\pi ij}{n}, \qquad \beta_i^n = \frac{1}{n}\sum_{j=0}^{n-1}\hat{\beta}_j'^n \cos\frac{2\pi ij + \pi j}{n}, \\ \gamma_i^n &= \frac{1}{n}\sum_{j=0}^{n-1}\hat{\gamma}_j'^n \cos\frac{2\pi ij - \pi j}{n}, \qquad \delta_i^n = \frac{1}{n}\sum_{j=0}^{n-1}\hat{\delta}_j^n \cos\frac{2\pi ij}{n}.\end{aligned} \tag{8}$$

For $n = 4$, we verify the consistency of the choice (6) by substituting (4) into (8).

$$\alpha_i^4 = \frac{1}{18}\big((u + \frac{19}{18})^2 + \frac{151}{324}\big), \qquad \delta_i^4 = \frac{1}{18}\big((u + \frac{11}{18})^2 - \frac{41}{324}\big).$$

Next, we decompose the polynomial into a vector of basis functions and a vector of coefficients:

$$a_{n,1}\Big((u+b_{n,1})^2+c_{n,1}\Big) = [1, u, 2u^2-1][a_{n,1}b_{n,1}^2+\frac{1}{2}a_{n,1}+a_{n,1}c_{n,1}, 2a_{n,1}b_{n,1}, \frac{1}{2}a_{n,1}]^T.$$

We mimic this decompostion for $n > 4$, by choosing $\hat{\alpha}_i^n = 0$ for $2 < i < n-2$. Then

$$\alpha_i^n = \frac{1}{n}\sum_{j=0}^{n-1} \hat{\alpha}_j^n \cos\frac{2\pi ij}{n} = [1, \cos\frac{2\pi i}{n}, \cos\frac{4\pi i}{n}][\frac{1}{n}\hat{\alpha}_0^n, \frac{2}{n}\hat{\alpha}_1^n, \frac{2}{n}\hat{\alpha}_2^n]^T.$$

Since $\cos\frac{2\pi i}{n} = u$, we have $\cos\frac{4\pi i}{n} = 2u^2 - 1$ and comparison of terms yields

$$2a_{n,1}b_{n,1} = \frac{2}{n}\hat{\alpha}_{n,1}, \quad \text{and} \quad \frac{1}{2}a_{n,1} = \frac{2}{n}\hat{\alpha}_2.$$

Solving $a_{n,1}, b_{n,1}$ in terms of $\hat{\alpha}_1^n, \hat{\alpha}_2^n, n$, and similarly for δ_i^n, yields (7). □

The rules of Lemma 4 preserve rotational and p-mirror symmetry. For $n > 4$, they reduce the free parameters from $2n + 4$ to six, namely $a_{n,i}, b_{n,i}, c_{n,i}, i = 1, 4$. (For $n = 3$ we use the original $2n + 4 = 10$ free parameters to enforce $\lambda_1^+ = \lambda, \quad \lambda_1^- = \lambda^2, \quad \lambda_0^+ = \lambda^2, \quad \lambda_0^- < \lambda^2$.) It remains need to determine the six free parameters so that for $n > 4$ the weigths $\alpha_j, \delta_j, v_1, v_2, e_0, f_0, v_0$ are nonnegative (β_j and γ_j are nonnegative since $\beta^4 \geq 0$ on $[-1, 1]$) and

$$\begin{aligned} \Lambda &= \text{diag}(1, \lambda_1^+, \lambda_{n-1}^+, \lambda_2^+, \lambda_{n-2}^+, \lambda_0^+, \nu_1, \nu_2, \ldots, \nu_{2n-5}) \\ &= \text{diag}(1, \lambda, \lambda, \lambda^2, \lambda^2, \lambda^2, \ldots, \lambda_i, \ldots), \quad \text{where} \quad 0 < |\lambda_i| < \lambda^2. \end{aligned} \tag{9}$$

By (1), and since $\hat{\gamma}_k$ and $\hat{\beta}_k$ are fixed, each of the constraints $\lambda_1^+ = \lambda$ and $\lambda_2^+ = \lambda^2$ is an equation in the parameters, $a_{n,1}$, $b_{n,1}$, $a_{n,4}$, $b_{n,4}$, explicitly so due to (7). The additional necessary constraint, $\lambda_0^+ = \lambda^2, \lambda_0^- < \lambda^2$ is enforced by choice of v_1, v_2 in the first 3 by 3 block of Q. The result is an underconstrained system with linear inequality constraints. We find, for example, the closed-form solution for $n = 5 \ldots 10$ stated in Lemma 5 and yielding the elliptic and saddle shapes shown in Figure 7.

Lemma 5. *For $n = 5..10$, the following choice satisfies (i)-(iv) and $\lambda = 1/3$:*

$$\begin{aligned} a_{n,1} &= .413, \quad b_{n,1} = .624, \quad c_{n,1} = .349 \\ a_2 &= \frac{4}{n}, \quad a_3 = \frac{4}{n} \\ a_{n,4} &= .026, \quad b_{n,4} = .286, \quad c_{n,4} = -.185 \end{aligned} \tag{10}$$

Table 1. Eigenvalues (see 9) other than $1, \frac{1}{3}, \frac{1}{3}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}$ for $n = 3..10$. $0 < |\epsilon| < .001$.

ν_i	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
ν_1	-.063	.037	.057	.057	.057	.057	.057	.057
ν_2	$\cdots$	.037	.057	.057	.057	.057	.057	.057
ν_3	$\cdots$	.012	-.023	-.012	-.002	.009	.019	.030
ν_4	$\cdots$	$\cdots$	-.001	-.001	-.001	-.001	-.001	-.001
ν_5	$\cdots$	$\cdots$	-.001	-.001	-.001	-.001	-.001	-.001
ν_6	$\cdots$	$\cdots$	$\cdots$	ϵ	ϵ	ϵ	ϵ	ϵ
ν_7	$\cdots$	$\cdots$	$\cdots$	ϵ	ϵ	ϵ	ϵ	ϵ
ν_8	$\cdots$	$\cdots$	$\cdots$	$\cdots$	ϵ	ϵ	ϵ	ϵ
ν_9	$\cdots$	$\cdots$	$\cdots$	$\cdots$	ϵ	ϵ	ϵ	ϵ
ν_{10}	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	ϵ	ϵ	ϵ
ν_{11}	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	ϵ	ϵ	ϵ
ν_{12}	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	ϵ	ϵ
ν_{13}	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	ϵ	ϵ
ν_{14}	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	ϵ
ν_{15}	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	ϵ

Proof. We need only verify (i), (ii) and (iii) listed in Section 2 since the scheme is symmetric by construction. Table 1 shows all eigenvalues other than $1, \frac{1}{3}, \frac{1}{3}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}$ for $n = 3 \ldots 10$. In particular, $\lambda_1 = \lambda_{n-1} = \frac{1}{3}$ is the subdominant eigenvalue. Table 2 verifies nonnegative weights summing to 1. The u-differences, scaled to unit size, fall strictly into the lower right quadrant and, by symmetry, the v-differences fall strictly into the upper right quadrant (Figure 6). The partial derivatives are a convex combination of the differences and hence all pairwise crossproducts are strictly positive. By [8], the characteristic map is regular. In addition, as shown in Figure 5, the first half-segment of the control net does not intersect the negative x-axis. By the convex hull property and [7] Theorem 21, injectivity of the characteristic map follows. □

The main contribution of the paper is technical: to provide a manageable set of parameters that make it easy to satisfy the formal constraints (i)–(iv). This set can be further

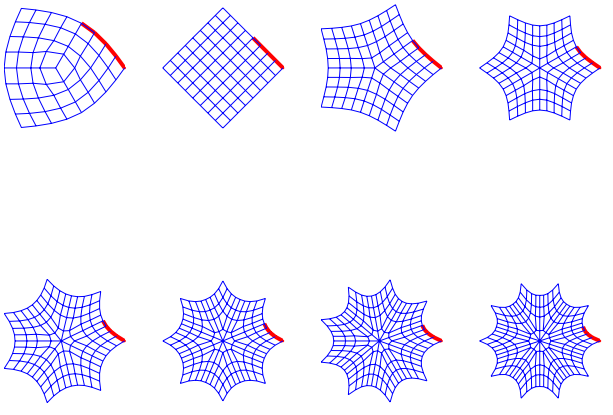


Fig. 5. Control polyhedron of the characteristic map for $n = 3..10$. Injectivity test: each red fat segment does not intersect the negative x-axis.

Table 2. The subdivision rule according to Lemma 5 has non-negative weights. $0 < \epsilon < .001$.

weights	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
v_0	.524	.495	.460	.470	.481	.491	.502	.512
v_1	.119	.104	.098	.078	.064	.053	.044	.038
v_2	.040	.022	.010	.010	.010	.010	.010	.010
e_0	.550	.417	.417	.417	.417	.417	.417	.417
α_0	.291	.261	.237	.198	.169	.148	.132	.119
β_0	.062	.055	.047	.042	.037	.033	.030	.027
α_1	.009	.088	.091	.103	.105	.103	.099	.094
β_1	.018	.005	.012	.017	.019	.020	.020	.020
α_2	$\cdots$	.026	.022	.017	.023	.032	.040	.045
β_2	$\cdots$	$\cdots$	.000	.002	.004	.007	.009	.010
α_3	$\cdots$	$\cdots$	$\cdots$	.025	.018	.012	.011	.014
β_3	$\cdots$	$\cdots$	$\cdots$	$\cdots$	.000	.001	.002	.003
α_4	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	.019	.015	.011
β_4	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	.000	.000
α_5	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	.015
f_0	.309	.351	.351	.351	.351	.351	.351	.351
δ_0	.180	.137	.089	.074	.063	.056	.049	.044
$\gamma_1 = \gamma_0$	.226	.219	.191	.167	.148	.132	.118	.108
δ_1	.017	.014	.021	.029	.033	.034	.034	.033
γ_2	.023	.022	.050	.068	.077	.080	.080	.079
δ_2	$\cdots$	.001	.017	.004	.002	.005	.008	.011
γ_3	$\cdots$	$\cdots$	.000	.006	.017	.027	.035	.041
δ_3	$\cdots$	$\cdots$	$\cdots$	.024	.000	.008	.003	.002
γ_4	$\cdots$	$\cdots$	$\cdots$	$\cdots$	.078	.003	.007	.013
δ_4	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	.018	.014	.009
γ_5	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	.000	.001
δ_5	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	.015

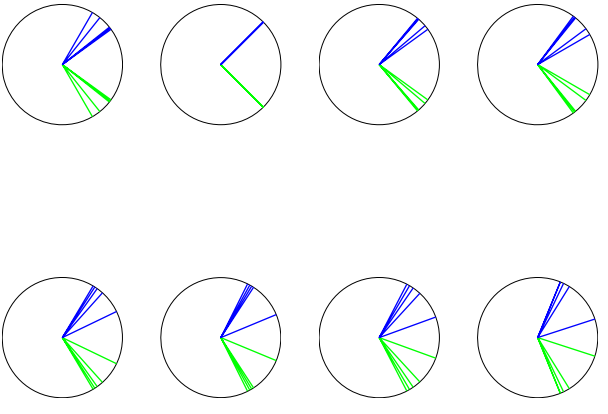


Fig. 6. The normalized differences in the u direction (*green*) and v direction (*blue*)

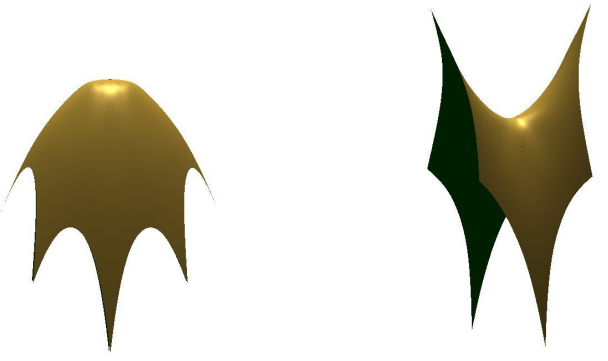


Fig. 7. Surfaces generated by the rules of Lemma 5

pruned by minimizing regions of meshes where the scheme exhibits hybrid behavior [5], i.e. the Gauss curvature is not uniquely of one sign in the limit. It should be noted that this is typically not enough to guarantee surface fairness, for example to avoid undue flatness when convexity is indicated by the control mesh.

Acknowledgment. This work was supported by NSF Grant CCF-0430891 and American University of Beirut URB-2006. Jörg Peters gave valuable technical guidance and assistance.

References

1. L. Barthe, and L. Kobbelt, Subdivision Scheme tuning around extraordinary vertices, *Computer Aided Geometric Design, 21: 561-583.*
2. E. Catmull, and J. Clark, Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes. *Computer Aided Design, 10(6): 350-355.*(1978)
3. C. Loop, Smooth Ternary Subdivision of Triangle Meshes *Curve and Surface Fitting* (Saint-Malo, 2002).
4. C. Loop, Bounded curvature triangle Mesh subdivision with the convex hull property. *The Visual Computer, 18 (5-6), 316-325.*
5. K. Karciauskas, J. Peters and U. Reif, Shape Characterization of Subdivision Surfaces – Case Studies, *Comp. Aided Geom. Design*, 21(6), 2004, 601–614.
6. A. Nasri, I. Hasbini, J. Zheng, T. Sederberg, Quad-based Ternary Subdivision, presentation Dagstuhl Seminar on Geometric Modeling, 29 May - 03 June, 2005
7. U. Reif and J. Peters. Structural Analysis of Subdivision Surfaces – A Summary *Topics in Multivariate Approximation and Interpolation*, Elsevier Science Ltd, K. Jetter et al., 2005, 149–190.
8. G. Umlauf, Analysis and Tuning of Subdivision Algorithms. *Proceedings of the 21st spring conference on Computer Graphics, 33-40.* (2005)

Simultaneous Precise Solutions to the Visibility Problem of Sculptured Models

Joon-Kyung Seong[1], Gershon Elber[2], and Elaine Cohen[1]

[1] University of Utah, Salt Lake City, UT84112, USA
seong@cs.utah.edu, cohen@cs.utah.edu
[2] Technion, Haifa 32000, Israel
gershon@cs.technion.ac.il

Abstract. We present an efficient and robust algorithm for computing continuous visibility for two- or three-dimensional shapes whose boundaries are NURBS curves or surfaces by lifting the problem into a higher dimensional parameter space. This higher dimensional formulation enables solving for the visible regions over all view directions in the domain simultaneously, therefore providing a reliable and fast computation of the *visibility chart*, a structure which simultaneously encodes the visible part of the shape's boundary from every view in the domain. In this framework, visible parts of planar curves are computed by solving two polynomial equations in three variables (t and r for curve parameters and θ for a view direction). Since one of the two equations is an inequality constraint, this formulation yields two-manifold surfaces as a zero-set in a 3-D parameter space. Considering a projection of the two-manifolds onto the $t\theta$-plane, a curve's location is invisible if its corresponding parameter belongs to the projected region. The problem of computing hidden curve removal is then reduced to that of computing the projected region of the zero-set in the $t\theta$-domain. We recast the problem of computing boundary curves of the projected regions into that of solving three polynomial constraints in three variables, one of which is an inequality constraint. A topological structure of the visibility chart is analyzed in the same framework, which provides a reliable solution to the hidden curve removal problem. Our approach has also been extended to the surface case where we have two degrees of freedom for a view direction and two for the model parameter. The effectiveness of our approach is demonstrated with several experimental results.

1 Introduction

A major part of rendering is related to the hidden surface removal problem, i.e., display only those surfaces which should be visible. The main contribution of this work can be summarized as follows:

- The exact boundary between visible and hidden parts of planar curves or surfaces is computed by solving a set of polynomial equations in the parameter space without any piecewise linear approximations.
- All possible view directions in the domain are considered, simultaneously, by lifting the problem into a higher dimensional space and solving a continuous visibility problem. This higher dimensional framework provides a reliable solution to the computation of the visibility chart.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 451–464, 2006.
© Springer-Verlag Berlin Heidelberg 2006

– The algorithm is easy to implement and robust by mapping the problem in hand to a zero-set solving that exploits the convex hull and subdivision properties of NURBS. Topological analysis of the visibility chart makes it easier to compute the global structure of the visibility chart.

Research into solving the hidden surface removal problem is one of the earliest areas of activity in computer graphics, computer-aided design and manufacturing, and many different algorithms have been developed [24,1,9,18,14,19]. Usually they are developed for polygonal data, so curved surfaces have traditionally been preprocessed and approximated as large collections of polygons [22,17]. In this paper, we present an algorithm for eliminating hidden curves or surfaces directly from freeform models without any polygonal approximations. Visibility computations of sculptured models have various applications not only in the area of rendering but also in such areas as mold design, robot accessibility, inspection planning and security.

Given a view direction, the hidden surface removal problem refers to determining which surfaces are occluded from that view direction. Most of the earlier algorithms in the literature are for polygonal data and hidden line removal [8,20,24]. In their work, because the displayed edges of the polygons are linear edges, the displayed curves, such as the silhouettes of an object viewed from a view direction, are not smooth. Curves can be displayed more smoothly by increasing the number of polygons used for the approximation, but this results in memory and computational expense.

Algorithms to resolve the hidden surface removal problem can be classified into those that perform calculations in object-space, those that perform calculations in image-space, and those that work partly in both, list-priority [24]. Object space techniques use geometric tests on the object descriptions to determine which objects overlap and where. Initiated by Appel's edge-intersection algorithm [1], the idea of quantitative invisibility which determines visible and invisible regions in advance was developed [9,18,11]. Image space approaches compute visibility only to the precision required to decide what is visible at a particular pixel, exemplified by [2]. Catmull develops the depth-buffer or z-buffer image-precision algorithm which uses depth information [4]. Also, Weiler and Atherton [25] and Whitted [26] develop ray tracing algorithms which transform the hidden surface removal problem into ray-surface intersection tests.

Given a model composed of algebraic or parametric surfaces, it can be polygonized and hidden lines can be removed from the polygonized surfaces [22,17]. However, the accuracy of the overall algorithm is limited by the accuracy of the polygonal approximation. Further, in both methods [22,17], visibility is determined for the endpoints of straight lines and hence, they fail to detect invisibility occurring in the interior region of a line when both endpoints are visible. To remove hidden lines from curved surfaces without polygonal approximation, Hornung et al. [11] extended the idea of quantitative invisibility to bi-quadratic patches, and Newton's method was employed to solve for intersections between curves. Elber and Cohen [7] applied Hornung's technique to nonuniform rational B-splines and extended it to treat trimmed surfaces. In particular, Elber and Cohen [7] extract the curves of interest by considering boundary curves, silhouette curves, iso-parametric curves and curves along C^1 discontinuity based on 2D curve-curve intersections. Nishita et al. [21] used their Bezier Clipping technique for the hidden curve elimination. These methods [11,7,21] are aimed at eliminating

the hidden curves from line drawings of surfaces (not shaded drawings). Krishnan and Manocha [16] presented an algorithm for the elimination of hidden surfaces using a combination of symbolic techniques and results from numerical linear algebra.

Elber et al. [6] presented an algorithm for computing two-dimensional visibility charts for planar curves. The visibility charts, however, are constructed by discretizing a continuous set of view directions [6]. Our algorithm is an extension of that work into the computation of continuous visibility charts. Krishnan and Manocha [16] solves the hidden surface removal problem for a discrete set of view directions only.

Our approach is unique in that of solving the visibility problem for all view directions in the domain, simultaneously.

Summary of Our Approach

We reduce the solution to the visibility problem to the problem of finding the zeros of a set of polynomial equations in the parameter space. For the curve case, visible curve locations are computed by solving 2 polynomial equations in 3 variables (t and r for curve parameters and θ for a view direction). Since one of the two equations is an inequality constraint, this framework yields 2-manifold surfaces as a 0-set in a 3-D parameter space. A curve's location is invisible if its corresponding parameter belongs to the projected region of the two-manifolds onto the $t\theta$-plane. The problem for computing hidden curve removal is then reduced to that of computing the projected region of the zero-set in the $t\theta$-domain. We recast this problem of computing boundary curves of the projected regions into that of solving three polynomial constraints in three variables, one of which is an inequality constraint.

The presented approach for the hidden curve removal can be extended to the surface case where we have 2 degrees of freedom (dof) for a view direction and two for surface parameters. Similarly to the curve case, visible surface's locations are computed by solving 3 polynomial equations in 6 variables, one of which is an inequality constraint. Assuming a freeform surface $S(u, v)$ is used to parameterize for all possible view directions $\mathcal{V}(\theta, \varphi)$, the 0-set of the 3 equations is constructed as four-manifolds in a 6-dimensional parameter space, and its projection into the $uv\theta\varphi$-domain prescribes the hidden parts of the surface $S(u, v)$. A surface's location, $S(u_0, v_0)$, is invisible from viewing direction $\mathcal{V}(\theta_0, \varphi_0)$ if its corresponding parameter, $(u_0, v_0, \theta_0, \varphi_0)$, belongs to the projected region of the 0-set. The boundary of the projected region is computed by introducing one more equation to the set of 3 equations, therefore generating 3-manifolds in the 4-dimensional parameter space. The visibility charts for the surface case are then constructed using the 3-manifolds in the $uv\theta\varphi$-parameter domain. A particular visibility query, which specifies θ and φ for a view direction, is resolved by extracting one-manifold curves in the surface's uv-parameter domain. Those curves in the uv-domain trim away hidden surface regions and thus only the visible surfaces are rendered from that view direction.

The topological structure of the visibility chart is further analyzed in the same framework, which provides a reliable solution to the computation of the visibility chart. The number of connected curve segments that delineate the hidden parts from the visible ones changes at critical points where the global topology changes in the visibility chart. *Aspect graphs* [3] are used in computer vision to topologically analize the visibility problem. In this paper, algebraic constraints for these critical points are derived as a set

of 3 polynomial equations in 3 variables for the curve case and precomputed for the global analysis of the visibility chart. Based on this topological information, it becomes easier to analyze the global arrangement of the visibility chart, avoiding the computation of complex combinatorial curve-curve intersections.

The rest of this paper is organized as follows. In Section 2, the hidden curves removal algorithm is discussed for planar curves. Section 3 presents its extension to the elimination of hidden surfaces. Some examples are presented in Section 4 and finally, in Section 5, this paper is concluded.

2 Continuous Visibility for Planar Curves

Let $\mathcal{V}(\theta)$ be a one-parameter family of viewing directions. The visibility for a planar curve $\mathbf{C}(t)$ is then solved by lifting the problem into a higher dimension, where the answer is represented using simultaneous solution of two polynomial equations.

Lemma 1. *A planar curve point $\mathbf{C}(t)$ is visible if and only if it satisfies the following two polynomial equations for all r,*

$$\mathcal{F}(t, r, \theta) = \mathcal{V}(\theta) \times (\mathbf{C}(t) - \mathbf{C}(r)) = 0,$$
$$\mathcal{G}_1(t, r, \theta) = \langle \mathcal{V}(\theta), \mathbf{C}(t) - \mathbf{C}(r) \rangle \leq 0.$$

Proof. Two equations, $\mathcal{F}(t, r, \theta) = 0$ and $\mathcal{G}_1(t, r, \theta) \leq 0$, are satisfied only if $\mathbf{C}(t)$ is closer to the view source than $\mathbf{C}(r)$ while two curve points are on the same line to the view direction $\mathcal{V}(\theta)$. Therefore, there may be no other curve point $\mathbf{C}(r)$ that blocks $\mathbf{C}(t)$ from $\mathcal{V}(\theta)$ if $\mathbf{C}(t)$ satisfies the above two equations for all r, which implies that $\mathbf{C}(t)$ is visible from the viewing direction. ■

Figure 1 demonstrates Lemma 1. Given a viewing direction $\mathcal{V}$, two curve points $\mathbf{C}(t)$ and $\mathbf{C}(r)$ in Figure 1(a) satisfy the first equation $\mathcal{F}(t, r, \theta) = 0$. This means that the vector from $\mathbf{C}(t)$ to $\mathbf{C}(r)$ is parallel to the view direction. The second condition is satisfied only if $\mathbf{C}(t)$ is closer to the view source than $\mathbf{C}(r)$. Thus, the curve point $\mathbf{C}(t)$ is visible for the view direction $\mathcal{V}$, while $\mathbf{C}(r)$ is not. For the curve point $\mathbf{C}(t)$ to be visible, $\mathcal{G}_1(t, r, \theta) \leq 0$ should be satisfied for all r. This implies that if there is any value of r such that $\mathcal{G}_1(t, r, \theta) > 0$, then the curve point $\mathbf{C}(t)$ is not visible. In Figure 1(b), $\mathbf{C}(t)$ is potentially visible from $\mathcal{V}$ if one considers the curve point $\mathbf{C}(s)$ as its corresponding pair. The point $\mathbf{C}(t)$, however, is not visible since there exists another curve point $\mathbf{C}(r)$ that fails at the second constraint of Lemma 1.

Elber et al. [6] solves two polynomial equations in two variables for a discrete set of view directions. If $\mathcal{V}$ is one such direction,

$$\mathbf{C}'(t) \times \mathcal{V} = 0,$$
$$(\mathbf{C}(t) - \mathbf{C}(r)) \times \mathcal{V} = 0.$$

Solution points of these two equations prescribe the visible portion of $\mathbf{C}$ for each $\mathcal{V}$, providing only a discrete solution. In this paper, we solve the problem of computing visible regions for all possible view directions $\mathcal{V}(\theta)$ in the domain, simultaneously, providing a continuous solution to the visibility problem.

For the clarity of explanation, we consider *invisible* curve segments instead.

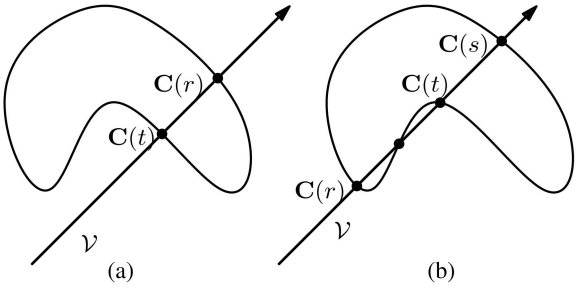


Fig. 1. (a) Given a viewing direction $\mathcal{V}$, a planar curve point $\mathbf{C}(t)$ is visible while $\mathbf{C}(r)$ is not. (b) A point $\mathbf{C}(t)$ has another curve point $\mathbf{C}(r)$ which makes it invisible from the view direction $\mathcal{V}$.

Corollary 1. *A planar curve point* $\mathbf{C}(t)$ *is invisible if and only if there exists another curve point* $\mathbf{C}(r)$ *such that the following two polynomial equations hold*

$$\mathcal{F}(t, r, \theta) = \mathcal{V}(\theta) \times (\mathbf{C}(t) - \mathbf{C}(r)) = 0, \tag{1}$$

$$\mathcal{G}_2(t, r, \theta) = \langle \mathcal{V}(\theta), \mathbf{C}(t) - \mathbf{C}(r) \rangle > 0. \tag{2}$$

Now, any r for which $\mathcal{G}_2(t, r, \theta) > 0$ holds renders curve point $\mathbf{C}(t)$ invisible. As this second equation, $\mathcal{G}_2(t, r, \theta) > 0$, is an inequality constraint, the solution of both constraints is a 2-manifold in 3-D parameter space. Furthermore, the solution is symmetric with respect to the $t = r$ plane so, we can consider one more inequality constraint, $t > r$, to speed up the equation-solving process by purging half the solution domain.

Denote by $\mathcal{M}$ the solution of Equations (1) and (2) that determines the hidden parts of the planar curve $\mathbf{C}(t)$. The projection of $\mathcal{M}$ into the $t\theta$-plane characterizes the regions where the curve is not visible. That is, if a parameter (t, θ) falls into the projected region of $\mathcal{M}$, then the corresponding curve point $\mathbf{C}(t)$ is not visible for the viewing direction $\mathcal{V}(\theta)$. Its complement, the uncovered region (under this projection) in the $t\theta$-plane, determines all the visible sections of $\mathbf{C}$ along continuously varying view directions. Figure 2 shows an example of such a visibility chart. Gray regions in Figure 2(a) represents the 2D projection of $\mathcal{M}$ for the planar curve $\mathbf{C}(t)$. Given a viewing direction $\mathcal{V}$, one can extract a set of visible curve segments from the uncovered (white) regions (see Figure 2(b)).

As one can see from Figure 2(b), visibility queries are resolved by extracting corresponding white regions from the visibility chart. Thus, solving the visibility problem for planar curves can be reduced to that of finding boundary curves of the projected regions of $\mathcal{M}$ in the parameter space. Since the projection is performed to the $t\theta$-plane, the boundary of the projected region under this projection occurs either at the boundaries of the zero-set $\mathcal{M}$ or at its local extrema. Since $\mathcal{M}$ is continuous and closed, it has no boundary and hence, the visibility problem reduces to finding r-extrema of the zero-set $\mathcal{M}$ which are the r-directional silhouettes of $\mathcal{M}$.

Definition 1. *Given a one-parameter family of viewing directions* $\mathcal{V}(\theta)$*, a* C^1*-continuous planar curve* $\mathbf{C}$*, and the solution manifold* $\mathcal{M}$ *of Equations (1) and (2) for* $\mathbf{C}$*;*

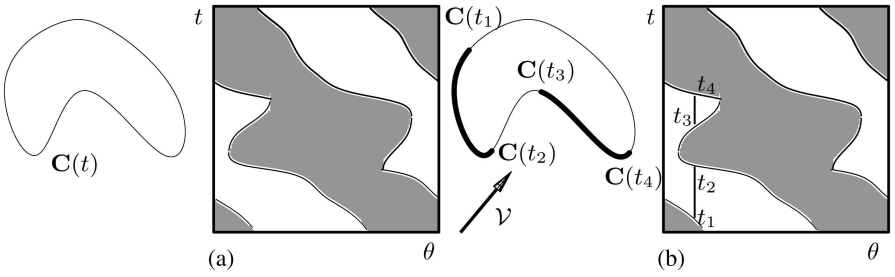


Fig. 2. (a) Given a planar curve $\mathbf{C}(t)$, the gray region in the $t\theta$-plane represents hidden curves of $\mathbf{C}$. (b) Visible curve segments can be extracted from the uncovered (white) regions.

1. *The r-directional silhouette curves, S^r, comprise the set of points on $\mathcal{M}$ whose r-directional partial derivative vanishes (bold lines in Figure 3(a) shows the projection of S^r in the $t\theta$-plane).*
2. *Denote by $S_I^r \subset S^r$ the set of points that falls in the interior of the projection of $\mathcal{M}$, among the set of r-directional silhouettes S^r (see dotted line segments in Figure 3(b)).*

Then, the sought boundary of $\mathcal{M}$, $\partial\mathcal{M}$, that delineates the visible segments of $\mathbf{C}$ from all possible views, can be computed using the two sets S^r and S_I^r as:

$$\partial\mathcal{M} = S^r - S_I^r.$$

Figure 3(c) presents $\partial\mathcal{M}$ in bold lines and $\mathcal{M}$ as a shaded region.

The r-directional silhouette curves, S^r, of $\mathcal{M}$ can be computed by finding the simultaneous solution of Equations (1), (2) and (3), where

$$\frac{\partial\mathcal{F}}{\partial r}(t, r, \theta) = 0. \tag{3}$$

Having two equality equations in three variables, solutions of the three equations are curves in the $tr\theta$-parameter space. As $\mathcal{F}$ and $\mathcal{G}_2$ are piecewise rational functions, the solution can be constructed by exploiting the convex hull and subdivision properties of NURBS, yielding a highly robust divide-and-conquer computation [5]. The solver [5] recursively subdivides rational functions along all parameter directions until a given maximum depth of subdivision or some other termination criteria is reached. At the end of the subdivision step, a discrete set of points are numerically improved into a highly precise solutions using a multivariate Newton-Raphson iterative stage. Finally, these discrete points are connected into a set of piecewise linear curves in the parameter space (See [23] for more details).

An entire curve segment or any portion of the curve segment in S^r can fall inside the projected region of $\mathcal{M}$ (see Figure 3(a)). We need to trim away S_I^r from S^r since they correspond to interior curve segments. An efficient and robust algorithm for purging S_I^r away is presented in this section and is based on the analysis of a topological change in the visibility charts. Given a continuous one parameter family of view directions

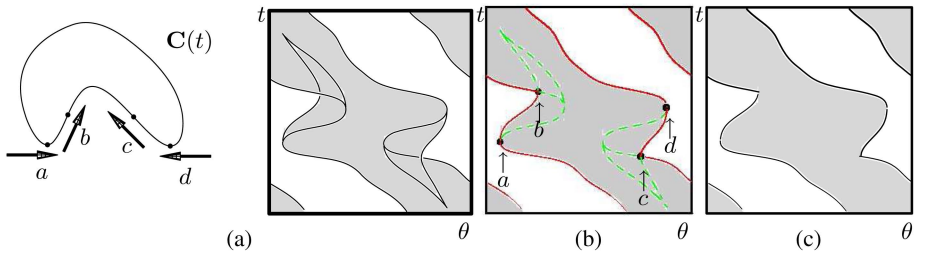


Fig. 3. (a) r-direction silhouette curves S^r projected into the $t\theta$-plane. (b) Dotted line segments represent S_I^r and (c) $\partial\mathcal{M} = S^r - S_I^r$ is shown in bold. Critical points are computed using a topological analysis and shown in (b). Their corresponding curve points and view directions are also shown in (a).

$\mathcal{V}(\theta)$, a topological change (i.e. a change in the number of connected components) can occur either globally or locally. Global topological changes occur where the viewing direction is parallel to a bi-tangent line segment of $\mathbf{C}$ connecting two (or more) points. Topological changes occur locally where the viewing direction is parallel to the tangent direction of $\mathbf{C}$, at an inflection point.

The bi-tangent line segment of $\mathbf{C}$ touches tangentially the curve at two or more different points. Bi-tangent directions can be computed by simultaneously solving the following three equations, in three variables:

$$\mathcal{F}(t, r, \theta) = 0,$$

$$\frac{\partial \mathcal{F}}{\partial t}(t, r, \theta) = \langle \mathcal{V}(\theta), N(t) \rangle = 0, \tag{4}$$

$$\frac{\partial \mathcal{F}}{\partial r}(t, r, \theta) = \langle \mathcal{V}(\theta), N(r) \rangle = 0. \tag{5}$$

Equations (4) and (5) constrain the viewing direction $\mathcal{V}(\theta)$ to touch $\mathbf{C}$ tangentially at two different points $\mathbf{C}(t)$ and $\mathbf{C}(r)$, respectively. The bi-tangent direction of $\mathbf{C}$ itself can be computed using two polynomial equations in two variables. In this context, however, the viewing direction $\mathcal{V}(\theta)$, which is parallel to the bi-tangent direction, must be computed for further processing. Inflection points of a planar curve occur at points where the sign of the curvature, a rational form if $\mathbf{C}$ is rational, changes. Solution points of $t = r$ clearly satisfy all the above equations and must be purged away.

Let $\mathcal{T}$ be a set of points (t, r, θ) in the $tr\theta$-parameter space that correspond to either bi-tangents or inflection points. We constrain point $(t, r, \theta) \in \mathcal{T}$ to be outside the projected region. The black bold dots in Figure 3(b) represents these critical points, at which the topological structure of the visibility chart changes. Thus, the r-directional silhouette curves, S^r, are trimmed at such critical points $(t, r, \theta) \in \mathcal{T}$. The curve segments S_I^r (Dotted line segments in Figure 3(b)) can be determined using a simple visibility check of a single point, testing whether the segment falls inside the projected region of $\mathcal{M}$ or not. Figure 3(c) shows the visible boundaries $\partial\mathcal{M}$ of the projected regions as a set of piecewise curves.

3 Continuous Visibility for Freeform Surfaces

The presented algorithm for computing visibility of planar curves can be extended for computing the hidden surfaces. Given two-parameters family of viewing directions $\mathcal{V}(\theta, \varphi)$, the visibility problem for the surface case is solved in a six-dimensional parameter space, $(u, v, s, t, \theta, \varphi)$. Much like the curve case, this higher dimensional formulation simultaneously considers all view directions in the domain, and provides a reliable solution to a particular visibility query. We first present a set of conditions for determining whether a surface location $\mathbf{S}(u, v)$ is visible or not.

Lemma 2. *A surface point $\mathbf{S}(u, v)$ is invisible if and only if there exists another surface point $\mathbf{S}(s, t)$ such that*

$$\mathcal{F}(u, v, s, t, \theta, \varphi) = \left\langle \mathbf{S}(u, v) - \mathbf{S}(s, t), \frac{\partial \mathcal{V}}{\partial \theta}(\theta, \varphi) \right\rangle = 0, \tag{6}$$

$$\mathcal{G}(u, v, s, t, \theta, \varphi) = \left\langle \mathbf{S}(u, v) - \mathbf{S}(s, t), \frac{\partial \mathcal{V}}{\partial \varphi}(\theta, \varphi) \right\rangle = 0, \tag{7}$$

$$\mathcal{H}(u, v, s, t, \theta, \varphi) = \langle \mathbf{S}(u, v) - \mathbf{S}(s, t), \mathcal{V}(\theta, \varphi) \rangle > 0, \tag{8}$$

where $\mathcal{V}(\theta, \varphi)$ is a polynomial approximation to the sphere that spans all possible viewing directions.

Proof. By Equations (6) and (7), the two surface points $\mathbf{S}(u, v)$ and $\mathbf{S}(s, t)$ are on the same line with the same direction to the view direction $\mathcal{V}(\theta, \varphi)$. By satisfying Equation (8), $\mathbf{S}(s, t)$ is closer to the view source than $\mathbf{S}(u, v)$, which makes $\mathbf{S}(u, v)$ invisible for that view direction. ■

Since Equation (8) is an inequality constraint, the simultaneous zeros of the three Equations (6) – (8) are 4-manifolds in a six-dimensional parameter space. Let $\mathcal{M}$ be the 4-manifold zero-set of Equations (6) – (8). Then, similarly to the curve case, the projection of the zero-set into the $uv\theta\varphi$-domain prescribes the hidden parts of the surface $\mathbf{S}(u, v)$. If (u, v, θ, φ) falls into the interior of the projected region of $\mathcal{M}$, then the corresponding surface location, $\mathbf{S}(u, v)$, is not visible from viewing direction $\mathcal{V}(\theta, \varphi)$. In other words, the uncovered region (under this projection), in the $uv\theta\varphi$-domain, determines all the visible sections of $\mathbf{S}(u, v)$ along continuously varying viewing directions. In Figure 4(a), a shaded region depicts the projection of the zero-set, $\mathcal{M}$, into the $uv\theta\varphi$-parameter space. A parameter $(u_1, v_1, \theta_1, \varphi_1)$ falls into the projected region in Figure 4(a) and thus, its corresponding surface point $\mathbf{S}(u_1, v_1)$ is invisible for viewing direction $\mathcal{V}(\theta_1, \varphi_1)$ (see Figure 4(b)). On the other hand, point $\mathbf{S}(u_2, v_2)$ is visible since parameter $(u_2, v_2, \theta_1, \varphi_1)$ is located outside the projected region.

Projected into the $uv\theta\varphi$ four-dimensional space, the boundaries of the projection of the zero-set $\mathcal{M}$ can be determined as the st-directional silhouettes of $\mathcal{M}$, by finding all the simultaneous zeros of Equations (6) – (9), where

$$\mathcal{I}(u, v, s, t, \theta, \varphi) = \langle \mathcal{V}(\theta, \varphi), \mathbf{N}(s, t) \rangle = 0, \tag{9}$$

and $\mathbf{N}(s, t)$ is a normal vector field of $\mathbf{S}(s, t)$. The common zero-set of Equations (6) – (9) is now a 3-manifold in a six-dimensional space, which is the boundary of the

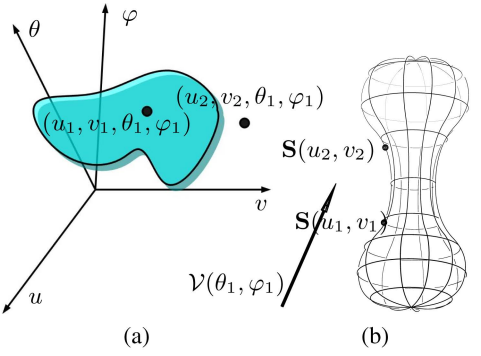


Fig. 4. (a) A shaded volume depicts a projection of the solution $\mathcal{M}$ into the $uv\theta\varphi$-parameter space. (b) $\mathbf{S}(u_1, v_1)$ is invisible for a viewing direction $\mathcal{V}(\theta_1, \varphi_1)$ since $(u_1, v_1, \theta_1, \varphi_1)$ falls into the projected volume. Compare it with $\mathbf{S}(u_2, v_2)$.

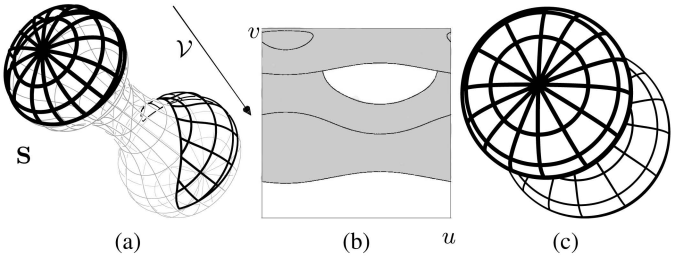


Fig. 5. (a) A surface $\mathbf{S}$ with a viewing direction $\mathcal{V}$. (b) A set of trimming curves in the uv-parameter domain. (c) Visible parts of $\mathbf{S}$ are shown for the given view direction.

projected volume of $\mathcal{M}$. Given a particular viewing query $\mathcal{V}(\theta_0, \varphi_0)$, two of the solution space's remaining degrees-of-freedom are fixed and we can extract 1-manifold solution curves from the projected region of $\mathcal{M}$. These curves in the parameter space correspond to curves that delineate the hidden surfaces from the visible ones.

It is quite difficult to either visualize or contour 3-manifolds in a six-dimensional space. By fixing a particular viewing direction, 1-manifold curves in a six-dimensional space result. So it is possible to use the algorithm presented by Seong et al [23] to extract all the visible parts of $\mathbf{S}(u, v)$. Figure 5(a) shows a surface $\mathbf{S}$ with a viewing direction $\mathcal{V}$. The boundary curves of visible sections in the uv-domain are computed using our approach (see Figure 5(b)). In Figure 5(c), gray-colored trimming surfaces represent hidden surfaces of the original surface and the bold ones are visible sections for the viewing direction. Shaded regions in the parameter domain (Figure 5(b)) correspond to the hidden surfaces in Euclidean space (Figure 5(c)).

4 Experimental Results

We now present examples of computing a visibility chart in a continuous domain for both planar curves and 3D surfaces. For all the figures, the gray-colored region

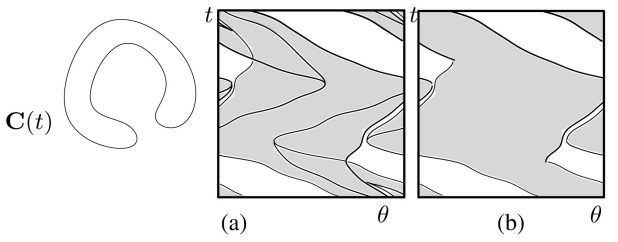


Fig. 6. (a) Given a planar curve $\mathbf{C}(t)$, the projected region of $\mathcal{M}$ and projected r-directional silhouette curves S^r are shown in gray and bold lines, respectively. (b) A set of visible segments, S^r_v, is shown in bold lines.

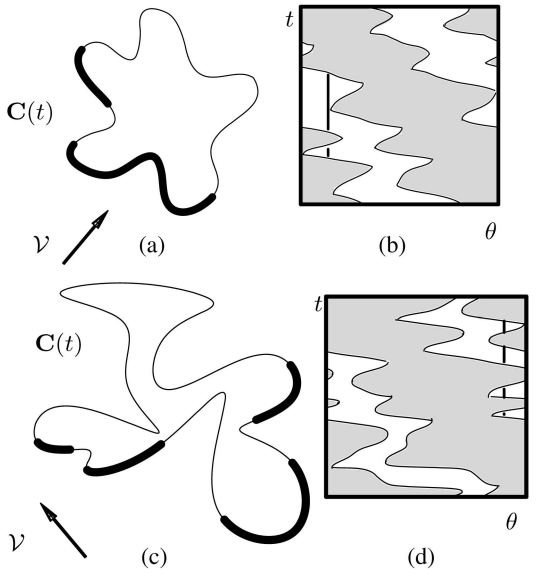


Fig. 7. (a), (c) A planar curve $\mathbf{C}(t)$ and the visible curve segments that are shown in bold lines. (b), (d) A continuous visibility charts computed by solving Equations (1) – (3).

represents the projection of the zero-set of the corresponding set of polynomial equations in the parameter space and characterizes hidden parts of planar curves or surfaces. Bold lines in curves or surfaces represents visible parts from the given view direction.

Figure 6 shows a planar curve and its visibility charts in a continuous domain. Bold lines in Figure 6(a) represent a set of r-directional silhouettes of the zero-set manifold. The boundary curves of the projected region are computed based on a topological analysis of the visibility charts and shown in Figure 6(b).

In Figures 7, (a) and (c) show two planar curves and (b) and (d) are the visibility charts for all viewing directions. For a particular viewing direction, $\mathcal{V}$, a set of visible curve segments are shown in bold lines in Figures 7(a) and (c). Figures 7(b) and (d) show the corresponding parameter domain in thick lines. The computation time for generating the visibility charts over all possible view directions for the curve case vary

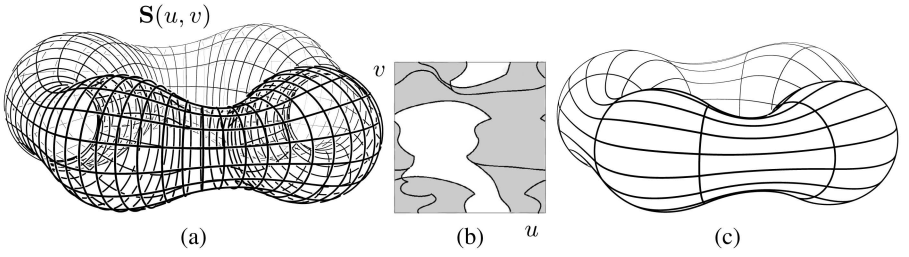


Fig. 8. (a) An envelope surface generated by sweeping a scalable ellipsoid along a space trajectory is shown. (b) A set of trimming curves in the uv-parameter domain is presented in bold lines. (c) Visible parts of the surface are shown for the given viewing direction.

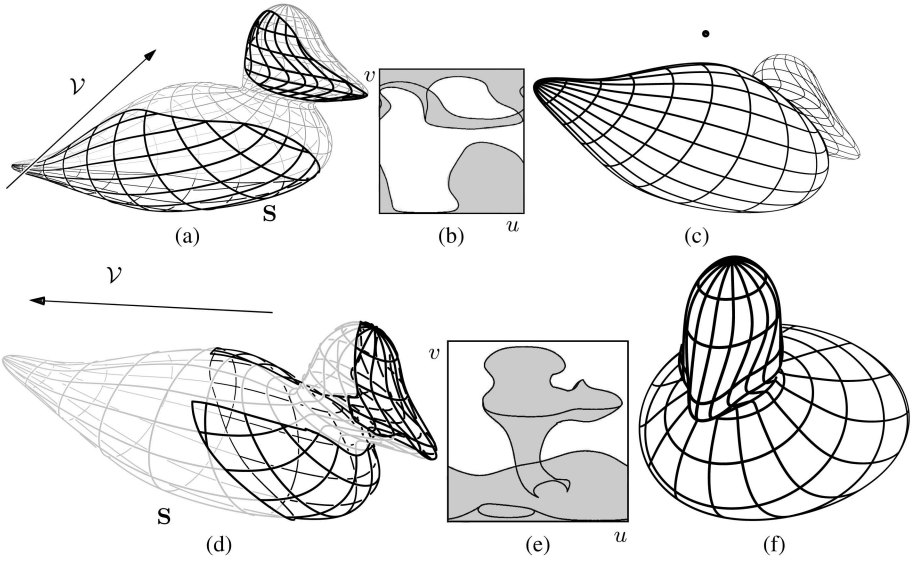


Fig. 9. (a), (d) A surface **S** is shown with a view direction. (b), (e) A set of trimming curves in the uv-parameter domain is presented in bold lines. (c), (f) Visible parts of the surface are shown for the given viewing direction.

according to the curve's complexity, taking from 1.3 to 6 seconds on a Pentium IV 2GHz desktop machine.

Figure 8(a) shows an envelope surface generated by sweeping a scalable ellipsoid along a space trajectory. A set of trimming curves is shown in Figure 8(b), which is the result of solving Equations (6) – (9) after fixing a viewing direction. Each trimmed surface sub-region is tested for visibility using a simple ray-surface intersection method. Figure 8(c) draws visible surface patches only.

The original surfaces in Figure 9(a) and (d) are bi-quartic NURBS having about 250 control points and shown with different view directions. Figure 9(b) and (e) show a set of trimming curves which are boundaries between visible parts and hidden surfaces in

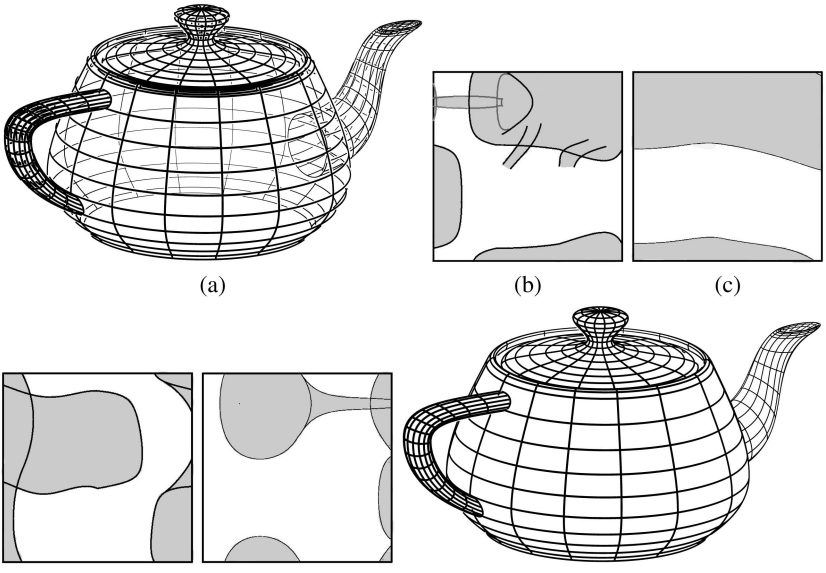


Fig. 10. (a) A teapot is presented by four surface patches. A set of trimming curves in the parameter domain of the body (b), handle (c), spout (d) and the cap (e). Trimmed surfaces are shown in (f) which are visible for the viewing direction.

the uv-parameter domain. Figure 9(c) and (f) show visible surface patches only along a specified viewing direction. On a 2GHz Pentium IV machine, computing the trimming curves in the uv-domain for Figures 8 – 10 took about 13 to 45 seconds.

The teapot in Figure 10 is represented by four bi-cubic NURBSs surfaces which are open (Figure 10(a)). Each of the four surface patches can be hidden by any of the other ones according to the viewing direction. In Figure 10(a), part of the body is blocked by both a handle and a cap for the given viewing direction (a figure is generated along the viewing direction). Furthermore, it blocks itself and makes shadow regions. Figure 10(b) shows the trimming curves in the parameter domain of the body. They are comprised of three set of curves. Trimming curves generated due to a cap are represented by gray-colored lines in Figure 10(b) and four open curve segments located in the middle part of the domain are generated by the handle. Since the surface patch of the handle is not closed, the trimming curves are also open. Thus, the geometric intersection curve between the handle and the body is needed for a proper trimming. All the other trimming curves in Figure 10(b) stems from the body itself. Figure 10(c)–(e) show a set of trimming curves for the handle, spout and the cap, respectively. Finally, Figure 10(f) draws all the visible parts.

5 Conclusion and Future Work

We have presented a robust and efficient scheme for computing hidden curve/surface removal, in the continuous domain. The approach is based on the derivation of a set of

algebraic constraints that determine the visibility of curve's or surface's locations. All view directions in the domain are considered simultaneously, and the algorithm provides a continuous chart for the visibility from all possible views. By simultaneously solving 2 polynomial equations for a curve case and 3 polynomial equations for a surface case, in the parameter space, the presented approach can detect all the hidden parts of the sculptured model for continuously varying view directions. The zero-set of the polynomial equations prescribes the hidden parts of the model and we construct a visibility chart by projecting the zero-set into an appropriate parameter space. Furthermore, the topological structure of the visibility chart is analyzed in the same framework, providing a reliable solution to the computation of the visibility chart.

The presented approach can be applied to trimmed models as well. The original trimming curves need to be considered in the computation of the boundary curves between visible and invisible parts in the case of trimmed models. Visibility computations for perspective views are desirable extensions to the method presented. To this end, we need to deal with even higher-dimensional solution spaces.

Acknowledgments

All the algorithms and figures presented in this paper were implemented and generated using the IRIT solid modeling system [12] developed at the Technion, Israel. This work was supported in part by NSF IIS0218809. All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

References

1. A. Appel. The Notion of quantitative Invisibility and the Machine Rendering of Solids. *Proceedings ACM National Conference*, 1967.
2. W.J. Bouknight. A procedure for Generation of Three-Dimensional Half-toned Computer Graphics Representations. *CACM*, Vol. 13, No. 9, 1969.
3. K. Bowyer and C. Dyer. Aspect graphs: An introduction and survey of recent results. *Proc. SPIE Conf. on Close Range Photogrammetry Meets Machine Vision*, 1395, pp. 200–208, 1990.
4. E. Catmull. A Subdivision Algorithm for Computer Display of Curved Surfaces. *Ph.D Thesis*, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, UT, 1974.
5. G. Elber and M. S. Kim. Geometric Constraint Solver Using Multivariate Rational Spline Functions. *Proc. of International Conference on Shape Modeling and Applications*, pp 216–225, MIT, USA, June 15-17, 2005.
6. G. Elber, R. Sayegh, G. Barequet and R. Martin. Two-Dimensional Visibility Charts for Continuous Curves. *Proc. of ACM Symposium on Solid Modeling and Applications*, Ann Arbor, MI, June 4-8, 2001.
7. G. Elber and E. Cohen. Hidden curve removal for free form surfaces. *Computer Graphics*, Vol. 24, No. 4, pp. 95–104, 1990.
8. J. Foley, A. Van Dam, J. Hughes, and S. Feiner. Computer Graphics: Principles and Practice. Addison Wesley, Reading, Mass., 1990.

9. R. Galimberti and U. Montanari. An algorithm for Hidden Line Elimination. *CACM*, Vol. 12, No. 4, pp. 206–211, 1969.
10. C. Hornung. A Method for Solving the Visibility Problem. *CG&A*, pp. 26–33, July 1984.
11. C. Hornung, W. Lellek, P. Pehwald, and W. Strasser. An Area-Oriented Analytical Visibility Method for Displaying Parameterically Defined Tensor-Product Surfaces. *Computer Aided Geometric Design*, Vol. 2, pp. 197–205, 1985.
12. IRIT 9.0 User's Manual, October 2000, Technion. http://www.cs.technion.ac.il/∼irit.
13. T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual Contouring of Hermite Data. *In Proceedings of SIGGRAPH 2002*, pp. 339–346, 2002.
14. T. Kamada and S. Kawai. An Enhanced Treatment of Hidden Lines. *ACM Transaction on Graphics*, Vol. 6, No. 4, pp. 308–323, 1987.
15. M. S. Kim and G. Elber. Problem Reduction to Parameter Space. *The Mathematics of Surface IX (Proc. of the Ninth IMA Conference)*, London, pp 82–98, 2000.
16. S. Krishnan and D. Manocha. Global Visibility and Hidden Surface Algorithms for Free Form Surfaces. Technical Report: TR94-063, University of North Carolina, 1994
17. L Li. Hidden-line algorithm for curved surfaces. *Computer-Aided Design*, Vol. 20, No. 8, pp. 466–470, 1988.
18. P. Loutrel. A Solution to the Hidden-line Problem for Computer Drawn Polyhedra. *IEEE Transactions on Computers*, Vol. C-19, No. 3, pp. 205–213, 1970.
19. M. Mckenna. Worst-Case Optimal Hidden-Surface Removal. *ACM Transaction on Graphics*, Vol. 6, No. 1, pp. 19–28, 1987.
20. M. Mulmuley. An efficient algorithm for hidden surface removal. *Computer Graphics*, Vol. 23, No. 3, pp. 379–388, 1989.
21. T. Nishita, S. Takita, and E. Nakamae. Hidden Curve Elimination of Trimmed Surfaces Using Bezier Clipping. *Proc. of the 10th International Conference of the Computer Graphics on Visual Computing*, pp. 599–619, Tokyo Japan, 1992.
22. Y. Ohno. A Hidden Line Elimination Method for Curved Surfaces. *Computer-Aided Design*, Vol. 15, No. 4, pp. 209–216, 1983.
23. J. K. Seong, G. Elber, and M. S. Kim. Contouring 1- and 2-Manifolds in Arbitrary Dimensions. *Proc. of International Conference on Shape Modeling and Applications*, pp 216–225, MIT, USA, June 15-17, 2005.
24. I. Sutherland, R. Sproull, and R. Schumacker. A Characterization of ten Hidden-Surface Algorithms. *Computer Surveys*, Vol. 6, No. 1, pp. 1–55, 1974.
25. K. Weiler and P. Atherton. Hidden Surface Removal Using Polygon Area Sorting. *SIGGRAPH77*, pp. 214–222, 1977.
26. T. Whitted. An Improved Illumination Model for Shaded Display. *ACAM*, Vol. 23, No. 6, pp. 343–349, 1980.

Density-Controlled Sampling of Parametric Surfaces Using Adaptive Space-Filling Curves

J.A. Quinn[1], F.C. Langbein[1], R.R. Martin[1], and G. Elber[2]

[1] Cardiff University, UK
{J.A.Quinn, F.C.Langbein, Ralph.Martin}@cs.cardiff.ac.uk
[2] Technion, Israel
Gershon@cs.technion.ac.il

Abstract. Low-discrepancy point distributions exhibit excellent uniformity properties for sampling in applications such as rendering and measurement. We present an algorithm for generating low-discrepancy point distributions on arbitrary parametric surfaces using the idea of converting the 2D sampling problem into a 1D problem by adaptively mapping a space-filling curve onto the surface. The 1D distribution takes into account the parametric mapping by employing a corrective approach similar to histogram equalisation to ensure that it gives a 2D low-discrepancy point distribution on the surface. This also allows for control over the local density of the distribution, e.g. to place points more densely in regions of higher curvature. To allow for parametric distortion, the space-filling curve is generated adaptively to cover the surface evenly. Experiments show that this approach efficiently generates low-discrepancy distributions on arbitrary parametric surfaces and creates nearly as good results as well-known low-discrepancy sampling methods designed for particular surfaces like planes and spheres. However, we also show that machine-precision limitations may require surface reparameterisation in addition to adaptive sampling.

1 Introduction

Many applications in geometric modelling and computer graphics require generation of evenly distributed points on surfaces. An even point distribution is important for two reasons: (i) to avoid aliasing artefacts which might be caused by regularly-spaced samples, and (ii) to provide efficiency in surface calculations—if the distribution is uneven, more samples are needed to guarantee a minimum point density. We discuss and analyse an efficient, practical algorithm to generate such point distributions on parametric surfaces in 3D, based on an approach suggested by [1]. The method also allows the user to control the local point density and further corrects for handling extreme parametric distortion.

Local density control is useful for applications such as *surface triangulation*, where we may want more samples in regions of higher curvature [2]. Existing applications where stochastic/low-discrepancy distributions are already used include radiosity [3] and ray tracing [4]. Another application is *point-based rendering* where surfaces are represented by point sets and are rendered employing

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 465–484, 2006.
© Springer-Verlag Berlin Heidelberg 2006

splats (small discs) which can be quickly generated using hardware shaders [5]. This provides a simple and efficient approach to non-photorealistic rendering and rendering of complex or dynamic surfaces [6]. Efficiently distributing, resampling and parameterising suitable point distributions on a surface is essential for interactivity and visible artefact reduction [7]. Stochastic methods are also widely used for measurement and quality control applications, e.g. to compute volume integrals and surface curvature of complex shapes. Other examples are *meshfree finite element analysis* [8,9], and *re-meshing* of meshes with a known parameterisation [10]. Applications based on low-discrepancy sampling typically require far fewer samples than a random approach, so the effort put into generating these samples will likely become worthwhile when performing repeated surface calculations such as physical simulations or when using dynamic surfaces.

We seek an *even* distribution of points on a surface, perhaps with respect to a specified density function. In particular we seek a *low-discrepancy* point set: discrepancy is a measure of the deviation of a point set from a uniform distribution. It is computed locally for a subset by taking the absolute difference between the ratio of points lying inside the subset and an area measure of the subset (e.g. the surface integral over the density function). The discrepancy of a point set is the supremum of this local discrepancy. A point set is of low-discrepancy if its discrepancy is minimal. Often the subsets considered are restricted to certain types such as rectangles [11,12]. A point set of low discrepancy covers the surface as evenly as possible and often has the desirable property that there are no large 'holes' in the distribution, while at the same time avoiding aliasing problems [4].

In this paper we describe an approach to generating low-discrepancy point distributions on parameterised surfaces by distributing points along a space-filling curve as suggested by Steigleder and McCool [1]. By generating a *space-filling curve* in the parameter domain, the problem of distributing points in 2D is reduced to sampling a curve appropriately. Sample points are placed along the space-filling curve using an idea similar to *histogram equalisation*. Adaptive generation of the space-filling curve allows us to handle parametric distortions where, e.g., a small area in the parameter domain is mapped onto a large area on the surface. However, for parameterisations in which extremely large areas of the surface are spanned by very small ranges of parameter values, we encounter limitations caused by the machine precision, as explained in Sect. 5.3.

The space-filling curve not only converts the 2D problem to a 1D problem, but also provides the additional benefit of good spatial localisation of the points, in the sense of knowing which points in the output sequence lie near which other points in the sequence [13]. This is clearly of advantage for problems which require, for example, rapid determination of the k-nearest neighbours of a given surface point.

We first review previous work and describe the algorithm. Then we briefly demonstrate the technique in the plane directly following [1], and evaluate the results obtained. Comparing the generated point sets to known low-discrepancy distributions indicates that they are close to the best, known low-discrepancy distributions, with consistently good results for various test shapes used to measure

discrepancy, and varying numbers of points. Most low-discrepancy sequences are optimised for discrepancy measures based on rectangular subsets, and while the approach is not quite as good as some other low-discrepancy methods for rectangular discrepancy, for the other-shaped discrepancy measures tested, it is better. We next demonstrate the effectiveness of the approach for sampling arbitrary parametric surfaces by computing discrepancy on the unit sphere with respect to spherical triangles. This provides strong evidence that this approach is a fast and effective way of low-discrepancy sampling on the sphere. Estimating the discrepancy for general surfaces in $\mathbb{R}^3$ is rather harder as it not only requires the calculation of exact surface areas, but also choice and construction of sampling shapes (such as a triangles) to assess the distribution on each particular surface. However, we give visual examples showing our methods applied to more general surfaces, with and without explicit user-provided density control functions. We then demonstrate how adaptive curve generation helps considerably on shapes with extreme parameterisations, but that it is ultimately not always sufficient, and that surface reparameterisation may also be required.

2 Previous Work

In this section we discuss previous approaches to generating low-discrepancy distributions in the plane and on parametric surfaces. We start by briefly reviewing general work that prompted our research, and then then discuss work more specifically related to our approach.

Niederreiter and Sobol sequences have been theoretically shown to produce an optimal low-discrepancy sequence [14] for rectangular subsets of $2D$ manifolds with discrepancy varying as $O(N^{-1}\log^2 N)$ for N points. They have been shown to be considerably better than random sampling in Monte-Carlo techniques both theoretically [15] and in practice for various geometric problems [16] and have also been applied to problems such as rendering [17]. In order to evaluate the quality of the point distributions generated by the current algorithm, we compared our results to low-discrepancy distributions generated by ACM TOMS Algorithms 738 [18] and 659 [19]. We also compared the results to a random distribution and a jittered distribution on the plane [20]. Generating independent pairs of random numbers gives a random distribution in the unit square [21] with expected discrepancy $O(N^{-1/2})$ [22]. Base-2 Hammersley distributions on the unit sphere were generated using the algorithm described in [23].

Our approach uses an adaptive version of an algorithm suggested by Steigleder and McCool [1] to generate density-controlled stratified samples in n-dimensions and employ them to sample surfaces. In this paper we provide experimental evidence that this approach allows the user to generate high quality, density-controlled distributions on arbitrary surfaces, correcting for parameteric distortions by adaptively generating the curve in the parameter domain. However, if the parametric distortion is too great, reparametrisation of the surface may also be required, as we discuss later. The sample points are produced along the curve using a technique similar to histogram equalisation.

Our algorithm uses a method similar to histogram equalisation to distribute points along a space filling curve lying in a surface to achieve the desired density distribution based on approximating the area of small surface patches (see Sect. 3). Other 1D inversion methods exist, such as the one presented in [24], but they do not generalise to arbitrary surfaces or provide the same utility as a space-filling curve.

Although algorithms such as [18] are available for generating low-discrepancy point sets in n-dimensional parallelpipeds starting from given seeds, we are not aware of methods specifically for generating low-discrepancy point sets on arbitrary surfaces in 3D, and which also allow the user to control the point density via a function. However, methods for generating such point-sets are known for specific surfaces. For example, [25] uses lines between low-discrepancy points on the surface of a sphere to calculate mass properties of objects, and [23] demonstrates the generation of Halton and Hammersley low-discrepancy sequences on the sphere. A similar approach using intersecting lines is used by [26] to generate a point cloud from a mesh. The first two papers use specific methods to generate points on the sphere, and do not readily generalise. The third provides no evidence that the point distribution generated from the mesh is of low discrepancy on the surface. Hartinger [27] describes a generalisation of a quasi-Monte Carlo technique originally proposed by [28] to generate points in the plane with an arbitrarily chosen distribution for computing integrals.

3 Point Distribution Algorithm

In this section we describe how to generate a low-discrepancy point set on a parametric surface S. Given a surface parametrisation $f : [0,1]^2 \to \mathbb{R}^3$ of S with the unit square as normalised parameter domain, a non-negative bounded density function $\delta : S \to \mathbb{R}_0^+$, and a desired number of points N, our algorithm generates a set of points p_l, $l = 1, \ldots, N$, distributed on S according to the density δ. More precisely, we generate a set of parameter values representing these points. We desire that, for each subset A of the surface S, the fraction of points p_l lying inside A should be as close as possible to the ratio between the surface integrals $\int_A \delta \, ds / \int_S \delta \, ds$ to ensure that the point set has low discrepancy (with respect to the desired density). For the special case $\delta \equiv 1$, the points are uniformly distributed with respect to surface area.

The basic idea of Steigleder and McCool's algorithm [1] is to convert the 2D distribution problem into a 1D distribution problem, by placing points along a space-filling curve that covers a square, which for our application is the parameter space. An approximation to a space-filling curve is generated in the unit square, and mapped onto S. This approximation can be described by a sequence of vertices v_l, $l = 0, \ldots, M$, lying on the curve in the surface. We then create a 1D point distribution q_k, $k = 1, \ldots, N$ in $[0,1]$ where the distances between these points indicates some desired initial distribution in the unit square (choices for this distribution will be discussed later, but, for example, we could use equal-distance points). This distribution is then mapped onto the 2D surface using

similar ideas to histogram equalisation: points p_l, $l = 1, \ldots, N$ are distributed on the curve in the surface so that they have approximately the same distances as the q_l in $[0, 1]$ in the plane. However, to ensure that fractional arc-lengths are preserved on the curve in the surface (rather than the parameter domain), we must measure the distances according to the local surface geometry. At the same time, we add a further factor to produce the desired point density function δ on the surface, by further adjusting the distance measure.

As space-filling curves map a 1D line onto a 2D surface, measuring distances as lengths along the curve is ill-defined. Instead, we approximate the area of the surface that is covered by a certain fraction of the space-filling curve, and use this as the distance measure on the curve which indicates how many points should be distributed on a segment of the curve. Each of the points v_l can be associated with a small patch A_l of the surface for which we approximate the ratio $s_l = \int_{A_l} \delta \, ds / \int_S \delta \, ds$. The s_l together with the v_l form a discrete distribution which indicates the desired local density of points. By calculating the cumulative sum of s_l as we move along the curve, we integrate over the discrete distribution indicating how the number of the points we wish to distribute has to be increased. This allows us to estimate the distances between the points on the curve in the surface and distribute them according to the distances between the q_l in $[0, 1]$. The rest of this section presents the main algorithm in more detail, especially concerning measuring distance on the curve. Then we describe various options for generating initial point distributions and the histogram equalisation.

Pseudocode for the overall point distribution algorithm is given in Fig. 1. It consists of the following main components: a space-filling curve generator, Fig. 2, that computes a set of vertices lying on an approximation to a space-filling curve for the unit-square; a generator which computes a 1D sequence of real numbers in the unit interval $[0, 1]$ indicating how the points are to be distributed on the curve; and an equalisation method to distribute points on the surface along the space filling curve by measuring distances along the curve based on the local surface area and the density function. First, the curve is generated, the parameterisation is applied and surface area is calculated along the curve. We then generate a 1D sequence of points, and map them to the curve, according to the integral of the surface area, taking into account a surface density function and the level of recursion of the curve.

First we call a recursive function PopulateCurveTree to generate an approximation of a space-filling curve in the unit square. A tree is generated using this method according to a specified curve type C (different space-filling curves can be used in principle, such as the *Hilbert Curve*, the *Hilbert II Curve*, etc.). We use a tree to facilitate the subdivision of the unit square, and after all subdivision, the final curve vertices are generated and stored in the leaf nodes.

The unit square is subdivided into a number of cells centred around each curve vertex (dependant on the curve). The method AssessSubdivision applies the supplied parameterisation f to each cell, and calculates the distances from the centre point to the corners and half-way points along each side. The maximum distance is taken, and compared to the supplied cell size a. If the value is greater

Algorithm. DISTRIBUTEPOINTS (f, δ, r, N, C)

Input: f—parametrisation $f : [0,1]^2 \to \mathbb{R}^3$ for surface S
δ—density function $\delta : S \to \mathbb{R}_0^+$
r—maximum recursion depth for space-filing curve approximation
a—parameter for the minimum required area associated with a curve vertex v
N—number of points that should be distributed on S
C—type of space-filling curve

Output: A low-discrepancy point distribution on S according to δ, given as a list of parameter values for f

1. $n \leftarrow$ GENERATEROOTNODE(C)
2. $T \leftarrow$ POPULATECURVETREE (n, r, a, 0)
3. $[v_1, \ldots, v_M] \leftarrow$ OUTPUT(T)
4. $S_0 \leftarrow 0$
5. **for** $l \leftarrow 1, \ldots, M$ **do**
6. $\quad S_l \leftarrow S_{l-1}+$DENSITY ($f$, δ, v_l)
7. $[q_1, \ldots, q_N] \leftarrow$CREATE1DPOINTS($N$)
8. $[p_1, \ldots, p_N] \leftarrow$EQUALISE($[q_1, \ldots, q_N]$, $[v_1, \ldots, v_M]$, $[S_1, \ldots, S_M]$)
9. **return** $[p_1, \ldots, p_N]$

Fig. 1. Point Distribution Algorithm

than a and the current depth g is less than the recursion cap r, a true result is returned and GENERATECHILDREN is called on the node. r should be set according to machine precision, or as a limit on the maximum memory requirements of the process. GENERATECHILDREN subdivides the cell, generating the number of children required according to the space-filling curve C, and applies the appropriate permutation. Permutations represent an ordering of the vertices in a given cell, and are calculated according to a set of simple rules based on the ordering of the parent cell and which child the current cell is.

If no subdivision is required, a final index, representing the cell's position along the curve is computed from the permutation and stored. When all subdivision is complete, a method OUTPUT is called which scales the cell indices to the maximum level of curve recursion used and hashes the values stored in the leaf nodes of the tree to a set of vertices v_l, $l = 1, \ldots, M$ indicating the line segments (see Fig. 1, line 3). In order to accurately estimate the distances between points on the curve in the surface for the distribution, M has to be sufficiently large (see Sect. 5.3). A suitable value for a can be chosen as half the required maximum distance between vertices of the space-filling curve on the surface. Appropriate choice of space-filling curve is discussed later in this section.

Next we compute the desired cumulative density of the point distribution by mapping each v_l onto the surface and estimating the ratio $s_l = \int_{A_l} \delta \, ds / \int_S \delta \, ds$ for a small surface patch A_l, using a method explained shortly. The s_l are approximated by the function DENSITY (Fig. 1, line 6). For each point $f(v_l)$ on the surface we get a cumulative density $S_l = \sum_{k=1}^{l} s_l$, $l = 1, \ldots, M$ (Fig. 1, lines 4–6). As we have an approximation of a space filling curve, we have $S_M \approx \int_S \delta \, ds$, if we choose suitable A_l. This means that the overall surface integral of δ is a scaling factor that can be ignored for distributing points as we know the number

Algorithm. PopulateCurveTree (n, r, a, g)
Input: n—root node for the specified curve
r—max space-filling curve recursion depth
a—parameter for the minimum required area associated with a curve vertex v
g—current depth of recursion
Output: A space filling curve defined by n, in a tree structure

1. $b \leftarrow$ AssessSubdivision(a)
2. **if** b is true and $g < r$
3. $[c_1, \ldots, c_N] \leftarrow$GenerateChildren($n$)
4. **for** $c \in [c_1, \ldots, c_N]$ **do**
5. PopulateCurveTree(c, r, a, $g + 1$)
6. **else**
7. Curve Index Stored in Node

Fig. 2. Curve Generation Algorithm

of points we intend to distribute. Hence, to approximate s_l we only approximate $\int_{A_l} \delta \, ds$.

We have considered several different methods to approximate this integral: (a) compute the area of the triangle $f(v_{l-1})$, $f(v_l)$, $f(v_{l+1})$ and multiply this area by the value of the density function at the centroid of this triangle; (b) use a small square centred at $f(v_l)$ instead of a triangle, constructed from extra vertices generated in the parameter domain with a size equal to a line segment of the curve; and (c) use $\delta(f(v_l))\sqrt{\det f_I(v_l)}$ where f_I is the first fundamental form of the surface, as $ds = \sqrt{\det f_I}\, du\, dv$. In general all three approaches work well, although (b) and (c) produced slightly superior results to (a), as demonstrated in Sect. 5.1. This may be partly due to the fact that for (a) the triangles overlap. For the examples reported in this paper we demonstrate results for (a) and (c). However, note that, ideally, the first fundamental form of the surface should be used and calculated automatically [29]. Also note, that for approach (c), we have to scale the results using: $A'_l = A_l(w^{k_{max}-k_{current}})$, where k_{max} is the maximum and $k_{current}$ the current depth of the space-filling curve recursion and w is the number of vertices in a curve 'cell' (i.e. the number of curve vertices where $k = 1$). This enables us to normalise the area value for the non-uniform curve.

After we have computed the cumulative density distribution S_l over the v_l, we generate a set of 1D points q_k, $k = 1, \ldots, N$ in $[0, 1]$ to provide some chosen initial distribution in the unit square (as explained later) by mapping these points on the space-filling curve in the parameter domain (Fig. 1, line 7). The S_l over the v_l describe how to find arc-lengths along the curve: the distance between $f(v_l)$ and $f(v_{l+1})$ on the curve in the surface is given by $S_{l+1} - S_l$ (the desired density). Thus, in the next step we map the distances between the q_k according to the the cumulative density distribution S_l over the vertices $f(v_l)$, $l = 1, \ldots, M$ on the curve in the surface via an equalisation approach (Fig. 1, line 8): a subset $p_l, l = 1, \ldots, N$ of the v_l is chosen such that the distances between the p_l correspond to the distances between the q_l.

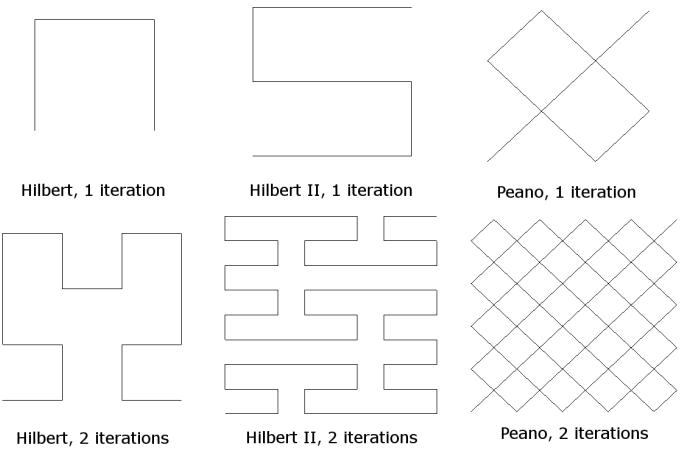


Fig. 3. The first two iterations of the Hilbert, Hilbert II and Peano curves

We now briefly describe in greater detail the space-filling curve approximations used in the first step of our algorithm. Various space-filling curves were investigated, including the Peano, Hilbert and Hilbert II curves [30] (see Fig. 3). Our adaptive sampling algorithm was used to generate these curves. However, as results in Sect. 4 demonstrate, the Hilbert curve is the space-filling curve of choice. If a uniform Hilbert curve is required, for use with very simple parameterisations, the algorithm described by Butz [31] and expanded by Lawder [32] can be used: it provides a very efficient way of generating points on the Hilbert curve using bit operations. In [1], *Jittered equally-spaced points* in the unit interval $[0, 1]$ were mapped onto the Hilbert curve. We generalised this technique and tested various methods for generating the 1D point sequences in the unit interval, mapped as fractional arc-lengths: *1D low-discrepancy sequence*, *Equally-spaced points*, *Jittered equally-spaced points* and *Randomly spaced points*. We expected the Jittered and Uniform points to demonstrate the best results, although one drawback to placing evenly spaced points on the curve is that if the number of points is commensurate with the number of vertices on the curve, we may get aliasing artefacts. Jittering the points removes the possibility of this occurring. For this reason, and because of the results discussed in Sect. 4, the third method is the method of choice for our final algorithm.

4 Experiments in the Plane

We performed some initial experiments in the plane to determine whether the choice of space-filling curve had a significant effect on the point distributions produced—even though Steigleder and McCool[1] suggest the use of a Hilbert curve, other space filling curves can also be used. These expand on the experimental results in [1] which showed how the star discrepancy scales for point sets of varying size, by comparing the point distributions generated in the unit square

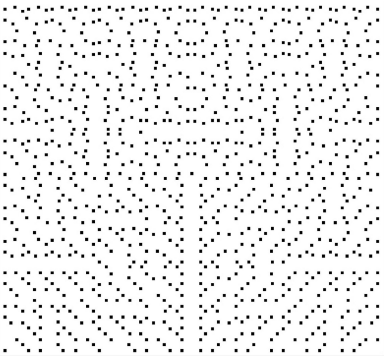


Fig. 4. Hilbert Curve 'Gaps'

with the reference point distributions listed in Sect. 2, determining the variation of discrepancy with number of sample points in each case. The star discrepancy is the most commonly employed measure due to its simplicity; it computes the discrepancy of a point set in the unit square based on axis-aligned rectangular subsets with one corner fixed to the origin. We also investigated generalisations of the star discrepancy, more specifically using triangles and quarter-circles, as in many applications, portions of surfaces of interest need not simply be rectangular.

4.1 Evaluation of Space-Filling Curves

Various choices exist for the space-filling curve. To determine the impact of this choice on the quality of the point distribution, we initially assessed the generated point sets both visually, and using experimentally measured discrepancy. Visually we aimed to ascertain any obvious regularities which might cause aliasing problems, or obvious gaps indicating high discrepancy—the human eye is very good at recognising patterns and holes in data.

Hilbert curve: The Hilbert curve showed large vertical and horizontal gaps in the distribution (see Fig. 4) when evenly-spaced or low-discrepancy 1D sequences were used. Aliasing could become a serious problem with such large holes in the distribution. Note in the example shown, the left half of the curve is identical to the right half with a big gap in the middle. Such gaps could cause visual and numerical problems. With a jittered 1D sequence, the curve did not exhibit these problems.

Hilbert II + Peano: Both the Hilbert II and Peano curves produced high quality results visually and in terms of measured discrepancy. Note that the Hilbert II curve does not have an axis of symmetry, unlike the Hilbert curve, and as a result does not suffer from the same aliasing problem.

The Hilbert II curve gave promising visual results as well as numerical discrepancy results and was hence chosen for further evaluation. Although the Hilbert

curve performed poorly in the visual evaluation for certain 1D sequences, its numerical discrepancy results were also good and scale well according to [1]. Furthermore, existing algorithms exist to generate it quickly, and previous results suggest that the Hilbert curve has the best *locality* properties (see Sect. 2). Hence, it was also chosen for further evaluation.

Although the Peano curve gave promising preliminary results, we discarded it for two reasons: firstly, its natural orientation is not aligned with the sides of the reference square, making its construction more complex to implement. Secondly, its constructor is self-intersecting: multiple vertices lie on the same point. Points in the square therefore may be covered more than once by the curve, increasing the likelihood of clumping. The converse is true for the other space-filling curves, as shown by the following argument. As the curve completely traverses the unit square and comes within a definite maximum distance of every point of it, if points are placed evenly along the curve, clumping cannot occur provided that any one segment of the curve has a minimum distance from other segments of the curve (apart, of course, from the ones preceding and following it). Hence, if no clumping can exist on the line, for a particular $1D$ sequence, the quality of the distribution would appear to be entirely due to the structure of the curve. This explains how space-filling curves, although always completely filling a space, can, and do, distribute the same sequence of points differently.

From now on, we will refer to the Hilbert curve sampled using a jittered 1D sequence as Hilbert-Jittered.

4.2 Evaluation of Numerical Discrepancy

Following the preliminary tests in the previous section, we investigated in detail the discrepancy properties of 2D point distributions generated by our implementation using the Hilbert and Hilbert II curves, sampled using random, evenly-spaced, jittered and low-discrepancy 1D point sequences. We found that the Jittered-Hilbert approach demonstrated the most consistent results, and so we only compare these with 2D point distributions produced at random, and using Niederreiter's method, Sobol's method and jittering. Note that when testing on the plane, even with adaptive curve generation, the curve would in fact be uniform, as the area is constant.

All of the tests were performed for point sets of size $N = 2^l$ and $N = 2^l + 2^{l-1}$, for $l = 0, \ldots, m$. Setting m to 19 gives 40 distribution sizes varying from 2 points to 1572864 points, resulting in a logarithmic range of data. Note that the number of vertices on the curve v_l generated from the recursive approximation depth k and the sizes of N were non-commensurate. However, testing with a uniform curve (a curve with a constant recursive depth) showed that even with the largest N and k set to 12 for the Hilbert curve and 8 for the Hilbert II curve, the curves consist of enough vertices such that the distance along the curve between each point placed is large enough to achieve the upper bound of the discrepancy of the set for each geometric subset.

Results are shown using graphs displaying discrepancy versus the logarithm of the number of points. Although theoretical discrepancy results are

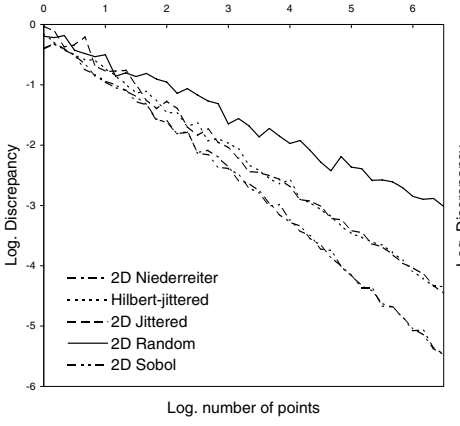


Fig. 5. Rectangular discrepancy comparison

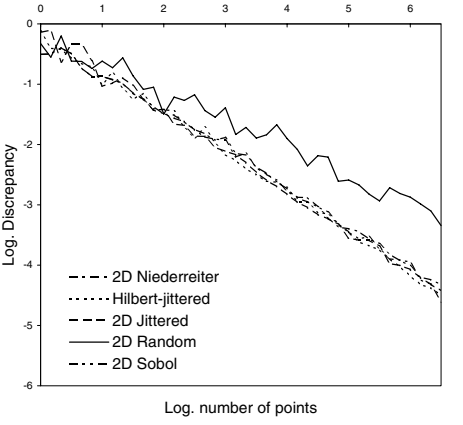


Fig. 6. Circular discrepancy comparison

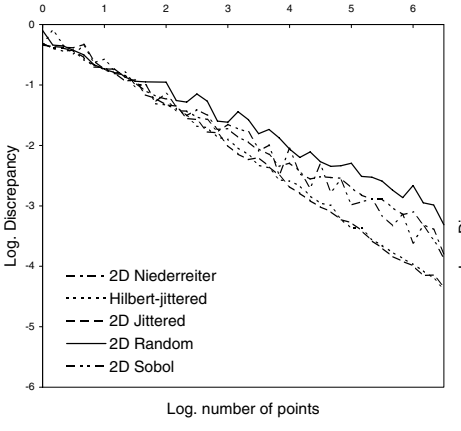


Fig. 7. Triangular discrepancy comparison

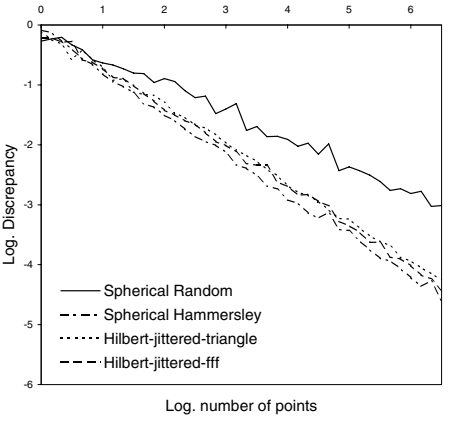


Fig. 8. Spherical discrepancy comparison

characterised by a power law times a logarithmic factor, the logarithmic factor is hard to determine experimentally due to its minor numerical effect. As can be seen in our graphs, on a double-logarithmic scale the experimentally determined discrepancy can be well approximated by a straight line. Thus, computing the slope of this line gives us an adequate way of comparing the behaviour of the different approaches.

For a random sequence the expected slope of a least-squares fitting line is $-1/2$. Clearly, we hope our point distributions to scale better than a random distribution. For N points in the unit square, the expected best relative error in area which can be achieved is $O(N^{-1}\log^2 N)$ [16], giving a lower bound of approximately -1 for the slope of the best-fit straight line, which we hope the method should approach as closely as possible.

Table 1. Gradients of least-squares best-fit discrepancy lines for distributions on the plane

1D Sequence	*Rectangular*	*Circular*	*Triangular*
2D Random	-0.49	-0.5	-0.49
Sobol	-0.90	-0.70	-0.58
Niederreiter	-0.90	-0.69	-0.57
2D Jittered	-0.75	-0.74	-0.72
Hilbert Jittered	-0.73	-0.73	-0.72

Figure 5 shows scaling of rectangular discrepancy for the 2D random, Niederreiter, Sobol, 2D jittered and Hilbert-jittered distributions. It is clear that the slopes for the Niederreiter and Sobol distributions are the steepest, outperforming the other distributions. The Hilbert-Jittered and two-dimensional jittered distributions performed very similarly; not as well as the Niederreiter and Sobol sequences, but closer to them than to the random sequences. Figure 6 makes a similar comparison using a circular discrepancy measure. The Niederreiter, Sobol, Hilbert-jittered and 2D Jittered sequences perform similarly. Figure 7 makes a further comparison using the triangular discrepancy measure. In this case, the 2D jittered and Hilbert-jittered sequences outperform the Niederreiter and Sobol sequences, which perform closer to the random sequence. The random sequence is worst, as expected, throughout all three tests. The slopes of the best-fit straight lines in each case are listed in Table 1.

4.3 Discussion

When considering rectangular discrepancy, the Niederreiter and Sobol sequences perform better than the Hilbert space-filling curve. We also note that the distribution generated performs similar to the 2D jittering technique as might be expected, probably due to the grid-like structure of the curves. When considering circular and triangular discrepancy, however, the picture is quite different. The Niederreiter and Sobol sequences perform considerably worse in these tests, especially the triangle test where they perform only slightly better than the random sequence, suggesting little robustness. The two best performing distributions for these two tests were the jittered 2D and Hilbert-jittered curve distributions. It appears that the Sobol and Niederreiter methods are *too* specialised to rectangular discrepancy, and the jittered methods are better for all-round usage. The curve method is also consistently superior to using a random distribution.

In [1] the star discrepancy is briefly investigated. Expanding on this, our experiments strongly suggest that the space-filling curve approach gives distributions in the plane with a low-discrepancy behaviour. We also demonstrate the robustness of the technique when testing the star-discrepancy with different geometric subsets. While we do not have a rigourous theoretical proof for this, [1] suggests that the approach is very similar to regular stratified sampling, only with irregularly shaped strata and, hence, has the same discrepancy.

5 Experiments with Surfaces in 3D

In this section, we test point generation on 3D parametric surfaces. We first compare the point distributions generated on the unit sphere with certain reference point distributions listed in Sect. 2. To do so we employ the spherical discrepancy measure described using spherical triangles. We compare results using the Hilbert-jittered space-filling curve approach with the spherical-Hammersley and spherical-random 2D point distributions. We end this section by demonstrating sampling of various surfaces generated by our algorithm using uniform and non-uniform density. Finally we consider the utility of adaptive space-filling curve generation, and its limitations, and show how reparametrisation may also be required.

5.1 Numerical Experiments

To assess the quality of techniques for generating point distributions on surfaces or volumes in 3D, their properties with respect to measuring areas or volumes are often employed. Computing a measure on surfaces equivalent to the star discrepancy is non-trivial and alternative approaches such as techniques described in [33] may be more suitable to compare the quality of the distributions. Hence, in the following we only compare numerical results for point distributions on the unit sphere.

We generated points on the sphere using the spherical Hammersley distribution, a spherical random distribution and our method based on Steigleder and McCool's algorithm. We assessed how the spherical discrepancy varied with point set size, for point sets of size $N = 2^l$ and $N = 2^l + 2^{l-1}$ for $l = 0, \ldots, 19$. Sect. 3 gives three techniques for approximating the local surface area at a point on the Hilbert curve, based on triangles, rectangles and infinitesimal area elements using the first fundamental form of the surface: $dA = \sqrt{EG - F^2}\, du\, dv$. We show results of variation in discrepancy with number of points, as before, for the triangular and first fundamental form methods, denoted by Hilbert-jittered-triangle and Hilbert-jittered-fff, to demonstrate the difference in quality of distribution between the least and most exact area approximation methods.

When the adaptive curve generation algorithm is used on the sphere, there is little variation in curve recursion depth across the surface of the sphere, resulting in a nearly uniform curve recursion level. However, for the purpose of these tests, we disabled the adaptive sampling, leaving us with a completely uniform curve recursion so that results are consistent.

Figure 8 shows the variation of discrepancy with point set size on a logarithmic scale. The spherical Hammersley sequence performs the best by a small margin, very closely followed by the Hilbert-jittered-fff and Hilbert-jittered-triangle methods, while the random distribution performs considerably worse than the other methods.The gradients of the best fit lines of the various distributions are given in Table 2, where we can see that the Hammersley sequence performs in a similar way to the Niederreiter and Sobol sequences in the plane when measuring discrepancy using quarter-circles. The gradient for the Hilbert-jittered-fff

Table 2. Gradient of least squares fit line of distributions on the sphere

Sequence	*Spherical*
Random	-0.49
Hammersley	-0.75
Hilbert-jittered-fff	-0.73
Hilbert-jittered-triangle	-0.71

approach is similar to that of the spherical Hammersley sequence, closely followed by the Hilbert-jittered-triangle approach. The spherical random sequence, however, performs poorly.

From the results, we can see that the best fit lines for the spherical Hammersley point set have very slightly better slopes than our approach. The random distribution on the sphere, however, performed considerably worse than the other three approaches; similar to the 2D random distribution. Hence, using the Hilbert-jittered distribution and our surface equalisation technique, we can produce a low-discrepancy distribution on the unit sphere comparable to other low-discrepancy sequences that have been specifically designed to work with a sphere. Our algorithm, however, works for any parameterised surface and attempts to correct for severe stretching of the parameter domain. In addition it allows the user to adjust the point distributions with a density-function and maintains a consistent localised ordering of points.

5.2 Visual Demonstrations

We provide visual results to show the method being applied to sample various surfaces, both to produce a uniform density, and a controlled density of points. We show points distributed on the unit square, on a Monkey Saddle, an Eight Surface and a Whitney Umbrella.

Figure 9 shows three images of $3,000$ points distributed in the unit square. The image on the left shows a uniform unit density δ. The middle image shows $\delta(u, v) = u$ and the image on the right shows $\delta(u, v) = u + v$, where u and v are coordinates in the unit square.

Figures 10, 11 and 12 show three images of the Monkey Saddle, the Eight Surface and half of the Whitney Umbrella respectively using the Hilbert-jittered-fff approach. Each figure shows a parametric mesh visualisation of the surface, two images demonstrating point distributions generated using our approach, and the adaptive Hilbert curve mapped onto the surface. Each middle image shows points with uniform density whilst the right-hand images show a distribution density proportional to the Gaussian curvature of the respective surface. The Monkey Saddle is sampled with $3,000$ points, the Eight surface with $10,000$ points and the Whitney Umbrella with $6,000$ points.

Note that even on the images with uniform density, some areas appear darker than others. This is a visualisation problem, rather than a fault in the distributions, depending on the angle of viewing of the surface. Due to the severity of this problem on the Whitney Umbrella near the central singularity, only half of

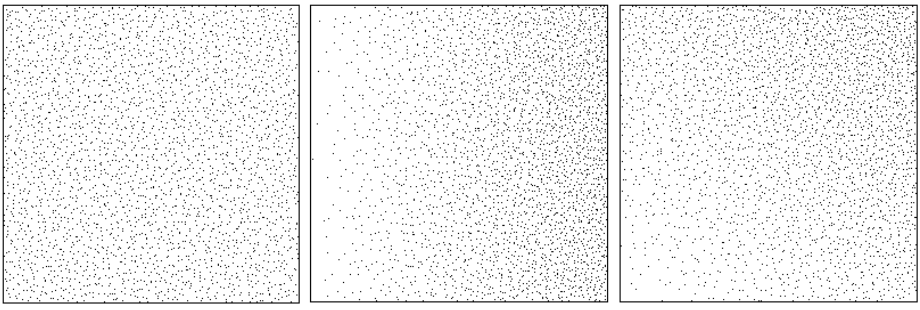


Fig. 9. Distribution in the unit square: uniform density; density $= u$; density $= u + v$

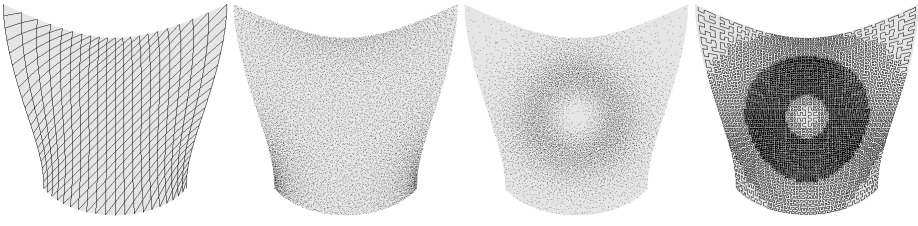


Fig. 10. The Monkey Saddle: parametric mesh; uniform density; curvature controlled density; adaptive Hilbert curve according to curvature

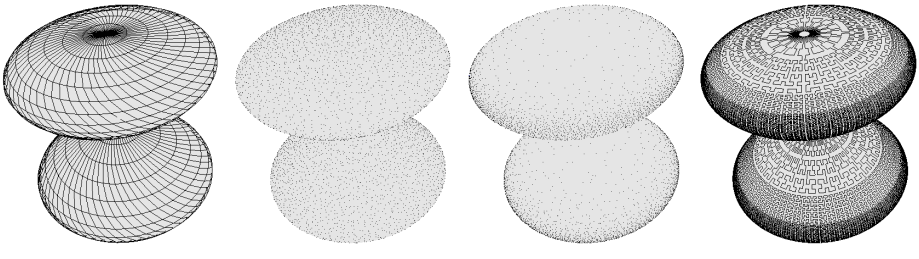


Fig. 11. The Eight Surface: parametric mesh; uniform density; curvature controlled density; adaptive Hilbert curve according to curvature

the surface has been drawn. Also note that the images of the adaptive curves were simplified in terms of recursive depth to make it easier to see the curve.

Our Java implementation on a 3Ghz Intel Pentium 4 with 1GB RAM takes about a second to generate the images shown above, with the curve generated adaptively according to surface curvature. For these examples, the recursive curve depth used was: $8 < k < 15$. In this context, where the surface can be parameterised straightforwardly, using the adaptive curve generation technique simply allows us to have far fewer curve vertices in order to sample at the required density. The extra computation involved in deciding whether to subdivide the curve at every branch in the tree, however, increases the algorithm complexity,

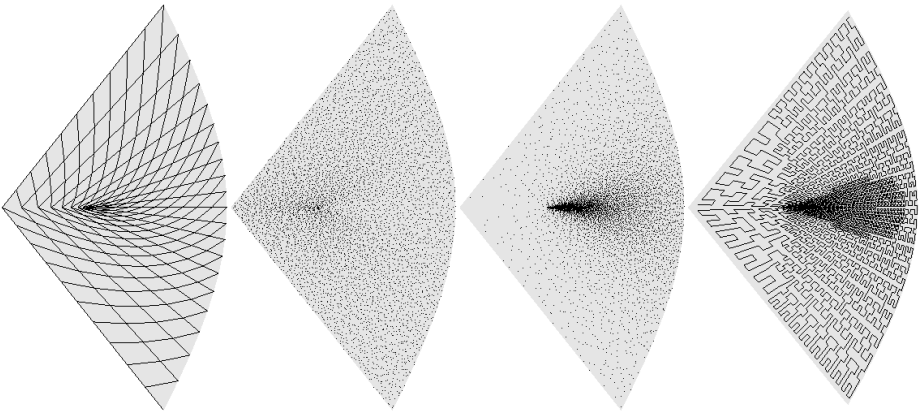


Fig. 12. Half of the Whitney Umbrella: parametric mesh; uniform density; curvature controlled density; adaptive Hilbert curve according to curvature

and thus runtime. However, if we have a non-uniform density (as in the examples shown in this section), runtime actually scales much better using the adaptive method as far less curve has to be generated in areas with a low density. The adaptive method also requires considerably less memory in this type of situation.

Showing that the approach given produces the same quality of distribution on any parametrically described surface is hard, but the visual results obtained above are plausible. Unlike earlier surface point distribution algorithms, this method can sample arbitrary surfaces, whilst providing direct sampling density control.

One limitation of our results is the reliance on the approximation of discrepancy as the only real quality measure of a distribution. This is accentuated by the number of test cases used: the plane and the sphere. However, as briefly discussed in Sect. 1, measuring the discrepancy on general surfaces robustly is a hard problem. We also demonstrate our results visually, but one can only garner so much from this. Essentially, it would be desirable to have another quantifiable metric by which we could compare results for various distributions tailored specifically for rendering and engineering applications. There are various options for application specific measures, and while none are perhaps quite as generically applicable as discrepancy, for the specific applications we intend to explore in the future, they might be of considerable use. Another possibility is Mitchell's [34] *Blue Noise* criterium: the sampling spectrum should be noisy and have few spikes, with a deficiency of low-frequency energy. The fulfilment of this criterium might be measured for the distributions.

5.3 Limitations of Adaptive Curve Generation

In this section, we discuss the limitations of adaptive curve generation technique. Figure 13 shows the Hilbert space-filling curve mapped onto the surface of a superellipsoid $x(u,v) = (cos^{1/3}u\ cos^{1/3}v)$, $y(u,v) = (cos^{1/3}u\ sin^{1/3}v)$, $z(u,v) =$

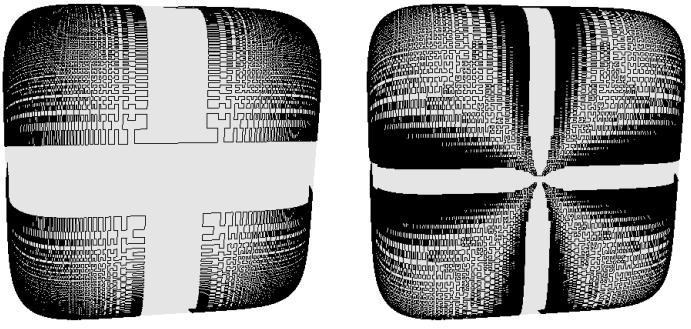


Fig. 13. A superellipsoid: without (262,000 vertices) and with (91,000 vertices) adaptive sampling

$sin^{1/3}u$) with and without adaptive curve generation. The non-adaptive curve consists of approximately $262,000$ vertices, whilst the adaptive curve consists of only $91,000$. It is clear that although there are still gaps between curve vertices when using the adaptive technique, they are far smaller than when using a uniformly generated curve.

This example also shows that there is a maximum density that can be reached for a given parametrisation due to machine precision limits. When generating the Hilbert curve we map 1D points to 2D points via bit indices using 2 algorithm. Thus, a 1D point represented by b bits, is mapped onto a 2D point where for each of its coordinates we only have $b/2$ bits. Thus, two distinct, computed points in the parameter domain differ in at last one coordinate by at least $e = 1/(2^{b/2}-1)$. Hence, to reach a minimum density of 1 point per surface area a, squares of the size e^2 in the parameter domain should be mapped to areas on the surface of at most size a. Or if we estimate the area in our algorithm via the first fundamental form, the norm of the first fundamental form should be smaller than a/e^2. Note that this ignores any numerical problems in evaluating the parametrisation.

To handle parameterisations which do not allow us to reach a certain density, an alternative to using multi-precision arithmetic is to reparameterise the surface locally or globally using polynomials. E.g. for the superquadric above, we can reparameterise it piecewise with a simple power law. In the interval $[0, \pi/2]$ we use $u = u'^3\pi/2$ to replace $\sin^{1/3}(u)$ with $\sin^{1/3}(u'^3\pi/2)$ for $u' \in [0, 1]$ and similar for the other intervals and for the cosine.

Figure 14 shows one corner of the superquadric shown in Fig. 13, with three curves mapped onto the surface. The image on the left shows a uniform Hilbert curve of depth $k = 7$. The middle image shows an adaptive Hilbert curve with $6 < k < 14$, and the image on the right shows a cubed reparameterisation of a uniform Hilbert curve of depth $k = 7$.

Generating the space-filling curve adaptively allows us to achieve localised higher densities on surfaces for a similar or lower memory requirement. The approach also allows us to correct for extreme parameterisations. However, Fig. 14 shows the limitations of this for an extreme parameterisation. To improve this

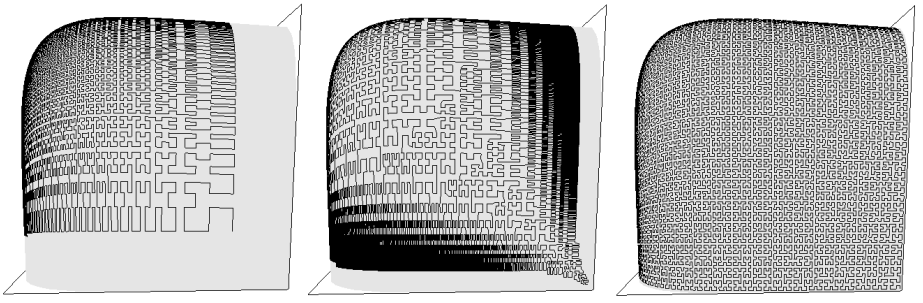


Fig. 14. One corner of a superellipsoid: normal distribution, uniform curve (left); normal distribution, adaptive curve (middle); cubed distribution, uniform curve (right)

situation for specific parameterisations, an alternative to using multi-precision arithmetic is to reparameterise the surface, either locally or globally. We have demonstrated that in the case of the superquadric, we can reparameterise it using a global power law (a simple polynomial fitted to part of the parameter to alter the distribution to better adapt to the sine or cosine terms). Future work will involve automating this reparameterisation approach, possibly on a per-cell basis during subdivision to take advantage of adaptive technique.

6 Conclusions

We have discussed the generation of density-controlled low-discrepancy distributions on arbitrary parametric surfaces employing a method due to Steigleder and McCool [1]. It allows us to map a 1D point sequence onto a surface via an adaptively generated space-filling curve. We have examined various choices for the 1D point distribution and space-filling curve. We found that both evenly-spaced and low-discrepancy 1D point sequences perform well when used with Hilbert and Hilbert II curves, where the Hilbert II curve is superior to the Hilbert curve. However, using a jittered evenly-spaced 1D point sequence in combination with adaptive generation of the Hilbert curve produces good point distributions in a short time for general surfaces and density functions.

Comparing the approach with other known low-discrepancy sequences in 2D and for special surfaces in 3D showed that specialised methods may perform better for some particular surface or discrepancy measure. However, the presented method is only slightly worse the specific distributions and outperforms these distributions if considered for general use. Overall this approach robustly generates low-discrepancy point distributions on surfaces, although reparameterisation may be needed to overcome limitations of the method arising from machine precision limits. We note that the use of the space-filling curve provides many inherent advantages, such as locality and connectivity.

One area of future work involves applying our method to a particular application and comparing how well it solves the problem to other approaches. Such problems include the computation of surface integrals of geometric models using

the Crofton formula [25], and other multivariate integration problems such as a Monte Carlo approach to computational fluid dynamics [35] and finite element analysis [8,9]. We may also test the approach with a ray-tracing or radiosity simulator to visually asses the distributions. These results would allow us to gauge the quality of various distributions in real-world problems.

One limitation of the method given is its reliance on a fully generated and stored adaptive Hilbert curve. We believe the method could be accelerated greatly if it were implemented to run directly on a GPU. [36] describes an algorithm to generate L-system subdivision curves using 32-bit precision pixel shaders. This technique maps well to programmable GPUs, and if further parallelised, could considerably increase the speed of space-filling curve generation.

References

1. Steigleder, M., McCool, M.: Generalized stratified sampling using the hilbert curve. Journal of Graphics Tools: JGT **8**(3) (2003) 41–47
2. Bern, M., Eppstein, D.: Mesh generation and optimal triangulation. In Hwang, F.K., Du, D.Z., eds.: Computing in Euclidean Geometry. World Scientific (1992)
3. Keller, A.: Instant radiosity. In: SIGGRAPH. (1997) 49–56
4. Cook, R.L.: Stochastic sampling in computer graphics. ACM Trans. Graphics **5**(1) (1986) 51–72
5. Rusinkiewicz, S., Levoy, M.: Qsplat: A multiresolution point rendering system for large meshes. In Akeley, K., ed.: Proc. ACM SIGGRAPH Comput. Graph. (2000) 343–352
6. Kobbelt, L., Botsch, M.: A survey of pointbased techniques in computer graphics. Computers and Graphics **28**(6) (2004) 801–814
7. Zwicker, M., Pauly, M., Knoll, O., Gross, M.: Pointshop 3D: An interactive system for point-based surface editing. In Hughes, J., ed.: Proc. ACM SIGGRAPH. (2002) 322–329
8. Zagajac, J.: A fast method for estimating discrete field values in early engineering design. In: Proc. 3rd ACM Symp. Solid Modeling and Applications. (1995) 420–430
9. Shapiro, V.A., Tsukanov, I.G.: Meshfree simulation of deforming domains. Computer-Aided Design **31**(7) (1999) 459–471
10. Floater, M.S., Reimers, M.: Meshless parameterization and surface reconstruction. Computer Aided Geometric Design **18**(2) (2001) 77–92
11. Dobkin, D.P., Eppstein, D.: Computing the discrepancy. In Proc. 9th ACM Symp. Computational Geometry (1993) 47–52
12. Zeremba, S.: The mathematical basis of monte carlo and quasi-monte carlo methods. SIAM Review **10**(3) (1968) 303–314
13. Gotsman, C., Lindenbaum, M.: On the metric properties of discrete space-filling curves. IEEE Trans. Image Processing **5**(5) (1996) 794–797
14. Niederreiter, H.: Random number generation and quasi-monte carlo methods. SIAM Review (1992)
15. Niederreiter, H.: Quasi-monte carlo methods and pseudo-random numbers. Bull. AMS **84** (1978) 957–1041
16. Davies, T.J.G., Martin, R.R., Bowyer, A.: Computing volume properties using low-discrepancy sequences. In: Geometric Modelling. (1999) 55–72

17. Keller, A.: The fast calculation of form factors using low discrepancy sequences. In Purgathofer, W., ed.: 12th Spring Conference on Computer Graphics, Comenius University, Bratislava, Slovakia (1996) 195–204
18. Bratley, P., Fox, B.L., Niederreiter, H.: Algorithm 738: Programs to generate niederreiter's low-discrepancy sequences. j-TOMS **20**(4) (1994) 494–495
19. Bratley, P., Fox, B.L.: Algorithm 659: Implementing sobol's quasirandom sequence generator. j-TOMS **14**(1) (1988) 88–100
20. Shirley, P.: Discrepancy as a quality measure for sample distributions. In: Eurographics '91. Elsevier Science Publishers, Amsterdam (1991) 183–94
21. Halton, J.H.: A retrospective and prospective survey of the monte carlo method. SIAM Review **12** (1970) 1–63
22. Kocis, L., Whiten, W.J.: Computational investigations of low-discrepancy sequences. ACM Trans. Math. Softw **23**(2) (1997) 266–294
23. Wong, T.T., Luk, W.S., Heng, P.A.: Sampling with hammersley and halton points. J. Graph. Tools **2**(2) (1997) 9–24
24. Secord, A., Heidrich, W., Streit, L.: Fast primitive distribution for illustration. In: Rendering Techniques. (2002) 215–226
25. Li, X., Wang, W., Martin, R.R., Bowyer, A.: Using low-discrepancy sequences and the crofton formula to compute surface areas of geometric models. Computer-Aided Design **35**(9) (2003) 771–782
26. Rovira, J., Wonka, P., Castro, F., Sbert, M.: Point sampling with uniformly distributed lines. Eurographics Symp. Point-Based Graphics (2005) 109–118
27. Hartinger, J., Kainhofer, R.: Non-uniform low-discrepancy sequence generation and integration of singular integrands. In H. Niederreiter, D.T., ed.: Proc.MC2QMC2004, Springer-Verlag, Berlin (2005)
28. Hlawka, E., Mück, R. In: A Transformation of Equidistributed Sequences. Academic Press, New York (1972) 371–388
29. Elber, G.: Free Form Surface Analysis using a Hybrid of Symbolic and Numeric Computation. PhD thesis, Dept. of Computer Science, University of Utah (1992)
30. Mandelbrot, B.: The fractal geometry of nature. Freeman, San Francisco (1982)
31. Butz, A.: Alternative algorithm for hilbert's space-filling curve. IEEE Trans. Computers **Short Notes** (1971) 424–426
32. Lawder, J.: Calculation of mappings between one and n-dimensional values using the hilbert space-filling curve. Technical Report JL1/00 (2000)
33. Cui, J., Freeden, W.: Equidistribution on the sphere. SIAM J. Sci. Comput. **18**(2) (1997) 595–609
34. Mitchell, D.R.: Generating antialiased images at low sampling densities. In Stone, M.C., ed.: SIGGRAPH '87 Conference Proceedings (Anaheim, CA, July 27–31, 1987), Computer Graphics, Volume 21, Number 4 (1987) 65–72
35. Alexander, F.J., Garcia, A.L.: The direct simulation monte carlo method. Comput. Phys. **11**(6) (1997) 588–593
36. Mech, R., Prusinkiewicz, P.: Generating subdivision curves with L-systems on a GPU. In: SIGGRAPH. (2003)

Verification of Engineering Models Based on Bipartite Graph Matching for Inspection Applications

F. Fishkel, A. Fischer, and S. Ar

Laboratory for CAD & Life Cycle Engineering
Faculty of Mechanical Engineering
Technion - Israel Institute of Technology
Haifa, Israel 32000
meranath@technion.ac.il

Abstract. Engineering Inspection (EI) requires automated verification of freeform parts. Currently, parts are verified by using alignment techniques on the inspected part and a CAD model. Applying the alignment on points or meshes is demanding and time-consuming. This work proposes a new alignment method to be applied on segments rather than on mesh elements. First, a discrete curvature analysis is applied on the meshes, and segments are extracted. Then, the inspected and CAD models are represented by segment graphs. Finally, a bipartite graph matching process is applied on the segment graphs, which are combined to be the two sides of a bipartite graph. As a result, a Combinatorial Matching Tree (CMT) is defined, and potential alignments are determined. The feasibility of the proposed segments alignment is demonstrated on real scanned engineering parts.

Keywords: Computational metrology, Mesh processing, Metrology, Reverse engineering, Bipartite graph matching.

1 Introduction

As 3D scanners become more accurate, industry increasingly requires automatic inspection of manufactured parts, where a scanned part is verified with respect to a CAD model. Part alignment is a crucial step in the verification and inspection of part geometry, for erroneous alignment can lead to non-accurate geometric comparison between parts.

In the literature and in practice, automatic alignment for inspection involves the following phases: data acquisition, range image registration, surface reconstruction, surface matching, and fine alignment. While some of the above phases may be omitted, they appear in most methods. These methods are applied on 3D surface point coordinates obtained by scanning devices. The problem of noisy data, common to all methods to varying degrees, is one of the main concerns of this research. Many studies have addressed the problem of registering and aligning clouds of points to a CAD reference model, bringing multiple scan range images to the same coordinate

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 485 – 499, 2006.
© Springer-Verlag Berlin Heidelberg 2006

system [4],[9],[11],[17]. Geometric methods solve the problem of range data alignment by matching points. Besl [5], proposed the Iterative Closest Point (ICP) algorithm, which requires a good initial "guess" of the alignment of the two given data sets in question. Hugli [10] presented a method to assess the range of good initial alignments. This method does not include fine alignment. Spanjaard [21] compared various registration methods and found they are sensitive to initial alignment conditions. Liu [17] presented a method for matching multiple views of range data for architecture applications, based on segmentation and graph matching techniques.

In this work, alignment is based on a segment matching technique. The following review surveys the literature relevant to the steps of the proposed algorithm.

Surface Reconstruction: The cloud of points obtained in the registration stage usually does not provide any information regarding the connectivity of the points. Point connectivity is a basic step needed for many topology-based algorithms, data structures and visualization techniques. The most common representation for the connectivity of scanned points is a triangular mesh, as can be seen in [1],[8],[24] and [26]. This representation is the one used by the method proposed here.

Segmentation: This work uses mesh segmentation to reduce complexity and take advantage of topological information. Rather than process large numbers of scanned vertices, the method takes advantage of the relatively small number of segments in a partitioned model. Segmentation quality is a parameter based on human visual recognition of the surface segmentation. Good functionality-based segmentation should discern between different components. It should separate regions on sharp edges or on areas where geometric properties change, such as curvature or normals. Most segmentation methods rely on mesh triangulation, and hence the quality of the results is highly dependent on triangulation quality, as described in [12],[18]. Most methods do not address the problem of segmenting a noisy mesh.

Curvature-Based Segmentation: Curvature-based segmentation is discussed extensively in literature as a topology-dependent method. The main stages of curvature-based segmentation include curvature analysis, segment finding based on vertex classification, and segment merging. Curvature-based methods for segmentation can be found in [14] and [27]. Nevertheless, these methods do not deal with scanning noise, and thus may yield bad results for noisy meshes. This research proposes a new method based on curvature analysis of noisy meshes and vertex classification using *curvature-mapping functions*, as described in Section 2.

Segment Matching: Graph matching belongs to the area of graph theory and has applications in many fields, among them computer vision, networking, data warehousing and biochemical applications. A comprehensive introduction to graph matching and a survey of many works in the area is found in [3]. Generic graph matching algorithms solve the maximal common sub-graph problem, as described in [15], [19], and [25]. Restricting the focus to special sub-classes of graphs, such as a segment graph, may result in more efficient matching procedures, as described in [6]. Classical methods for error tolerant graph matching algorithms can be found in [7] and [20]. In this work, the problem of matching segments of two segmented meshes is

posed as a graph-matching problem. Since segments of both models may differ in form, number and connectivity, the problem is a graph-matching optimization problem. This means that the nodes (segments) of two different graphs must be matched with some error tolerance.

Alignment and Inspection: Classic inspection techniques are described in [22], and the possible benefits of computational alignment for inspection are discussed in [2]. Most methods are based on two main stages for aligning the two models: the initial or coarse alignment, and the final or fine alignment.

This paper proposes a new method for coarse alignment of inspected parts. The paper is organized as follows. Section 2 describes the problem of verification using the proposed approach of graph matching. It then describes the concepts behind the algorithms of segmentation and graph matching, and their implementation. Section 3 presents results and performance analysis. Section 4 gives a summary of this work and offers conclusions.

2 General Approach and Implementation

This section describes the approach to coarse alignment in detail. Given two manifold triangular meshes, one of the inspected part and one of the CAD reference model, the goal is to align one to the other. Since input meshes are generally large—thousands of vertices per mesh—the problem of alignment should naturally be simplified. In the proposed method, initial alignment is carried out at the segment level to avoid the problem of alignment at the vertex level. The main steps in the proposed alignment process are as follows:

(a) Segment each of the meshes according to curvature levels.
(b) Create two segment graphs, one for each mesh, with nodes representing the segments.
(c) Determine similarity of nodes/segments from the two graphs, using a *segments similarity function* (SSF).
(d) Construct a bipartite graph, based on the segment graphs.
(e) Find and enumerate bipartite graph matchings using Combinatorial Matching Trees (CMTs).
(f) Register segments according to the bipartite graph matchings found.
(g) Align registered segments by minimizing the distance of matched segments.

Figures 1-5 provide examples of the different stages. Figure 1 shows the curvature map of a scanned part (Fig. 1.a) and the corresponding segmentation of the mesh (Fig. 1.b). Figure 2 demonstrates the affects of different curvature mapping techniques on the segmentation. Figure 3.a shows a bipartite graph, where each side is a segment graph. Figure 3.b shows the Combinatorial Matching Tree (CMT) created for the graph of Figure 3.a. Figure 4 shows one step in the creation of that CMT. Figure 5 shows the sub-trees extracted from the CMT of Figure 3.b. Each sub-tree is a segment matching combination. Following is a detailed description of the stages of the proposed method.

Mesh Segmentation

The meshes corresponding to each model are segmented to reduce the problem size. After segmentation, a relatively small number of segments represent the mesh of an engineering part, making the process of achieving the initial alignment more efficient. The proposed segmentation method uses curvature analysis. The idea is to classify the mesh vertices into segments according to their curvature, with respect to their neighbors' curvature. Once the curvature value of each vertex is known, the range of these values is classified into finite curvature levels. Then, neighbor vertices belonging to the same curvature level will be part of the same segment. Segments that have too few vertices are merged into neighboring segments.

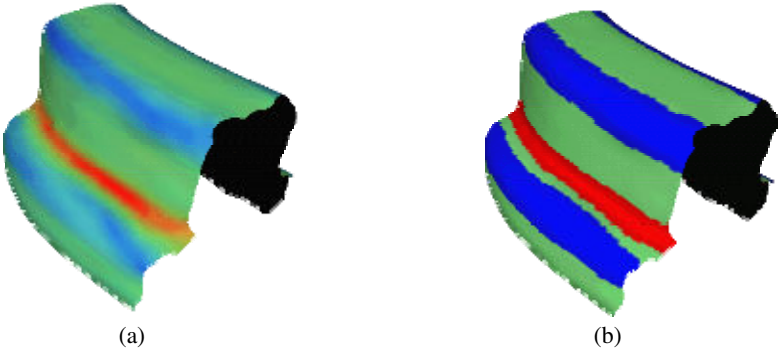
The result of the algorithm is an undirected segment graph for each mesh. Each node of the graph represents a segment composed of vertices of the mesh with similar curvature, while the edges represent neighbor relationships between segments. Following is the segmentation algorithm.

Segmentation Algorithm

Input: A triangular manifold mesh of a model
Output: A segment graph

(1) Compute curvature values at each vertex of the mesh.
(2) Map the curvature range into the finite range [0, 1].
(3) Classify the range [0,1] into curvature levels, and assign each mesh vertex a number indicating its curvature level.
(4) Create segments by grouping neighboring vertices having the same curvature level
(5) Create the segment graph.
(6) Merge segments with very few vertices or very small area to a neighbor segment, and update the segment graph.

Fig. 1 shows an example of a curvature map and its segmentation. Fig. 1.a shows the mesh after step (2) of the segmentation algorithm, where the curvature has been mapped into the [0,1] range. Fig. 1.b, shows the model after step (6), where the segmentation has been determined.



(a) (b)

Fig. 1. Scanned Mechanical Part: (a) Curvature Map and (b) Segmentation

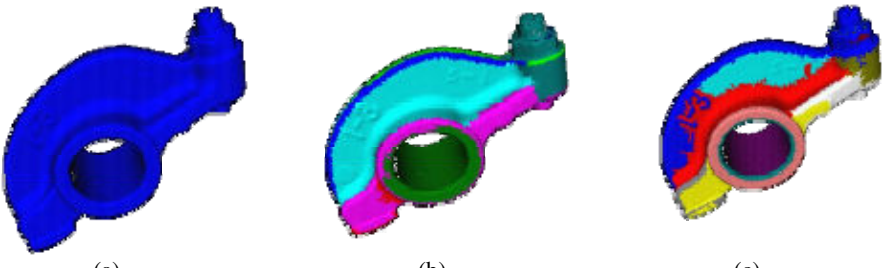
Curvature Calculation, Mapping, and Segmentation

The Taubin extended method, given in [16] was used to achieve good curvature estimations, even for noisy meshes. The original Taubin method [23] calculates the curvature of a vertex based on relations between it and its first ring of neighbors. The extended method [16] considers an extended neighborhood. For each vertex to be assigned a discrete curvature level, the curvature values domain is mapped into the range [0,1]. Then, this range is divided into equally sized sub-ranges. Each mesh vertex is assigned a curvature level, and the vertices are then grouped into segments based on these levels. The distribution of the vertices of the models into curvature levels depends on the specific mapping chosen. In this work, three different mapping methods were tried: Linear Mapping, Gaussian Mapping and a new method devised , the Gap Contraction Mapping.

A linear mapping is calculated using the finite extreme curvature values actually seen in a given model, with infinity mapped to 1; and minus infinity mapped to 0. The main problem observed with a linear mapping is with objects having both regions of high curvature, such as features, and regions of low curvature. This may result in an unbalanced segmentation, with too few segments. Another possibility is to expand the curvature domain by a Gaussian function. This mapping function contracts large curvature values and expands those near the average. This method often gives good results, but may fail in a variety of ways for models where the curvature distribution is not a Gaussian. The target in developing the Gap Contraction curvature mapping method was to spread the values as evenly as possible in the [0,1] range. The intuition being that this should lead to meaningful segmentation, as indeed is shown by the tests. Of the three methods tried, the gap contraction mapping consistently lead to the best results, as far as identifying details.

Fig. 2 shows the segmentation results for a scanned mechanical part, after curvature mapping by the three different methods. Segmentation of the linearly mapped part was unsuccessful – only one segment was identified. Gaussian mapping leads to better segmentation, still omitting many small details. As can be seen, the gap contraction mapping leads to the best results.

Regardless of the method, segmentation of a noisy mesh may result in many small segments, and thus in over-segmentation. This fragmentation may be diminished by merging very small segments into larger neighbor segments. Thus, the mesh consists of large segments that can be interpreted by functionality. This segmentation is used as a basis for graph matching, as described in the following sub-section.



(a) (b) (c)

Fig. 2. (a) Linear Mapping – 1 segment; (b) Gaussian Mapping – 12 segments; (c) Gap Contraction Mapping – 16 segments

2.1 Graph Matching

For each mesh model, a segment graph is defined. Each node of a segment graph is composed of mesh vertices and faces of a particular segment. Let us define the graphs G_S and G_T, as the segment graphs corresponding respectively to the meshes, M_S and M_T, where M_S is the CAD mesh - the *source*, and M_T is the inspected mesh -the *target*.

First, a *segment similarity-function* is defined, which makes it possible to quantify the similarity of two segments, one from G_S and one from G_T. Two segments are *potentially matching segments* if their similarity is higher than a predefined threshold. G_S and G_T together represent an undirected weighted *bipartite graph,* or *bigraph*, $G = (V(G_S) \cup V(G_T), E)$ such that $V(G_S)$ and $V(G_T)$ are the two *sides* of the bigraph. The edges of the *bigraph, E,* connect pairs of segments, one from each segment graph. A bigraph edge, e_{ij}, has a weight w_{ij}, which is a real number, reflecting the segment similarity between the two segments connected by e_{ij}. In the bigraph, we do not include all possible edges, only those corresponding to *potentially matching segments*. Fig. 3.a is an example of a bipartite graph consisting of two graphs G_S and G_T. Without loss of generality, in this example, G_S and G_T are identical. The red edges connect matching segments.

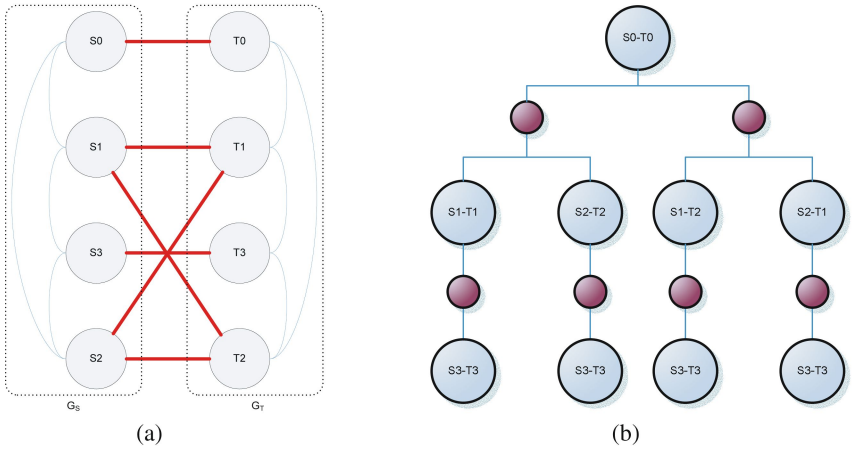


Fig. 3. Bipartite Graph, and (b) Combinatorial Matching Tree (CMT)

A *matching* in a graph $G = (V,E)$ is a subset of the edges $M \subset E$ such that no two edges in M share a common vertex. A matching in a bigraph, a *bipartite matching*, assigns vertices of one side to those of the other. In the context of this work, a bipartite matching corresponds to a possible pre-alignment of two segmented meshes. The process of matching two segment graphs that represent similar models may lead to the discovery of more than one matching. This is due to factors such as sub-pattern repetition, symmetry, noise and production defects. In practice, only a few of these matchings may be valid and useful for the pre-alignment stage. Note that not only perfect matchings may be valid. Each matching found is assigned a *matching value*, as described later in this section. The geometric validity of the matchings with a high

matching value can only be determined by geometric registration, a process performed later.

For two segments to be compared, the similarity between them must be quantified. Many isotropic parameters characterize a segment, such as average curvature, surface area or neighboring segment similarity. Euclidian distances between compared segments may also be useful. We define the distance between two segments *s* and *t*, from the two graphs respectively, as a function of similarity between segments – *f(s,t)* - with values in the [0,1] range. This function depends on the curvature values of each segment and on their area and relative distance from their neighbor segments. This is the *segment similarity function*.

Next, a hierarchical data structure called a *combinatorial matching tree* (CMT) is defined. Such trees enumerate all candidate-matchings serving as a basis for pre-alignment configurations. Once all potential matchings (candidate pre-alignments) are extracted from the trees, those with the best matching value may be checked. The process is then as follows. Create Combinatorial Matching Trees (CMTs), extract potential segment matchings from the trees, and find the best matchings from the trees. This method of constructing CMTs and extracting matchings from them is an exhaustive search method. In the future, the intent is to experiment with efficient network flow algorithms for the matching problem, as described in chapter 7 of [13].

Creating a Combinatorial Matching Tree (CMT)

Combinatorial Matching Trees are created to enumerate all relevant, partial bipartite matchings. Two kinds of nodes compose a Combinatorial Matching Tree: matching nodes and combination branching nodes (CBNs). Matching nodes are linked to two matching segments, one from each segment graph, and the similarity distance between them. A CBN acts like a grouping node to separate the possible partial matchings (matching combinations) at each level of the tree. Fig. 3.b shows one Combinatorial Matching Tree corresponding to the bipartite graph shown in Fig. 3.a, where CBNs are the smaller nodes. This example shows the root node matching S0 from G_S to T0 from G_T. Under it are two CBNs, separating the two possible matching combinations in this case. Two matching segments may be similar but still not a geometrically correct match. For instance, similar segments may appear at various regions of an object with a repetitive pattern. Therefore, the algorithm searches for all the matchings and builds several CMTs, choosing segment matches with high matching similarity as roots of the tree. Each CMT is created by a recursive algorithm. The algorithm traverses the two segment graphs in parallel, as the two sides of a bipartite graph. Each pair of matching segments is added to the tree as a node. Following is the algorithm for creating Combinatorial Matching Trees.

Combinatorial Matching Tree (CMT) Algorithm

Input: Two segment graphs, corresponding to a source model and a target model.
Output: CMTs enumerating all potential matchings

(1) Initialize: find the best matching segment pairs, and create a tree node for each. These will be tree roots.
(2) For each node from step (1), iterate over steps (3)-(7).
(3) Mark the two segments of the tree node, and call this tree node *the current node*.

(4) Find all combinations of pairs of unmarked matching segments neighboring the current node's segments. If there are none, go to step (7).
(5) Mark all segments found in (4).
  (a) Create a branching node for each combination of matching pairs found in step (4).
  (b) Link all branching nodes created in (5.a) as children of the current node.
  (c) Create the segment match nodes for each pair found in (4), and link them as children of the branching node.
(6) For each match node created in step (5.c) recourse to step (3).
(7) Unmark all neighbors of the current node's segments.

Fig. 4 illustrates one iteration of the tree-creation algorithm. Segments S0 and T0 were chosen as the root of the tree due to their high similarity. In this example, neighbor segments S1 and S2 match neighbor segments T1 and T2. There are two possible combinations among these matching segments. Combination S1-T1, S2-T2 and combination S1-T2, S2-T1 lead to the creation of two combination branching nodes under the root of the tree, acting as separators between the combinations.

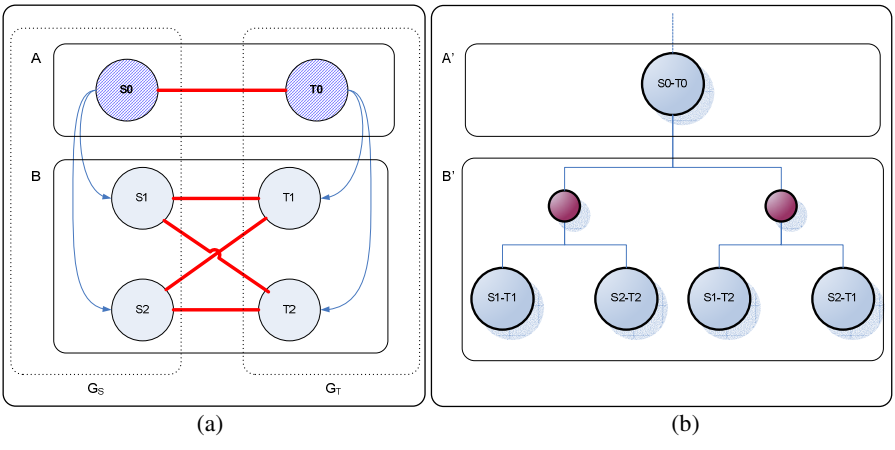


Fig. 4. Combinatorial Matching Tree Creation, One iteration; (a) Partial Bigraph, (b) Top of CMT

Once the CMTs are constructed, the matching solutions are defined by extracting CMT sub-trees.

Extracting Best Matchings from a CMT

A CMT represents the potential combinations of matching segments. Each sub-tree of the CMT corresponds to a partial matching of two segment graphs. This section presents a *tree-walking algorithm* that extracts the potential matchings from a CMT. To find the bipartite matchings, sub-trees are first extracted from a CMT by traversing it, starting at the root and iteratively going over all the children of each node from left to right.

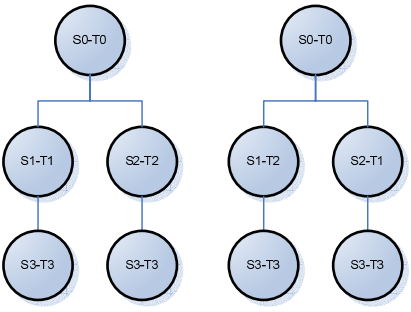


Fig. 5. Sub-Trees Extracted from CMT

The matching algorithms use an experimentally determined threshold. Therefore, many of the matchings produced can correspond to geometrically invalid solutions and are an *invalid matching*. To determine the validity of a matching, geometric registration of the models is required. However, this is computationally expensive, so the goal is to reduce the number of solutions that need be checked geometrically.

The *matching value* of a matching is defined as an algebraic combination of the following parameters of a matching: number of segment matches, total sum of matching segment distances, and total sum of the area of the matching segments of both models. Several optimization functions were tested, and the average matching distance was chosen, that is, the total sum of matching segment distances divided by the number of segment matches. This function gave the best results. The following section details the results.

3 Experimental Results and Performance Analysis

In the evaluation of segment-matching algorithm performance, the following aspects were checked: validity of the solutions extracted from the CMT, calculation of graph complexity, and testing the threshold impact on the number of matched segments. For testing, two types of input pairs of meshes were chosen: identical and slightly unequal at different levels. Identical meshes serve to analyze performance under ideal conditions. Applying the matching algorithm on non-identical but similar meshes serves to test the performance in practical matching situations. All tests were performed on a PC Pentium 4, with 2.0 GHz and 256MB of memory.

In inspection, the mesh of scanned points is usually different from its CAD model. However, in order to discover the defects and production errors in the inspected mesh, the two models must be aligned. The alignment is useful if it is based on matching segments that indeed correspond.

In this work, various meshes were tested, some resulting from real model scans and some synthesized, where defects and errors were introduced into the inspected mesh. These defects were intended to simulate real production defects: small and large area defects, and even pattern discontinuities. Results from the experiments highlight several features of the proposed method:

- For identical pairs of meshes, at least one matching solution was found, regardless of the distance threshold parameter.
- Low threshold values resulted in low time complexity, but at the price of very few segment matches being found.
- Higher threshold values reveal many possible matching solutions, at the expense of time and memory complexity. Nevertheless, running times remain generally low, as the number of segments is often small, due to the segment merging stage of the segmentation algorithm.
- The parameter with the greatest impact on running time was the average number of neighbors of each segment. This number is generally small, and may be reduced by increasing the number of curvature levels of the segmentation.

Example 1. Matching a synthetic surface
This example demonstrates the impact of the distance coefficients on the matching results. A synthetic trigonometric function was used as a model. Four defects were introduced into the verified model, see Fig. 6.

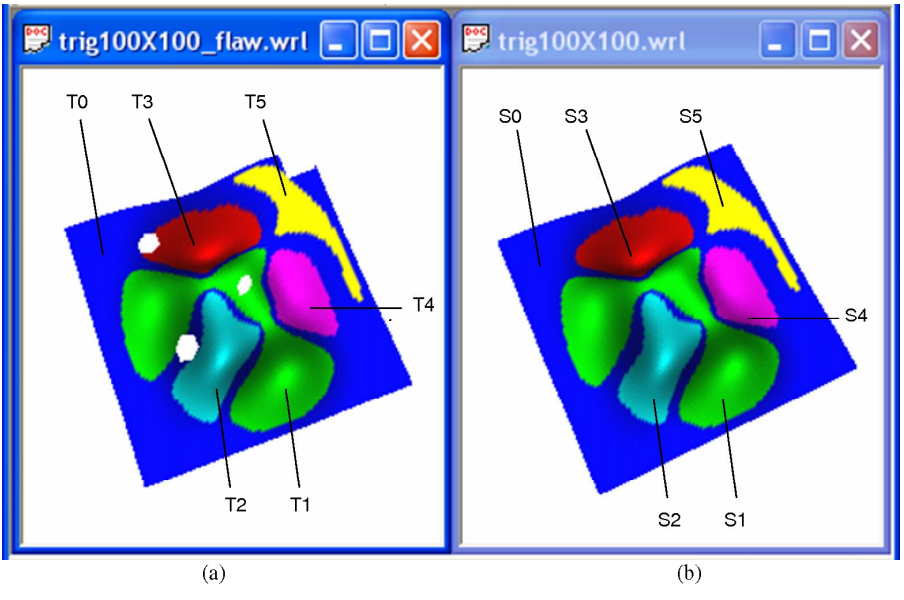


Fig. 6. A Synthesized Part - (a) Inspected Model and (b) CAD Model

Table 1 presents the results for different combinations of coefficients within a threshold of 5%. Note that the higher the area coefficient is, the higher the similarity. Nevertheless, incorrect matches are obtained when the neighborhood coefficient is set to zero. Fig. 7 shows the Matching Graph for the second case in the table. All resulting graphs look the same, but differ in the distance values.

Table 1. Example 1 – Parameters and Results

Source Mesh	#vertices=10201 - #faces=20000				
Target Mesh	#vertices=9977 - #faces=19491				
Threshold	5%				
Neighborhood Distance Coefficient	1.0	0.7	0.5	0.3	0.0
Area Distance Coefficient	0.0	0.3	0.5	0.7	1.0
Minimal Segment Similarity	95.75%	96.45 %	96.93%	97.40%	98.10%
Average Segment Similarity	98.57%	98.87%	99.07%	99.27%	99.57%
Total Number of Matched segments	6 / 6	6 / 6	6 / 6	6 / 6	6 / 6
Correct Solutions / Total Solutions	1 / 1	1 / 1	1 / 1	1 / 1	1 / 2
Running time	2 seconds	1 second	1 second	1 second	1 second

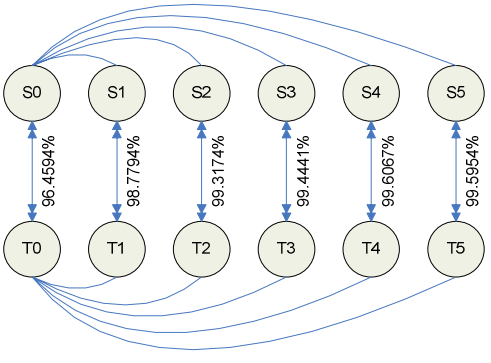


Fig. 7. Example 1 – Matching Graph Solution for Threshold = 5%, a1=0.7,a2=0.3

Example 2. Curvature Mapping and Matching Identical Models

Fig. 8 shows two identical meshes, after having run the segment-matching algorithm. Identical color indicates matched segments. This is the same model, with 30,132 vertices and 60,264 faces, used to demonstrate the curvature mapping and segmentation in the previous section. However, the parameter tuning for this experiment was different. The algorithm found 6 segments, and matched them all correctly. These results are representative of all tests on identical meshes: for pairs of identical models, at least one matching solution was found, regardless of the value of the distance threshold parameter. Thus "proving" that the matching algorithm works, finding meaningful solutions.

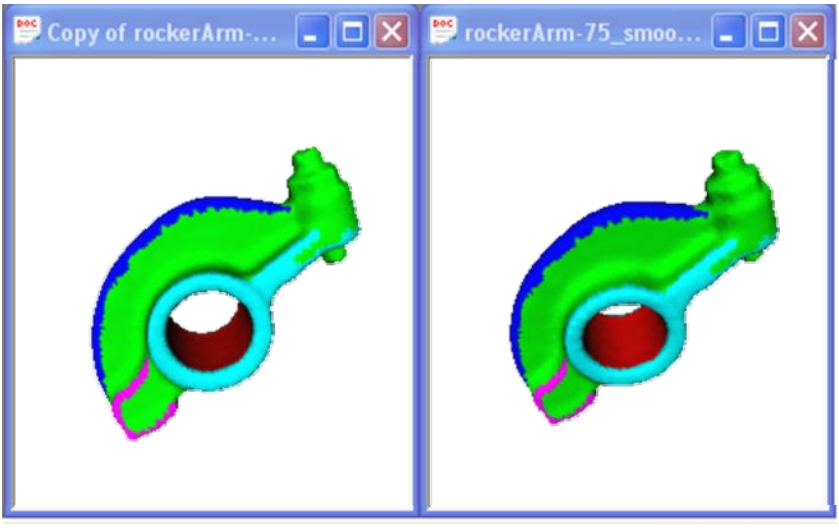


Fig. 8. Example 2 – Identical scanned noisy models, after running Graph Matching Algorithm

Example 3. Matching a mechanical part

Fig. 9 shows the input meshes for this example. The picture on the right shows the reference mesh, while the left picture presents a copy of it with a synthetic defect at the top. The quality of the solutions depends on the threshold chosen for the distance similarity among segments. Table 2 presents the parameters and results after running the segmentation and graph matching algorithms.

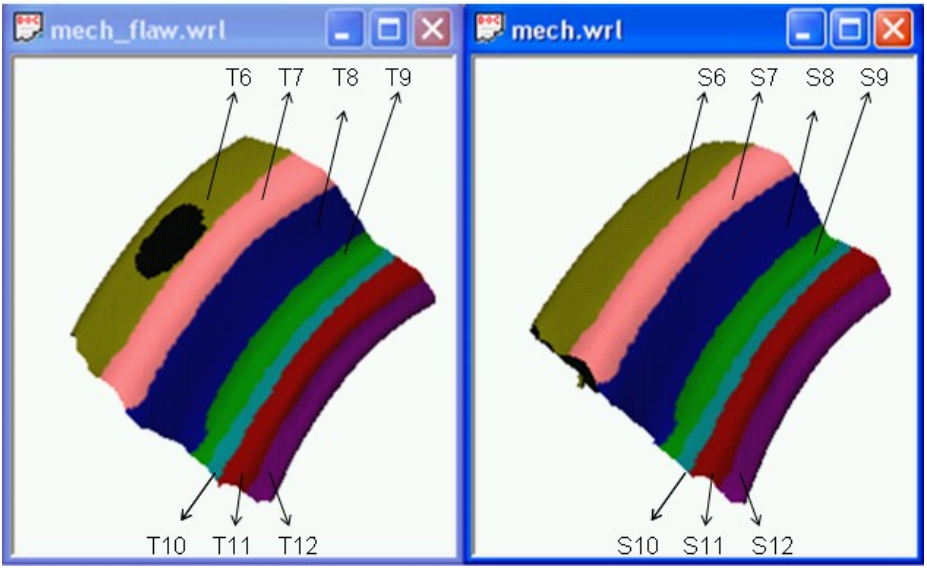


Fig. 9. Example 3 - Visualization of Matching Results

Table 2. Example 3: Mechanical part – Parameters and Results

Source Mesh	#vertices = 15204 - #faces = 29846	
Target Mesh	#vertices = 14821 - #faces = 29004	
Neighborhood Distance Coefficient	0.7	
Area Distance Coefficient	0.3	
Threshold	0.1 %	1 %
Minimal Segment Similarity	99.9726 %	99.0353 %
Total Number of Matched segments	12 / 13	13 / 13
Number of Sub Solutions	2	1
Running time	4 seconds	3 seconds

With a higher threshold value, the defective segment (T6) is included in the solution, see Fig. 10. With a smaller threshold, distant segments are not matched. This example generates two unlinked sub-solutions, since the missing match S6-T6 breaks the chain-like graph, see Fig. 11.

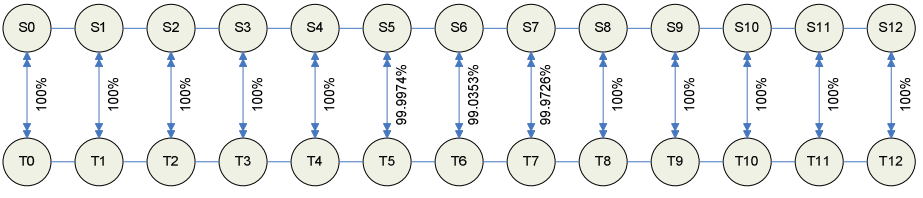


Fig. 10. Example 3 – Graph Matching Solution for Threshold = 1%

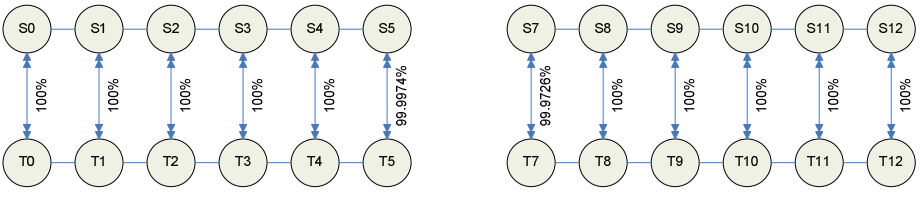


Fig. 11. Example 3 – Graph Matching Sub-Solutions for Threshold = 0.1%

4 Summary and Conclusions

This paper has proposed a new alignment method for verification as a base for engineering inspection. The novelty of this method is that it is applied to segments rather than mesh elements, and uses graph matching to align segments. The process starts with applying discrete curvature analysis to the two meshes of the inspected and the reference CAD models. Then, according to this curvature, the algorithm extracts segments of each model. Using this segmentation, a representation by a segment

graph is created for each of the models. Finally, the segment graphs are combined into one bipartite graph, and the proposed bipartite graph-matching algorithm is applied to this graph. Then a combinatorial matching tree (CMT) is defined. This is a data structure enumerating all relevant segment-matching sets. Each of these matching sets determines a coarse alignment of the two meshes. Finally, the feasibility of the proposed segment alignment method is demonstrated on synthesized models and on real scanned engineering parts.

The proposed method has several advantages. The segmentation method and the use of thresholds on the number of vertices in a segment together help reduce the effects of noise in the data and avoid very small segments and over-segmentation. This yields high quality segmentation, consistent with the geometric or functional meaning of segments. Therefore, matching of the segments at the graph matching stage is good, for segments that should coincide are matched. Experimenting with the proposed method has shown that the coarse alignment results are satisfactory and were obtained efficiently and in real time when compared to other methods. The alignment process is improved by creating a family of solutions and choosing the best among those. In the future, the algorithms should be tuned to find optimal solutions.

References

[1] Amenta, N., Bern, M., Kamvysselis, M.: A New Voronoi-Based Surface Reconstruction Algorithm. Siggraph (1998) 415-421

[2] Barhak, J.: Utilizing Computing Power to Simplify the Inspection Process of Complex Shapes. Israel-Italy Bi-National Conference, Tel Aviv, Israel (2004)

[3] Bengoetxea, E.: Inexact Graph Matching Using Estimation of Distribution Algorithms. Th`ese de Doctorat Sp´ecialit´e Signal et Images, Ecole Nationale Sup´erieure des T´el´ecommunications, Paris, France (2002)

[4] Bergevin, R., Laurendeau, D, Poussart, D.: Registering Range Views of Multipart Objects. Computer Vision and Image Understanding, Vol. 61, No 1, Januart, (1995) 1-16

[5] Besl, P.J., McKay, N.D.: A Method for Registration of 3D Shapes. IEEE Transactions on pattern Analysis and Machine Intelligence, 14(2) (1992) 239-256

[6] Bunke, H.: Graph Matching: Theoretical Foundations, Algorithms, and Applications. International Conference on Vision Interface (2000) 82–88

[7] Eshera, M.A., Fu, K.S.: A Graph Distance Measure for Image Analysis. IEEE Trans. SMC 14 (1984) 398-408

[8] Gopi, M., Krishnan, S.: Fast and Efficient Projection Based Approach for Surface Reconstruction. 15th Brazilian Symposium on Computer Graphics and Image Processing, SIBGRAPI (2002)

[9] Higuchi, K., Delingette ,H., Hebert, M., Ikeuchi, K.: Merging Multiple Views Using a Spherical Representation. In Proc. IEEE 2nd CAD-Based Vision Workshop, Champion, PA, (1994) 124-131

[10] Hugli, H. Schutz, C.,: Geometric Matching of 3D Objects: Assessing the Range of Successful Initial Configurations. 3-D Digital Imaging and Modeling (1997)

[11] Johnson, A., Hebert, M.: Surface Registration by Matching Oriented Points. Proc. Int. Conf. on Recent Adv. in 3-D Digital Imaging and Modeling (1997) 121-128

[12] Katz, S., Tal, A.: Hierarchical Mesh Decomposition using Fuzzy Clustering and Cuts. In SIGGRAPH, ACM Transactions on Graphics, Volume 22 (2003) 954-961

[13] Kleinberg, J., Tardos, E.: Algorithm Design. Addison-Wesley, (2006).
[14] Lavoué, G., Dupont, F., Baskurt, A.: Constant Curvature Region Decomposition of 3D-Meshes by a Mixed Approach Vertex-Triangle. Journal of WSCG Vol.12, No.1-3, ISSN 1213-, Plzen, Czech Republic (2004)
[15] Levi, G.: A Note on the Derivation of Maximal Common Subgraphs of Two Directed or Undirected Graphs. Calcolo, Vol. 9 (1972) 341-354
[16] Lipschitz, B.: Discrete Curvature Estimation of Scanned Noisy Objects for Verification of Scanned Engineering Parts with CAD Models. Technion – Israel Institute of Technology, Research Thesis (2002)
[17] Liu, R., Hirzinger, G.: Marker-free Automatic Matching of Range Data. In: R. Reulke and U. Knauer (eds), Panoramic Photogrammetry Workshop, Proceedings of the ISPRS working group V/5, Berlin (2005)
[18] Mangan, A., Whitaker, R.: Partitioning 3D Surface Meshes using Watershed Segmentation. IEEE Transactions on Visualization and Computer Graphics, 5, 4 (1999) 308-321
[19] McGregor, J.: Backtrack Search Algorithms and the Maximal Common Subgraph Problem. Software-Practice and Experience, Vol. 12 (1982) 23-34
[20] Sanfeliu, A., Fu, K.S.: A Distance Measure Between Attributed Relational Graphs for PatternRecognition. IEEE Trans. SMC, Vol. 13 (1983) 353-363
[21] Spanjaard, S., Vergeest, J.S.M.: Comparing Different Fitting Strategies for Matching Two 3D Point Sets using a Multivariable Minimizer. Proceedings of Computers and Information in Engineering Conference, Pittsburgh, USA, ASME, New York, DETC'01/CIE-21242 (2001)
[22] Srinivasan, V.: Elements of Computational Metrology. Proceedings of the DIMACS Workshop on Computer-Aided Design and Manufacturing, American Mathematical Society (2005)
[23] Taubin, G.: Estimating the Tensor of Curvature of a Surface from a Polyhedral Approximation. Proc. of fifth international conference on computer vision (1995) 902–907
[24] Turk, G., Levoy, M.: Zippered Polygon Meshes from Range Images. Computer Graphics Proceedings, Annual Conference Series, Siggraph, (1994) 311-318
[25] Ullman, J.R.: An Algorithm for Subgraph Isomorphism. Journal of the Association for Computing Machinery, Vol. 23, No. 1 (1976) 31-42
[26] Várady, T., Martin, R. R., Cox, J.: Reverse Engineering of Geometric Models. An Introduction. Computer-Aided Design,Vol.29,No.4 (1997) 255-268
[27] Zhang, Y., Paik, J., Koschan A., Abidi, M.A.: A Simple and Efficient Algorithm for Part Decomposition of 3D Triangulated Models Based on Curvature Analysis. Proc. Int. Conf. Image Processing, Vol. III, Rochester, NY (2002) 273-276

A Step Towards Automated Design of Side Actions in Injection Molding of Complex Parts

Ashis Gopal Banerjee and Satyandra K. Gupta

Department of Mechanical Engineering and The Institute for Systems Research
University of Maryland
College Park, MD 20742, USA
ashis@umd.edu
skgupta@eng.umd.edu

Abstract. Side actions contribute to mold cost by resulting in an additional manufacturing and assembly cost as well as by increasing the molding cycle time. Therefore, generating shapes of side actions requires solving a complex geometric optimization problem. Different objective functions may be needed depending upon different molding scenarios (e.g., prototyping versus large production runs). Manually designing side actions is a challenging task and requires considerable expertise. Automated design of side actions will significantly reduce mold design lead times. This paper describes algorithms for generating shapes of side actions to minimize a customizable molding cost function.

1 Introduction

Injection molding is one of the most widely used plastic manufacturing processes nowadays [1]. It is a near net-shape manufacturing process that can produce parts having good surface quality and accuracy. Moreover, this process is suitable for mass volume production due to fast cycle time. Often complex injection molded parts include undercuts, patches on the part boundaries that are not accessible along the main mold opening directions. Undercuts are molded by incorporating side actions in the molds. Side actions are secondary mold pieces (core and cavity form the two main mold pieces) that are removed from the part using translation directions that are different from the main mold opening direction (also referred as parting direction). Use of side actions is illustrated in Fig. 1.

Significant progress has been made in the field of automated mold design. Chen et al [2] linked demoldability (i.e. the problem of ejecting all the mold pieces from the mold assembly such that they do not collide with each other and with the molded part) to a problem of partial and complete visibility as well as global and local interference. If a surface is not completely visible, then either it is blocked locally by parts of the same surface, or globally by other surfaces. They reduced global accessibility of undercuts (or pockets that are the non-convex regions of a part) to a local problem and used visibility maps. Hui [3] classified undercuts into external and internal, which require side and split cores for removing them from the main mold pieces respectively. He primarily focused on obtaining optimal parting directions from a set

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 500 – 513, 2006.
© Springer-Verlag Berlin Heidelberg 2006

of heuristically generated candidate directions by minimizing the number of such side and split cores. Since then, Yin et al [4] have tried to recognize undercut features for near net-shape manufactured parts by using local freedom of sub-assemblies in directed blocking graphs. Another way of recognizing and especially categorizing undercuts was put forward by Fu et al. [1]. A ray tracing/intersection method for finding out release directions in injection-molded components was suggested by Lu et al. [5]. Chen and Rosen [6] suggested partitioning part faces into regions, consisting of generalized pockets and convex faces, each of which can be formed by a single mold piece.

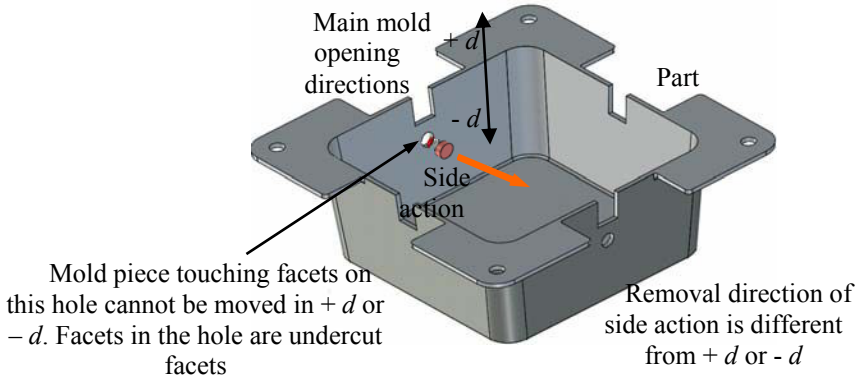


Fig. 1. Side action to remove the undercut facets in a direction different from the main mold opening direction

Dhaliwal et al. [7] described exact algorithms for computing global accessibility cones for each face of a polyhedral object. Using these, Priyadarshi and Gupta [8] developed algorithms to design multi-piece molds. Ahn et al. [9] have given theoretically sound algorithms to test whether a part is castable along a given direction in $O(n \log n)$ time and compute all such possible directions in $O(n^4)$ time. Building on this, Elber et al. [10] have developed an algorithm based on aspect graphs to solve the two-piece mold separability problem for general free-form shapes, represented by NURBS surfaces. Recently, researchers [11, 12] have presented new programmable graphics hardware accelerated algorithms to test the moldability of parts and help in redesigning them by identifying and graphically displaying all the mold regions (including undercuts) and insufficient draft angles.

Due to space restrictions, it is difficult to present a detailed review of all the other work in the mold design area. So, we will identify some of the important work in this field. Representative work in parting surface and parting line design includes [13] and [14] respectively. Representative work in the area of undercut feature recognition was done by [15], whereas same in the area of side core design includes [16].

Thus, we can summarize that the problem of finding optimum parting direction has been studied extensively and is very useful in casting applications. However, in case of injection molded plastic parts, most designers develop part designs with a mold

opening direction in mind. This is because mold opening direction influences all aspects of part design. Typically, plastic parts are either flat or hollow-box shaped and they need to have relatively thin sections. These shapes have an implied mold opening direction. Moreover, they cannot have vertical walls; and some amount of taper has to be imparted to them in order to remove them from the mold assembly. Therefore the main tasks in injection mold design automation are (1) determination of the main parting line and parting surface, (2) recognition of undercuts, and (3) design of side actions. Existing methods for the first two tasks provide satisfactory solutions.

However, existing techniques for side action design do not work satisfactorily if a complex undercut region (1) needs to be partitioned to generate side actions, or (2) has finite accessibility. Fig. 2 shows two parts for which generating side actions is challenging due to these two reasons. In this paper, we present a new algorithm to handle these types of cases. We currently only consider those types of side actions that are retracted from the undercut in a direction perpendicular to the main mold opening direction. In fact, majority of side actions used in industrial parts meet this restriction. Typically, a single side action produces a connected undercut region. Therefore, we treat each connected undercut independently and generate appropriate number of side actions for molding it.

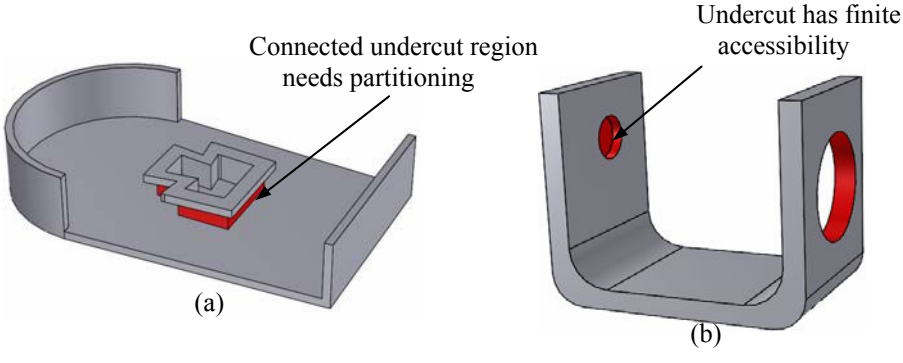


Fig. 2. Parts that pose challenges for existing side action design algorithms

2 Problem Formulation and Theoretical Foundations

Retraction is defined by a point (x, y) in $\mathbb{R}^2$. We are interested in two types of retractions namely, core-retractions and cavity-retractions.

Retraction length is defined as the length of the position vector of the 2D point that defines the corresponding retraction.

A given retraction (x, y) is a feasible core retraction for an undercut facet f, if sweeping f first by a translation vector $\vec{t_1} = x\hat{i} + y\hat{j}$ and then by another one $\vec{t_2} = l_a\hat{k}$, (where l_a is a large positive number) does not result in any intersection with the part. Fig. 3 shows examples of a feasible and an infeasible core retraction.

A given retraction (x, y) is a feasible cavity retraction for an undercut facet f, if sweeping f first by a translation vector $\vec{t}_1 = x\hat{i} + y\hat{j}$ and then by another one $\vec{t}_2 = -l_a\hat{k}$, does not result in any intersection with the part.

Feasible core retraction space (illustrated in Fig. 4) for an undercut facet f is the set of all feasible core retractions for that particular facet. *Feasible cavity retraction space* is defined in a similar manner.

Side Action Set Generation Problem: Given a polyhedral object, its *mold enclosure* (bounding box enclosing the part), main mold opening directions and undercut facets, determine the side action(s). Side actions require two translational motions for complete disengagement from the part. The first one is assumed to lie in a plane orthogonal to the main mold opening directions and the second one coincides with the either of the two main mold opening directions.

Input

- A faceted (triangulated) solid geometric model of a polyhedral connected part P oriented such that the main mold opening directions are along $+z$ and $-z$ respectively.
- Mold enclosure.
- n facets marked or labeled as undercuts (we use techniques described in [12]).

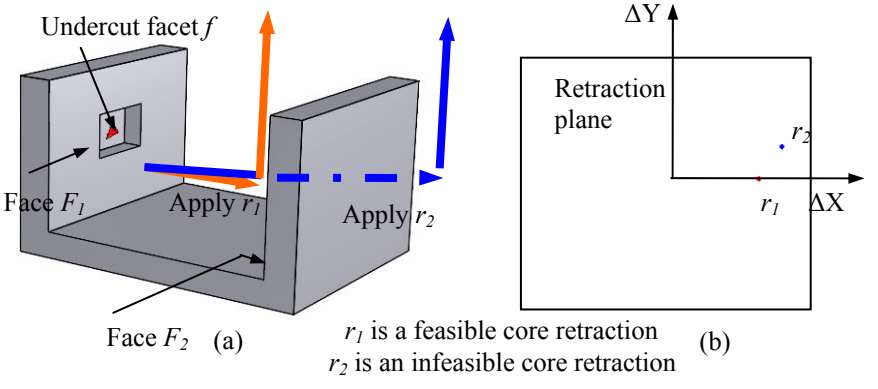


Fig. 3. Feasible and infeasible core retractions

Output: A set of side actions, such that each set is defined by a 4-tuple {s, F, (x, y), T}. The first entity s represents a solid body corresponding to the side action. The second entity F represents the set of undercut facets that will be associated with this side action. The third entity (x, y) is a retraction and the fourth entity (an integer) describes whether this is a core or a cavity retraction (+1 for core and -1 for cavity).

Output requirements

- An undercut facet f must be included in one and only one side action facet set F.
- All the facets marked as undercuts must be associated with one of the side action facet sets F.

- (x, y) is a feasible core or cavity retraction for all facets in F based on the type defined in T.
- Any two side action solids will not intersect.
- The side actions generated will minimize the following objective (molding cost) function:

$$C = \gamma \sum_{i=1}^{N} \left| x_i \hat{i} + y_i \hat{j} \right|^k + NC' + \chi \sum_{i=1}^{N} \left| x_i \hat{i} + y_i \hat{j} \right| \tag{1}$$

where N is the cardinality of the output set, (x_i, y_i) is the retraction in the i^{th} element of the set, and γ, k, C', χ are molding parameters. Here γ is a proportionality constant that relates the machining cost to the complexity of the solid shape, k is an exponent (obtained experimentally) associating shape complexity with the retraction length, C' is the cost of actuating and assembling side action associated with core or cavity-retraction and χ relates retraction length to the cost incurred due to increase in molding cycle time. It is important to note that depending on the geographical region and the nature of the molding operation, values of these parameters will be significantly different for the same part. In other words, these parameters might have altogether different values for production-run molds and prototyping molds even for the same part.

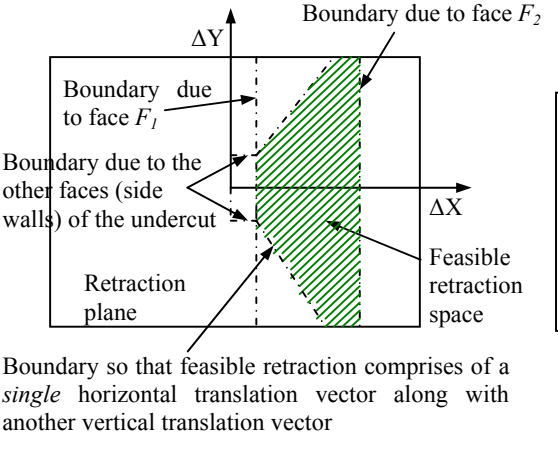


Fig. 4. Feasible core retraction space

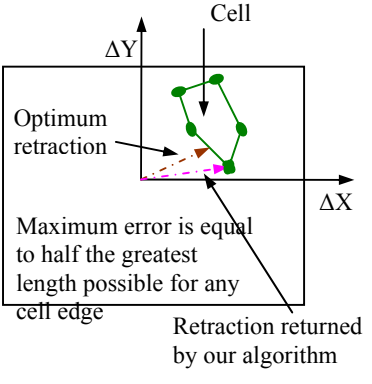


Fig. 5. Maximum error in retraction length

For each undercut facet we will compute its feasible retraction space (see Section 3 for details on this step) that can be represented as a set of one or more disjoint polygons in 2D translation space (ΔXY plane). This translation space can be bounded on all the four sides by the mold enclosure and is termed as the *retraction plane*. Eventually, we get a set of feasible retraction spaces on the retraction plane. The objective function for this problem calls for minimizing the number of side actions as

well as the retraction lengths. Therefore, usually a compromise needs to be worked out by identifying a suitable set of feasible retractions.

The set of feasible retraction spaces can be partitioned into a set of cells and let A to be the spatial (planar) arrangement defined by them. This arrangement A is computed by intersecting and splitting the feasible retraction spaces for all the facets. Hence, for each cell a in A, the set of undercut facets for which this cell will be a subset of its corresponding feasible retraction space is known. This set of facets is termed as the retractable facets for the cell a. In other words, all retractions defined by points located within a are feasible for all the retractable facets. Since by selecting such a retraction, we will be able to deal with a large number of retractable facets at one go, we will then perform a search over all cells and find the optimal combination of retractions.

Although, we follow the described methodology in spirit, an attempt to explicitly generate A leads to implementation challenges due to robustness problems. Hence, we focus on finding a discrete set of promising retractions and performing search over them. In the following sections, we explain our methodology to do the same. But before we proceed, let us establish some important foundations and properties.

Lemma 1. Let r be a retraction used in the optimal solution and F be the set of facets associated with r. Then r will lie on the boundary of a cell in A.

Proof. Since we are minimizing the objective (molding cost) function, if retraction r belongs to the optimal solution, then the corresponding retraction length must have the least possible value. Since the associated facet set F corresponds to a particular cell in A, r must lie on the boundary of that cell, as the point having minimum distance from the origin (i.e. retraction length) for any polygonal cell will be a boundary point. Hence the assertion of lemma directly follows from this observation.

This lemma clearly points out that we only need to consider the boundary of the cells in A and hence the interior of the cells can be safely ignored. Now let us consider a set R that consists of two types of elements - the original corner vertices of all the feasible retraction space polygons and the intersection points obtained by pair-wise intersection of all edges in the feasible retraction space polygons. This set R includes all the vertices of the arrangement A.

Lemma 2. Let F be a facet group in the optimal solution. Then there will exist a retraction r in R such that r is a feasible retraction for all facets in F.

Proof. Since F is a facet group present in the optimal solution, if retraction r is a feasible retraction for all facets in F, then it must be contained (either on the boundary or within the interior) in the particular cell corresponding to F (see Lemma 1).

Now, the set of retractable facets corresponding to a particular cell does not change within its interior. It definitely changes at the 0-faces, i.e. at the original corner vertices of the free space polygons or at the intersection points. The status of the edges is rather hard to determine. While the boundary edges (edges belonging to a single cell only) have the same set of facets as the respective cell interiors, edges that are common to multiple cells have an ambiguous status. However, this is immaterial here since any change in the retractable facet set along an edge is already accounted

by the two vertices forming the edge. Since the set R encompasses all the 0-faces of the cells, r is bound to be present in R and thus, the lemma follows.

Based on Lemma 2, we can utilize R as our search space instead of actually computing A. However, if the solution actually lies on the edge of a cell, we might not get the optimal solution. In order to minimize this error, we facet the edges of the feasible retraction space polygons a priori (before computing the line segment intersections) so that no two neighboring vertices are more than ε apart from each other on any of the sides. Still the retraction lengths in our solution might be marginally greater than or equal to the optimum retraction lengths. However, Theorem 1 formalizes that such an error will be bounded.

Theorem 1. Let S^* be the optimal solution and let S' be a solution that has the same facet sets but each retraction length is increased by $0.5\,\varepsilon$. Then the solution produced by our algorithm will be no worse than S'.

Proof. As Fig. 5 points out, maximum error in retraction length occurs when the optimum solution lies at the mid-point of an edge and the solution explored by our algorithm by searching over the set R, corresponding to the same facet set is obviously one of the edge vertices. If the optimum retraction corresponds to any other point on the edge, then the closer neighboring vertex will be considered by our algorithm. There are two possibilities. First, our algorithm will return this solution, resulting in a small error. This difference is, of course, bounded by half the maximum possible edge length, i.e. by $0.5\,\varepsilon$. Second, our algorithm will find a better solution than this one.

From the above argument, it easily follows that since solution S' has identical facet sets as the optimum solution S^*, but each retraction length is increased by $0.5\,\varepsilon$, our algorithm will not generate solution worse than S'.

Choosing ε to be equal to 1 mm, there is very little qualitative difference between the solution generated by our algorithm and the optimum one. Practically this error has no effect, since a discrepancy of 0.5mm in the retraction length hardly matters due to use of fast actuators attached to side actions. Thus, this error is insignificant.

3 Constructing Feasible Retraction Space

From the discussion in the previous section, it is clear that we first need to construct the feasible retraction space for every individual undercut facet. This is done in two stages. Initially, a feasible space needs to be computed on the retraction plane such that a single horizontal translation vector can pull the facet there without being obstructed by any other facet lying on the way. So, all the potential facets capable of causing obstruction need to be identified. Since the translation vector in our case is restricted to lie in a horizontal plane, it makes sense to consider all the facets lying partially or completely within the z-range of the undercut facet under consideration as potential obstacles. Z-range refers to the 3D space bounded by the maximum and the minimum z coordinate values of the facet vertices. In case a facet does not lie completely within the z-range, only the part of it lying inside is considered. This involves truncation of triangular facets to form convex polygons of 3, 4 or 5 sides.

According to Aronov and Sharir [17], 3D free configuration space *FP* of a convex "robot" *B* moving in space occupied by k' obstacles $\{A_1, \ldots, A_{k'}\}$ is given as the complement *C* of *U*, where $U = \bigcup_{i=1}^{k'} P_i$ be the union of the so-called expanded obstacles P_i. Here P_i is the Minkowski sum of A_i and $-B$, for $i = 1,\ldots, k'$.

Collision polyhedron for a facet f with respect to another facet f' is defined as the set of points in 3D translation space such that if corresponding translations are applied to *f*, then the translated *f* intersects with *f'*.

Collision polygon for a facet f with respect to another facet f' is defined as the set of points in 2D translation space such that if corresponding translations are applied to *f*, then the translated *f* intersects with *f'*.

In this case, replacing *B* by the undercut facet under consideration and A_i^s by the k' facets falling within its z-range, set of collision polyhedrons are obtained. This Minkowski sum is computed easily by obtaining the convex hull of the vector differences of each pair of vertices,. The collision polyhedrons are then intersected by a horizontal plane located at $\Delta z = 0$ to obtain collision polygons in the retraction plane. However, the fact that the facet must be able to reach this feasible space by means of a single translation vector also needs to be taken into account here. That is why, mere construction of Minkowski polyhedrons and then intersection with a horizontal plane do not give the final obstructed space.

Sweep-based collision polygon for a facet f with respect to another facet f' is defined as the set of points in 2D translation space such that if corresponding translations $\vec{t}$ are applied to *f*, the swept polygon *P* will intersect *f'*, where $P = \{p + \xi \vec{t} : p \in f, 0 \le \xi \le 1\}$.

Collision free 2D translation space for a facet f is defined as the set of points in 2D translation space such that if corresponding translations $\vec{t}$ are applied to *f*, the swept polygon *P* (defined as before) will not intersect with any of the collision facets *f'*.

The hard shadows of all the k' collision polygons need to be computed in order to determine the sweep-based collision polygons. Alternatively, same thing is done by plane sweeping the collision polygons until they reach the retraction plane boundaries. Lastly, the collision free 2D translation space is obtained by subtracting the union of the forbidden spaces from the bounded retraction plane. The steps are schematically shown in Fig. 6.

In order to ensure that the horizontally translated undercut facet can also be pulled vertically, it is necessary to construct another feasible space on the retraction plane. This transformation space is called *2D translation space for upward vertical accessibility for a facet f*. It is defined as the set of points in 2D translation space such that if this translation is applied to *f*, then it can be released vertically upwards. Similarly, a 2D translation space for downward vertical accessibility can also be defined for facet *f*. This direction of possible release provides a natural way of classifying feasible retractions into feasible core retractions and feasible cavity retractions. In order to compute this space, the entire mold free space (regularized Boolean difference between the mold enclosure and the part) is voxelized. Each voxel is then analyzed for vertical accessibility and a procedure similar to the previous one

is adopted. Finally, the intersection of the collision-free 2D translation space and the 2D translation space for upward (or downward) vertical accessibility is taken to identify the feasible core (or cavity) retraction space for every individual undercut facet. If voxels are accessible along both $+z$ as well as $-z$ directions, then they are either merged with voxels accessible along $+z$ or $-z$ depending upon the nature of the neighboring voxels. In general we prefer feasible cavity retractions because side actions associated with cavity retractions are easier to realize in practice.

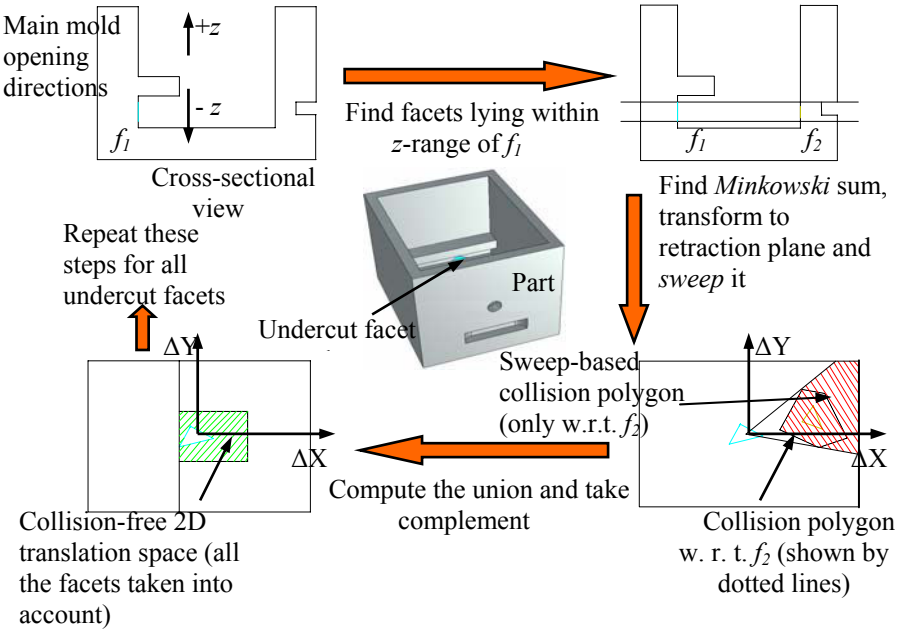


Fig. 6. Constructing collision-free 2D translation space

4 Computing Discrete Set of Candidate Retractions

As discussed in Section 2, the aim is to obtain the discrete set of all possible retractions. Firstly, edge faceting is carried out on all the feasible retraction space polygons if necessary and then all the line segment intersection points are computed. The corner points of all the retraction space polygons are also retained. Next we need to prune the set of retractions to obtain a so-called non-dominated set. Such a set will only consist of those retractions which are better than any other in the entire set with respect to either the retraction length or the number of associated facets. This is done by first sorting all the retractions in order of increasing retraction lengths. Then we search for all the retractions that have identical retraction lengths and again sort them in order of their number of retractable facets. We delete all the retractions that have lesser number of associated facets than the current maximum value. Initially, this current maximum is equal to zero and we keep on updating it as we progressively consider one retraction length bucket after another.

Once all the non-redundant retractions have been computed, we can start constructing a tree to represent our feasible solution space. In order to do that, we need to sort the undercut facets in a particular way, so that the process of placing the nodes gets facilitated. A heap is built for the n undercut facets where they are arranged depending upon their minimum retraction lengths (with highest preference being given to those having maximum values of minimum retraction lengths). Ties are broken on the basis of lesser number of associated retractions. This heap is used as an efficient priority queue.

Moreover, a linked list is used so that access from facets to retractions can be done easily in constant time. For a particular element in the linked list, i.e. an undercut facet, all the associated retractions are maintained in a sorted order depending upon their lengths, with highest preference being given to the one(s) having least magnitude. This is utilized while placing the actual nodes in the search tree. All these data structures are created so that queries become efficient while traversing the search tree to obtain an optimal solution to the undercut region grouping problem. They ensure that we do not need to perform more than $O(n \log n)$ computations at any node in any level of the tree, instead of the usual $O(n^2)$ calculations necessary if we carry out exhaustive comparisons and enumerations.

5 Generating Side Actions

A depth-first branch and bound algorithm is used to determine the optimal set of undercut regions. This requires us to employ intelligent heuristics to quickly steer search to a good initial solution, limit branching, and prune as many search paths as possible. The notion of a *bottleneck* facet plays a key role in realizing these heuristics. It may be observed that certain facets become the main bottlenecks in generating a solution. These facets have very limited number of feasible solutions and they impose constraints on the goodness of the overall solution. The overall solution has to address these facets and hence it is desirable to process them first. These facets belong to the category such that the minimum retraction length for them is maximum among all the facets. On top of this, they must have rather narrow feasible retraction spaces, which, in turn, will be reflected in the fact that they will have smaller number of retractions associated with them.

Once bottleneck facets have been identified, we will start constructing the search space with any one of them. An empty node is created as the root node and all the vertices associated with the chosen bottleneck facet (highest priority element in our heap) are placed as top level nodes in the search space. Since we will access the vertices directly using the linked list, they will be placed in order of increasing retraction lengths. If we consider a particular retraction, its associated retractable facets form the first undercut region. In the next level, we need to determine the bottleneck facets among the remaining ones and place the vertices attached to any one of them as nodes. We proceed in this manner until all facets are covered. Of course, we need to be careful not to include two vertices such that the horizontal translation vectors corresponding to the retractions intersect with each other in the final solution. Such a path, if encountered, will be termed as infeasible and will be pruned. If some

of the retractable facets associated with a retraction are already covered then they will not be added to this solution.

We keep track of the current best solution. If during the search, cost of a partial solution exceeds the cost of current best solution, then this path is pruned. If during the search a better solution than the current best solution is found then the current best solution is updated. Search can terminate in two ways. Firstly, it terminates when all promising nodes have been explored. In this case it produces a solution very close to the optimal solution. Secondly, it stops when the user specified time limit has been exceeded. In this case search returns the current best solution.

The bottleneck facet scheme restricts the amount of branching (or number of nodes) at a particular search level. Now coming to the tree depth issue, it is rare to find parts in which a connected undercut region requires more than three side actions. Usually cost function parameters are such that solutions involving $N + 1$ side actions are more expensive than a solution involving N side actions. Our heuristics enable us to quickly locate feasible solutions and once a feasible solution has been found, very few nodes are explored at the next depth level. Hence, for virtually all practical parts we should be able to find optimal solutions in a reasonable amount of time. As with any depth first branch and bound algorithm, the time complexity increases exponentially with the depth of the search tree. However, since we do not expect practical cases involving more than three side actions for a single undercut region, the exponential growth is not much of a concern in this particular application.

Once all the regions have been generated, they are swept along the associated horizontal translation vectors (corresponding to the retractions) and additional patches are included at the top, bottom and the arrow-tip end of the vector to form a compact, 2-manifold solid. These capping patches consist of planar faces only. The boundaries of these solids are then triangulated to represent them in a faceted format and they form the desired set of side actions.

6 Results

All the algorithms were implemented in C++ using Visual Studio.NET 2003 in Windows XP Professional operating system. CGAL [18] version 3.0.1 was used as the geometric kernel. In order to speed up computation, all the vertex coordinates were first converted into integral values and then the Cartesian kernel with int number type was used. CAD model for every part was triangulated and converted into .stl format which was then taken as input by the program. The main mold opening direction was inputted separately. A preprocessor program was written to recognize all undercut facets. All the programs were run in a Pentium M processor machine having 512 MB of RAM and processor clock speed of 1.6 GHz.

A series of computational experiments have been carried out on 4 parts to characterize the performance of the algorithms. Since we cannot compare time values across different parts, we decided to facet each part using four different levels of accuracy. Now comparisons with respect to computation time, number of nodes in search space and so on can be made for the same part having varying number of

facets. Results of these experiments for γ = 200 (in \$/mm^2), k = 2, C' = 5,000 or 10,000 (in \$) depending upon whether it is a cavity or core retraction, and χ = 18.227 (in \$/mm) are shown in Table 1 below. These values are based on a specific injection molding scenario. The side action solids generated for the four test parts have been displayed in Fig. 7.

Certain basic trends are discernible from the values in the table. Of course, both the feasible retraction space and candidate retraction computation times increase as we go for higher number of facets to represent the part. However, for parts A and B where the numbers of undercut facets remain same, although the overall numbers of facets increase, this trend is markedly different from parts C and D. The retraction space computation time increases linearly, whereas candidate retraction calculation time remains more or less constant in the former case. In the latter case, feasible retraction space construction time increases almost quadratically, while the retraction computation time increases at a rate slightly greater than linear, but less than quadratic, indicating possibly a linear-logarithmic growth. Overall an optimal set of 2 or 3 side actions were generated for four sample parts in about 30-50s. This is a reasonably good performance and can serve as the foundation step towards our eventual goal of fully automatic side action design.

Table 1. Results of computational experiments

Part	Model	Total # of facets	# of undercut facets	Feasible retraction space computation time (in s)	Candidate retraction generation time (in s)	Depth-first branch and bound computation time (in s)
A	# 1	224	36	3.5	2.2	27.0
	# 2	256	36	4.0	2.4	28.0
	# 3	336	36	4.8	2.8	30.0
	# 4	568	36	6.1	3.3	32.0
B	# 1	378	122	6.0	5.2	36.0
	# 2	570	122	7.4	5.5	36.0
	# 3	716	122	8.5	5.7	36.5
	# 4	882	122	10.0	5.8	37.0
C	# 1	414	175	5.9	6.4	2.3
	# 2	478	188	6.3	6.5	2.3
	# 3	576	194	7.1	6.7	2.4
	# 4	882	249	12.2	7.6	2.5
D	# 1	376	60	0.3	3.5	3.2
	# 2	814	122	1.0	5.9	3.3
	# 3	1324	156	2.1	9.0	3.5
	# 4	2002	218	4.4	16.3	3.9

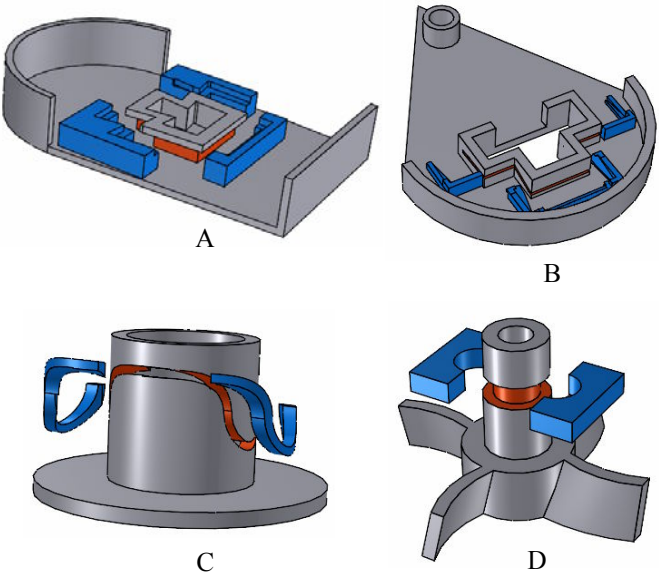


Fig. 7. Side action solids for 4 different test parts (A, B, C, D) shown in retracted state

7 Conclusions

New algorithms to automatically generate shapes of side actions have been presented in this paper. The major contributions of our work can be summarized as follows. It is capable of designing side actions for complex undercuts that are finitely accessible. Then it successfully partitions connected undercut regions (for which no single side action exists) into smaller regions, such that each of them can be molded by separate side actions and a customizable molding cost function is minimized. Many of the steps in the computation of feasible retraction space and discrete set of candidate retractions have linear or linear-logarithmic worst-case asymptotic time complexities. Few grow quadratically with an increase in the total number of part facets as well the number of undercut facets. Finally, if a connected undercut region can be molded by 3 or fewer number of side actions, then empirical results suggest that our algorithm is capable of finding a solution very close to the optimal solution in a reasonable amount of time for most practical parts.

This paper focuses on a particular type of side action, commonly known as side core in molding terminology. Further work needs to be done to generalize our method to design other kinds of side actions, namely, split cores, lifters etc. We will also continue working on improving our edge-faceting scheme, so that we can come up with stronger theoretical results to rule out such operations for a majority of feasible retraction space polygonal edges. In addition, we plan to incorporate better forward looking cost bounding functions to prune larger number of nodes in the search tree.

Acknowledgements. This work has been supported by NSF grant DMI-0093142. However, the opinions expressed here are those of the authors and do not necessarily reflect that of the sponsor. We would also like to thank the reviewers for their comments that improved the exposition.

References

[1] Fu, M. W., Fuh, J. Y. H., Nee, A. Y. C.: Undercut feature recognition in an injection mould design system. Computer Aided Design, Vol. 31, No. 12, (1999), 777-790
[2] Chen, L-L., Chou, S-Y., Woo, T. C.: Partial Visibility for Selecting a Parting Direction in Mould and Die Design. Journal of Manufacturing Systems, Vol. 14, No. 5, (1995), 319-330
[3] Hui, K. C.: Geometric aspects of the mouldability of parts. Computer Aided Design, Vol. 29, No. 3, (1997), 197-208
[4] Yin, Z. P., Ding, H., Xiong, Y. L.: Virtual prototyping of mold design: geometric mouldability analysis for near net-shape manufactured parts by feature recognition and geometric reasoning. Computer Aided Design, Vol. 33, No. 2, (2001), 137-154
[5] Lu, H. Y., Lee, W. B.: Detection of interference elements and release directions in die-cast and injection-moulded components. Proceedings of the Institution of Mechanical Engineers Part B Journal of Engineering Manufacture, Vol. 214, No. 6, (2000), 431-441
[6] Chen, Y., Rosen, D. W.: A Region Based Method to Automated Design of Multi-Piece Molds with Application to Rapid Tooling, Journal of Computing and Information Science in Engineering, Vol. 2, No. 2, (2002), 86-97
[7] Dhaliwal, S., Gupta, S. K., Huang, J., Priyadarshi, A.: Algorithms for Computing Global Accessibility Cones. Journal of Computing and Information Science in Engineering, Vol. 3, No. 3, (2003), 200-209
[8] Priyadarshi, A.K., Gupta, S. K.: Geometric algorithms for automated design of multi-piece permanent molds. Computer Aided Design, Vol. 36, No. 3, (2004), 241-260
[9] Ahn, H-K., de Berg, M., Bose, P., Cheng, S-W., Halperin, D., Matousek, J., Schwarzkopf, O.: Separating an object from its cast. Computer Aided Design, Vol. 34, No. 8, (2002), 547-559
[10] Elber, G., Chen, X., and Cohen, E.: Mold Accessibility via Gauss Map Analysis. Journal of Computing and Information Science in Engineering, Vol. 5, No. 2, (2005), 79-85
[11] Kharderkar, R., Burton G., McMains, S.: Finding Feasible Mold Parting Directions Using Graphics Hardware. In Proceedings of the 2005 ACM symposium on Solid and Physical Modeling, Cambridge, MA, (2005), 233-243
[12] Priyadarshi, A.K., Gupta, S. K.: Finding Mold-Piece Regions Using Computer Graphics Hardware. In Proceedings of Geometric Modeling and Processing, Pittsburgh, PA, (2006)
[13] Ravi, B., and Srinivasan, M. N.: Decision criteria for computer-aided parting surface design. Computer Aided Design, Vol. 22, No. 1, (1990), 11-18
[14] Wong, T., Tan, S. T., Sze, W. S.: Parting line formation by slicing a 3D CAD model. Engineering with Computers, Vol. 14, No. 4, (1998), 330-343
[15] Ye, X. G., Fuh, J. Y. H., Lee, K. S.: A hybrid method for recognition of undercut features from moulded parts. Computer Aided Design, Vol. 33, No. 14, (2001), 1023-1034
[16] Shin, K. H., Lee, K.: Design of Side Cores of Injection Mold from Automatic Detection of Interference Faces. Journal of Design and Manufacturing, Vol. 3, No. 4, (1993), 225-236
[17] Aronov, B., Sharir, M.: On Translational Motion Planning of a Convex Polyhedron in 3-Space. Siam Journal of Computing, Vol. 26, No. 6, (1997), 1785-1803
[18] Cgal.org.: Computational Geometry Algorithms Library. http://www.cgal.org, (2004)

Finding All Undercut-Free Parting Directions for Extrusions

Xiaorui Chen and Sara McMains

Department of Mechanical Engineering
University of California, Berkeley CA 94720, USA
{xrchen, mcmains} @me.berkeley.edu

Abstract. For molding and casting processes, geometries that have undercut-free parting directions (UFPDs) are preferred for manufacturing. Identifying all UFPDs for arbitrary geometries at interactive speeds remains an open problem, however; for polyhedral parts with n vertices, existing algorithms take at least $O(n^4)$ time. In this paper, we introduce a new algorithm to calculate all the UFPDs for extrusions, an important class of geometry for manufacturing in its own right and a basic geometric building block in solid modeling systems. The algorithm is based on analyzing the 2D generator profile for the extrusion, building on our previous results for 2D undercut analysis of polygons. The running time is $O(n^2 \log n)$ to find the exact set of UFPDs or $O(n)$ to find a slightly conservative superset of the UFPDs, where n is the geometric complexity of the 2D generator profile. Using this approach, the set of possible UFPDs for a part containing multiple extruded features can be reduced based upon an analysis of each such feature, efficiently identifying many parts that have no UFPDs and reducing the search time for complete algorithms that find all UFPDs.

1 Background and Related Work

In molding or casting manufacturing processes, molten plastic or metal is reshaped and solidified in a hollow mold. Simple reusable molds consist of two rigid halves that move in opposite directions during the mold closing and opening operations. The direction of motion of the mold halves is called the *parting direction*. The two mold halves meet at the *parting surface* (see Fig. 1), which may need to be non-planar for more complex parts. The part geometry is said to be *2-moldable (monotone) in a direction* $\boldsymbol{d}$ if the mold halves forming it can be translated to infinity along $\boldsymbol{d}$ and $-\boldsymbol{d}$, respectively, without collision with the interior of the part. The part shown in Fig. 1(a) is 2-moldable in the vertical direction; the same part with a different orientation shown in Fig. 1(b) is not 2-moldable in the vertical direction. Surfaces that prevent the removal of the mold halves with respect to a particular parting direction are called *undercuts*. The existence of undercuts not only increases the mold cost but also shortens the mold life. Therefore, all else being equal, geometries with undercut-free parting directions (UFPDs) are preferred. Since early feedback on UFPDs reduces the cost to redesign a part, in our research we concentrate on finding all UFPDs at

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 514–527, 2006.
© Springer-Verlag Berlin Heidelberg 2006

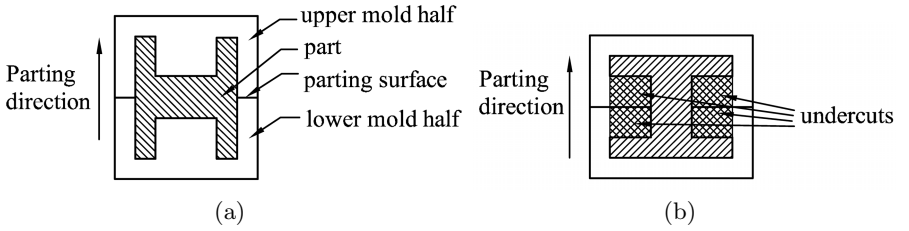


Fig. 1. Terminology. (a) mold for a simple 2D part; (b) an orientation of the same part with undercuts.

interactive speeds for an arbitrary geometry during the early design stage. The coordinate frame can then be rotated so that the desired UFPD is aligned with the parting direction.

Whether a given part geometry allows a UFPD has been studied by many researchers. Early work considered only a limited number of potential parting directions, such as the three principle axes [1,2] and the bounding box axes [3], or used a heuristic search approach to choose potential directions to test [4,5]. These algorithms are not guaranteed to find a feasible parting direction even if one does exist if it is not in the set tested. Other researchers developed parting direction algorithms based on convex hull differences [6,7,8,9,10,11,12]. They identify connected surfaces that are potential undercuts, which are called "pockets," by performing a regularized [13] subtraction of the part from its convex hull. The preferred parting directions are chosen by resolving as many undercuts as possible. This approach, however, cannot find existing UFPDs when portions of a single "pocket" can only be formed by different halves of the mold. Graph-based feature recognition is another approach to finding undercuts [14,15,16,17,18]. However, this approach breaks down for complex interacting features. Multi-piece and/or sacrificial molds are not considered in this paper; we restrict our discussion to two-piece, rigid, reusable molds with opposite removal directions.

Complete algorithms for finding if *any* UFPD exists for a given geometry have been presented in both two and three dimensions. Rappaport and Rosenbloom give an $O(n)$ time algorithm to determine if a 2D polygon with n vertices is moldable in arbitrary (not necessarily opposite) removal directions, and an $O(n \log n)$ time algorithm for opposite removal directions [19]. Bose et al. present algorithms to determine the existence of a parting direction for simple polyhedra [20]. They assume planar parting surfaces in their work. In prior work, we showed how to find all UFPDs for a 2D polygon bounded by straight line and/or curved edges in $O(n)$ time [21]. Ahn et al. present a complete algorithm to find all the combinatorially distinct 2-moldable directions for a 3D (faceted) polyhedron in time $O(n^5 \log n)$ and a more efficient but more complicated algorithm that runs in time $O(n^4)$ [22]. However, in practice, their implementation instead reverts to testing heuristically chosen directions based on input edge orientation and additional randomly chosen test directions. Elber et al. give an exact solution for a model bounded by only NURBS surfaces, but it is restricted to a completely smooth boundary that is C^3 everywhere [23]. Khardekar et al. developed

a programmable graphics hardware accelerated algorithm to find the combinatorially distinct UFPDs for a given faceted geometry. Their implementation can graphically display the undercut for a particular parting direction in linear time with respect to the number of facets in the solid model [24]. But their complete algorithm to find all UFPDs or to definitely state that no UFPDs exist still takes $O(n^5)$ time.

In order to take advantage of a complete algorithm for finding UFPDs on arbitrary 3D input such as the one presented by Ahn et al. or Khardekar et al., while speeding up the running time, we propose to reduce its search space. In this paper, we study extrusions, a basic geometric building block in solid modeling systems, showing how to detect all UFPDs for individual extruded features. The UFPDs thus detected can then be passed to a complete algorithm for further testing with respect to the full part geometry. The running time is $O(n^2 \log n)$ to find the exact set of UFPDs or $O(n)$ to find a slightly conservative superset of the UFPDs, where n is the geometric complexity of the 2D generator profile. Using this approach, the set of possible UFPDs for a part containing multiple extruded features can be reduced for each such feature, efficiently identifying many parts that have no UFPDs and reducing the search time for complete algorithms that find all UFPDs.

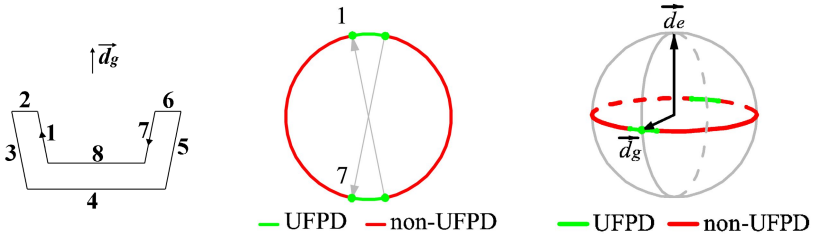
We next define our assumptions and terminology and summarize our prior work on undercut analysis for polygons before presenting our algorithm for finding all UFPDs for extruded features.

2 Assumptions, Preliminaries and Background

2.1 Assumptions and Preliminaries

In this paper, we assume that the extrusion is formed by extruding a 2D generator profile along its plane normal direction by a given distance. We denote the extrusion direction as $\boldsymbol{d_e}$. For the sake of clarity, we assume that the coordinate frame has been rotated to align $\boldsymbol{d_e}$ with the vertical $+z$ axis. The 2D generator profile is a polygon without self-intersections but possibly with holes. The polygon is bounded by straight line segments that only intersect their neighbor segments at endpoints, whereas non-adjacent edges do not intersect each other. We use the right-hand rule convention that the edges of the polygon are oriented in such a way that the interior of the polygon lies on the left when moving along the edges — that is, the polygon is oriented counterclockwise. The normal of each oriented edge is a unit vector pointing towards the exterior of the polygon. A direction in the plane of the 2D generator profile is denoted by $\boldsymbol{d_g}$.

To capture all the directions in 2D Euclidean space, we use a Gaussian circle; each direction in 2D can be represented by a point on the Gaussian circle by normalizing the direction to a unit vector and placing its tail at the origin. Similarly, we use a Gaussian sphere to represent every possible direction in 3D. For the sake of simplicity, when we refer to a direction $\boldsymbol{d}$ represented by a point on the Gaussian sphere, we may call it point $\boldsymbol{d}$. The extrusion direction $\boldsymbol{d_e}$ is mapped on the Gaussian sphere to the north pole and its inverse is mapped to



(a) (b) (c)

Fig. 2. (a) A polygon P; (b) UFPDs and non-UFPDs of P on Gaussian circle, with each arc bounded by two points representing directions of edge 1 and 7 or their inverse directions; (c) UFPDs and non-UFPDs of P on Gaussian sphere, showing the longitudinal great circle for $\boldsymbol{d_g}$

the south pole. Noting that the 2D generator profile of the extrusion is normal to $\boldsymbol{d_e}$, every direction in the plane of this 2D generator profile can be represented by a point on the equator. We call the great circle through the poles and a direction $\boldsymbol{d_g}$ on the equator $\boldsymbol{d_g}$'s *longitudinal great circle* (Figure 2(c)). For any point on the sphere other than the poles, there is a unique longitudinal great circle containing it. We denote an arbitrary direction represented by a point on a given equatorial direction $\boldsymbol{d_g}$'s longitudinal great circle that is not on one of its poles $\boldsymbol{d_{eg}}$ (Figure 4). For purposes of visualization, we render the Gaussian sphere and its equator, if shown, in grey.

2.2 Finding All UFPDs for Polygons

Since the algorithm proposed in this paper is based on our prior work for undercut analysis on polygons [21], we will summarize here the input and output of that analysis.

The polygon undercut analysis projects the normal directions of polygon edges and their connectivity onto the Gaussian circle, which in combination uniquely determine if a direction is a UFPD or a non-UFPD. The input to the algorithm is a simple oriented polygon (i.e., without holes or self-intersections) [1]. The output is a set of arcs on the unit circle, with each arc denoting a continuous set of UFPDs for the input polygon. That is, for each point on the output arcs, the polygon is 2-moldable in the direction it represents. Each arc is bounded by and includes its two endpoints, with each representing an edge direction of the input polygon or its inverse direction. The two endpoints of an arc may coincide with each other when the arc is a single point on the Gaussian circle or is a full Gaussian circle itself. An example of the algorithm output is illustrated in Fig. 2(b). (The algorithm actually computes arcs bounded by points corresponding to normals of polygon edges, a clockwise rotation of 90° around the origin relative to the output shown here, which is in terms of edge directions.)

[1] In 2D, a polygon with holes is not 2-moldable in any direction. Therefore, no directions are UFPDs for such a polygon.

3 UFPDs for Polyhedrons

Before describing how UFPDs are found for extrusions, we first introduce some useful properties of UFPDs for polyhedra in general.

A direction $\boldsymbol{d}$ is a UFPD for a given polyhedron if and only if every line parallel to $\boldsymbol{d}$ intersects the boundary of the polyhedron at most two times, where an intersection may be either a point or a line segment [22,12] (see Fig. 3(a)). Noting that a polyhedron can be decomposed into a union of an infinite number of infinitesimally thin cross sections that are both parallel to each other and parallel to a fixed plane containing $\boldsymbol{d}$, we call these cross sections *a family parallel to* $\boldsymbol{d}$. The cross sections shown in Fig. 3(b) and (c) with boundaries highlighted illustrate two different families of cross sections (only the cross sections passing through vertices are drawn in the figure). Every line l parallel to $\boldsymbol{d}$ lies on the plane of exactly one such cross section for a given family. It is parallel to the planes of the other cross sections in the family and does not intersect them. Therefore, the number of intersections between l and the boundary of the polyhedron equals the number of intersections between l and the boundary of the cross section on whose plane l lies. This observation is the basis for the following theorem.

Theorem 1. *A direction* $\boldsymbol{d}$ *is a UFPD for a given polyhedron if and only if every line parallel to* $\boldsymbol{d}$ *intersects the boundary of any cross section in a given family parallel to* $\boldsymbol{d}$ *at most two times.*

The family of cross sections parallel to a direction is not unique. Fig. 3 shows an extrusion. In Fig. 3(b), the cross sections are parallel to both the test direction $\boldsymbol{d_2}$ and the extrusion direction $\boldsymbol{d_e}$; in Figure 3(c), the cross sections are parallel to $\boldsymbol{d_2}$ but normal to $\boldsymbol{d_e}$. Note that by Theorem 1, it is sufficient to test a potential UFPD against *any* single family of cross sections parallel to the test direction.

For an arbitrary polyhedron, we could use Theorem 1 and our 2D classification algorithm to classify all directions on a given great circle of the Gaussian sphere (as either UFPDs or non-UFPDs). We could do so by slicing the polyhedron with planes parallel to the great circle and running the 2D algorithm on each cross section, then finding the intersection of the UFPD sets calculated for each. Since parallel cross sections intersecting the same faces would have

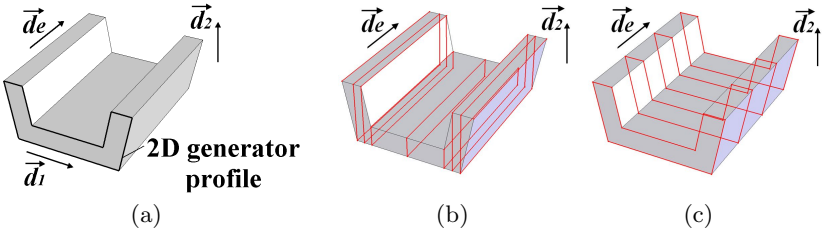


Fig. 3. (a) Direction $\boldsymbol{d_1}$ is not a UFPD, $\boldsymbol{d_2}$ is a UFPD; (b) A family of cross sections parallel to $\boldsymbol{d_2}$ and $\boldsymbol{d_e}$; (c) A family of cross sections parallel to $\boldsymbol{d_2}$ and normal to $\boldsymbol{d_e}$

identical UFPD sets, only $O(n)$ cross sections would need to be checked. Unfortunately, for the general case, there are an infinite number of such great circles to test. For extrusions, however, we can divide the Gaussian sphere into groups of longitudinal great circles with similar properties, using Theorem 1 to show that all directions in each group have the same classification in most cases. The exact classifications within each group can be computed from the 2D generator profile. The details of the algorithm are described below.

4 Finding All UFPDs for Extrusions

Recall that for the 2D generator profile of the extrusion, all UFPDs and non-UFPDs can be represented by points on the equator of the Gaussian sphere (see Fig. 2). These equatorial directions have the same classification (as UFPDs or non-UFPDs) for the extrusion as their classification for the 2D generator profile, since the family of cross sections perpendicular to the extrusion direction all have identical geometry. The extrusion direction $\boldsymbol{d_e}$ and its inverse are always UFPDs. The classification of any other direction on the Gaussian sphere is a function of which longitudinal great circle contains it. We have two cases: (1) the point at the intersection of its longitudinal great circle and the equator represents a UFPD for the 2D generator profile; (2) the point at the intersection of its longitudinal great circle and the equator represents a non-UFPD for the 2D generator profile. In the following two sections, we discuss how to classify all points as either UFPDs or non-UFPDs for these two cases.

4.1 Extending UFPDs of the 2D Generator Profile

Lemma 1. *Every point $\boldsymbol{d_{eg}}$ on a longitudinal great circle c passing through a UFPD point $\boldsymbol{d_g}$ for the 2D generator profile represents a UFPD for the extruded feature, including $\boldsymbol{d_g}$.*

Proof. Consider any point $\boldsymbol{d_{eg}}$ on c, exclusive of the poles, and a family of cross sections parallel to c. We will first show that each such cross section C is either a line segment with length equal to the extrusion height, or a rectangle with height equal to the extrusion height. Since C is itself an extrusion along $\boldsymbol{d_e}$, the line segment would be an extrusion of a point on the 2D generator profile, while the rectangle would be an extrusion of a line segment on or through the interior of the 2D generator profile. Thus we will prove, by contradiction, that the intersection of C and the 2D generator profile (interior and boundary) is either a point or a line segment.

Assume that C intersects the boundary of the 2D generator profile more than two times, where each intersection may be either a point or a line segment. These intersections lie on one line l. Because the planes of C and the 2D generator profile are both parallel to $\boldsymbol{d_g}$, their intersection line l is parallel to $\boldsymbol{d_g}$. Therefore, by our assumption there exists a line parallel to $\boldsymbol{d_g}$ that intersects the boundary of the 2D generator profile more than two times. But this would make $\boldsymbol{d_g}$ a *non*-UFPD for the 2D generator profile, a contradiction. Therefore, any cross

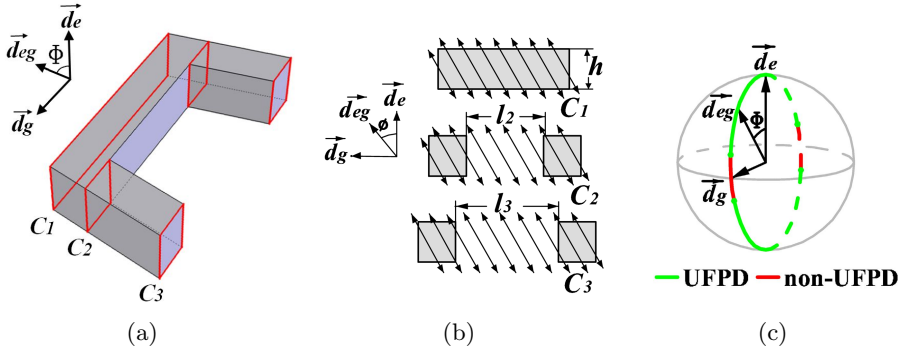


Fig. 4. $\boldsymbol{d_g}$ is a non-UFPD for the 2D generator profile. (a) Extrusion feature; (b) a family of cross sections; (c) Gaussian sphere showing $\boldsymbol{d_g}$'s longitudinal great circle.

section C parallel to c intersects the boundary of the 2D generator profile at most two times. By enumerating all the cases, we can see that C intersects the 2D generator profile (interior and boundary) at either a point or a line segment because an intersection at two disconnected pieces would be adjacent to a cross section that intersected the boundary of the 2D generator profile more than two times. Thus C is either a line segment or a rectangle, and any line, including lines parallel to $\boldsymbol{d_{eg}}$, intersects a line segment or the boundary of a rectangle at most two times.

Since C is an arbitrary cross section parallel to c, which is parallel to $\boldsymbol{d_{eg}}$, all such cross sections together are also a family parallel to $\boldsymbol{d_{eg}}$. Thus by Theorem 1, $\boldsymbol{d_{eg}}$ is a UFPD for the extrusion for any direction on the longitudinal great circle of a direction $\boldsymbol{d_g}$ that is a UFPD for the 2D generator profile.

4.2 Resolving Non-UFPDs of the 2D Generator Profile

At first glance, it may appear as if every point on the longitudinal great circle of a non-UFPD for the 2D generator profile (not including the north and south poles) is also a non-UFPD for the extrusion, as Kurth and Gadh assumed [25]. However, this is not the case, as the counterexample shown in Fig. 4 demonstrates. The direction $\boldsymbol{d_g}$ is a non-UFPD for the 2D generator profile. If we look at each cross section parallel to the longitudinal great circle determined by $\boldsymbol{d_g}$, its boundary is one or more disjoint rectangles. We denote l_i as the horizontal length of the smallest gap between two adjacent rectangles in cross section C_i. All the lines in the plane of C_i with a slope in the vertical plane larger than l_i/h, where h is the extrusion height, intersect the boundary of C_i at most two times. The ratio between $l_{min} = \min(l_i)$ and h determines all lines that intersect with *every* cross section boundary at most two times: any direction $\boldsymbol{d_{eg}}$ with $\phi < \tan^{-1}(l_{min}/h)$ is a UFPD for the extrusion, by Theorem 1. The symmetrical continuous sets of UFPDs on the longitudinal great circle are shown in Fig. 4(c).

Thus we can obtain the set of UFPDs on the longitudinal great circle of $\boldsymbol{d_g}$, a non-UFPD for the 2D generator profile, as follows. Assume the coordinate frame

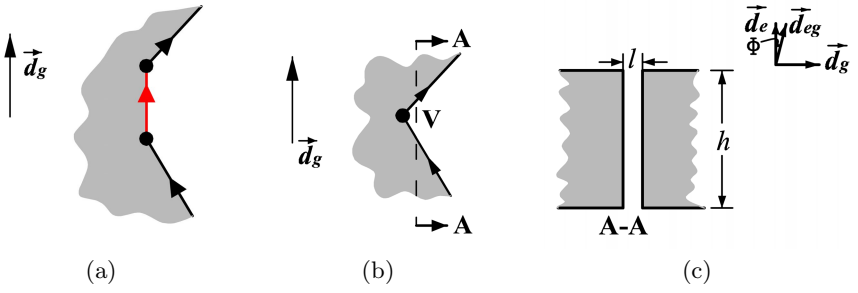


Fig. 5. (a) A vertical edge with concave endpoints; (b) a concave vertex; (c) cross section A-A, showing that no points are UFPDs on the longitudinal great circle of $\boldsymbol{d_g}$ when l approaches zero. Grey area denotes interior of the 2D generator profile.

has been rotated to orient $\boldsymbol{d_g}$ in the vertical direction. First, we consider each vertical line segment that connects two points on the 2D generator profile and that either lies totally outside the 2D generator profile or coincides with an edge of the 2D generator profile having two concave endpoints (Fig. 5(a)). The length l_i of the line segment equals the gap between two rectangles on the boundary of a cross section passing the line segment, as shown in Figure 4(b). The shortest of all such line segments is l_{min}. The maximum angle ϕ can then be calculated: $\phi_{max} = \tan^{-1}(l_{min}/h)$. The same arguments apply to 2D generator profiles with holes, for which all equatorial directions are non-UFPDs, since the analysis is based purely on cross sections. Note that the cross sections may contain more than one rectangle having gaps in between them, whether or not the 2D generator profile contains holes.

We only need to consider a finite number of line segments. If the regularized difference between the convex hull of the 2D generator profile and the 2D profile itself is decomposed into a trapezoidal map [26], assuming that $\boldsymbol{d_g}$ is vertical, the length of the vertical line segments change linearly and monotonically within a trapezoid. Therefore, only line segments coinciding with the left or right edge of a trapezoid and bounded by the 2D generator profile need to be considered. For each such line segment, at least one of its endpoints coincide with a vertex of the 2D generator profile. We will describe the details of our algorithm for calculating the minimum distance efficiently in Section 5.1.

It is worth noting the case illustrated in Fig. 5(b) and (c). For a concave vertex V whose two incident edges lie on the same side of a vertical line l passing through V (both on the left or both on the right), Fig. 5(c) shows a cross section slightly to the right of V. We can see that l approaches zero when the cross section approaches V. Thus, ϕ_{max} approaches zero and no direction (with the exception of the north and south poles, which are always UFPDs for extruded features) on the longitudinal great circle is a UPFD for the extrusion. Furthermore, if $\boldsymbol{d_g}$ is not parallel to any edge of the 2D generator profile, each 2D undercut with respect to $\boldsymbol{d_g}$ must contain at least one such vertex, since it must contain an up-facet connected to a down-facet at a concave vertex. That is, ϕ_{max} is always

zero when a non-UFPD $\boldsymbol{d_g}$ is not parallel to any edge of the 2D generator profile. Therefore, we only need to calculate the minimum distances for the finite number of orientations where $\boldsymbol{d_g}$ is parallel to at least one of the edges of the profile.

5 Algorithm for Finding All UFPDs for an Extrusion

Based on the previous discussion, we can now find all UFPDs for an extruded feature. The pseudocode is listed in Algorithm 1, which in turn calls the function MinDistance() to calculate the minimum distance for a non-UFPD of the 2D generator profile. The full algorithm takes $O(n^2 \log n)$ time, where n is the number of vertices on the 2D generator profile. The 2D undercut analysis to find the equatorial set of UFPD arcs on the Gaussian sphere takes time $O(n)$ (see [21]). Every point on the longitudinal great circle of a direction on these arcs, including the arc endpoints, represents a UFPD for the extrusion. Finding the minimum distance l_{min} for a non-UFPD of the 2D profile takes time $O(n \log n)$ using the algorithm detailed in Section 5.1. Since the number of such non-UFPDs that need to be tested is $O(n)$, finding the minimum distances for all of them takes $O(n^2 \log n)$. Therefore, the running time for the full algorithm is $O(n^2 \log n)$.

5.1 Finding the Minimum Distance with Respect to a Direction

In order to find the minimum distance l_{min} with respect to $\boldsymbol{d_g}$ efficiently, we adapt the plane sweep algorithm for line segment intersection detection presented in [26]. One of the main data structures is an event queue Q, which stores all events that will change the l_{min} calculation as the sweep line L sweeps from left to right on the plane of the 2D generator profile. The events occur when L crosses one or more vertices of the 2D profile. Initially, each vertex of the 2D generator profile is pushed onto Q, sorted by x-coordinate. When several vertices have the same x-coordinate, only one event is added to Q. For each event, in addition to storing the vertices at this event, we also store all edges with endpoints at these vertices. The second data structure is a status tree T, which stores a list of edges that intersect the current sweep line L. These edges are sorted by the y-coordinates of their intersections with L. Both data structures use balanced binary search trees.

Events are handled as follows. We first update T by adding the edges that have vertices on L and lie to its right. Then we check each vertex on the sweep line. If a vertex V is concave and its incident edges lie on the same side of L (Fig. 6(a)), we set the minimum distance l_{min} to 0 and exit. If V is convex and its incident edges lie on the same side of L, we search T to get the edge directly above (resp. below) and project V vertically onto the edge (Fig. 6(b)). Denote the projected vertex U_1 (resp. U_2). If the length of the shorter vertical line segment (either VU_1 or VU_2) is less than the current value of l_{min}, update l_{min}. Otherwise if the edges incident on V lie on different side of L (Fig. 6(c)), we project V vertically onto the edge directly above or below (but not both) depending on whether the normals of the edges incident on V point upward or downward. If the normals

Algorithm 1. FindingUFPDs()

Input : P, 2D generator profile for an extrusion;
h, extrusion height.
Output: Set of arcs A on the Gaussian sphere equator whose longitudinal great circles are UFPDs;
set of points D on the equator and their associated UFPD angles Φ.

2D undercut analysis on P to find UFPD arcs A on the equator of the Gaussian sphere.
foreach *direction* $\boldsymbol{d_g}$ *on equator parallel to one or more edges of* P, *if* $\boldsymbol{d_g}$ *not in* A **do**
  l_{min} =MinDistance($\boldsymbol{d_g}$, P).
  if $l_{min} > 0$ **then**
    $\phi = \tan^{-1}(l_{min}/h)$.
    Add ($\boldsymbol{d_g}$, ϕ) to (D, Φ).
end
Return (A, D, Φ).

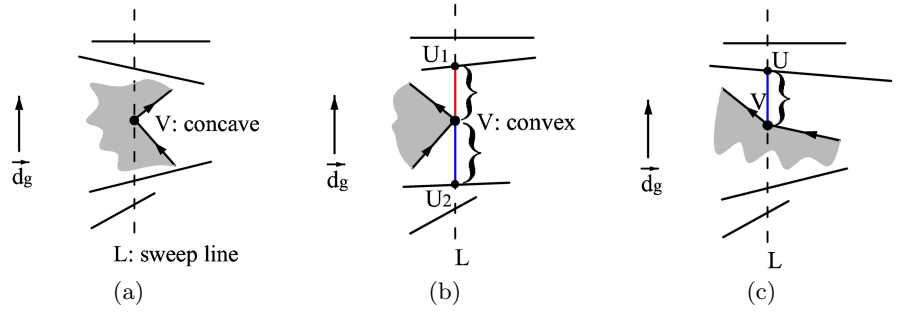


Fig. 6. (a) V is concave and incident edges lie on same side of L; (b) V is convex and incident edges lie on same side of L; (c) incident edges of V lie on different sides of L

point upward, the edge directly above is used; otherwise, the edge directly below is used. We then compare the distance between V and the projected point with the current value of l_{min} and update it. After processing the vertex event, we remove the edges on T that have vertices on L and lie to its left.

To use the adapted plane sweep algorithm described above, we need to rotate the 2D generator profile so that the test direction $\boldsymbol{d_g}$ is aligned with the y-axis. The rotation takes $O(n)$ time. Building the event queue, a binary search tree, takes $O(n \log n)$ time. An event query operation takes $O(\log n)$ time, with one query for each of the $O(n)$ events. Hence event query operations for the overall plane sweep algorithm take $O(n \log n)$ time. Each of the $O(n)$ edges is inserted into and removed from the status tree T only once, with $O(\log n)$ time for each operation. Thus the operations on T also take $O(n \log n)$ time. Therefore, the full plane sweep algorithm takes $O(n \log n)$ time.

5.2 Relaxed Version for Finding Candidate UFPDs

Finding the minimum distances for non-UFPDs of the 2D generator profile is the only step that takes $O(n^2 \log n)$ time, and hence dominates the running time for the full algorithm. If the algorithm will be used only for preprocessing a geometry that is more complex than a single extrusion in order to reduce the search space for a complete algorithm, even if a non-UFPD for an extrusion is passed on as a candidate UFPD, the complete algorithm will be able to classify it as a non-UFPD. Therefore, if the algorithm is to be used for preprocessing it may be more efficient not to calculate the minimum distances, but instead pass on all points on the longitudinal great circle of a non-UFPD that is parallel to one or more edges on the 2D generator profile. There are at most n such longitudinal great circles. This relaxed version for calculating a slightly conservative superset of all UFPDs for an extrusion only takes $O(n)$ time.

6 Summary and Conclusion

Finding all UFPDs at interactive speeds gives designers maximum flexibility choosing a parting direction early in the design process, when redesign cost is lowest. In this paper, we show how to find all UFPDs for an extrusion feature via analyzing its 2D generator profile. For a UFPD $\boldsymbol{d_g}$ of the 2D profile, every point on the longitudinal great circle of $\boldsymbol{d_g}$ represents a UFPD for the extrusion. For a non-UFPD of the 2D profile, if it is parallel to one or more edges on the 2D profile, portions of its longitudinal great circle represent UFPDs; otherwise no point on its longitudinal great circle is a UFPD. These UFPDs on the great circular arcs were previously found only by complete algorithms for arbitrary parts that took at least $O(n^4)$ time. However, such UFPDs are sometimes the only mutually 2-moldable directions for a part containing multiple extruded features, as shown in Fig. 7. Using the previous incomplete algorithms, no UFPDs would be found for this part, since neither the axes, edge directions, nor face normals are UFPDs, and there is a large connected "pocket" that the convex hull difference approaches would not be able to resolve. Using our algorithm (see Fig. 8), we can find several UFPDs, which allows us to avoid undercuts, by calculating the Boolean intersection of the sets of UFPDs for each extrusion. Taking directions in the intersection as candidate UFPDs for the full geometry of the part and verifying using Khardekar et al.'s complete algorithm [24] confirmed that this intersection is exactly all the UFPDs for this particular example.

In general, the intersection of UFPDs for extrusion features will be a superset of the UFPDs for a part containing the extrusions. Thus testing all the directions in the intersection region will still be necessary for verification, but the search space will often be significantly smaller than the entire Gaussian sphere that previously had to be considered by complete algorithms. If the intersection is empty, the part can be immediately identified as non-2-moldable without further testing; in this case, designers can either go back to redesign the part geometry or choose an optimal parting direction based on other criteria such as the number

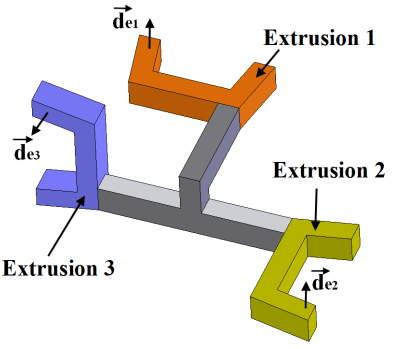


Fig. 7. A part containing three identical translated and rotated extrusion features

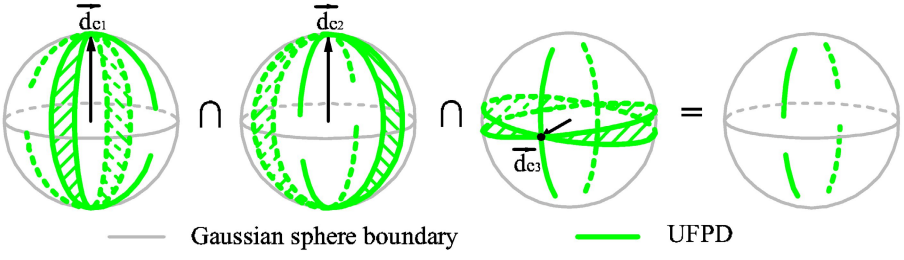


Fig. 8. The UFPDs for each extrusion and their Boolean intersection

of undercuts and undercut volume. Our exact solution for finding UFPDs for an extrusion takes $O(n^2 \log n)$ time and the relaxed version takes only $O(n)$ time.

The algorithm can be easily extended to 2D generator profiles with concave curved edges. Since each point on a concave curve is a concave vertex, the minimum distance along the tangential direction at that point is zero (see Fig. 5(b) and (c)). For convex curved edges, extension will be more complex.

Whether undercuts exist is not the only criteria when choosing an optimal parting direction for a complex geometry. Other factors, such as the complexity of the parting surface, also play an important role [1,27]. Generally UFPDs are the preferred parting directions. But designers and manufacturers may choose non-UFPDs with planar parting surfaces instead if all UFPDs require non-planar parting surfaces. Our future work aims to define optimal parting directions for an arbitrary geometry based on multiple criteria. We will also study finding all UFPDs for other feature types such as revolved, swept, or lofted surfaces.

Acknowledgement

This research was supported in part by MICRO 05-066, NSF CAREER Award DMI 0547675, the Hellman Family Foundation, and the Prytanean Alumnae Faculty Award. Thanks also to the reviewers for their valuable comments.

References

1. Ravi, B., Srinivasan, M.N.: Decision criteria for computer-aided parting surface design. Computer-Aided Design **22**(1) (1990) 11–18
2. Wong, T., Tan, S.T., Sze, W.S.: Parting line formation by slicing a 3D CAD model. Engineering with Computers **14**(4) (1998) 330–343
3. Chen, Y.H.: Determining parting direction based on minimum bounding box and fuzzy logics. Int. J. Mach. Tools Manufact. **37**(9) (1997) 1189–1199
4. Hui, K.C., Tan, S.T.: Mould design with sweep operations - a heuristic search approach. Computer-Aided Design **24**(2) (1992) 81–91
5. Hui, K.C.: Geometric aspects of the mouldability of parts. Computer-Aided Design **29**(3) (1997) 197–208
6. Chen, L.L., Chou, S.Y., Woo, T.C.: Parting directions for mould and die design. Computer-Aided Design **25**(12) (1993) 762–768
7. Woo, T.C.: Visibility maps and spherical algorithms. Computer-Aided Design **26**(1) (1994) 6–16
8. Chen, L.L., Chou, S.Y.: Partial Visibility for Selecting a Parting Direction in Mold and Die Design. Journal of Manufacturing Systems **14**(5) (1995) 319–330
9. Weinstein, M., Manoochehri, S.: Geometric Influence of a Molded Part on the Draw Direction Range and Parting Line Locations. Journal of Mechanical Design **118**(3) (1996) 29–39
10. Wuerger, D., Gadh, R.: Virtual prototyping of die design. Part one: Theory and formulation. Concurrent Engineering : Research and Applications **5**(4) (1997) 307–315
11. Wuerger, D., Gadh, R.: Virtual Prototyping of Die Design. Part Two: Algorithmic, Computational, and Practical Considerations. Concurrent Engineering : Research and Applications **5**(4) (1997) 317–326
12. Ha, J., Yoo, K., Hahn, J.: Characterization of polyhedron monotonicity. Computer-Aided Design **38**(1) (2006) 48–54
13. Requicha, A.A.G.: Representations for Rigid Solids: Theory, Methods, and Systems. ACM Computing Surveys **12**(4) (1980) 437–464
14. Ganter, M.A., Skoglund, P.A.: Feature extraction for casting core development. In: 17th Design Automation Conference presented at the 1991 ASME Design Technical Conferences, Miami, FL, American Society of Mechanical Engineers (1991) p 93–100
15. Fu, M.W., Fuh, J.Y.H., Nee, A.Y.C.: Generation of optimal parting direction based on undercut features in injection molded parts. IIE Transactions **31**(10) (1999) 947–955
16. Fu, M.W., Fuh, J.Y.H., Nee, A.Y.C.: Undercut feature recognition in an injection mould design system. Computer-Aided Design **31**(12) (1999) 777–790
17. Ye, X.G., Fuh, J.Y.H., Lee, K.S.: A hybrid method for recognition of undercut features from moulded parts. Computer-Aided Design **33**(14) (2001) 1023–1034
18. Yin, Z., Ding, H., Xiong, Y.: Virtual prototyping of mold design: geometric mouldability analysis for near-net-shape manufactured parts by feature recognition and geometric reasoning. Computer-Aided Design **33**(2) (2001) 137–154
19. Rappaport, D., Rosenbloom, A.: Moldable and castable polygons. Computational Geometry: Theory and Applications **4**(4) (1994) 219–233
20. Bose, P., Bremner, D.: Determining the Castability of Simple Polyhedra. Algorithmica **17**(1-2) (1997) 84–113

21. McMains, S., Chen, X.: Finding undercut-free parting directions for polygons with curved edges. ASME Journal of Computing and Information Science in Engineering **6**(1) (2006) 60–68
22. Ahn, H.K., de Berg, M., Bose, P., Cheng, S.W., Halperin, D., Matousek, J., Schwarzkopf, O.: Separating an object from its cast. Computer-Aided Design **34**(8) (2002) 547–59
23. Elber, G., Chen, X., Cohen, E.: Mold Accessibility via Gauss Map Analysis. Journal of Computing and Information Science in Engineering **5**(2) (2005) 79–85
24. Khardekar, R., Burton, G., McMains, S.: Finding Feasible Mold Parting Directions Using Graphics Hardware. Computer-Aided Design **38**(4) (2006) 327–341
25. Kurth, G.R., Gadh, R.: Virtual prototyping of die-design: determination of die-open directions for near-net-shape manufactured parts with extruded or rotational features. Computer Integrated Manufacturing System **10**(1) (1997) 69–81
26. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational geometry: algorithms and applications. Springer, New York (2000)
27. Boothroyd, G., Dewhurst, P., Knight, W.: Product design for manufacture and assembly. M. Dekker, New York (2002)

Robust Three-Dimensional Registration of Range Images Using a New Genetic Algorithm

John Willian Branch[1], Flavio Prieto[2], and Pierre Boulanger[3]

[1] Escuela de Sistemas, Universidad Nacional de Colombia – Sede Medellín
jwbranch@unal.edu.co
[2] Departamento de Eléctrica, Electrónica y Computación,
Universidad Nacional de Colombia – Sede Manizales
faprietoo@unal.edu.co
[3] Department of Computing Science – University of Alberta, Canada
pierreb@cs.ualberta.ca

Abstract. Given two approximately aligned range images of a real object, it is possible to carry out the registration of those images using numerous algorithms such as ICP. Registration is a fundamental stage in a 3D reconstruction process. Basically the task is to match two or more images taken in different times, from different sensors, or from different viewpoints. In this paper, we discuss a number of possible approaches to the registration problem and propose a new method based on the manual pre-alignment of the images followed by an automatic registration process using a novel genetic optimization algorithm. Results for real range data are presented. This procedure focuses, on the problem of obtaining the best correspondence between points through a robust search method between partially overlapped images.

Keywords: Registration, range image, ICP algorithm, normal, genetic algorithm.

1 Introduction

The misalignment that is unavoidably produced when two or more images have been taken from different views, and without any control of the relative positions of the sensor and the object, becomes the central problem of registration. The purpose of the registration process is to align these views in such a way that the object's shape is recovered with the highest precision.

For a little more than one decade, with the introduction of the ICP Algorithm [1] there have been many variations to mitigate its deficiencies. This algorithm formulated a basic schema to obtain the alignment while minimizing the cost function and is based on the squares summation of the distance between points on the image.

Another approach to the registration of images consists of determining a set of matches through a search process instead of the classical approach based on distances.This approach consists in finding a solution close to the global minimum in a reasonable time. This can be done by means of a Genetic Algorithm (GA).

We propose a procedure based on a Genetic Algorithm for the registration of a pre-aligned image pair. This procedure focuses on the problem of obtaining the best

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 528 – 535, 2006.
© Springer-Verlag Berlin Heidelberg 2006

match between points through a robust search method on images that are partially overlapped. This set of matches allows the calculation of transformation which precisely registers the images.

This paper is organized as follows: Section 2 presents a literature review. Section 3 describes the methodology used to do the registration of a pre-aligned images pair using a Genetic Algorithm. Section 4 presents experiments realized, and in the Section 5, the conclusions of this work are presented.

2 Literature Review

Genetic algorithms have already been used for registration of 3D data. In their recent survey on genetic algorithms in computer-aided design [2], Renner and Ek´art mention a few related studies. In particular, Brunnstrom and Stoddart [3] proposed a method that integrates the classical ICP method with a genetic algorithm to couple free form surfaces. Here an alignment is obtained with a genetic algorithm, which is later refined with the ICP. The main problem treated by Brunnstrom and Stoddart is to find a corresponding set of points between the two views. Robertson and Fisher [4] proposed a parallel genetic algorithm which reduces the computational time, but its solution is not more accurate than the ones obtained with the first method. Silva et al [5] proposed a method for the registration of range images, making two key contributions: the hybridization of a genetic algorithm with the heuristic optimization method of hill climbing, and a measurement of the performance of the interpretation of the surfaces different from the classical metric, based on the calculation of the mean square error between corresponding points on the two images after the registration. Yamany et al. [6] used a genetic algorithm for registration of partially overlapping 2D and 3D data by minimising the mean square error cost function. The method is made suitable for registration of partially overlapping data sets by only considering the points such that $p_i \in G_1 \cup G_2$, where G_1 and G_2 are space bounding sets for the two datasets. Unfortunately, the authors give very few details about their genetic algorithm, focusing on the Grid Closest Point transformation they use to find the nearest neighbour. Salomon et al. [7] apply a so-called differential evolution algorithm to medical 3D image registration. Differential evolution uses real-valued representation and operates directly on the parameter vector to be optimised. Otherwise, only the reproduction step is different from GA. On the other hand, this method requires much more computation than the simpler algorithm we use. In [7], differential evolution is used to register two roughly pre-aligned volumetric images of small size. The relative rotation is within ±20°, which is comparable to the range our TrICP can already cope with. We need and propose a preregistration algorithm that can cope with arbitrary orientations.

A recent study on the use of genetic algorithms for range data registration appeared in [8]. Euclidean parameter space optimization is considered. To handle partially overlapping data, the median of the residuals is used as error metric. This improves the robustness but renders the method inapplicable to data overlaps below 50%. An advanced operator of dynamic mutation is introduced, which reportedly improves registration error and helps avoid premature convergence. An attempt is made to improve the precision by using dynamic boundaries. After the GA has converged, the

search space is reduced and the GA is applied again. However, using genetic algorithms for precise registration does not seem reasonable since faster and more precise iterative methods exist for registration of pre-aligned datasets.

3 Registration of a Pre-alignment Image Pairs to a 3D Surface Model Using Genetic Algorithms

The views to be registered are pre-aligned in order to obtain an initial overlapping area in both images. As it can be seen in Figure 1 the following steps for each point of sample size N taken in the overlapping area of one of the views, has a correspondent point that is searched for around the nearest points of the other view to be registered. This search is done because the best couple of points to obtain a transformation using Horn's method [9] are not always the points with less distance within an overlapping area. Two views could be badly aligned and present points with very short distances; however when joining the views using these points as a guide, their registration could be off. The initially pre- aligned images could be askew and the correspondent points with which the views would match best when applying a transformation, could be very close to the points with a minimum distance.

Given two images of ranges A and B where A is the image model and B is the image to be registered, searching the best points in A that match with a sample of points selected in B, is done by a genetic algorithm. The design is as follows.

3.1 Sampling

It is a random selection of *N* points that belongs to the overlapped area in B and establishes, for each one of them, a subset of points or sub-domain in A. The sub-domains contain *m* points near the closest point in A for each point in B. This approach of sub-domains reduces the space search and betters the global efficiency of the algorithm. The establishment of the domains has a critical computational step; that is, searching the closest point in A to each one of the points of the selected sample in B because this implies both calculating and comparing the distances to all the points which make up the overlapping area in A. Such a search is improved by implementing a K-d tree structure. Figure 1 graphically shows the establishment of a sub-domain.

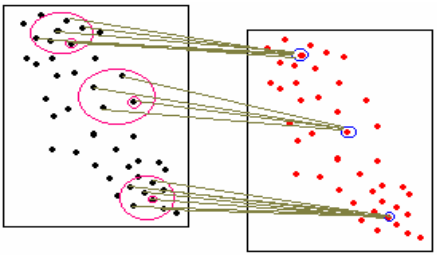


Fig. 1. Establishment of sub-domains, a) view A, b) view B

3.2 Diagram of Representation

It is represented as a chromosome of size N, that is, to each one of the points of the selected samples in view B there is a corresponding gene of the chromosome. Each gene contains an index that identifies a point within the neighborhood corresponding to a point as defined in view A. Figure 2 illustrates this representation.

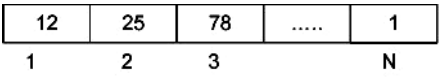


Fig. 2. Diagram of representation of a chromosome

Gene 1 corresponds to the first point of the sample, whereas gene 2 corresponds to the second point of the sample and subsequently to the N-th point of the sample taken in view B. For instance, in Figure 2 gene 1 contains value 12, which means that point 12 is found within the sub-domain corresponding to the first point of the sample in B. Twenty-five (25) is an index of a point-from-view A that belongs to a neighborhood of points close to point 2 of the sample taken in view B. Each point of the sample taken in view B has a defined neighborhood of points in view A from which the respective gene will take values.

3.3 Aptitude Function

The aptitude function measures the standard deviation of the distribution of the distances between the points of the overlapping areas originating in the registration of the views. Each individual can be seen as a set of points with their respective couples translated into a transformation by Horn's method. The transformation is applied to the two views and the standard deviation of this registration is assigned as the aptitude of an individual. The more accurate the individual, the smaller the error (1):

$$\varepsilon = \sqrt{\frac{\sum_{i=1}^{N} (P_i - R_i)^2}{N}} \tag{1}$$

Parameter P denotes each point in the overlapping area in view A obtained by applying each transformation. Parameter R is each point in the overlapping area in view B after applying the transformation.

3.4 Genetic Operators

The proposal presented for a two-view registration applies a simple cross with only one crossing point, in which the parents' genetic content is exchanged on each side of the crossing point in order to generate two new individuals. In turn, the mutation operator varies the information of each gene according to the mutation probability, taking into account the defined neighborhoods for each point represented. That is, if

gene *i* represents the *i-th* value of the sample taken in view B, and it has to be mutated, a respective point in the defined neighborhood is selected at random in view A, and it is changed by the former value.

4 Experiments and Results

All tests were performed using a computer with a 3.0G Intel processor and 1.0G RAM memory, running under the Microsoft XP operating system. The program was written in C++ and using Open GL to obtain the graphic representation of images. The data used were obtained using a Kreon sensor located at the Advanced Man-Machine Interface Laboratory at the University of Alberta, Canada.

The least average results were obtained with a 40% probability for the crossing operator and a 70% probability for the mutation operator. The size of the population was established to be 100 individuals, each one of which is formed by 10 pairs of points. Due to the fact that the GA model works on a specific problem, in order to find the best relationship between points that allows a transformation that correctly registers a pair of images with the objective of validating the correct performance of the methods ICP, ICP+Normals [10] and ICP+GA (See Figure 3), tests were performed to assure a point-to-point correspondence between the images, guaranteeing the existence of a unique solution to the problem. The convergence test was carried out fixing an error and running the method iteratively until it reached a convergence of 1x10E-6. The results of these experiments showed that the ICP+GA method converges more quickly as it is observed in the Figures 4 and 5.

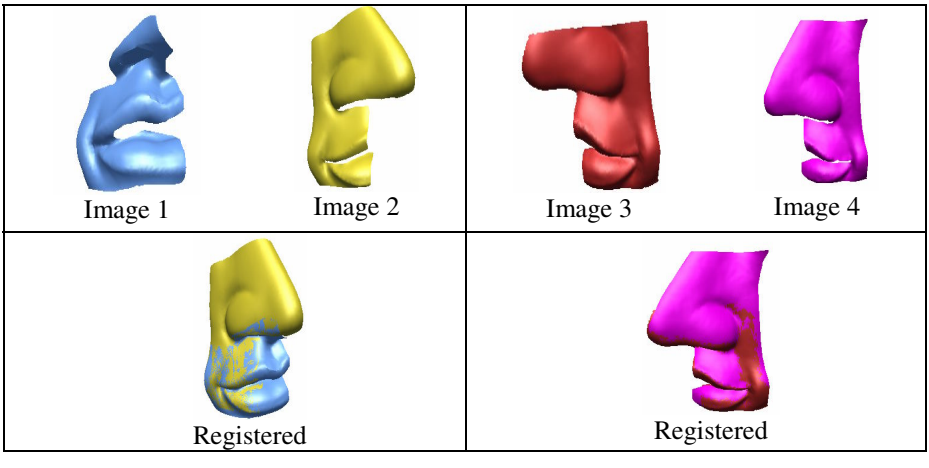


Fig. 3. Sample images 1 to 4 and their corresponding registration

Tests were made to compare the methods in the quality of the final registration using bad and good pre-alignments as benchmarks. The objective was to even measure the behavior of convergence of the methods when the images were rotated into their correct positions. Each test was carried out, keeping in mind that although

the were sufficiently rotated as to considered the pre-alignment like bad, there was no translation because of the neighborhoods of searches were constructed based on measures of Euclidean distance. In order to generalize the behavior in the final values of the registration were carried out 20 registrations with similar conditions of bad pre-alignment. Although, in some cases the methods not obtain values significantly different, one can observe that in general the method ICP +GA obtains the smallest error values that the other methods.

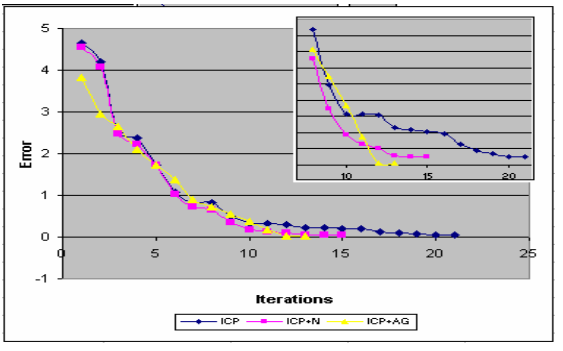


Fig. 4. Convergence Test 2 of the methods (Convergence=1x10E -6) with images 1 and 2. (ICP = 21, ICP+N= 15, ICP+AG=13 iterations)

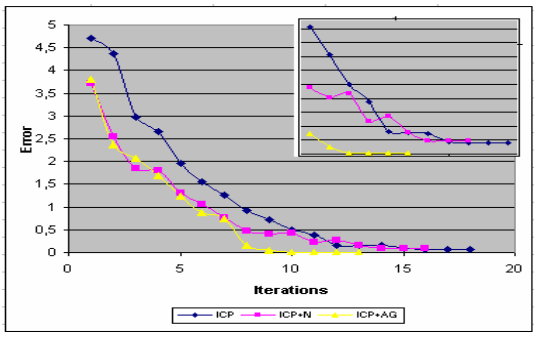


Fig. 5. Convergence Test 3 of the methods (Convergence=1x10E -6) with images 3 and 4. (ICP = 18, ICP+N= 16, ICP+AG=13 iterations)

Additionally, the robustness of the method was proven, as it determined the maximum value of the angle for which the different methods converged in a good registration. Figure 6 shows the errors for different registrations with the variations in angles of each coordinate. The different methods obtain a correct registration for angles less than 40 degrees. For these cases the ICP+AG always produces the best registration. For angles greater than 40 degrees, the traditional methods present a significant increase in the error and we consider that it is not possible to reach a

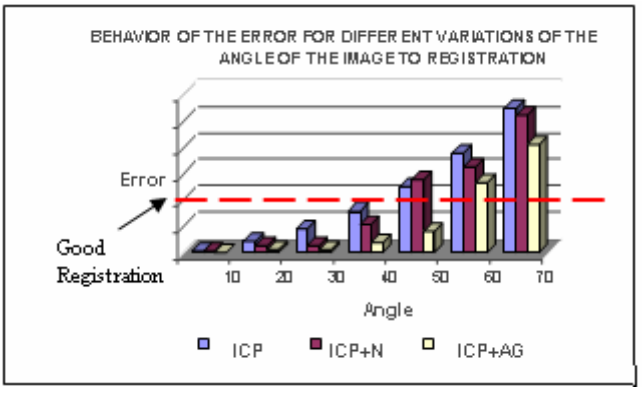


Fig. 6. Error of registration for different uniform variations in the angle. Dashed line shows the good pre-alignment limit.

correct registration for angles greater than 40 degrees. However the ICP+AG provide a good registration up to 50 degrees. Although the error continues to decrease after that point when using the other methods, there still is not correct registration.

5 Conclusions and Future Work

A semiautomatic method has been proposed for the registration of multiple view range images with low overlap that is capable of finding an adequate registration without needing a fine preliminary pre-alignment of the images. This method is based on a genetic algorithm to perform a search of the best correspondence between a set of sample points, starting from an approach based on sub-domains that reduces the space search of the genetic algorithm which implies global algorithm efficiency.

The comparison of the results obtained through the different experiments shows a more precise convergence (using proposed method (ICP+GA) than the classical ICP method and one of its variants (ICP+Normals) can provide. However, the proposed method takes more computational time to find the solution.

For future work, the exploration of a parallel version to reduce the computational cost of the proposed method is suggested.

References

1. Besl, P.: A method for Registration of 3-D Shapes. IEEE Trans. Pattern Analysis and Machine Intelligence, 14 (1992).
2. Renner, G. Ek´art, A. Genetic Algorithms. Computer-Aided Design, (2003) 709–726.
3. Brunnstrom, K.: Genetic Algorithms for Free-Form Surface Matching. (1996)
4. Robertson, C.: Parallel Evolutionary Registration of Range Data. Computer Vision and Image Understanding, 87 (2002).

5. Silva, L.: Precision Range Image Registration Using a Robust Surface Interpretation Measure and Enhanced Genetic Algorithm. IEEE Trans. Pattern Analysis and Machine Intelligence, 27 (2005) 762-776.
6. Yamany, S.: New Genetic-Based Technique for Matching 3D Curves and Surfaces. Pattern Recognition, 32 (1999) 1817–1820.
7. Salomon, M.: Differential Evolution for Medical Image Registration. International Conference on Artificial Intelligence, (2001) 201–207.
8. Chow, C.: Surface Registration Using a Dynamic Genetic Algorithm. Pattern Recognition, 37 (2004) 105–117.
9. Horn, B. K. P.: Closed-Form Solution of Absolute Orientation. J. Opt. Soc. A, 4 (1987) 629-642.
10. Chen, Y.: Object modeling by registration of multiple range images. Image and Vision Computing, 10 (1992).

Geometrical Mesh Improvement Properties of Delaunay Terminal Edge Refinement

Bruce Simpson[1] and Maria-Cecilia Rivara[2]

[1] David Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1
rbsimpson@uwaterloo.ca
[2] Department of Computer Science, Universidad de Chile, Blanco Encalada 2120, Santiago, Chile
mcrivara@dcc.uchile.cl

Abstract. The use of edge based refinement in general, and Delaunay terminal edge refinement in particular are well established for adaptive meshing, but largely on a heuristic basis. In this paper, we present some theoretical results on geometric improvement, and it limitations, for these methods. Angle bounds for simple longest edge bisection are reviewed and extended. Terminal edges are local maximal edges in a mesh; two additional bounds that apply to simple bisection of terminal edges in Delaunay meshes are presented. The angle properties of Delaunay insertion of the midpoint of a terminal edge are described.

1 Introduction

Delaunay terminal edge refinement, specified in §2 below, is a member of the family of edge-based adaptive mesh refinement methods; references to which can be found in [1, 5, 12]. The use of edge based refinement in general, and Delaunay terminal edge refinement in particular are well established for planar meshing, but largely on a heuristic basis. In this paper, we present a series of theoretical results on the geometric mesh improvement properties of these methods. Iterative refinement methods for generating such meshes typically take a triangulation of D, M_0, as input. M_0 is not connected to any approximation task necessarily. It is simply a representation of D and may have arbitrarily small angles and/or edges. For some applications, the merit of an unstructured mesh for discretizing a domain D is influenced by the geometric quality of its triangles, e.g. Berzins, [2]. Part of the task of generating a mesh is to improve these measures, which typically involves better aspect ratios in the triangles. It is well known that for the goals of efficient and appropriate meshes for piecewise linear approximation, the measures of length and angles should be made in an error based metric, e.g. George and Borouchaki [4], or Simpson [12].

Strong mesh improvement properties for Delaunay circumcenter based refinement have been established by Chew, [3], Ruppert [6], and Shewchuk [9]. In particular, under appropriate conditions on D, the methods are guaranteed to produce meshes with the minimum angle larger that a specified angle tolerance.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 536–544, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Our discussion presents, in §2, an overview of triangle properties local to Delaunay terminal edge bisection, including some new results. We will usually shorten 'Delaunay terminal' to 'Deter' in the sequel. The terminal edge concept is explained in §2.2. In §3, we analyse the angle distribution in the mesh resulting from a Deter bisection and study a case of repeated Delaunay terminal edge refinement of a triangle with one small edge.

2 Local Features of Deter Edge Bisection

We first look at the properties of the angles produced in a simple longest edge bisection of t. We usually use 'LEBis' for this bisection. Individual properties have been reported in a variety of references, [7, 5, 8]. In §2.1, we believe we have included all previously published properties, provided some simpler proofs for some cases, and added new properties in Theorem 2.1 b) and c) and Theorem 2.2.

In §2.2, we explain the components of Deter refinement and our terminology for them. We then present two bounds on the angles in the pairs of triangles incident on a Delaunay terminal edge.

2.1 Basic Properties of LEBis

We introduce a standardized notation for this splitting by labeling the vertices of t as A, B, C and normalizing this labeling by requiring

$$|B - C| \ \leq \ |C - A| \, ; \ |C - A| \ \leq \ |B - A| \, ; \ M = (A + B)/2$$

where M is the where M is the midpoint of the longest edge that is to be split. The two new child triangles of t are labelled t_A and t_B, and the angles of t_A (t_B) are labelled α_j (β_j) ; for $j = 0, 1, 2$.

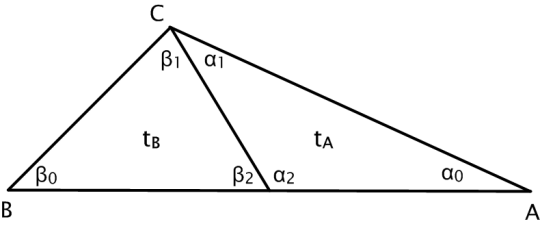


Fig. 1. Notation for longest edge bisection

The following lemma and theorems present some simple properties of a LEBis of any t.

Lemma 2.1. *Each of the assertions in the following groups is equivalent to any other in the group.*

a)	*b)*	*c)*
t is right angled	*t is acute*	*t is obtuse*
$\alpha_1 = \alpha_0$	$\alpha_1 < \alpha_0$	$\alpha_1 > \alpha_0$
$\beta_1 = \beta_0$	$\beta_1 < \beta_0$	$\beta_1 > \beta_0$
$\lvert A - M\rvert = \lvert C - M\rvert$	$\lvert A - M\rvert < \lvert C - M\rvert$	$\lvert A - M\rvert > \lvert C - M\rvert$

Theorem 2.1. *The following angle bounds apply*

$$a) \quad \alpha_1 \geq \alpha_0/2 \quad ; \quad b) \quad \beta_1 \geq \pi/6 \quad ; \quad c) \quad \beta_2 \geq 3\alpha_0/2$$

Theorem 2.2. *The following angle bounds apply conditionally*

a) if t is obtuse, then $\beta_2 \geq 2\alpha_0$
b) if $\alpha_0 < \pi/6$,then $\beta_1 > min(\beta_0, \beta_2)$
c) if $\alpha_0 > \arcsin(1/3) = 19.5°$, then $\beta_1 < \pi/2$, i.e. t_B is acute.

2.2 Deter Edge Bisection

One of the tactics in refinement for geometric improvement is to refine the largest triangles first. If the size measure for largest is the length of the longest edge, then refining terminal edges of the mesh is an approximation to using the longest first ordering. A terminal edge in a mesh is a local maximum edge length in a graph sense. Figure 2, a) shows edge AB as an example of an internal terminal edge with two neighbouring triangles, t_2, t_3 and b) shows edge CD as a boundary terminal edge with one neighbouring triangle. This, and the following, concepts were introduced and used in references [5, 8, 7].

We now explain the terminology of the paper. Terminal edge bisection is simple LEBis of each triangle incident on a terminal edge as described in §2.1 above. Delaunay terminal edge bisection is is a modification of terminal edge bisection in which the mesh being refined is Delaunay, or constrained Delaunay, and the insertion is a Delaunay point insertion. 'Lepp' is an acronym for ' the longest edge propagation path'. Given a triangle, t_0 that is to be refined, Lepp locates a terminal edge near t_0, in a graph sense. Figure 2 illustrates this process for the two triangles marked t_0 on the left and t_0^* on the right. Finally, Deter refinement of a triangle t will refer to finding a terminal edge associated with t using Lepp and performing Deter edge bisection on it. As these examples show, Deter refinement of t_0 may not modify t_0, in which case the process can be

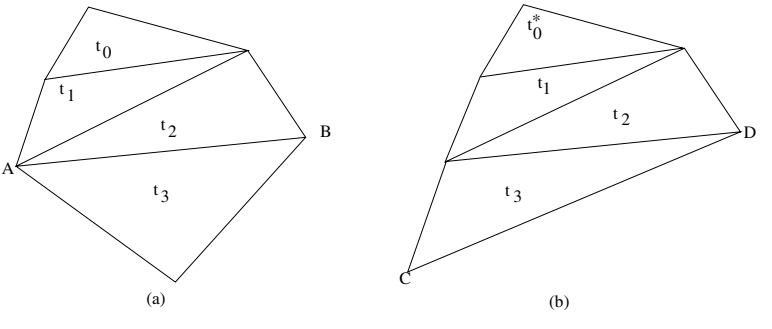


Fig. 2. (a) AB is an interior terminal edge shared by terminal triangles (t_2, t_3) associated to Lepp$(t_0) = \{t_0, t_1, t_2\}$; (b) CD is a boundary terminal edge with terminal triangle t_3 associated to Lepp$(t_0^* = \{t_0^*, t_1, t_2, t_3\}$

repeated in the refined mesh. Algorithmic details of Deter refinement, including repeated application to a given t, are given in [5, 8, 7, 13].

Our discussion of simple LEBis applies to terminal boundary edges of a mesh in full generality. However, special 'encroachment' rules are needed to ensure mesh improvement for refining boundary edges, [3, 6]. In this paper, we will restrict our attention to refinement of internal edges. The fact that the neighbouring triangles, t_1 and t_2 of an internal terminal edge are a Delaunay pair places significant restrictions their configuration. So, for an internal edge, Delaunay terminal edge bisection is more constrained than terminal edge bisection applied to two independent triangles separately. This can been seen in Figure 3.

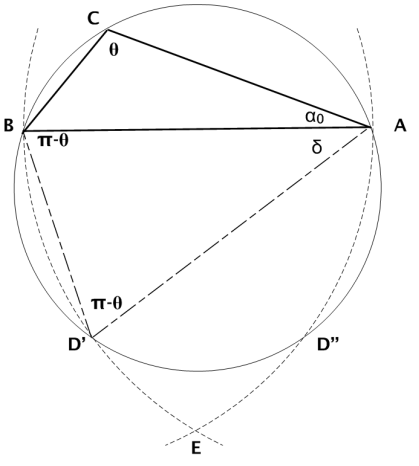


Fig. 3. The configuration of triangles at an internal terminal edge

The figure shows only t_1; we denote the vertex of t_2 opposite edge AB by D, which is not shown. The dashed circular arcs are part of the circles of radius $|B - A|$ centered at A and B respectively. We will use $CC(t)$ to denote 'the circumcircle of t'. $CC(t_1)$ is shown with a solid perimeter. Because edge AB is terminal and the triangles are Delaunay, D must lie in the small region at the bottom of the diagram below the short arc of $CC(t_1)$ from D' to D'' and inside the two dashed arcs that meet at E.

Simple implications of this diagram are the following lemma and corollary.

Lemma 2.2. *For any pair of Delaunay terminal-triangles t_1, t_2 sharing an internal terminal edge, largest angle(t_i) $\leq 2\pi/3$ for $i = 1, 2$.*

Corollary 2.1. *For child triangle t_A of LEBis in a Delaunay mesh, if $\max(\alpha_0, \alpha_1) < \pi/6$ then edge CA is not a terminal edge.*

For a pair of triangles (t_1, t_2) sharing a Delaunay edge, the sum of the angles opposite the common edge cannot exceed π, Consequently, at most one of the t_k can be obtuse. The following theorem shows that if the edge is a terminal edge, then the more obtuse t_1 is, the larger the smallest angle of acute t_2 is.

Theorem 2.3. *Let t_1 and t_2 be incident on an internal terminal edge. Let t_1 be obtuse, with largest angle $\theta > \pi/2$ and let $\alpha_0(2)$ be the smallest angle of t_2. Then*

$$\alpha_0(2) \geq 2\theta - \pi$$

This theorem, and Figure 3 illustrate restrictions on the configuration of triangles that share a terminal edge, e.g. if $\theta = 7\pi/12$, then $\alpha_0(2) \geq \pi/6$.

Intuitively, as Figure 1 suggests, LEBis produces an improved triangle,t_B, and a triangle,t_A which is not improved. So mesh improvement can depend on subsequent processing of t_A. It may happen that the Delaunay insertion of M removes t_A from the mesh. The implications of of this possibility are discussed in the next section. If not, i.e. if edge AC is an internal Delaunay edge , it may not be a terminal edge. Intuitively, it would be expected that the configurations of the two triangles incident on edge AC would not commonly meet the conditions presented above for it to be a terminal edge, in general. Corollary 2.1 is a particular instance of this. So, in general, it would be expected that repeated Deter refinements of t_A would, sooner or later, result in edge AC being removed from the mesh by a Delaunay insertion following the bisection of some other nearby terminal edge.

3 Mesh Properties of Deter Edge Refinement

We now look at properties of Deter edge bisection associated with the region of the mesh affected by the refinement. We start, in §3.1, with a study of the angles in the updated mesh, assuming that t_A is not Delaunay in the mesh resulting from a LEBis of t. However, it may be that no update is necessary. , i.e. that t_A is already Delaunay in the refined mesh. The second subsection discusses a configuration that that could be applicable to repeated Deter refinement in this case.

3.1 Delaunay Insertion of Point *M*

We describe the Deter edge bisection process as the simple LEBis of each trian gle, t, incident on the terminal edge, which results in an updated mesh M_{SB}, followed by the conversion of M_{SB} to a Delaunay mesh, M_{CD}.

To describe the conversion of M_{SB}, we will use the terminology of George and Borouchaki ,[4]. The cavity of the vertex M in M_{SB} is the set of triangles, t, such that $M \in \mathrm{CC}(t)$. It has a polygonal boundary that is star-shaped with respect to vertex M. We will denote the boundary vertices by P_k for $k = 0$, to N in clockwise order about M starting with $P_0 = C$. Since A, B and C are on this boundary, $N \geq 2$. The result of the Delaunay insertion of vertex M is that the triangles in the cavity of M are removed from M_{SB} and triangles MP_kP_{k+1} replace them in M_{CD}.

We let NA be the index of A in the list of boundary vertices of the cavity of M i.e. $P_{NA} = A$. The subset of the cavity of M that is bounded by the first $NA+1$ vertices and the edges AM and MC will be referred to as the partial cavity

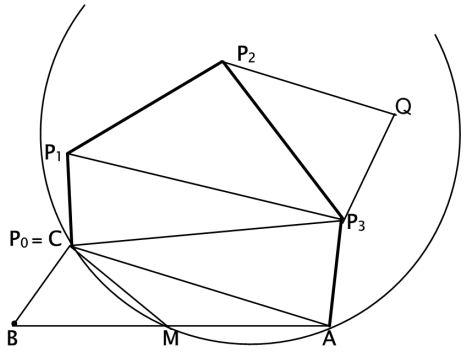


Fig. 4. Example of the partial cavity of vertex M in mesh M_{SB} with $NA = 4$

of M. An example is shown in Figure 4; this figure also shows $\mathrm{CC}(t_A)$ of child triangle $t_A = CMA$ with the P_k in its interior. This illustrates the statement of the following lemma. We have also shown a mesh vertex, Q , and triangle P_2QP_3 which are not in the cavity of M although they are in $CC(t_A)$. So the converse of the lemma is not true.

Lemma 3.1. *If $NA > 1$, P_k is in $CC(t_A)$ for $1 < k < NA$.*

We will study the angles of the new triangles incident on M. Let $\alpha_{min}(M)$ be the the minimum angle of the triangles in the partial cavity of M excluding t_A Each triangle $t \in M_{SB}$ in the cavity of M, except t_A, has vertices, P_i, P_j, P_k for $i < j < k$. If t has an edge on the boundary of the cavity, then $i = j - 1$. In this case, $M \in \mathrm{CC}(t)$ implies that the angle at M in M_{CD} opposite edge P_{j-1}, P_j is larger than the angle opposite edge P_{j-1}, P_j in t. So, in particular, the angle at M is larger than $\alpha_{min}(M)$. Intuitively, we can see that the closer a cavity edge, $P_{j-1}P_j$, is to M the larger this angle improvement will be. Conversely, if $\mathrm{CC}(t)$ is very close to $\mathrm{CC}(t_A)$ then very little angle improvement can occur.

The following theorem details the worse case limits of angle improvement. Its proof provides insight into the mechanisms of angle improvement resulting from Delaunay insertion.

Theorem 3.1. *Angle $CP_1M \geq \alpha_0$ and the other two angles of triangle CP_1M exceed $\alpha_{min}(M)$.*
Angle $MP_{NA-1}A \geq \alpha_1$ and the other two angles of triangle $MP_{NA-1}A$ exceed $\alpha_{min}(M)$.
If $NA > 2$, then in the set of triangles $P_jP_{j+1}M$ for $1 \leq j \leq NA - 2$, every angle exceeds $\alpha_{min}(M)$.

Corollary 3.1. *If t is obtuse, then no new angles smaller than the existing ones in the unrefined mesh result from Deter bisection of t.*

3.2 A Case of Repeated Delaunay Refinement

In this section, we show that if t is shaped so that $|B-C| < |C-M|$, i.e. if $|B-C|$ is the shortest edge of t_B, then Delaunay insertion of M into the mesh can only produce new edges that are longer than $|B-C|$. We then look at repeated Deter refinement applied to a special case of t.

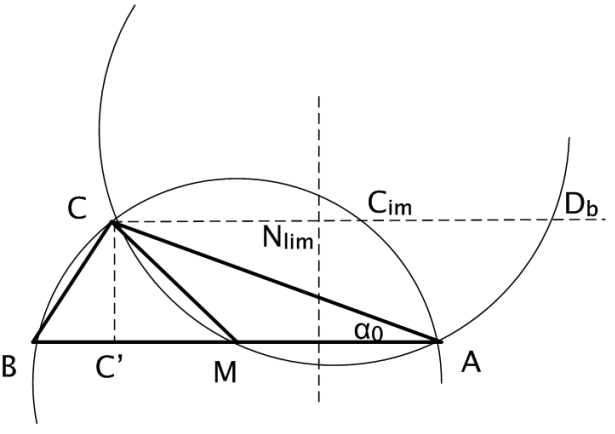


Fig. 5. Configuration of terminal triangle t and its neighbour

Figure 5 shows the terminal triangle $t = ABC$, and an arc of its circumcircle $CC(t)$. The point C' is the projection of C onto edge BA. The figure also shows the insertion point M, and an arc of $CC(t_A = CMA)$. We assume that $|C-M| > |A-M|$, and consequently, that t is acute and that $\alpha_1 < \alpha_0$. Other features of the figure are used in the proofs; see technical report [14].

Lemma 3.2. *If $|C-M| > |B-C|$, the circle of radius $|B-C|$ about M lies inside $CC(ABC)$.*

Corollary 3.2. *If $|B-C|$ is the shortest edge of t_B then Delaunay insertion of M into the current mesh can only produce edges longer than $|B-C|$.*

We now use this lemma in a theorem that demonstrates a special case of t for which we can prove that no new small edges are produced in repeated Deter refinements of t. Let D be the vertex of the triangle that shares edge CA with t. D must be outside $CC(ABC)$.

Theorem 3.2. *If $\alpha_0 \leq$ angle ACD ; $|B-C|$ is the shortest edge of t_B, and edge CA is not terminal as an edge of t_A, then the circle of radius $|B-C|$ about M is empty for repeated applications of Deter refinement to t_A.*

Corollary 3.3. *Under the conditions of Theorem 3.2, if t_B is acceptable, i.e. $\beta_2 \geq \theta_{tol}$, then no edge smaller than $|B-C|$ is introduced at M by Deter refinement of the mesh.*

4 Conclusions

Our motivation in this study has been to understand how, or in what circumstances, Deter refinement can produce a submesh near a small angled triangle t with improved triangles. There are two ways in which the method can create this submesh, either at some stage of repeated Deter refinement, the longest edge of t is a terminal edge, in which case its midpoint is Delaunay inserted into the mesh, or the longest edge is removed from the mesh by the Deter bisection of the terminal edge of Lepp(t). This paper addresses the first case. As mentioned in §2.2, simple terminal edge bisection of a small angled triangle, t, produces a demonstrably improved triangle, t_B and an unimproved triangle, t_A both incident on the new vertex M. Improvement of the submesh near t may come from the Delaunay insertion of M. Our analysis of §3.1 identifies the only circumstances under which this does not happen; i.e. t must be acute and the neighbour of t on side AC must have a special configuration. These circumstances do not preclude Deter refinement from successfully improving the mesh, of course, but suggests that it may not be possible to identify improvement on the basis of Deter refinement aplied to one small angled triangle. On the other hand, we demonstrate in §3.2, a special case in which it is possible to show at least non-degeneration of the local submesh based on properties of t and its neighbour.

References

[1] T. J. Baker, Triangulations, Mesh Generation and Point Placement Strategies. *Computing the Future,* ed. *D Caughey*,John Wiley, 61-75.

[2] M. Berzins, Mesh Quality: A Function of Geometry, Errror Estimates or Both?, *Engineering with Computers*, 15, 1999, 236-247.

[3] L.P.Chew, Guaranteed-Quality Mesh Generation for Curved Surface *9th Annual Symposium on Comp Geometry*, ACM, 1993, 274-280,

[4] P L George and H Borouchaki, *Delaunay Triangulation and Meshing.* Hermes, 1998.

[5] M. C. Rivara. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. *International Journal for Numerical Methods in Engineering*, **40**, 1997, 3313–3324.

[6] J Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. of Algorithms*, **18**, 1995, 548–585.

[7] M. C. Rivara and M. Palma. New LEPP Algorithms for Quality Polygon and Volume Triangulation: Implementation Issues and Practical Behavior. *In Trends unstructured mesh generation*, Eds: S. A. Cannan . Saigal, AMD, **220**, 1997, 1–8.

[8] M. C. Rivara, N. Hitschfeld, and R. B. Simpson. Terminal edges Delaunay (small angle based) algorithm for the quality triangulation problem. *Computer-Aided Design*, **33**, 2001, 263–277.

[9] J R Shewchuk, Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. *First Workshop on Applied Computational Geometry*, ACM, 1996, 124-133.

[10] M.C. Rivara and N. Hitschfeld,LEPP-Delaunay algorithm: a robust tool for producing size-optimal quality triangulations, *Proc. of the 8th Int. Meshing Roundtable*, October 1999, 205-220.

[11] I.G. Rosenberg and F. Stenger, A lower bound on the angles of triangles constructed by bisecting the longest side, *Mathematics of Computation*, 29, 1975, 390-395.

[12] R.B. Simpson, Anisotropic Mesh Transformations and Optimal Error Control. *Applied Numerical Mathematics*, **14**, 1994, 183-198.

[13] R.B. Simpson, N. Hitschfeld and M.C. Rivara, Approximate quality mesh generation, *Engineering with computers*, **17**, 2001, 287-298.

[14] R. B. Simpson and M. C. Rivara, GeometricalMesh improvement Properties of Delaunay Terminal Edge Refinement. *Technical Report CS-2006-16*, David Cheriton School of Computer Science, University of Waterloo.

Matrix Based Subdivision Depth Computation for Extra-Ordinary Catmull-Clark Subdivision Surface Patches

Gang Chen and Fuhua (Frank) Cheng

Graphics & Geometric Modeling Lab, Department of Computer Science,
University of Kentucky, Lexington, Kentucky 40506-0046
{gchen5, cheng}@cs.uky.edu
www.cs.uky.edu/∼cheng

Abstract. A new subdivision depth computation technique for extra-ordinary Catmull-Clark subdivision surface (CCSS) patches is presented. The new technique improves a previous technique by using a matrix representation of the second order norm in the computation process. This enables us to get a more precise estimate of the rate of convergence of the second order norm of an extra-ordinary CCSS patch and, consequently, a more precise subdivision depth for a given error tolerance.

1 Introduction

Given a Catmull-Clark subdivision surface (CCSS) patch, *subdivision depth computation* is the process of determining how many times the control mesh of the CCSS patch should be subdivided so that the distance between the resulting control mesh and the surface patch is smaller than a given error tolerance.

A good subdivision depth computation technique requires precise estimate of the distance between the control mesh of a CCSS patch and its limit surface. Optimum distance evaluation techniques for regular CCSS patches are available [3,6]. Distance evaluation for an extra-ordinary CCSS patch is more complicated. A first attempt in that direction is done in [3]. The distance is evaluated by measuring norms of the *first order forward differences* of the control points. But the distance computed by this approach is usually bigger than what it really is for regions already flat enough and, consequently, leads to over-estimated subdivision depth.

An improved distance evaluation technique for extra-ordinary CCSS patches is presented in [4]. The distance is evaluated by measuring norms of the *second order forward differences* (called *second order norms*) of the control points of the given extra-ordinary CCSS patch. However, it has been observed recently that, for extra-ordinary CCSS patches, the convergence rate of second order norm changes with the subdivision process, especially between the first subdivision level and the second subdivision level. Therefore, using a fixed convergence rate in the distance evaluation process for all subdivision levels would over-estimate the distance and, consequently, over-estimate the subdivision depth as well.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 545–552, 2006.
© Springer-Verlag Berlin Heidelberg 2006

In this paper we present an improved subdivision depth computation method for extra-ordinary CCSS patches. The new technique uses a matrix representation of the maximum second order norm in the computation process to generate a recurrence formula. This recurrence formula allows the smaller convergence rate of the second subdivision level to be used as a bound in the evaluation of the maximum second order norm and, consequently, leads to a more precise subdivision depth for the given error tolerance.

2 Problem Formulation and Background

Given the control mesh of an extra-ordinary CCSS patch and an error tolerance ϵ, the goal here is to compute an integer d so that if the control mesh is iteratively refined (subdivided) d times, then the distance between the resulting mesh and the surface patch is smaller than ϵ. d is called the *subdivision depth* of the surface patch with respect to ϵ.

2.1 Distance Between Patch and Control Mesh

For a given interior mesh face $\mathbf{F}$, let $\mathbf{S}$ be the corresponding Catmull-Clark Subdivision Surface (CCSS) patch in the limit surface $\bar{\mathbf{S}}$. The *distance* between an interior mesh face $\mathbf{F}$ and the corresponding patch $\mathbf{S}$ is defined as the maximum of $\|\mathbf{F}(u,v) - \mathbf{S}(u,v)\|$:

$$D_{\mathbf{F}} = max_{(u,v)\in\Omega} \|\mathbf{F}(u,v) - \mathbf{S}(u,v)\| \tag{1}$$

where $\Omega \equiv [0,\ 1] \times [0,\ 1]$ is the parameter space of $\mathbf{F}$ and $\mathbf{S}$. $D_{\mathbf{F}}$ is also called the distance between $\mathbf{S}$ and its control mesh.

2.2 Depth Computation for Extra-Ordinary Patches

The distance evaluation mechanism of the previous subdivision depth computation technique for extra-ordinary CCSS patches utilizes second order norm as a measurement scheme as well [4], but the pattern of *second order forward differences* (SOFDs) used in the distance evaluation process is different from the one used for regular patches [4].

Second Order Norm and Recurrence Formula. Let $\mathbf{V}_i$, $i = 1, 2, ..., 2n+8$, be the control points of an extra-ordinary patch $\mathbf{S}(u,v) = \mathbf{S}_0^0(u,v)$, with $\mathbf{V}_1$ being an extra-ordinary vertex of valence n. The control points are ordered following J. Stam's fashion [7] (Figure 1(a)). The control mesh of $\mathbf{S}(u,v)$ is denoted $\mathit{\Pi} = \mathit{\Pi}_0^0$. The *second order norm* of $\mathbf{S}$, denoted $M = M_0$, is defined as the maximum norm of the following $2n + 10$ SOFDs:

$$\begin{aligned} M = max\{ & \{\|2\mathbf{V}_1 - \mathbf{V}_{2i} - \mathbf{V}_{2((i+1)\%n+1)}\| \mid 1 \le i \le n\} \\ \cup\ & \{\|2\mathbf{V}_{2(i\%n+1)} - \mathbf{V}_{2i+1} - \mathbf{V}_{2(i\%n+1)+1}\| \mid 1 \le i \le n\} \\ \cup\ & \{\|2\mathbf{V}_3 - \mathbf{V}_2 - \mathbf{V}_{2n+8}\|,\ \|\ 2\mathbf{V}_4 - \mathbf{V}_1 - \mathbf{V}_{2n+7}\|,\ \|\ 2\mathbf{V}_5 - \mathbf{V}_6 - \mathbf{V}_{2n+6}\|, \\ & \|\ 2\mathbf{V}_5 - \mathbf{V}_4 - \mathbf{V}_{2n+3}\|,\ \|\ 2\mathbf{V}_6 - \mathbf{V}_1 - \mathbf{V}_{2n+4}\|,\ \|\ 2\mathbf{V}_7 - \mathbf{V}_8 - \mathbf{V}_{2n+5}\|, \\ & \|\ 2\mathbf{V}_{2n+7} - \mathbf{V}_{2n+6} - \mathbf{V}_{2n+8}\|,\ \|\ 2\mathbf{V}_{2n+6} - \mathbf{V}_{2n+2} - \mathbf{V}_{2n+7}\|, \\ & \|\ 2\mathbf{V}_{2n+3} - \mathbf{V}_{2n+2} - \mathbf{V}_{2n+4}\|,\ \|\ 2\mathbf{V}_{2n+4} - \mathbf{V}_{2n+3} - \mathbf{V}_{2n+5}\| \}\ \} \end{aligned} \tag{2}$$

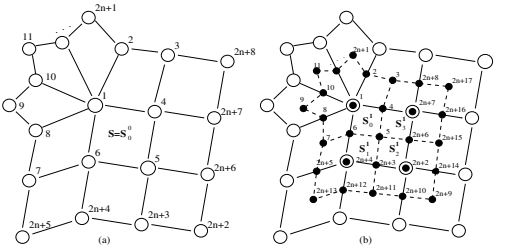


Fig. 1. (a) Ordering of control points of an extra-ordinary patch. (b) Ordering of new control points (solid dots) after a Catmull-Clark subdivision.

If we perform a Catmull-Clark subdivision step [1] on the control mesh of $\mathbf{S}$, we get four new subpatches: $\mathbf{S}_0^1$, $\mathbf{S}_1^1$, $\mathbf{S}_2^1$ and $\mathbf{S}_3^1$. $\mathbf{S}_0^1$ is an extra-ordinary patch but $\mathbf{S}_1^1$, $\mathbf{S}_2^1$ and $\mathbf{S}_3^1$ are regular patches (see Figure 1(b)). We use M_1 to denote the second order norm of $\mathbf{S}_0^1$. This process can be iteratively repeated on $\mathbf{S}_0^1$, $\mathbf{S}_0^2$, $\mathbf{S}_0^3$, ... etc. We have the following lemma for a general $\mathbf{S}_0^k$ and its second order norm M_k [4].

Lemma 1: For any $k \geq 0$, if M_k represents the second order norm of the extra-ordinary sub-patch $\mathbf{S}_0^k$ after k Catmull-Clark subdivision steps, then M_k satisfies the following inequality

$$M_{k+1} \leq \begin{cases} \frac{2}{3}M_k, & n = 3 \\ \frac{18}{25}M_k, & n = 5 \\ (\frac{3}{4} + \frac{8n-46}{4n^2})M_k, & n > 5 \end{cases}.$$

Distance Evaluation. Let $\mathbf{L}(u, v)$ be the bilinear parametrization of the center face of $\mathbf{S}(u, v)$'s control mesh $\mathbf{F} = \{\mathbf{V}_1, \mathbf{V}_6, \mathbf{V}_5, \mathbf{V}_4\}$

$$\mathbf{L}(u, v) = (1 - v)[(1 - u)\mathbf{V}_1 + u\mathbf{V}_6] + v[(1 - u)\mathbf{V}_4 + u\mathbf{V}_5], \quad 0 \leq u, v \leq 1$$

and let $\mathbf{S}(u, v)$ be parameterized following the Ω-partition based approach [7] then the maximum distance between $\mathbf{S}(u, v)$ and its control mesh satisfies the following lemma [4].

Lemma 2: The maximum of $\| \mathbf{L}(u, v)$–$\mathbf{S}(u, v) \|$ satisfies the following inequality

$$\| \mathbf{L}(u, v) - \mathbf{S}(u, v) \| \leq \begin{cases} M_0, & n = 3 \\ \frac{5}{7}M_0, & n = 5 \\ \frac{4n}{n^2-8n+46}M_0, & 5 < n \leq 8 \\ \frac{n^2}{4(n^2-8n+46)}M_0, & n > 8 \end{cases} \quad (3)$$

where $M = M_0$ is the second order norm of the extra-ordinary patch $\mathbf{S}(u, v)$.

Subdivision Depth Computation. Lemma 2 can be used to estimate the distance between a level-k control mesh and the surface patch for any $k > 0$.

Theorem 3: Given an extra-ordinary surface patch $\mathbf{S}(u, v)$ and an error tolerance ϵ, if k levels of subdivisions are iteratively performed on the control mesh

of $\mathbf{S}(u,v)$, where $k = \lceil log_w \frac{M}{z\epsilon} \rceil$ with M being the second order norm of $\mathbf{S}(u,v)$ defined in (2),

$$w = \begin{cases} \frac{3}{2}, & n=3 \\ \frac{25}{18}, & n=5 \\ \frac{4n^2}{3n^2+8n-46}, & n>5 \end{cases} \quad \text{and} \quad z = \begin{cases} 1, & n=3 \\ \frac{25}{18}, & 5 \le n \le 8 \\ \frac{2(n^2-8n+46)}{n^2}, & n>8 \end{cases}$$

then the distance between $\mathbf{S}$ and the level-k control mesh is smaller than ϵ.

3 New Subdivision Depth Computation Technique for Extra-Ordinary Patches

The SOFDs involved in the second order norm of an extra-ordinary CCSS patch (see eq. (2)) can be classified into two groups: *group I* and *group II*. Group I contains those SOFDs that involve vertices in the vicinity of the extra-ordinary vertex (see Figure 2(a)). These are the first $2n$ SOFDs in (2). Group II contains the remaining SOFDs, i.e., SOFDs that involve vertices in the vicinity of the other three vertices of $\mathbf{S}$ (see Figure 2(b)). These are the last 10 SOFDs in (2). It is easy to see that the convergence rate of the SOFDs in group II is the same as the regular case, i.e., 1/4 [3]. Therefore, to study properties of the second order norm M, it is sufficient to study norms of the SOFDs in group I.

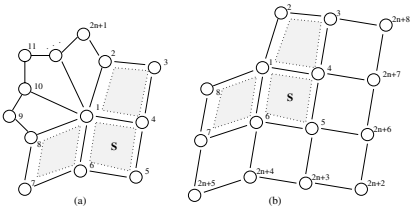


Fig. 2. (a) Vicinity of the extra-ordinary point. (b) Vicinity of the other three vertices of $\mathbf{S}$.

3.1 Matrix Based Rate of Convergence

The second order norm of $\mathbf{S} = \mathbf{S}_0^0$ can be put in matrix form as follows:

$$M = \|\mathrm{A}\mathbf{P}\|_\infty$$

where A is a $2n * (2n+1)$ matrix

$$\mathrm{A} = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & -1 & 0 & 0 \cdots & 0 & 0 \\ 2 & 0 & 0 & -1 & 0 & 0 & 0 & -1 \cdots & 0 & 0 \\ & & & & \vdots & & & & & \\ 2 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \cdots & -1 & 0 \\ 0 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \cdots & 0 & -1 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \cdots & 0 & 0 \\ & & & & \vdots & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \cdots & 2 & -1 \end{bmatrix}$$

and $\mathbf{P}$ is a control point vector

$$\mathbf{P} = [\mathbf{V}_1,\ \mathbf{V}_2,\ \mathbf{V}_3, \ldots,\ \mathbf{V}_{2n+1}]^T.$$

A is called the *second order norm matrix* for extra-ordinary CCSS patches. If i levels of Catmull-Clark subdivision are performed on the control mesh of $\mathbf{S} = \mathbf{S}_0^0$ then, following the notation of Section 2, we have an extra-ordinary subpatch $\mathbf{S}_0^i$ whose second order norm can be expressed as:

$$M_i = \left\|\mathrm{A}\Lambda^i\mathbf{P}\right\|_\infty$$

where Λ is a subdivision matrix of dimension $(2n+1) * (2n+1)$. The function of Λ is to perform a subdivision step on the $2n+1$ control vertices around (and including) the extra-ordinary point (see Figure 2(a)). We are interested in knowing the relationship between $\|\mathrm{A}\mathbf{P}\|_\infty$ and $\left\|\mathrm{A}\Lambda^i\mathbf{P}\right\|_\infty$. We need the following important result for $\mathrm{A}\Lambda^i$. The proof of this result is shown in [2].

Lemma 4: $\mathrm{A}\Lambda^i = \mathrm{A}\Lambda^i\mathrm{A}^+\mathrm{A}$, where A^+ is the *pseudo-inverse matrix* of A.

With this lemma, we have

$$\frac{\left\|\mathrm{A}\Lambda^i\mathbf{P}\right\|_\infty}{\|\mathrm{A}\mathbf{P}\|_\infty} = \frac{\left\|\mathrm{A}\Lambda^i\mathrm{A}^+\mathrm{A}\mathbf{P}\right\|_\infty}{\|\mathrm{A}\mathbf{P}\|_\infty} \le \frac{\left\|\mathrm{A}\Lambda^i\mathrm{A}^+\right\|_\infty \|\mathrm{A}\mathbf{P}\|_\infty}{\|\mathrm{A}\mathbf{P}\|_\infty} = \left\|\mathrm{A}\Lambda^i\mathrm{A}^+\right\|_\infty$$

Use r_i to represent $\left\|\mathrm{A}\Lambda^i\mathrm{A}^+\right\|_\infty$. Then we have the following recurrence formula for r_i

$$\begin{aligned} r_i \equiv \left\|\mathrm{A}\Lambda^i\mathrm{A}^+\right\|_\infty &= \left\|\mathrm{A}\Lambda^{i-1}\mathrm{A}^+\mathrm{A}\Lambda\mathrm{A}^+\right\|_\infty \\ \le \left\|\mathrm{A}\Lambda^{i-1}\mathrm{A}^+\right\|_\infty &\ \|\mathrm{A}\Lambda\mathrm{A}^+\|_\infty = r_{i-1}\ r_1 \end{aligned} \tag{4}$$

where $r_0 = 1$. Hence, we have the following lemma on the convergence rate of second order norm of an extra-ordinary CCSS patch.

Lemma 5: The second order norm of an extra-ordinary CCSS patch satisfies the following inquality:

$$M_i \le r_i\ M_0 \tag{5}$$

where $r_i = \left\|\mathrm{A}\Lambda^i\mathrm{A}^+\right\|_\infty$ and r_i satisfies the recurrence formula (4).

The recurrence formula (4) shows that r_i in (5) can be replaced with r_1^i. However, experiment data show that, while the convergence rate changes by a constant ratio in most of the cases, there is a significant difference between r_2 and r_1. The value of r_2 is smaller than r_1^2 by a significant gap. Hence, if we use r_1^i for r_i in (5), we would end up with a bigger subdivision depth for a given error tolerance. A better choice is to use r_2 to bound r_i, as follows.

$$r_i \le \begin{cases} r_2^j, & \mathrm{i} = 2j \\ r_1 r_2^j, & \mathrm{i} = 2j+1 \end{cases} \tag{6}$$

3.2 Distance Evaluation

Following (12) and (13) of [4], the distance between the extra-ordinary CCSS patch $\mathbf{S}(u,v)$ and its control mesh $\mathbf{L}(u,v)$ can be expressed as

$$\begin{array}{l}\|\mathbf{L}(u,v)-\mathbf{S}(u,v)\| \le \sum_{k=0}^{m-2}\|\mathbf{L}_0^k(u_k,v_k)-\mathbf{L}_0^{k+1}(u_{k+1},v_{k+1})\| \\ +\|\mathbf{L}_0^{m-1}(u_{m-1},v_{m-1})-\mathbf{L}_b^m(u_m,v_m)\|+\|\mathbf{L}_b^m(u_m,v_m)-\mathbf{S}_b^m(u_m,v_m)\|\end{array} \tag{7}$$

where u_m,v_m and b are defined in [4].

By applying Lemma 5, Lemma 6 and Lemma 1 of [4] on the first, second and third terms of the right hand side of the above inequality, respectively, we get

$$\begin{array}{rl}\|\mathbf{L}(u,v)-\mathbf{S}(u,v)\| & \le c\sum_{k=0}^{m-2}M_k+\frac{1}{4}M_{m-1}+\frac{1}{3}M_m \\ & \le M_0(c\sum_{k=0}^{m-2}r_k+\frac{1}{4}r_{m-1}+\frac{1}{3}r_m)\end{array}$$

where $c = 1/\min\{n,8\}$. The last part of the above inequality follows from Lemma 2. Consequently, through a simple algebra, we have

$$\|\mathbf{L}(u,v)-\mathbf{S}(u,v)\| \le \begin{cases} M_0[c(\frac{1-r_2^j}{1-r_2}+\frac{1-r_2^{j-1}}{1-r_2}r_1)+\frac{r_1r_2^{j-1}}{4}+\frac{r_2^j}{3}], & \text{if } m=2j \\ M_0[c(\frac{1-r_2^j}{1-r_2}+\frac{1-r_2^j}{1-r_2}r_1)+\frac{r_2^j}{4}+\frac{r_1r_2^j}{3}], & \text{if } m=2j+1\end{cases}$$

It can be easily proved that the maximum occurs at $m=\infty$. Hence, we have the following lemma.

Lemma 6: The maximum of $\|\mathbf{L}(u,v)-\mathbf{S}(u,v)\|$ satisfies the following inequality

$$\|\mathbf{L}(u,v)-\mathbf{S}(u,v)\| \le \frac{M_0}{\min\{n,8\}}\frac{1+r_1}{1-r_2}$$

where $r_i = \|\mathrm{A}\Lambda^i\mathrm{A}^+\|_\infty$ and $M = M_0$ is the second order norm of the extra-ordinary patch $\mathbf{S}(u,v)$.

3.3 Subdivision Depth Computation

Lemma 6 can also be used to evaluate the distance between a level-i control mesh and the extra-ordinary patch $\mathbf{S}(u,v)$ for any $i>0$. This is because the distance between a level-i control mesh and the surface patch $\mathbf{S}(u,v)$ is dominated by the distance between the level-i extra-ordinary subpatch and the corresponding control mesh which, accoriding to Lemma 6, is

$$\|\mathbf{L}_i(u,v)-\mathbf{S}(u,v)\| \le \frac{M_i}{\min\{n,8\}}\frac{1+r_1}{1-r_2}$$

where M_i is the second order norm of $\mathbf{S}(u,v)$'s level-i control mesh, $\mathbf{M}_i$. Hence, if the right side of the above inequality is smaller than a given error tolerance ϵ, then the distance between $\mathbf{S}(u,v)$ and the level-i control mesh is smaller than ϵ. Consequently, we have the following subdivision depth computation theorem for extra-ordinary CCSS patches.

Theorem 7: Given an extra-ordinary surface patch $\mathbf{S}(u,v)$ and an error tolerance ϵ, if

$$i \equiv \min\{2l,\ 2k+1\}$$

levels of subdivision are iteratively performed on the control mesh of $\mathbf{S}(u,v)$, where

$$l = \lceil \log_{\frac{1}{r_2}} (\frac{1}{\min\{n,8\}} \frac{1+r_1}{1-r_2} \frac{M_0}{\epsilon}) \rceil \ ,$$
$$k = \lceil \log_{\frac{1}{r_2}} (\frac{r_1}{\min\{n,8\}} \frac{1+r_1}{1-r_2} \frac{M_0}{\epsilon}) \rceil$$

with $r_i = \|\mathrm{A}\Lambda^i\mathrm{A}^+\|_\infty$ and M_0 being the second order norm of $\mathbf{S}(u,v)$, then the distance between $\mathbf{S}(u,v)$ and the level-i control mesh is smaller than ϵ.

4 Examples

The new subdivision depth technique has been inplemented in *C++* on the Windows platform to compare its performance with the previous approach. *MatLab* is used for both numerical and symbolic computation of r_i in the implementation. Table 1 shows the comparison results of the previous technique, Theorem 3, with the new technique, Theorem 7. Two error tolerances 0.01 and 0.001 are considered and the second order norm M_0 is assumed to be 2. For each error tolerance, we consider five different valences: 3, 5, 6, 7 and 8 for the extra-ordinary vertex. As can be seen from the table, the new technique has a 30% improvement over the previous technique in most of the cases. Hence, the new technique indeed improves the previous technique significantly.

To show that the rates of convergence are indeed difference between r_1 and r_2, their values from several typical extra-ordinary CCSS patches are also included in Table 1. Note that when we compare r_1 and r_2, the value of r_1 should be squared first.

Table 1. Comparison between the old and the new technique

	$\epsilon = 0.01$		$\epsilon = 0.001$		convergence rate	
N	Old	New	Old	New	r_1	r_2
3	14	9	19	12	0.6667	0.2917
5	16	11	23	16	0.7200	0.4016
6	19	16	27	22	0.8889	0.5098
7	23	14	33	22	0.8010	0.5121
8	37	27	49	33	1.0078	0.5691

5 Conclusions

A new subdivision depth computation technique for extra-ordinary CCSS patches is presented. The computation process is performed on matrix representation of the second order norm, which gives us a better bound of the convergence rate and, consequently, a tighter subdivision depth for a given error tolerance. Test results show that the new technique improves the previous technique by about 30% in most of the cases. This is a significant result because of the exponential nature of the subdivision process.

Acknowledgement. Reserach work of the authors is supported by NSF under grant DMS-0310645 and DMI-0422126.

References

1. Catmull E, Clark J, Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes, *Computer-Aided Design 10*, 6, 350-355, 1978.
2. Chen G, Cheng F, Matrix based Subdivision Depth Computation for Extra-Ordinary Catmull-Clark Subdivision Surface Patches (complete version), http://www.cs.uky.edu/∼cheng/PUBL/ sub_depth_3.pdf
3. Cheng F, Yong J, Subdivision Depth Computation for Catmull-Clark Subdivision Surfaces, *Computer Aided Design & Applications 3*, 1-4, 2006.
4. Cheng F, Chen G, Yong J, Subdivision Depth Computation for Extra-Ordinary Catmull-Clark Subdivision Surface Patches, to appear in *Lecture Notes in Computer Science*, Springer, 2006.
5. Halstead M, Kass M, DeRose T, Efficient, Fair Interpolation Using Catmull-Clark Surfaces, *Proceedings of SIGGRAPH 1993*, 35-44.
6. Lutterkort D, Peters J, Tight linear envelopes for splines, *Numerische Mathematik 89*, 4, 735-748, 2001.
7. Stam J, Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values, *Proceedings of SIGGRAPH 1998*, 395-404.

Hierarchically Partitioned Implicit Surfaces for Interpolating Large Point Set Models

David T. Chen[1], Bryan S. Morse[2], Bradley C. Lowekamp[1], and Terry S. Yoo[1]

[1] National Library of Medicine, Bethesda, MD, USA
[2] Brigham Young University, Provo, UT, USA

Abstract. We present a novel hierarchical spatial partitioning method for creating interpolating implicit surfaces using compactly supported radial basis functions (RBFs) from scattered surface data. From this hierarchy of functions we can create a range of models from coarse to fine, where a coarse model approximates and a fine model interpolates. Furthermore, our method elegantly handles irregularly sampled data and hole filling because of its multiresolutional approach. Like related methods, we combine neighboring patches without surface discontinuities by overlapping their embedding functions. However, unlike partition-of-unity approaches we do not require an additional explicit blending function to combine patches. Rather, we take advantage of the compact extent of the basis functions to directly solve for each patch's embedding function in a way that does not cause error in neighboring patches. Avoiding overlap error is accomplished by adding phantom constraints to each patch at locations where a neighboring patch has regular constraints within the area of overlap (the function's radius of support). Phantom constraints are also used to ensure the correct results between different levels of the hierarchy. This approach leads to efficient evaluation because we can combine the relevant embedding functions at each point through simple summation. We demonstrate our method on the Thai statue from the Stanford 3D Scanning Repository. Using hierarchical compactly supported RBFs we interpolate all 5 million vertices of the model.

1 Introduction

Many applications in computer graphics require smoothly interpolating a large set of points on or near a surface. These points may originate as unorganized point sets such as from a 3-D scanning system. They may also come in organized or semiorganized sets from the vertices of polygonal models, which once interpolated can provide a smoother surface than the polygonal one and can be converted to other representations, including a more finely polygonalized one if desired. Such point sets may also come from computer vision analysis of an image volume or from interactive modeling tools.

Numerous techniques have emerged for converting point sets to implicit models that interpolate (or approximate) these points [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]. Broadly, we call these *interpolating implicit surfaces*.[1] These methods take the same general approach: known points on the surface define where the implicit surface's embedding function should have a value of 0; known off-surface points, surface normals (either known or fitted), or other assumptions define where the embedding function has

[1] These have also been known in the literature as *variational implicit surfaces*, *implicit surfaces that interpolate*, and *constraint-based implicit surfaces* by various authors.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 553–562, 2006.
© Springer-Verlag Berlin Heidelberg 2006

nonzero values; and the embedding function is then interpolated using scattered data interpolation techniques such as radial basis functions (RBFs) [1,2,4,5,6,7,10], (implicit) moving least squares [14], or partition-of-unity blending of local fitting using these or other interpolation methods [8,11,12].

Many implementations [1,2,6] use thin-plate splines, but this requires the solving of a large, full, generally ill-conditioned system of equations and quickly becomes computationally impractical for large models. Other RBFs with infinite support [7, for example] have similar limitations. Various methods have been used to accelerate RBF approaches, including using compactly supported RBFs to make the required system sparse [4,9,10,13,16] or approximating a large set of constraints by a well-selected subset [3,5]. Others accelerate the surface fitting by subdividing the surface into smaller patches, fitting a surface (or the embedding function for that surface) to each patch, then combining the patches through blending [8,11,12,14].

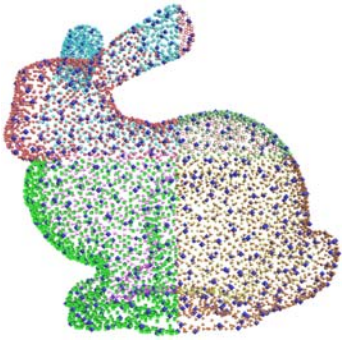


Fig. 1. A simple two-level hierarchy of the Stanford bunny model. The constraints of the root RBF are dark, larger octohedra, and each child partition is indicated in different shapes, styles and colors.

This paper presents a novel method for efficiently creating compactly supported RBF-based implicit representations from the vertices of a polygonal model using hierarchical spatial partitioning as illustrated in Figure 1. Unlike other approaches that combine local interpolations through compactly supported blending functions, no explicit blending function is required—each level and partitioned patch is calculated so that a simple linear combination of them produces an exact interpolation.

2 Related Work and Background

Our implementation of compactly-supported RBFs follows most closely that detailed in [4], which is based on the general approach of Turk *et al.* [2,6].

The basic method begins with a set of points known to lie on the desired implicit surface and constrains the interpolated embedding function to have a value of 0 at these points. Using the method of [6], non-zero constraints (often called "normal constraints") are placed at fixed offsets in the direction of the known or desired normals at these points. This produces a set of point/value constraints $P = \{(\mathbf{c}_i, h_i)\}$ such that $h_i = 0$ for all $\mathbf{c}_i$ on the surface and $h_i = 1$ for all $\mathbf{c}_i$ at a fixed offset from that surface. An embedding function $f(\mathbf{x})$ is then interpolated from these constraints such that $f(\mathbf{c}_i) = h_i$.

This interpolation is done using an RBF $\phi(r)$ by defining the embedding function f as a weighted sum of these basis functions centered at each of the constraints:

$$f(\mathbf{x}) = \sum_{(\mathbf{c}_i, h_i) \in P} d_i \, \phi(\|\mathbf{x} - \mathbf{c}_i\|) \tag{1}$$

where d_i is the weight of the radial basis function positioned at $\mathbf{c}_i$.[2] To solve for the unknown weights d_i, substitute each constraint $f(\mathbf{c}_i) = h_i$: into Eq. 1:

$$\forall \mathbf{c}_i : f(\mathbf{c}_i) = \sum_{(\mathbf{c}_j, h_j) \in P} d_j \phi(\|\mathbf{c}_i - \mathbf{c}_j\|) = h_i \tag{2}$$

By using compactly supported RBFs one can make this system of equations sparse [4,9]. By efficiently organizing the points spatially, one can also reduce the time required to compute the system itself. As the size of the model increases, one can commensurately reduce the radius of support for the RBFs, thus increasing efficiency while keeping the data density approximately constant.

The primary drawback to using compactly supported radial basis functions alone for surface modeling is that the embedding function is 0 outside one radius of support from the surface. This does not preclude polygonalization, ray-tracing, or many other uses of the surface because it is relatively easy to separate zero sets that result from lack of support. However, it does limit their use for CSG and other operations for which implicit surfaces are useful. The compact support also causes them to fail in areas with low data density, in the limit failing where the surface has holes larger than the support. (See [10] for an excellent discussion of the limitations of compactly supported RBFs for surface modeling, with additional empirical analysis in [16].) These limitations can be overcome using hierarchical, or multilevel, approaches, such as [17,18] for scattered data interpolation, and [10,13] for compactly supported RBFs.

Another way to accelerate the surface fitting is to spatially subdivide the surface points into separate patches, then interpolate (or approximate) each patch and blend the results using partition-of-unity or similar blending techniques [8,11,12,14]. By blending local approximations instead of directly trying to fit the entire model at once, this method provides efficient processing for very large models. Key to these partition-of-unity or other blending methods is the use of a compactly supported weighting function to blend separate patches or the effects of individual points in a neighborhood.

We demonstrate a new method for creating and blending interpolations for separate patches that uses compactly supported RBFs to interpolate the patches and, due to their compactly supported nature, does not require the a separate explicit blending function.

3 Method

As with other hierarchical partitioning approaches, our method builds a large-scale approximating embedding function then successively refines it with smaller-scale incremental functions. The two main components of our method are selecting the points for each node in the hierarchy and creating *phantom constraints* to handle overlapping function domains. Using phantom constraints to clamp each embedding function allows us to combine them simply by addition rather than requiring a blending function.

3.1 Building a Hierarchy

To build a hierarchy we use an octree to span the input points, which is traversed from the top down. Points are first selected for the root, producing an embedding function for

[2] For some RBFs, including thin-plate splines, an additional polynomial may be required.

a base model. Then points for the each of the children of the root are selected, adding detail at a finer resolution. After solving the refining embedding functions for the eight children, we proceed to the grandchildren, and so on, stopping when all points have been included in the hierarchy. When building any given node of the hierarchy, the functions for the nodes above it have already been solved.

Selecting Points for a Node. Points in a node's octant are selected based on a random Poisson-disk distribution. However, our initial implementation, the traditional Poisson-disk distribution where there is a minimum Euclidean distance between any two points, tended to undersample regions of high curvature. We needed to allow sample points to be closer together in high curvature regions.

Comparing the normal directions of nearby points is an efficient estimate of local curvature. A region with points that have disparate normals requires a higher sampling rate. To achieve adaptive sampling we use a modified distance function based on the points' normals. If two points have identical normals, the distance between them is the same as the Euclidean distance. However, if their normals differ we would like them to appear to be farther apart. The net effect is to place samples closer together in areas of higher curvature.

We have experimentally developed an admittedly arbitrary modified distance function that scales the Euclidean distance between points $\mathbf{x}_1$ and $\mathbf{x}_2$ by a quadratic function of the angle θ between the normal vectors:

$$\mathrm{dist}(\mathbf{x}_1, \mathbf{x}_2, \theta) = \|\mathbf{x}_1 - \mathbf{x}_2\| \left(\tfrac{1}{2}\cos(\theta)^2 - \tfrac{7}{2}\cos(\theta) + 4 \right) \tag{3}$$

Selecting Points for the Root. Selecting points for the root embedding function is especially important because error in the root propagates throughout the hierarchy. The more error there is in a parent node, the more "energy" required at a child node to bend the embedding function to fit. Since the root node affects all other nodes, we are more particular in selecting its points.

At the root node, in addition to the modified Poisson-disk distribution mentioned earlier, we attempt to select points that are "representative" of a local region. The goal is to pick points that capture the larger-scale shape of a region, pushing smaller scale detail or noise to nodes lower in the hierarchy.

For the root node, candidate points are screened by comparing the normal direction for each point with the normals of points around it and rejecting those that are too disparate. Specifically, if the average dot product of a candidate point's normal with the normals around it is below a specific threshold (0.1), the point is not selected.

Figure 1 shows a two-level hierarchy of RBFs of the Stanford bunny. The hierarchy consists of a root RBF and eight children. The nodes' constraints are differentiated by style. The root's constraints are dark blue, slightly larger octahedra.

Embedding Function for a Node. Once points have been selected for interpolation within a node, interior, surface, and exterior constraints are placed for each point. The embedding function also requires a level set value for each constraint and a radius of support for the compact RBF. The root embedding function should produce the correct

results at the locations of the root's constraints. Therefore the constraint values at the root are determined solely by the type of constraint. By default the functions' values at surface, interior and exterior constraints should be 0, 1 and -1 respectively.

Using the notation of Eqs. 1 and 2, we can write the root embedding function f_0 defined by the set of root constraints $P_0 = \{(\mathbf{c}_i, h_i)\}$ using root-level RBF $\phi_0(r)$:

$$f_0(\mathbf{x}) = \sum_{(\mathbf{c}_i,h_i)\in P_0} d_{0i}\, \phi_0(\|\mathbf{x}-\mathbf{c}_i\|) \tag{4}$$

where the root-level weights d_{0i} are determined by solving the system of equations

$$\forall \mathbf{c}_i \in P_0 : f_0(\mathbf{c}_i) = h_i \tag{5}$$

The embedding function of a child node is an increment that corrects the parent function at the location of the child node's constraints. For example, at a child node's surface constraint the net function should to evaluate to 0. However, the parent function evaluates to some value α. Therefore the child's function should evaluate to $-\alpha$ to correct the error. Thus at each child constraint location, the hierarchy of embedding functions above the child node is evaluated, and a value is given to the child constraint that corrects the result of the nodes above it.

Thus, we may write a single child level's embedding function f_1 defined by the child node's constraints $P_1 = \{(\mathbf{c}_i, h_i)\}$ and child-level RBF $\phi_1(r)$, along with the parent node's constraints P_0 and embedding function f_0, as follows:

$$f_1(\mathbf{x}) = \sum_{(\mathbf{c}_i,h_i)\in P_0\cup P_1} d_{1i}\, \phi_1(\|\mathbf{x}-\mathbf{c}_i\|) \tag{6}$$

where the child-level weights d_{1i} are determined by solving the system of equations

$$\forall \mathbf{c}_i \in P_0 \cup P_1 : f_1(\mathbf{c}_i) = h_i - f_0(\mathbf{c}_i) \tag{7}$$

Note that each level k of the hierarchy uses its own RBF $\phi_k(r)$ and weights d_{ki}.

Since Eq. 5 already holds for the root constraints, the root embedding function f_0 already evaluates correctly at the root constraints and no correction is required by the child embedding function:

$$\forall \mathbf{c}_i \in P_0 : f_1(\mathbf{c}_i) = 0 \tag{8}$$

This process may be continued to additional levels of the hierarchy and extended to include multiple nodes at each level. Solving for and combining embedding functions for multiple nodes at each level is addressed in Section 3.2.

Once all the constraint values for a node have been determined, the radius of support for the compact RBF for that node must be determined. We attempt to keep the same number of points per node, and nodes at different levels in the hierarchy cover different-sized regions. Naturally different-level nodes should have compact RBFs with different radii. In our approach the user selects the radius for the root, and each descendant is given a radius proportional to that root radius and to its own size.

Typically compact RBFs are used for the embedding functions of all of the nodes in the hierarchy, but using only compactly supported RBFs have the problem of the

function being undefined in some regions. Any location that is outside of all constraints' radii of support will not have a defined embedding function. Therefore we also allow the option of using a thin plate spline (TPS) RBF for the root node (see Figure 3), eliminating the problem. Although a TPS is much more expensive to solve and evaluate, the embedding function for the root node uses only a limited subset of points, making it still practical even for otherwise large models.

3.2 Phantom Constraints

Managing embedding functions with overlapping extent is a common problem that occurs when attempting to partition a point set. To interpolate all the points in a data set, we must guarantee that the combination of all embedding functions that impinge on a point produces the exact value we require. Our task is simplified by the compact RBF's limited extent. Therefore at any given point only relatively few embedding functions need to be combined and evaluated.

Our approach is to place *phantom constraints* in a given node to clamp its embedding function. Phantom constraints are placed in regions where the extent of the embedding functions for different nodes overlap, requiring us to suppress the influence of the node's embedding function. In this way, phantom constraints serve much the same purpose as the blending function in partition-of-unity approaches but without explicit blending during evaluation of the implicit surface's embedding function. The locations for phantom constraints fall into two categories: locations that have been inherited from regular constraints in ancestral nodes, and locations from regular constraints in adjacent sibling nodes. Using a top-down approach means constraint locations from descendant nodes can be ignored.

A child node's embedding function is an incremental change applied to the sum of its ancestor embedding functions as mentioned in Section 3.1. For our purposes an ancestor node is any node in the octree above a given node whose embedding function overlaps with that node, not just direct ancestors. Since a child embedding function is an increment to the functions above it, a regular constraint of the child is given a value that corrects the summed ancestor functions. Also, phantom constraints with values of 0 are placed in the child RBF at all ancestor constraint locations within the child's bounds to ensure the child's function does not produce erroneous results at these locations.

Similarly, the interpolation within a node should not be incorrectly affected by neighboring sibling nodes. Therefore, any regular constraints of neighboring siblings that overlap with the extent of a node's embedding function become corresponding phantom constraints for the node.

Extending the notation of Eqs. 4–8, we define $P_{lk} = \{(\mathbf{c}_i, h_i)\}$ as the set of constraints for node k of level l. We also define $\hat{P}_{lk} = \{(\mathbf{c}_i, h_i)\}$ as the set of phantom constraints relevant to this node and the function $\hat{f}_{lk}$ as the sum of all other embedding functions relevant to this node (i.e., those higher up in the hierarchy whose support-expanded regions overlap this node's support-expanded region). We may thus write the embedding function for this child node in terms of these constraints and the node's RBF $\phi_{lk}(r)$ as

$$f_{lk}(\mathbf{x}) = \sum_{(\mathbf{c}_i, h_i) \in P_{lk} \cup \hat{P}_{lk}} d_{lki}\, \phi_{lk}(\|\mathbf{x} - \mathbf{c}_i\|) \tag{9}$$

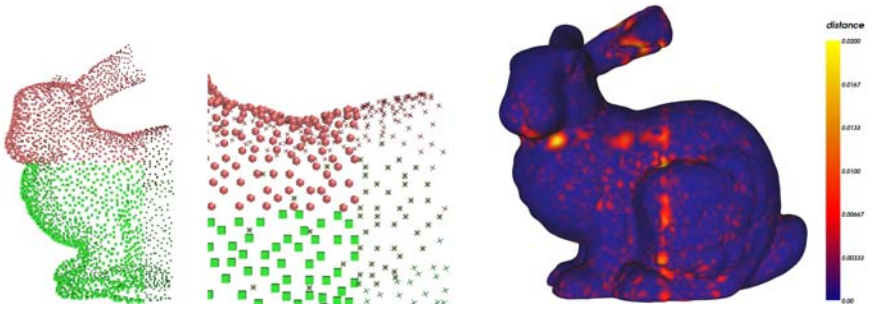


Fig. 2. Phantom constraints. a) the constraints of two neighboring child nodes of the bunny. The solid shapes are regular constraints, and the crosses are phantom constraints, rotated based on node. b) the effects of phantom constraints on the embedding function. The left side of the bunny does not have phantom constraints from neighboring nodes, while the right side does. The color shows the distance error between the embedding function and the original surface.

where the child node's weights d_{lki} are determined by solving the system of equations

$$\forall \mathbf{c}_i \in P_{lk} \cup \hat{P}_{lk} : f_{lk}(\mathbf{c}_i) = h_i - \hat{f}_{lk}(\mathbf{c}_i) \tag{10}$$

Again, this node's embedding function provides incremental refinement only and does not change the result at the phantom constraints from other nodes:

$$\forall \mathbf{c}_i \in \hat{P}_{lk} : f_{lk}(\mathbf{c}_i) = 0 \tag{11}$$

Figure 2a shows two overlapping child nodes of the bunny. The solid shapes represent regular constraints, while the crosses represent phantom constraints. The phantom constraints contained with the bounds of a node's octant have been inherited from the root node, while those outside the octant come from neighboring sibling nodes.

Figure 2b illustrates the error that can occur from overlapping embedding functions that do not have phantom constraints. The left side of the bunny has no phantom constraints from neighboring octants, while the right side does have phantom constraints. The surface is colored by the distance error between the embedding function and the original surface mesh. Clearly there is much more error on the left side, particularly where octants abut. Also the error bleeds into the right side of the bunny because the functions on the left are not evaluating to 0 to the right.

Figure 3 shows slices through four embedding function of the bunny. Images 3a and 3b use a compact RBF for the root node, while 3c and 3d use a thin plate spline at the root. In the left pair the textured region is where the compact RBF is not defined. Images 3a and 3c are slices through the embedding functions of just the root nodes. The images that slice through two level hierarchies (3b and 3d) clearly show sharper boundaries and more detail. For instance, the bottoms of the bunny are less rounded.

Adding phantom constraints outside of a given node's bounds expands the region of space where the node's RBF must be evaluated. However, this expansion can be nullified by only defining the embedding function in the original region defined by the regular constraints. At any location with only phantom constraints within the RBF's radius, the embedding function returns 0.

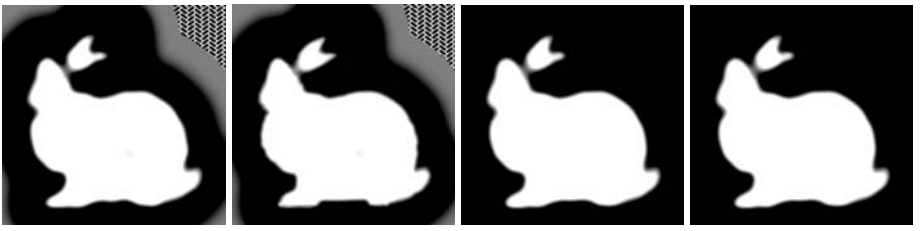


Fig. 3. Slices through the embedding functions. a) a compact RBF root. b) a two level hierarchy, with a compact root. c) a thin plate spline root. d) a two level hierarchy, with a thin plate spline root. The texture regions are where the compact RBF is undefined.

4 Results

To demonstrate the efficacy of our method, we applied it to the Thai statue from Stanford's 3D Scanning Repository. The model consists of 5 million vertices to which we added 426,245 vertices on the bottom, which was not scanned. Our implicit model has 17.5 million constraints in a 5 level octree. Figure 4 shows the five levels in the hierarchy and an iso-surface extracted from the embedding function.

Section 4.1 analyzes the statistics and error characteristics of our method. Section 4.2 provides some implementation details.

4.1 Statistics

Figure 5 shows statistics for the implicit hierarchy. The upper section of the table shows statistics for the entire model, and the middle section shows statistical averages per implicit evaluation. The bottom section shows the error in the implicit surface. To compute the statistics in the middle and bottom sections, the embedding function was evaluated at every surface constraint location in the data set.

The implicit error of a constraint is the unsigned difference between the value a constraint should have and the value returned by the embedding function. By default, surface constraints should have a value of 0.

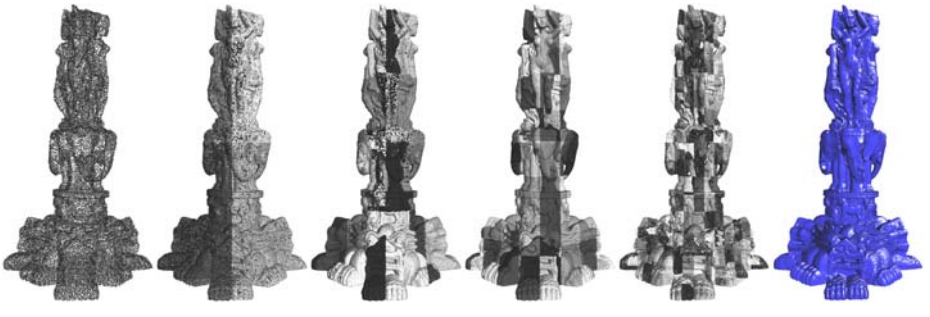


Fig. 4. Stanford's Thai statue. The 5 levels of the hierarchy of constraints and an extracted iso-surface. The hierarchy contains 17.5 million constraints.

The distance error is the distance between a surface constraint's location and a root (zero) of the embedding function. The root was found by searching the embedding function along the constraint's normal direction. The data set has an extent 395.9 along its longest axis, and the exterior constraints were offset by a distance of 10^{-4}. Values of 0 for surface constraints and -1 for exterior constraints of the embedding function results in gradients on the order of 10^4. Thus one would expect implicit errors on the order of 10^4 times greater than the distance errors, as the statistics demonstrate.

		Thai statue
Hierarchy statistics	*vertices*	4,999,996
	regular constraints	10,852,316
	phantom constraints	6,632,801
	total constraints	17,485,117
	tree nodes	799
	tree height	5
	build time (minutes)	127.35
Averages per evaluation	*regular constraints*	961.77
	phantom constraints	202.00
	number of nodes	6.59
Error	*avg. implicit error*	1.4980E-06
	max. implicit error	1.4950E-04
	avg. distance error	2.1230E-10
	max. distance error	2.2700E-08

Fig. 5. Statistics for the hierarchical model of the Thai statue

In our examples, adding phantom constraints increases the number of constraints in the data sets by an average of 61.1% so that phantom constraints make up 37.9% of the constraints. However, since the phantom constraints tend to occur towards the bottom of the hierarchy, i.e. in the nodes with smaller extents, the average number of phantom constraints encountered per function evaluation is lower. In all they represent only 17.4% of the constraints when evaluating the embedding function.

4.2 Implementation

The example implicit hierarchy was built on a SGI Altix system with four 1.4 GHz Itanium 2 processors and 8 GB of main memory. The most time consuming sections of code, solving each node's sparse matrix and computing all the constraints' values, were parallelized. Computing a constraint's value is required for non-root nodes, since its value depends on the embedding functions above it in the hierarchy. The matrices were solved using the LDL solver in SGI's Scientific Computing Software Library.

5 Conclusions

We have presented a technique for generating implicit surfaces that interpolate large point sets. This method employs a hierarchical spatial partitioning that imposes a successive series of embedding functions constrained so that when they are added to one another, they interpolate the point set. The approach begins with the careful selection of a representative subset of the point set from which an interpolating implicit surface that provides a base model can be created. This base model interpolates the core subset of data points and serves as the foundation for the coarse-to-fine hierarchy. The data space is recursively divided into an octree with additional data points selected, and more detailed embedding functions are derived for each child octant that, when added to the base model, accurately interpolate the more complete, higher resolution model.

Neighboring spatial partitions are supplemented with phantom constraints that assure smooth transitions between adjoining embedding functions. No additional blending functions are required because our compactly supported radial basis functions have a limited radius of influence, imposing a predictable margin between partitions and

gradual diminishing of effect between them. Furthermore, this method elegantly handles irregularly sampled data and hole filling because of its multiresolutional approach.

A longer version of this paper with color images can be found at the following URL: `http://erie.nlm.nih.gov/hrbf`.

References

1. Savchenko, V.V., Pasko, A.A., Okunev, O.G., Kunii, T.L.: Function representation of solids reconstructed from scattered surface points and contours. Computer Graphics Forum **14**(4) (1995) 181–188
2. Turk, G., O'Brien, J.F.: Shape transformation using variational implicit functions. Computer Graphics **33**(Annual Conference Series) (1999) 335–342
3. Yngve, G., Turk, G.: Creating smooth implicit surfaces from polygonal meshes. Technical Report GIT-GVU-99-42, Georgia Institute of Technology (1999)
4. Morse, B.S., Yoo, T.S., Rheingans, P., Chen, D.T., Subramanian, K.R.: Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In: Shape Modeling International 2001, Genoa, Italy (2001) 89–98
5. Carr, J.C., Beatson, R.K., Cherrie, J.B., Mitchell, T.J., Fright, W.R., McCallum, B.C., Evans, T.R.: Reconstruction and representation of 3D objects with radial basis functions. In: Proceedings of SIGGRAPH 2001. (2001) 67–76
6. Turk, G., O'Brien, J.F.: Modelling with implicit surfaces that interpolate. ACM Transactions on Graphics **21**(4) (2002) 855–873
7. Dinh, H., Turk, G., Slabaugh, G.: Reconstructing surfaces by volumetric regularization using radial basis functions. IEEE Trans. on Pattern Analysis and Machine Intelligence (2002)
8. Wendland, H.: Fast evaluation of radial basis functions: Methods based on partition of unity. In Chui, C.K., Schumaker, L.L., Stöckler, J., eds.: Approximation Theory X: Wavelets, Splines, and Applications, Vanderbilt University Press, Nashville, TN (2002) 472–483
9. Kojekine, N., Hagiwara, I., Savchenko, V.: Software tools using CSRBFs for processing scattered data. Computers & Graphics **27**(2) (2003) 311–319
10. Ohtake, Y., Belyaev, A., Seidel, H.: A multi-scale approach to 3d scattered data interpolation with compactly supported basis functions. In: Shape Modeling International 2003. (2003)
11. Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., Seidel, H.: Multi-level partition of unity implicits. ACM TOG (Proc. SIGGRAPH 2003) **22**(3) (2003) 463–470
12. Tobor, I., Reuter, P., Schlick, C.: Multiresolution reconstruction of implicit surfaces with attributes from large unorganized point sets. In: Proceedings of Shape Modeling International (SMI 2004). (2004) 19–30
13. Ohtake, Y., Belyaev, A., Seidel, H.P.: 3d scattered data approximation with adaptive compactly supported radial basis functions. In: Shape Modeling International 2004. (2004)
14. Shen, C., O'Brien, J.F., Shewchuk, J.R.: Interpolating and approximating implicit surfaces from polygon soup. In: Proceedings of ACM SIGGRAPH 2004, ACM Press (2004) 896–904
15. Nielson, G.M.: Radial hermite operators for scattered point cloud data with normal vectors and applications to implicitizing polygon mesh surfaces for generalized CSG operations and smoothing. In: 15th IEEE Visualization 2004 (VIS'04). (2004) 203–210
16. Morse, B., Liu, W., Otis, L.: Empirical analysis of computational and accuracy tradeoffs using compactly supported radial basis functions for surface reconstruction. In: Proceedings Shape Modeling International (SMI'04). (2004) 358–361
17. Floater, M., Iske, A.: Multistep scattered data interpolation using compactly supported radial basis functions. Journal of Comp. Appl. Math. **73** (1996) 65–78
18. Iske, A., Levesley, J.: Multilevel scattered data approximation by adaptive domain decomposition. In: Numerical Algorithms. Volume 39. (2005) 187–198

A New Class of Non-stationary Interpolatory Subdivision Schemes Based on Exponential Polynomials

Yoo-Joo Choi[1], Yeon-Ju Lee[2], Jungho Yoon[2], Byung-Gook Lee[3], and Young J. Kim[4,⋆]

[1] Dept. of CS., Seoul Univ. of Venture and Info., Seoul, Korea
yjchoi@suv.ac.kr
[2] Dept. of Math., Ewha Womans Univ., Seoul, Korea
{lee08, yoon}@ewha.ac.kr
[3] Div. of Internet Engineering, Dongseo Univ., Busan, Korea
lbg@dongseo.ac.kr
[4] Dept. of CS., Ewha Womans Univ., Seoul, Korea
kimy@ewha.ac.kr

Abstract. We present a new class of non-stationary, interpolatory subdivision schemes that can exactly reconstruct parametric surfaces including exponential polynomials. The subdivision rules in our scheme are interpolatory and are obtained using the property of reproducing exponential polynomials which constitute a *shift-invariant space*. It enables our scheme to exactly reproduce rotational features in surfaces which have trigonometric polynomials in their parametric equations. And the mask of our scheme converges to that of the polynomial-based scheme, so that the analytical smoothness of our scheme can be inferred from the smoothness of the polynomial based scheme.

1 Introduction

Subdivision surfaces are defined in terms of successive refinement rules that can generate smooth surfaces from initial coarse meshes. More formally, starting with the coarse control points $\mathbf{P^0} = \{ \mathbf{p_n^0} \mid n \in Z^d \}$, recursive application of the subdivision rule S_k defines a new denser set of points $\mathbf{P^k} = \{ \mathbf{p_n^k} \mid n \in Z^d \}$ which can be written as

$$\mathbf{P^k} = S_k \cdots S_0 \mathbf{P^0}, \quad k \in Z_+.$$

Here, a subdivision scheme is said to be stationary if S_k is the same regardless of k; otherwise it is called non-stationary [1]. Interpolatory subdivision schemes refine data by inserting values corresponding to intermediate points using a linear combination of neighboring points. As a result, in the limit, they keep the original data exactly. Furthermore, interpolatory subdivision has become a tool for multi-resolution analysis and wavelet construction on general manifolds (even in complicated geometric situations) using the *lifting scheme* [2].

Recently, concepts of refinement taken from signal processing have been applied to refinement of a triangular mesh in digital geometry processing. In the context of signal processing, a subdivision scheme can be seen as upsampling followed by filtering [3].

⋆ Corresponding author. Tel.: +82-2-3277-4068.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 563–570, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Moreover, in the univariate case it has been shown that a non-stationary subdivision scheme based on exponential or trigonometric polynomials with a suitable frequency factor works more effectively on highly oscillatory signals than stationary interpolatory scheme based on polynomials [4]. Thus, for instance, the Butterfly scheme [5], which is based on cubic polynomial interpolation, may not accurately reproduce highly oscillatory triangular mesh data. Motivated by these issues, we introduce a new class of non-stationary interpolatory subdivision schemes that can exactly reproduce a complicated, parametric surface including exponential polynomials in the sense of complex numbers; thus, our scheme can handle trigonometric and exponential functions as well as polynomials. The main idea of our scheme is that exponential polynomials constitute a *shift-invariant* space, and the mask of our subdivision scheme is constructed in such a way as to find the values of the exponential polynomials that correspond to the initial control points. As a result, the subdivision converges to the original surface as the subdivision level increases. Moreover, the shift-invariant property ensures that local weights corresponding to local control points are invariant, regardless of their locations, which ultimately enables parametric surfaces to be generated exactly. Furthermore, thanks to the linearity of our subdivision rules, a complicated surface generated from both polynomials and trigonometric polynomials can be exactly reproduced from a set of coarse initial points.

Not much work has been reported on the accurate reconstruction of given complicated surfaces by subdivision. Hoppe et al. [6] presented a method to reconstruct piecewise smooth surface models from scattered range data using a variation of Loop's scheme [7]. Morin et al. [8] addressed the issue of reconstructing rotational features in surfaces as special cases. Jena et al. [9] and Joe Warren [10] presented a non-interpolatory scheme based on the exponential splines for curve. But the exponential splines are non-interpolatory scheme for curves. Spline schemes such as [9] aim at reproducing only trigonometric curves like a circle, elliptic, helix but not surfaces. Jena et al. [11] presented a non-interpolatory scheme for tensor product bi-quadratic trigonometric spline surfaces.

2 Interpolatory Subdivision Scheme

2.1 Construction Rules

The construction of our subdivision scheme is based on the property of reproducing a certain class of exponential polynomials, which actually constitute a parametric surface F(u,v) defined on a planar parametric domain $\Omega \subset R^2$, where $F(u,v)$ can contain polynomials, trigonometric and exponential functions. Interpolatory subdivision schemes refine data by inserting new vertices, corresponding to intermediate points, into a planar triangulation by using linear combinations of neighboring vertices. The sampling density of the initial control points is assumed to satisfy the Nyquist rate [12]. The subdivision rules proposed in this paper are constructed so as to accommodate the class S of functions capable of producing parametric surfaces such as trigonometric surfaces and rotational features. In order to formulate a non-stationary rule, the space S is required to be *shift invariant* [13]: that is,

$$f \in S \quad \text{implies} \quad f(\cdot - \alpha) \in S, \quad \alpha \in Z. \tag{1}$$

Central ingredients in our construction are exponential polynomials of the form

$$\phi(u,v) = u^{\alpha_1} v^{\alpha_2} e^{\beta_1 u} e^{\beta_2 v}, \quad (u,v) \in \Omega \subset R^2, \tag{2}$$

where $\alpha_i = 0, \cdots, \mu$ for some non-negative integer μ and complex numbers β_i $(i = 1,2)$.

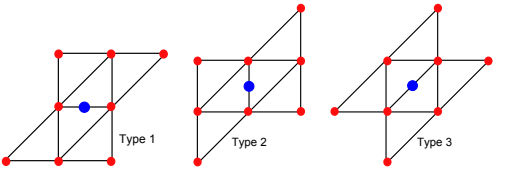


Fig. 1. Stencils based on butterfly shapes. The blue dot represents the new insertion point.

Let

$$S = \text{span}\{\phi_n(u,v) | n = 1, \cdots, N\}$$

be a shift-invariant space with linearly independent ϕ_n's of the form given in Eq. 2. For each level (say k), the non-stationary subdivision rule is constructed by solving the linear system:

$$\phi_n(p2^{-k-1}) = \sum_{\ell \in \mathscr{X}} a^{[k]}_{p-2\ell} \phi_n(\ell 2^{-k}), \quad \phi_n \in S,\ k = 0,1,\cdots, \tag{3}$$

where $p/2$ is the insertion point in the triangulation at level 0, and $\mathscr{X}$ is its corresponding stencil (see Fig.1). This linear system can be written in matrix form as

$$\mathbf{a}^{[k]} = {\mathbf{B}^{[k]}}^{-1} \mathbf{b}^{[k]}, \tag{4}$$

where $\mathbf{a}^{[k]} = (a^{[k]}_{p-2\ell} : \ell \in \mathscr{X})$, $\mathbf{B}^{[k]} = (\phi_n(\ell 2^{-k}) : \ell \in \mathscr{X},\ n = 1, \cdots, N)$ and $\mathbf{b}^{[k]} = (\phi_n(p2^{-k-1}) : n = 1, \cdots, N)$. Refining the triangular mesh involves three groups of vertices (termed stencils) to evaluate three types of new vertices depending on their locations, as shown in Fig.1. There are then two important arguments to be discussed:

- **Uniqueness:** For each stencil (say $\mathscr{X}$) of the rule it is required that $\dim(S|_{\mathscr{X}}) = \dim S$. This guarantees a unique solution to the linear system (Eq. 3).
- **Non-stationary rule:** The shift-invariant property implies that the rule is the same everywhere at the same level k but may vary between different levels.

The configuration of the stencil $\mathscr{X}$ and the space S may differ, depending on the target surface $F(u,v)$. We begin with a butterfly-shaped stencil (shown in Fig.1) which can be considered as a non-stationary version of the well-known Butterfly scheme [5]. We will see that this new scheme provides the same smoothness and approximation order as the original Butterfly scheme, with the additional advantages that: (1) it reproduces certain types of rotational features; (2) there are flexibilities in the choice of S and frequency factors β_i $(i = 1,2)$ in Eq. 2. In fact, any surface whose parametric equations constitute a shift-invariant space with fewer than eight basis functions can be reconstructed exactly using this butterfly stencil. As simple examples we take the sphere and torus.

Example 1. (Sphere and Torus) Let us define S as:

$$S := \text{span}\{\ \sin u, \cos u, \sin v, \cos v,\ \sin u \sin v, \cos u \cos v, \sin u \cos v, \cos u \sin v\}.$$

Then the three types of refinement rule are obtained by solving the linear system of Eq. 3 using the butterfly stencil. Recursive application of this subdivision rule exactly generates a sphere whose parametric equation $F(u,v)$, $0 \le v \le 2\pi,\ 0 \le u \le \pi$, is:

$$x(u,v) = r \sin u \cos v, \quad y(u,v) = r \sin u \sin v, \quad z(u,v) = r \cos u.$$

Similarly, the non-stationary scheme reconstructs a torus whose $F(u,v)$ is given by

$$x(u,v) = (1+\cos u)\cos v,\ y(u,v) = (1+\cos u)\sin v,\ z(u,v) = \sin u,$$

where $0 \le u, v \le 2\pi$.

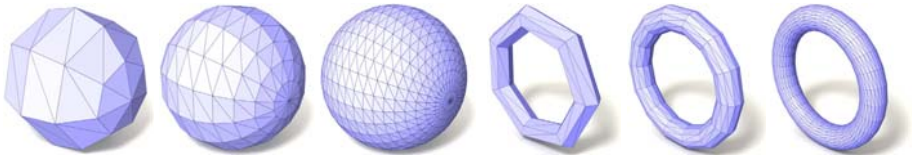


Fig. 2. Exact reconstruction of a sphere and a torus

Later we will introduce additional types of stencil which can reproduce more complicated parametric surfaces.

2.2 Asymptotic Equivalence

As the level of refinement increases, the mask at level k, $\{a_n^{[k]}\}$, of our non-stationary scheme converges to that, $\{a_n\}$, of a stationary scheme based on polynomial interpolation; this property is also known as 'asymptotic equivalence' between the two schemes in the sense of

$$|a_n^{[k]} - a_n| = O(2^{-k})$$

as k increases [14]. The asymptotic equivalence property guarantees that, in the limit, the non-stationary subdivision converges uniformly to a continuous surface. This observation plays a key role in proving that the non-stationary scheme has the same smoothness as the stationary Butterfly scheme.

2.3 Smoothness and Approximation Order

The central ingredient in the analysis of non-stationary schemes is the asymptotic equivalent relation between schemes as shown in Thm 1. The analytical properties of a stationary scheme are well-understood [15], and so the smoothness of a non-stationary scheme can be inferred from the stationary scheme to which it is asymptotically equivalent. On the other hand, another important issue in devising subdivision algorithms is how to attain the original function as closely as possible when the initial data is sampled from the underlying function. A high quality reconstruction scheme should guarantee that the approximation error decreases as the sample rates increase. We can find the following results from [14]:

Theorem 1. *Let* $\{S_{a^{[k]}}\}$ *is the non-stationary interpolatory subdivision scheme with a butterfly stencil. Then, we have*

1. *The scheme* $\{S_{a^{[k]}}\}$ *is* C^1*, i.e., it has the same smoothness as the stationary Butterfly subdivision scheme* $\{S_a\}$.
2. *The scheme* $\{S_{a^{[k]}}\}$ *has the approximation order* 4 *on any compact* K *in* R^2.

3 Reconstruction of Mathematical Surfaces

In this section, we extend the basic interpolatory subdivision scheme to a more general scheme which is able to reconstruct mathematical parametric surfaces. For a given parametric surface F, we first explain how to construct a shift-invariant space S along with its basis functions that constitute F, and then provide a general algorithm that creates a stencil $\mathscr{X}$ upon which our subdivision scheme to reconstruct F is based.

3.1 Finding the Shift Invariant Space

From a given parametric surface F, our goal is to find its shift invariant space, S, in the smallest dimension: we want to minimize the dimension of S since we want to use as few vertices as possible in the subdivision rules.

To construct the shift-invariant space S, we first search for a finite collection of functions, B, whose elements generate F via a linear combination. To find B, we initialize B as an empty set and enumerate all the linearly independent monomials constituting F. Then we incrementally add each of these monomials to B, provided that the monomial is linearly independent to the elements already added to B. Once we have B, we find S that generates S as follows ;

1. Initially, we set S to be the same as span B.
2. We pick an element f_i from S and perform a constant shifting of f_i as shown in Eq. 1; i.e., compute $f_i(\cdot - \alpha)$.
3. We enumerate all the monomials f_i^j constituting $f_i(\cdot - \alpha)$.
4. For each f_i^j, if it can be generated by the current S, then we do not include it in S and continue to check other f_i^js; otherwise we add it to S.
5. Repeat between 2 and 4 until there is no possible new addition to S.
6. Finally, we have the shift invariant space $S := \text{span}S$.

3.2 Stencil Generation Algorithm

Once we have formed the shift-invariant space S for a given surface F, and an appropriate stencil $\mathscr{X}$, then the linear system in Eq. 4 must have a unique solution. This means that the stencil $\mathscr{X}$ must satisfy $\dim(S|_{\mathscr{X}}) = \dim S$ and, equivalently, $\dim \mathbf{B} = \dim S$ in Eq. 4; i.e., $\mathbf{B}$ is invertible. The solution of this system provides a mask set corresponding to the stencil $\mathscr{X}$ in the subdivision rule. However, the stencil that satisfies the linear system in Eq. 3 is not necessarily unique. Moreover, we need to keep the stencil symmetric and concentrated around the vertex to be refined, so that the resulting mask set is symmetric and has as small a support as possible. This property preserves the locality of subdivision rules and reduces the computational costs when these rules are applied.

We construct a stencil $\mathscr{X}$ starting from a newly inserted point p. First, we choose two stencil vertices (say v_1 and v_2) connected to p and continue to search for other stencil vertices by expanding v_1 and v_2 to their neighborhood (more specifically, n-ring neighbors) while keeping the stencil shape symmetric. Here, the n-ring neighbors of v are defined as vertices that are reachable from v by traversing no more than n edges in the mesh. This search process continues until the associated matrix $\mathbf{B}$ satisfies $\dim \mathbf{B} = \dim S$. This search process can be efficiently implemented using breadth first traversal of the initial subdivision mesh.

We will now provide an example of generating stencils for a complicated surface: the Möbius strip.

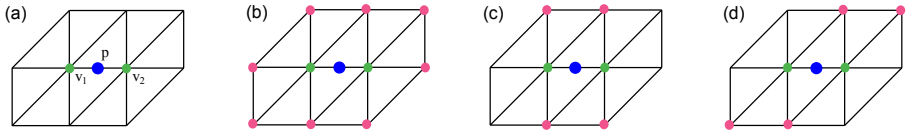


Fig. 3. Type 1 stencil for a Möbius strip. (a) The newly inserted point p is colored blue and the two green points are chosen as the immediate neighbors (v_1, v_2). (b) The one-ring neighbors of v_1, v_2 are colored pink. (c) A candidate stencil, turns out to be invalid. (d) A valid stencil.

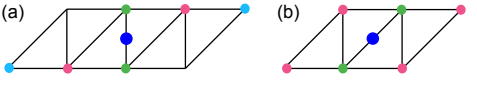


Fig. 4. Type 2 and 3 stencils for a Möbius strip

Example 2. (Möbius strip) The parametric equation $F(u,v)$ for Möbius strip is given by

$$\begin{aligned} x(u,v) &= a\cos u + v\cos(u/2), \\ y(u,v) &= a\sin u + v\cos(u/2), \\ z(u,v) &= v\sin(u/2), \end{aligned}$$

where $0 \le u \le 2\pi$, $-w \le v \le w$, and w and a are constants.

Then the space that generates the parametric surface with the smallest dimension is

$$B := \{\sin u, \cos u, v\sin(u/2), v\cos(u/2)\},$$

and the corresponding shift-invariant space S that generates $F(u,v)$ is

$$S := \text{span}\{\sin u, \cos u, \sin(u/2), \cos(u/2), v\sin(u/2), v\cos(u/2)\}.$$

To find the stencil, we initiate a search from the newly inserted point p as shown in Fig. 3-(a) (Type 1 stencil). Then, we choose two closest vertices (the green dots in Fig. 3-(a)), v_1, v_2, connected to p. Since $\dim S = 6$, we now need to find four more vertices.

We may choose any vertex among the one-ring neighbors (pink dots in Fig. 3-(b)) of $\{v_1, v_2\}$; however, to preserve the symmetry, we may choose four vertices like those in Fig. 3-(c). But the resulting stencil does not satisfy $\dim \mathbf{B} = 6$, so we discard these vertices and seek others. Finally, we locate the four vertices shown in Fig. 3-(d); since the resulting stencil satisfies $\dim \mathbf{B} = 6$, we can stop the search. We can find stencils for different types of points in a similar way, as shown in Fig. 4-(a) (Type 2) and 4-(b) (Type 3). Notice that in case of Type 2 point, we include two-ring neighbors (light blue dots) to create a symmetric stencil, because $\dim \mathbf{B} = 6$ cannot be satisfied with one-ring neighbors alone.

Fig. 5 shows the reconstruction results of two benchmarking models. A figure-8 klein bottle is defined by

$$
\begin{aligned}
x(u,v) &= (a + \cos(u/2)\sin v - \mathit{sin}(u/2)\sin(2v))\cos u,\\
y(u,v) &= (a + \cos(u/2)\sin v - \mathit{sin}(u/2)\sin(2v))\sin u,\\
z(u,v) &= \sin(u/2)\sin v + \cos(u/2)\sin(2v),
\end{aligned}
$$

where $0 \le u \le 2\pi$, $0 \le v \le 2\pi$ and some constant a.

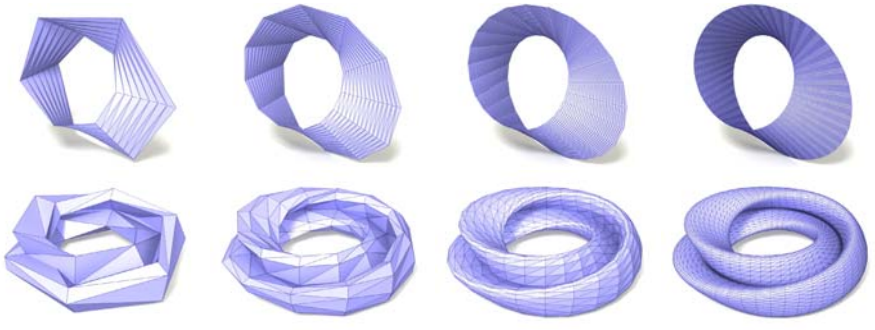


Fig. 5. Reconstruction of complicated surfaces. The rows, from top to bottom, show the reconstruction of a möbius strip and a figure-8 klein bottle. The images, from left to right, show the initial mesh, and subdivision levels 1, 2, and 3.

4 Limitations and Future Work

Our scheme cannot reconstruct parametric surfaces that contain non-exponential polynomials such as a logarithmic functions and division terms, which actually require non-uniform masks of subdivision for the exact reconstruction of such functions. Nor can it handle a mesh with arbitrary topology. There are several areas for future work. First of all, extending our scheme to a mesh with arbitrary topology is an immediate challenge. We would also like to work on regenerating exact surface normals by subdivision. And if we could reconstruct surfaces containing singular points we would be able to address many additional interesting applications.

Acknowledgements. Young J. Kim is sponsored in part by the grant KRF-2004-205-D00168 of the KRF funded by the Korean government, the ITRC program and the

MOST STAR program. Jungho Yoon and Yeon-Ju Lee are supported in part by the grant KRF-2005-015-C00015 funded by Korea Government (MOEHRD, Basic Reasearch Promotion Fund).

References

1. Zorin, D., Schröder, P.: Subdivision for modeling and animation. SIGGRAPH Course Notes (2000)
2. Sweldens, W.: The lifting scheme : a construction of second generation wavelets. SIAM J. Math. Anal. (1998) 511–546
3. Guskov, I., Sweldens, W., Schroder, P.: Multiresolution signal processing for meshes. Proc. of ACM SIGGRAPH (1999) pp. 325 – 334
4. Dyn, N., Levin, D., Luzzatto, A.: Refining Oscillatory Signals by Non-Stationary Subdivision Schemes. In: Modern Developments in Multivariate Approximation. Volume 145 of Internat. Ser. Numer. Math. Birkhäuser (2002)
5. Dyn, N., J.A., Gregory, Levin, D.: A butterfly subdivision scheme for surface interpolation with tension control. ACM Trans. Graph. **9** (1990) 160–169
6. Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J., Stuetzle, W.: Piecewise smooth surface reconstruction. In: Proceedings of ACM SIGGRAPH. (1994) 295–302
7. Loop, C.: Smooth subdivision surfaces based on triangles. Master's thesis, Department of Mathematics, University of Utah (1987)
8. Morin, G., Warren, J., Weimer, H.: A subdivision scheme for surfaces of revolution. Comp. Aided Geom. Design **18** (2001) 483–502
9. Jena, M.J., Shunmugaraj, P., Das, P.J.: A sudivision algorithm for trigonometric spline curves. Comp. Aided Geom. Desig. **19** (2002) 71–88
10. Warren, J., Weimer, H.: Subdivision methods for geometric design. Academic press (2002)
11. Jena, M.J., Shunmugaraj, P., Das, P.J.: A non-stationary subdivision scheme for generalizing trigonometric spline surfaces to arbitrary meshes. Comp. Aided Geom. Desig. **20** (2003) 61–77
12. McClellan, J.M., Schafer, R.W., Yoder, M.A.: DSP First: A Multimedia Approach. Prentice Hall (1998)
13. Chenny, E., Light, W., Light, W.: A Course in Approximation Theory. Brooks Cole (1999)
14. Yoon, J.: Analysis of non-stationary interpolatory subdivision schems based on exponential polynomials. Ewha womans university tech. document (http://graphics.ewha.ac.kr/subdivision/sup.pdf) (2005)
15. Dyn, N.: Subdivision Schemes in Computer-Aided Geometric Design. In: Advances in Numerical Analysis Vol. II: Wavelets, Subdivision Algorithms and Radial Basis Functions. Oxford University Press (1992)

Detection of Closed Sharp Feature Lines in Point Clouds for Reverse Engineering Applications

Kris Demarsin[1], Denis Vanderstraeten[2], Tim Volodine, and Dirk Roose

[1] Department of Computer Science, Celestijnenlaan 200A, B-3001 Heverlee, Belgium
kris.demarsin@cs.kuleuven.be
[2] Metris N.V., Interleuvenlaan 86, B-3001 Leuven, Belgium

Abstract. The reconstruction of a surface model from a point cloud is an important task in the reverse engineering of industrial parts. We aim at constructing a *curve network* on the point cloud that will define the border of the various surface patches. In this paper, we present an algorithm to extract *closed* sharp feature lines, which is necessary to create such a closed curve network. We use a first order segmentation to extract candidate feature points and process them as a graph to recover the sharp feature lines. To this end, a minimum spanning tree is constructed and afterwards a reconnection procedure closes the lines. The algorithm is fast and gives good results for real-world point sets from industrial applications.

1 Introduction

Feature lines can be mathematically defined via local extrema of principal curvatures along corresponding principal directions. These feature lines can be used for visualization purposes: point clouds are visually easier to understand if the feature lines are highlighted in the visualization. In addition, the quality of a mesh can be improved when the feature lines are known. Shape recognition and quality control are other application areas of feature line extraction.

Many feature line extraction algorithms rely on a triangular mesh as input, e.g. [2], [6], [7], [8], and [14]. Few algorithms only use a point cloud, e.g. [1] and [5]. However, these existing methods usually result in pieces of unconnected feature lines, making it hard to segment a point cloud or mesh into surface patches, based on these lines.

Since we aim at constructing a curve network on a point cloud, that will define the border of the various surface patches, this paper focusses on finding closed sharp feature lines. We use a region growing method, which is a modification of the method of Vanco et al. ([9], [10], and [11]), to segment a point cloud in clusters of points (*segments*) and to detect sharp edges. We build and manipulate a graph of these segments, resulting in closed sharp feature lines that fit the segments such that the algorithm can be used as a pre-process step to find the areas where a surface patch can be defined. We are interested in point clouds from industrial applications, where closed sharp feature lines can be detected. It is not our goal to segment clouds with free form surfaces or fillets with a

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 571–577, 2006.
© Springer-Verlag Berlin Heidelberg 2006

large radius. The algorithm differs from the existing feature line algorithms by the fact that it reconstructs *closed sharp* feature lines. The advantages of the algorithm are that (a) it is meshless, i.e. only the coordinates of the points are used, (b) it does not use curvature information, that is difficult to estimate in a noisy environment, (c) it intelligently clusters the points to create a graph that is much smaller than the original cloud, thus making it practical for large point clouds and, (d) it constitutes a pre-process step for surface reconstruction.

The algorithm is explained in the next section. In Sect. 3 we illustrate some results of the algorithm applied to realistic point clouds, i.e. point clouds obtained from scanning industrial parts. We formulate the conclusions in Sect. 4.

2 Algorithm Overview

Given a point cloud, we extract closed polygonal lines indicating the sharp edges. Algorithm 1 gives the different steps of the algorithm which will be explained briefly in this section. We illustrate the algorithm with a point cloud representing two intersecting cylinders. The results of each step are depicted in Fig. 1, 2 and 3, where the black lines approximate the sharp edges. For a more detailed explanation of the algorithm we refer to [4].

Algorithm 1. High level description of the algorithm

1. Segment point cloud using the normals $\Rightarrow$ point clusters (*segments*) (Fig. 1)
2. Build graph G_{all} connecting neighboring segments (Fig. 2 and 3(a))
3. Add edges, indicating a piece of a sharp feature line, to $G_{all} \Rightarrow G_{extended}$ (Fig. 3(b))
4. Build the pruned minimum spanning tree of $G_{extended} \Rightarrow G_{pruned_mst}$ (Fig. 3(c))
5. Prune short branches in $G_{pruned_mst} \Rightarrow G_{pruned_branches}$ (Fig. 3(d))
6. Close the sharp feature lines in $G_{pruned_branches} \Rightarrow G_{closed}$ (Fig. 3(e))
7. Smooth the sharp feature lines in $G_{closed} \Rightarrow G_{smooth}$ (Fig. 3(f))

The first step of the algorithm divides a point cloud in different clusters of points. We use the Delaunay neighborhood [12] as an approximation of the 1-ring neighborhood. The normal vectors are estimated by the PCA analysis of these 1-ring neighbors, as explained in [13]. The segmentation method we use is a modification of the region growing method described by Vanco et al. ([9], [10], and [11]): we use one threshold angle which specifies the maximum acceptable angle between two adjacent normals in one segment. At a sharp edge, the normal estimation depends heavily on the computed 1-ring neighborhood, since this neighborhood is very local and these neighbors are located on both sides of the sharp edge. This means that the variation of the normals along a sharp edge is high, resulting in large segments with low variation of the normals bounded by small segments with high normal variation. Since these small segments indicate the sharp edges, we build a graph at segment level in the next step of the algorithm. Figure 1 illustrates the result of the first order segmentation, applied to the point cloud of the two intersecting cylinders, with each point colored

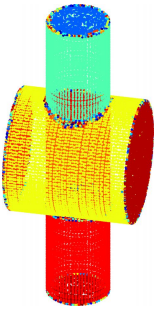


Fig. 1. First order segmentation of two intersecting cylinders

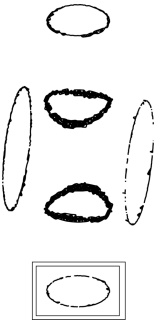


Fig. 2. The graph G_{all}; the area bounded by the rectangle is used to illustrate the following steps of the algorithm in detail

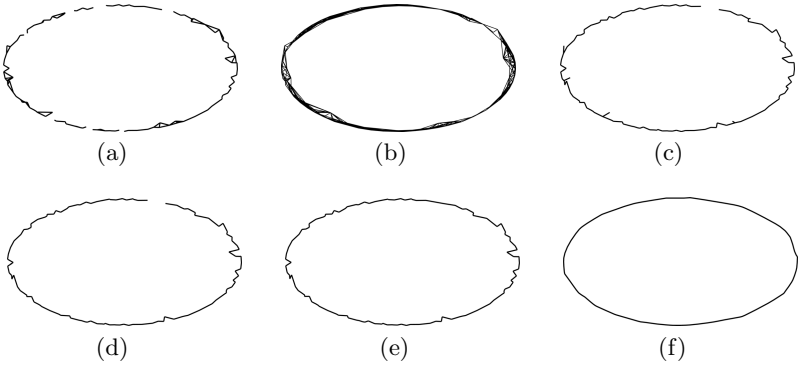


Fig. 3. Result of each step of the algorithm illustrated with the detail of G_{all} indicated by the rectangle in Fig. 2. (a) G_{all}; (b) $G_{extended}$; (c) G_{pruned_mst}; (d) $G_{pruned_branches}$; (e) G_{closed}; (f) G_{smooth}.

corresponding to the segment it belongs to. Contrary to the method of Vanco et al. ([9], [10], and [11]), each cylindrical piece, bounded by sharp edges, consists of only one large segment with many small segments defining the boundary.

In step 2 we construct the connected graph G_{all} where each vertex represents a segment and each edge connects two segments that contain at least one point with overlapping 1-ring neighborhood (see the graph of small segments in Fig. 2 and 3(a)). From now on, we only process the graph and the point cloud is not needed anymore. Since our goal is to extract closed lines, we add edges to the graph G_{all} connecting two small segments that share two large neighboring segments, resulting in less unwanted 'gaps'. The corresponding graph $G_{extended}$ is illustrated in Fig. 3(b). To reduce the cycles we construct the minimum spanning tree of $G_{extended}$, with well chosen weights, and we remove the edges involving a large segment, which results in G_{pruned_mst}, see Fig. 3(c). To remove unneces-

sary endpoints, i.e. vertices in the graph with exactly one incident edge, we now prune the graph, see Fig. 3(d), resulting in the graph $G_{pruned_branches}$. In the next step, we use a 'connect' algorithm to link each endpoint with a suited point in the graph such that no small cycles are generated. The corresponding graph G_{closed} can be seen in Fig. 3(e). If no suitable point is found to connect with an endpoint, we remove all the edges starting from that endpoint until a point with more than two incident edges is reached. The connect algorithm can be seen as a last clean up step: 'noisy' branches are pruned and all lines are closed. The method we use to get a smooth graph G_{smooth}, see Fig. 3(f), is explained in [3]. More details concerning the algorithm can be found in [4].

3 Results

Figure 4 illustrates the final result for a detail of a mobile phone, a typical example of a point cloud used in industrial applications. This point cloud has been generated by a laser scanner and thus some noise is present. We note that the algorithm does not guarantee good results when two sharp feature lines are located too close to each other depending on neighborhood selection and point density: there is an unwanted gap at the right caused by the pruning, and there exist some edges connecting two sharp feature lines. The final result for a larger part of a mobile phone, also real data from a scanner, is illustrated in Fig. 5. Contrary to the previous point clouds, this point cloud does not respresent a solid, i.e. it has a boundary which we extract and include in the graph. By comparing the final graph with the segmentation, we see how well the detected lines fit the segmentation, but we see also that a few sharp feature lines are not detected because they consist of cycles which are too short. Additionally, where two lines are located too close to each other, they are extracted as one line.

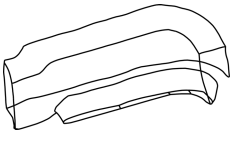


Fig. 4. Final result for the mobile phone point cloud

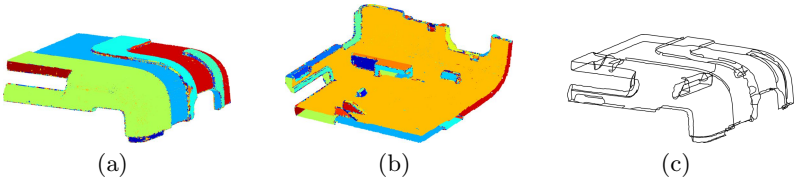


Fig. 5. Results for a larger detail of a mobile phone. (a) and (b) illustrate two different views of the segmentation; (b) G_{smooth}.

Table 1. Information about the segmentation for the different point clouds

	Size of point cloud	Average size small segments	Number of small segments	Number of large segments
Phone small	11034	1.24	1610	37
Cylinders	26846	1.11	1696	10
Phone large	110053	1.62	5247	69

Table 2. Complexity and time consumption of the different steps. The segmentation is decomposed in two steps: the normal estimation ($\overrightarrow{n}$) and the region growing (RG). The timings are in seconds and generated on an Intel Pentium 4, 3.20 GHz.

Edges	G_{all}	$G_{extended}$	G_{pruned_mst}	$G_{pruned_branches}$	G_{closed}
Phone small	5206	8776	1604	1225	1170
Cylinders	6086	8984	1690	1123	1076
Phone large	16198	27538	5199	4040	4579
Vertices					
Phone small	1647	1647	1607	1228	1162
Cylinders	1706	1706	1696	1129	1076
Phone large	5316	5316	5215	4056	4527

Time (seconds)	$\overrightarrow{n}$	RG	G_{all}	$G_{extended}$	G_{pruned_mst}	$G_{pruned_branches}$	G_{closed}	**Total**
Phone small	1.69	0.11	0.18	0.39	0.05	0.04	0.09	**2.55**
Cylinders	3.71	0.2	0.23	0.49	0.06	0.04	0.08	**4.81**
Phone large	16.01	1.06	0.66	2.32	0.17	0.12	0.96	**21.30**

Table 1 presents information about the segmentation, e.g. because of the high normal variation at the sharp edges, the average size of a small segment is close to unity. In the case of the cylinders, we see that the segmentation results in the correct number of large segments: the segmentation fits the extracted sharp feature lines perfectly. Table 2 gives for every step of the algorithm the number of vertices and edges of the corresponding graph. In the case of the two intersecting cylinders, we start with a point cloud of 26846 points and then we build a graph G_{all} of 1706 vertices and 6086 edges. After adding edges to G_{all}, every following step reduces the memory consumption of the graph: a huge reduction in the number of edges happens when building G_{pruned_mst} and $G_{pruned_branches}$. In general, in the close step, more edges are removed than added, since noisy branches are pruned. Note that for the large mobile phone point cloud the boundary is included just before the close step. The table also illustrates the time consumption of the algorithm. The segmentation step requires more time compared to the other steps of the algorithm, because this step has to grow through all the points of the point cloud and the normal for each point needs to be estimated. We could make the segmentation much faster by estimating the normal as the

normal of the least squares plane through the k nearest neighbors. However, for realistic point clouds, a small k might generate neighbors only on one side of the sharp edge causing no small segments, and a large k makes it impossible to accurately detect the transition from a smooth surface to a sharp edge.

4 Conclusion and Future Work

We presented an algorithm to extract sharp edges from a point cloud without estimating the curvature and without triangulating the point cloud. Additionally, all extracted lines are *closed* at the end of the algorithm. We start with a very simple region growing method with well chosen normals, resulting in an initial segmentation based on the sharp edges. Afterwards, we build and manipulate a graph of the segments. Using a graph structure at the level of segments yields faster execution times and less memory consumption, making the algorithm suitable for large point clouds. Once we build the graph of the segments, the point cloud is not needed anymore and we only need to process the graph in the following steps: adding extra edges, construction of the minimum spanning tree, pruning, closing and smoothing the sharp feature lines.

Although the segmentation step is time consuming for large point clouds, together with the *closed* lines, it constitutes a pre-process step in finding a curve network. In the future, we plan to construct this network, which consists of a set of loops, where each loop defines the boundary of an area where a patch can be fitted. When all segments bounded by sharp edges are known, in a next step, we can continue with each segment individually, e.g. to detect also tangent continuous but curvature discontinuous features like fillets.

Acknowledgements

The two mobile phone point clouds are courtesy of Metris N.V. Belgium.

References

1. Gumhold, S., Wang, X., MacLeod, R.: Feature Extraction from Point Clouds. Proceedings of the 10th International Meshing Roundtable (2001) 293–305
2. Watanabe, K., Belyaev, A.G.: Detection of Salient Curvature Features on Polygonal Surfaces. Computer Graphics Forum **20(3)** (2001) 385–392
3. Volodine, T., Vanderstraeten, D., Roose, D.: Smoothing of meshes and point clouds using weighted geometry-aware bases. Department of Computer Science, K.U.Leuven, Belgium, Report TW 451 (2006)
4. Demarsin, K., Vanderstraeten, D., Volodine, T., Roose, D.: Detection of closed sharp feature lines in point clouds for reverse engineering applications. Department of Computer Science, K.U.Leuven, Belgium, Report TW 458 (2006)
5. Pauly, M., Keiser, R., Gross, M.H.: Multi-scale Feature Extraction on Point-sampled Surfaces. Comput. Graph. Forum **22(3)** (2003) 281–290
6. Ohtake, Y., Belyaev, A., Seidel, H.-P.: Ridge-Valley Lines on Meshes via Implicit Surface Fitting. SIGGRAPH (2004) 609–612

7. Ohtake, Y., Belyaev, A.: Automatic Detection of Geodesic Ridges and Ravines on Polygonal Surfaces. The Journal of Three Dimensional Images **15(1)** (2001) 127–132
8. Hildebrandt, K., Polthier, K., Wardetzky, M.: Smooth Feature Lines on Surface Meshes. Symposium on Geometry Processing (2005) 85–90
9. Vanco, M., Brunnett, G., Schreiber, Th.: A Direct Approach Towards Automatic Surface Segmentation of Unorganized 3D Points. Proceedings Spring Conference on Computer Graphics (2000) 185–194
10. Vanco, M., Brunnett, G.: Direct Segmentation for Reverse Engineering. Proceedings International Symposium on Cyber Worlds (2002) 24–37
11. Vanco, M., Brunnett, G.: Direct Segmentation of Algebraic Models for Reverse Engineering. Computing **72(1-2)** (2004) 207–220
12. Floater, M. S., Reimers, M.: Meshless parameterization and surface reconstruction. Computer Aided Geometric Design **18(2)** (2001) (77-92)
13. Hormann, K.: Theory and Applications of Parameterizing Triangulations. PhD thesis, Department of Computer Science, University of Erlangen (2001)
14. Stylianou, G., Farin, G.: Crest lines extraction from 3D triangulated meshes. Hierarchical and Geometrical Methods in Scientific Visualization (2003) 269–281

Feature Detection Using Curvature Maps and the Min-cut/Max-flow Algorithm

Timothy Gatzke and Cindy Grimm

Washington University in St. Louis, St. Louis MO 63130, USA

Abstract. Automatic detection of features in three-dimensional objects is a critical part of shape matching tasks such as object registration and recognition. Previous approaches often required some type of user interaction to select features. Manual selection of corresponding features and subjective determination of the difference between objects are time consuming processes requiring a high level of expertise. The *Curvature Map* represents shape information for a point and its surrounding region and is robust with respect to grid resolution and mesh regularity. It can be used as a measure of local surface similarity. We use these curvature map properties to extract feature regions of an object. To make the selection of the feature region less subjective, we employ a min-cut/max-flow graph cut algorithm with vertex weights derived from the curvature map property. A multi-scale approach is used to minimize the dependence on user defined parameters. We show that by combining curvature maps and graph cuts in a multi-scale framework, we can extract meaningful features in a robust way.

1 Introduction

Advances in three-dimensional (3-D) scanning capability are providing ready access to 3-D data. Automatic detection of features in 3-D objects is critical for tasks such as object registration and recognition. For example, identifying corresponding regions between two similar surfaces is a necessary first step toward alignment and registration of those surfaces. A fundamental question is: What constitutes a feature? Man-made objects often have well-defined features such as edges, but features on natural shapes, such as the wrist bones shown in Figure 1, are more subjective. Furthermore, such shapes can have subtle variations, the importance of which may not be obvious.

We aim to detect subtle shape features in a robust way with a fully automated process. The types of features we expect to be useful are peaks, pits, ridges, and valleys. Important features may be of various sizes within one object. We need not (in fact, cannot) detect every feature, and the features we do detect may or may not be unique. We just need to identify enough features to resolve any ambiguities during shape matching. It is desirable for feature detection to be consistent, robust, independent of the mesh resolution, and relatively insensitive to noise.

Previous approaches often required some type of user interaction to select features. Manual selection of corresponding features and subjective determination of the difference between objects are time consuming processes requiring a high level of expertise. In contrast, our approach is entirely automatic.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 578–584, 2006.
© Springer-Verlag Berlin Heidelberg 2006

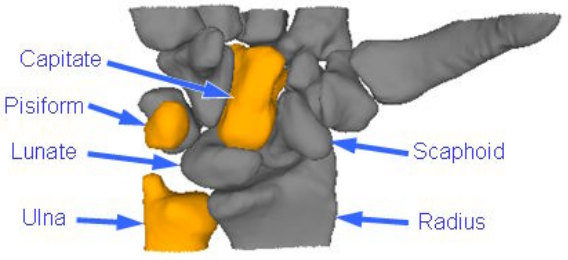


Fig. 1. Bones making up the human wrist. Natural objects have subtle shape variations that are challenging to characterize.

1.1 Approach

In this paper we present a feature detection algorithm based on the *Curvature Map* [1], which at a point represents shape information for the point and its surrounding region. A min-cut/max-flow graph cut algorithm, popular for image segmentation tasks, is employed to identify features at various scales. Results from multiple graph cuts are combined in a novel manner to produce a final feature set. A two-step multi-scale approach eliminates the need for user interaction, and for tuning parameters based on a particular application. This algorithm can extract meaningful features in a robust way.

Section 2 focuses on related work in object recognition, feature detection, and segmentation. In Section 3 we give an overview of the algorithm. Details omitted due to space constraints can be found in [2]. Results for various shapes, and conclusions and possible areas for future work, are presented in Sections 4 and 5 respectively.

2 Related Work

The two main areas of research related to this work are shape representations or signatures, and feature segmentation. Object recognition, correspondence, and registration often rely on similarity measures to quantify the similarity or dissimilarity between objects by computing distances between shape representations, such as sets of points, feature vectors, histograms, signatures, or graph representations. Methods that are more applicable to 2D images rather than 3D object representations will not be discussed here. See [3] for a survey of methods applied to medical images.

Graph representations, such as skeletons [4,5] and multi-resolution Reeb graphs [6], like algorithms based on point sets [7,8], can be useful for computing similarity and registration. But these methods are primarily global rather than local and often can be sensitive to the distribution of the mesh points.

Signatures may be global or local, and provide a compact representation that results in more efficient comparison at the expense of their ability to discriminate shape. Methods used for shape retrieval, such as shape distributions [9], spin images [10], and spherical spin images [11], tend to be global measures, and generally provide limited discrimination between similar shapes.

Signatures of a more local nature include statistical signatures [12] and shape contexts [13], but the use of local point-to-point distances and angles, and sampling of

points respectively, limits the suitability of these methods for detailed shape comparison. The point fingerprint [14], which defines an irregularity measure for geodesic circles around a point, and the surface curvature signature [15] rely on high curvature feature points. Unlike these approaches, we are looking for subtle shape differences that require more than signatures just at 'interesting' points.

Feature regions can be extracted based on critical points (peaks, pits, and passes) and associated ridge and valley lines. In [16], smoothing was required as a preprocessing step. Peak (pit) areas surrounded by valley (ridge) cycles then provide the candidate feature areas to be selected interactively. The uncertainty as to an appropriate amount of smoothing and the narrow definition of a feature are drawbacks to this approach.

Volume decomposition based on topology [17] or morphological tools [18] provides volumetric features rather than surface features. Surface segmentation methods, which identify local regions of an object, have been based on the sign of the curvature [19], isosurfaces and extreme curvatures [20], and watersheds of a curvature function [21,22]. Methods that identify salient features [23,24] have also been developed. However, these methods do not yield the types of features we are interested in for shape matching.

Graph cut algorithms have been used to segment images [25] and medical datasets [26]. They are effective at assigning the vertices of a graph to either a feature (foreground) or background set, based on graph properties such as the gradient of the image intensity. Some of these methods employ an interactive step, where the user identifies feature and background seed points, to guide the algorithm to the objects that are to be separated. By treating our mesh as a graph, we can apply the graph cut algorithm and identify features based on the resulting segmentation.

3 Feature Detection Method

The basic feature shapes we are looking for include the peak, pit, ridge, and valley. The common link between these features is the dependence on the magnitude of the mean curvature. The curvature map [1] provides a context for each point that can be used to define a local shape property to help identify these features.

3.1 Local Shape Property

For a vertex p, the 1-D curvature map, $Kmap(p)$, is defined by two curves representing the average mean and Gaussian curvature as functions of distance from the vertex. We will refer to these curves as $Mean(Kmap(p))$ and $Gauss(Kmap(p))$ respectively. We define our local shape property S as

$$S(p) = \int_0^R Mean(Kmap(p))(r)dr$$

where R represents the radius corresponding to the maximum feature size.

We also considered functions based on the Gaussian curvature component of the curvature map, but given a suitable threshold, the mean curvature function gave the most consistent identification of the features in our test cases.

Algorithm 1. Multi-Scale Feature Detection

```
Read Curvature Map (K_map) for Mesh M
for K_map radius R from R_min to R_max do
  Compute S as the integral of the K_map mean curvature component from 0 to R
  for a range of weight factor α do
    Create graph cuts C_abs, C_pos, C_neg on the absolute, positive, and negative values of S
    Identify the features in C_abs, C_pos, C_neg
    for each vertex v in Mesh M do
      Count feature occurrences N_abs, N_pos, N_neg in C_abs, C_pos, C_neg
    end for
    for each edge do
      count how many times both endpoints occur in the same region
      Note: Used to generate edge weights for the later max-flow/min-cut runs
    end for
  end for
end for
for a range of weight factor α do
  Create graph cuts C_abs, C_pos, C_neg from normalized counts N_abs, N_pos, N_neg
  Identify and merge features from C_abs, C_pos, C_neg into composite feature sets
  G_abs, G_pos, G_neg
end for
Merge G_neg and G_pos into G_abs to create the Master Feature Set G
```

Although the local shape property often highlights the expected features, finding an appropriate threshold requires manual adjustment, and the results still depend on the curvature map radius R. In addition, no single threshold could extract both the positive curvature features (peak and ridge) and the negative curvature features (pit and valley). These factors motivated our search for an improved feature detection approach.

3.2 Multi-scale Algorithm

Combining our local shape property with the min-cut/max-flow graph cutting technique [25] creates a multi-scale approach for feature detection as presented in Algorithm 1. Varying the curvature map radius R detects features at different scales, while increasing the weights by a scale factor α detects less prominent features.

Ranges for these parameters are discussed in [2]. For our examples, we use 8 K_{map} radii cross 10 scale factors, resulting in 80 graph cuts each for the absolute value, positive, and negative of the shape property, plus 30 in the second step, for a total of 270 graph cuts. Fortunately, the graph cut algorithm is very efficient, with the 270 graph cuts on a 10,000 vertex mesh taking less than 40 seconds on a 2.8GHz Pentium 4 processor.

Re-running the graph cut algorithm on the occurrence count maintains focus on the strongest features. Once we have created the graph cut, we form features from contiguous groups of vertices in the feature set of the graph cut. For combining sets of features, a simple greedy approach lets features grow, but without allowing neighboring features to merge. This ensures that all of the features do not get merged into a single feature, as might occur for a very large scale factor.

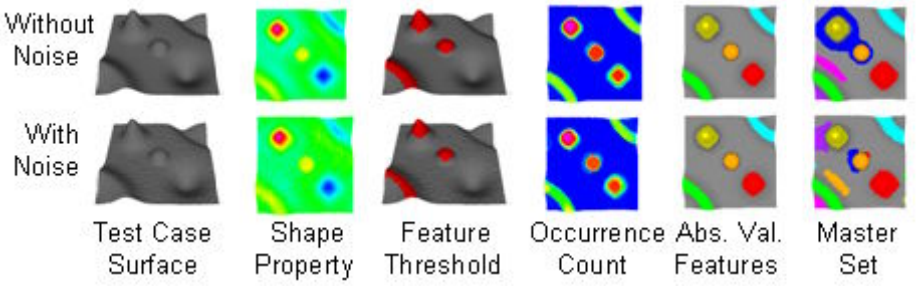


Fig. 2. Test case without and with Gaussian noise added. The function and final feature set are similar for the two cases, especially for the primary features.

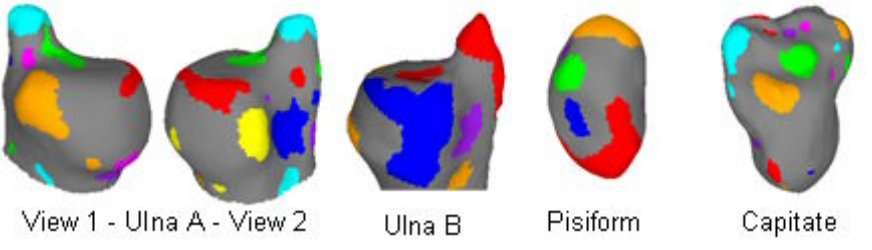


Fig. 3. Master Feature Sets for selected bone meshes. The Ulna is challenging due to the limited number of pronounced features and the significant difference between the scales of the features. Similar features were detected for Cases A and B even though the resolution of the meshes is very different. Reasonable features were also identified for the Pisiform and Capitate.

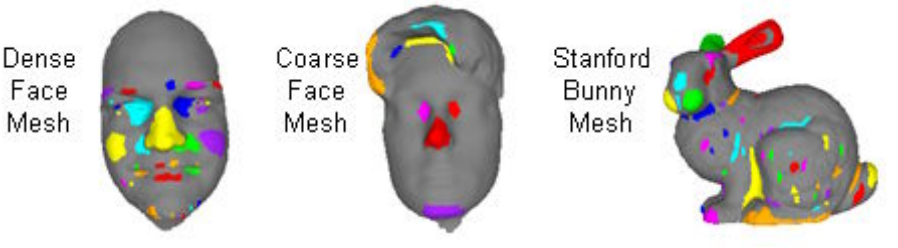


Fig. 4. Features detected for a dense face scan, coarse face scan, and the Stanford bunny. The larger features, which are also generally the strongest features, agree with the intuitive notion of features which may be useful for matching shapes.

4 Results

Figure 2 shows the similar feature structures produced for a simple test surface with and without the addition of Gaussian noise.

The features for several bone meshes are shown in Figure 3. These bones have fairly subtle features. The feature layouts for Ulna A (View 2) and Ulna B are similar despite significant differences in mesh resolution and being from different subjects.

Although the face scans and bunny, presented in Figure 4, produced a number of very small features, the larger feature regions, such as the nose and eyes (face), and ears, feet, and tail (bunny), seem to be features that could be useful for shape matching. Also, features are ordered by strength so that the most significant features can be used first in operations such as shape matching, and the weaker features may not be needed.

5 Conclusions and Future Work

We have presented a two-step multi-scale feature detection approach that uses a local shape function based on the *Curvature Map*. It employs an efficient min-cut/max-flow graph cutting algorithm and greedy algorithm to merge feature sets. The method is robust with respect to noise, and consistently yields a reasonable set of features. Most importantly, there is no user interaction or parameter tuning required.

The method could benefit from alternate algorithms for merging feature sets. The greedy approach works fairly well, but may cause some over-segmentation, since it does not allow two features to coalesce into one, which might be desirable in some instances.

Acknowledgments. This work was partially supported by NSF Grant 049856. The bone data was provided through NIH Grant AR44005, PI: J.J. Crisco, Brown Medical School /Rhode Island Hospital. The authors would also like to thank Vladimir Kolmogorov for the min-cut/max-flow code, Cyberware for the human head scans, and the Stanford Scanning Repository for the bunny data set.

References

1. Gatzke, T., Zelinka, S., Grimm, C., Garland, M.: Curvature maps for local shape comparison. In: Shape Modeling International. (2005) 244–256
2. Gatzke, T.D., Grimm, C.M.: Feature detection using curvature maps and the min-cut/max-flow graph cut algorithm. Technical Report WUCSE-2006-22, Washington University, St. Louis Missouri (2006)
3. van den Elsen, P., Maintz, J., Pol, E., Viergever, M.: Medical image matching - a review with classification. In: IEEE Engineering in Medicine and Biology. (1993) 26–39
4. Bloomenthal, J., Lim, C.: Skeletal methods of shape manipulation. In: International Conference on Shape Modeling and Applications. (1999) 44
5. Klein, P.N., Sebastian, T.B., Kimia, B.B.: Shape matching using edit-distance: an implementation. In: Symposium on Discrete Algorithms. (2001) 781–790
6. Hilaga, M., Schinagawa, Y., Kohmura, T., Kuni, T.L.: Topology matching for fully automatic similarity estimation of 3d shapes. In: Computer Graphics (SIGGRAPH). (2001) 203–212
7. Besl, P.J., McKay, N.D.: A method for registration of 3-d shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence **14**(2) (1992) 239–256
8. Rusinkiewicz, S., Levoy, M.: Efficient variants of the icp algorithm. In: Third International Conference on 3D Digital Imaging and Modeling (3DIM). (2001) 145–152
9. Osada, R., Funkhouser, T., Chazelle, B., Dobkin, D.: Shape distributions. In: ACM Transactions on Graphics, 21(4). (2002) 807–832
10. Johnson, A., Hebert, M.: Recognizing objects by matching oriented points. In: CVPR '97. (1997) 684–689

11. Ruiz-Correa, S., Shapiro, L.G., Meila, M.: A new signature-based method for efficient 3-d object recognition. In: CVPR (1). (2001) 769–776
12. Planitz, B.M., Maeder, A.J., Williams, J.A.: Intrinsic correspondence using statistical signature-based matching for 3d surfaces. In: Australian Pattern Recognition Society (APRS) Workshop on Digital Image Computing (WDIC). (2003)
13. G., M., Belongie, S., Malik, H.: Shape contexts enable efficient retrieval of similar shapes. In: CVPR 1. (2001) 723–730
14. Sun, Y., Paik, J.K., Koschan, A., Page, D.L., Abidi, M.A.: Point fingerprint: A new 3-d object representation scheme. IEEE Trans. Systems, Man, and Cybernetics, Part B **33**(4) (2003) 712–717
15. Yamany, S.M., Farag, A.A.: Surface signatures: An orientation independent free-form surface representation scheme for the purpose of objects registration and matching. IEEE Trans. Pattern Anal. Mach. Intell. **24**(8) (2002) 1105–1120
16. Takahashi, S., Ohta, N., Nakamura, H., Takeshima, Y., Fujishiro, I.: Modeling surperspective projection of landscapes for geographical guide-map generation. Computer Graphics Forum **21**(3) (2002) 259–268
17. Mortara, M., Patanè, G., Spagnuolo, M., Falcidieno, B., Rossignac, J.: Blowing bubbles for multi-scale analysis and decomposition of triangle meshes. Algorithmica **38**(1) (2003) 227–248
18. Maintz, J.B.A., van den Elsen, P.A., Viergever, M.A.: Registration of 3d medical images using simple morphological tools. In: IPMI. (1997) 204–217
19. McIvor, A.M., Penman, D.W., Waltenberg, P.T.: Simple surface segmentation. In: DICTA/IVCNZ97, Massey University, New Zealand (1997) 141–146
20. Vivodtzev, F., Linsen, L., Bonneau, G.P., Hamann, B., Joy, K.I., Olshausen, B.A.: Hierarchical isosurface segmentation based on discrete curvature. In: Proceedings of VisSym'03-Data Visualization 2003, New York, New York, ACM Press (2003) 249–258
21. Mangan, A.P., Whitaker, R.T.: Partitioning 3d surface meshes using watershed segmentation. In: IEEE Transactions on Visualization and Computer Graphics 5(4). (1999) 308–321
22. Page, D.L., Koschan, A.F., Abidi, M.A.: Perception-based 3d triangle mesh segmentation using fast marching watersheds. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '03) **2** (2003) 27–32
23. Gal, R., Cohen-Or, D.: Salient geometric features for partial shape matching and similarity. j-TOG **25**(1) (2006) 130–150
24. Lee, C.H., Varshney, A., Jacobs, D.W.: Mesh saliency. Trans. Graph. **24**(3) (2005) 659–666
25. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Trans. Pattern Anal. Mach. Intell. **26**(9) (2004) 1124–1137
26. Boykov, Y., Jolly, M.P.: Interactive organ segmentation using graph cuts. In: MICCAI '00: Proceedings of the Third International Conference on Medical Image Computing and Computer-Assisted Intervention, London, UK, Springer-Verlag (2000) 276–286

Computation of Normals for Stationary Subdivision Surfaces

Hiroshi Kawaharada and Kokichi Sugihara

Department of Mathematical Informatics,
Graduate School of Information Science and Technology,
University of Tokyo
kawarada@simplex.t.u-tokyo.ac.jp, sugihara@mist.i.u-tokyo.ac.jp

Abstract. This paper proposes a method for computing of normals for stationary subdivision surfaces.

In [1,2], we derived a new necessary and sufficient condition for C^k-continuity of stationary subdivision schemes. First, we showed that tangent plane continuity is equivalent to the convergence of difference vectors. Thus, using "normal subdivision matrix" [3], we derived a necessary and sufficient condition of tangent plane continuity for stationary subdivision at extraordinary points (including degree 6). Moreover, we derived a necessary and sufficient condition for C^1-continuity.

Using the analysis, we show that at general points on stationary subdivision surfaces, the computation of the exact normal is an infinite sum of linear combinations of cross products of difference vectors even if the surfaces are C^1-continuous. So, it is not computable. However, we can compute the exact normal of subdivision surfaces at the limit position of a vertex of original mesh or of j-th subdivided mesh for any finite j even if the surfaces are not regular.

1 Introduction

Subdivision [4,5] is a well-known method for geometric design and for computer graphics, because the subdivision makes smooth surfaces with arbitrary topology. A subdivision scheme is defined by a rule of change of connectivity and subdivision matrices. Many researchers study the conditions of the continuity of subdivision surfaces depending on the subdivision matrices [5,6,7,8,9,10,11,12,13].

Therefore, for a long time it had been an important open problem to derive necessary and sufficient conditions for C^k-continuity for subdivision schemes.

For smoothness of stationary subdivision schemes at extraordinary points, Reif [7] derived a sufficient condition for C^1-continuity. Moreover, Prautzch [8] derived sufficient conditions and necessary conditions for C^k-continuity. However, a necessary and sufficient condition was not obtained.

Zorin [6] derived a necessary and sufficient condition for C^k-continuity with some assumptions. However, the derivation of his condition is not simple, because the condition is described in terms of subdivision matrix S_k and eigen basis functions and a parametric map.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 585–594, 2006.
© Springer-Verlag Berlin Heidelberg 2006

In [1,2], on the other hand, we recently derived another necessary and sufficient condition for C^k-continuity. Our condition is described in terms of a certain matrix derived from the subdivision matrix, instead of the subdivision matrix itself. We use only linear algebra and hence the derivation is simple and easily understandable.

In this paper, we will show the exact normal of subdivision surface at the limit position of a vertex of the original mesh or of the j-th subdivided mesh for any finite j. In our method, the exact normal at such point is obtained as a linear combination of cross products of difference vectors.

Moreover, if the subdivision scheme is C^1-continuous, the exact normal at general point is the infinite sum of linear combinations of cross products of difference vectors.

Of course, there are many method of computation of approximated normal (for example, [14]). Also, there are proposals to compute the normals exactly [15,16], but these methods are limited to specific types of subdivisions such as Loop or Catmull-Clark. Our method, on the other hand, can compute the exact normals for any stationary subdivisions in general.

2 Subdivision Schemes

In this section, we review subdivision schemes in general.

2.1 Subdivision Matrix

A subdivision scheme is defined by subdivision matrices and a rule of connectivity change. The subdivision scheme, when it is applied to 2-manifold irregular meshes, generates smooth surfaces at the limit. In Fig. 1, a face is divided into four new faces. This is a change of connectivity. In this paper, the change of connectivity is fixed to this type, but other types of connectivity change can be discussed similarly.

Next, let us consider how to change the positions of the old vertices, and how to decide the positions of the new vertices. They are specified by matrices called "subdivision matrices". The subdivision matrices are defined at vertices and they depend on degree k of the vertex (the degree is the number of edges connected to the vertex). For example, Fig. 1 denotes a vertex v_0^j which has five edges. Let $v_1^j, v_2^j, \cdots, v_5^j$ be the vertices at the other terminals of the five edges. Then, subdivision matrix S_5 is defined as follows:

$$\begin{pmatrix} v_0^{j+1} \\ v_1^{j+1} \\ \vdots \\ v_5^{j+1} \end{pmatrix} = S_5^j \begin{pmatrix} v_0^j \\ v_1^j \\ \vdots \\ v_5^j \end{pmatrix},$$

where v_0^{j+1} is the new locations of the vertex v_0^j after the $j+1$-st subdivision, while $v_1^{j+1}, \cdots, v_5^{j+1}$ are the newly generated vertices.

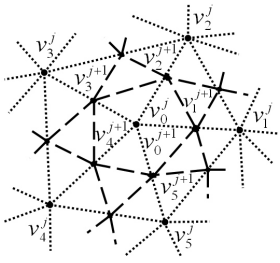


Fig. 1. Subdivision matrix

Here, the subdivision matrix S_5^j is a square matrix. The superscript j means the j-th step of the subdivision. Here, neighbor vertices of a vertex v are called vertices on the 1-disc of v. The subdivision matrix is generally defined not only on vertices in the 1-disc, but also on vertices in the 2-disc, the 3-disc, $\cdots$. Here, we discuss only subdivision matrices that depend on vertices in the 1-disc. However, we can discuss other subdivision matrices, similarly. In this paper, we assume that the subdivision matrix is independent of j. A subdivision scheme of this type is called "stationary".

In this way, the subdivision matrix is written for a vertex. However, since a newly generated vertex is computed by two subdivision matrices at the ends of the edge, the two subdivision matrices must generate the same location of the vertex. So, the subdivision matrices have this kind of restriction.

Here, the degree k of a vertex (i.e., the number of edges connect to a vertex) is at least two. A vertex whose degree is two is a boundary vertex. The degree of a vertex of 2-manifold meshes is at least three. In this paper, we do not discuss boundaries of meshes. So, we assume that the degree is at least three.

2.2 Limit Position of a Vertex

Here we compute the limit position of a vertex v_0^j. This problem was solved already.

Now, the subdivision scheme is written as:

$$p^{j+1} = S_k p^j,$$

where $p^j = (v_0^j, v_1^j, \cdots, v_k^j)^\top$. Here, we assume the subdivision surface is C^0-continuous. Then,

$$p^\infty = (v_0^\infty, v_0^\infty, \cdots)^\top.$$

Thus, let $S_k = V_0^{-1} H V_0$, where H is the Jordan normal form. Now, clearly, S_k has an eigen value $\lambda_1 = 1$ with right eigen vector $(1, \cdots, 1)^\top$ from affine invariance. By C^0-continuity, λ_1 has a single cyclic subspace of size 1 and $|\lambda_i| < \lambda_1,\ i = 2, 3, \cdots$, where $\lambda_i,\ i = 2, 3, \cdots$ are eigen values of S_k (see [1,2]). So,

$$p^\infty = S_k^\infty p^0 = \left(\begin{array}{c|c} 1 & \\ \vdots & * \\ 1 & \end{array}\right) \left(\begin{array}{c|ccc} 1 & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & 0 & \\ 0 & & & \end{array}\right) V_0 p^0 = \left(\begin{array}{c|c} 1 & \\ \vdots & 0 \\ 1 & \end{array}\right) V_0 p^0.$$

Therefore, we can compute v_0^∞. Let e_1 be the first left eigen vector of S_k (e_1 is the first row of V_0). Then, $e_1 \cdot p^0$ is the limit position v_0^∞.

3 C^1-Continuity and Limit Normal

In this section, we present a sufficient and necessary condition of C^1-continuity for stationary subdivision in [1,2] and show a method of computation of the limit normal.

Here, we assume C^0-continuity at v_0^∞.

3.1 Normal Subdivision Matrix

At the $j+1$-st step of the subdivision, new vertices are computed by subdivision matrix S_k as $(v_0^{j+1}, v_1^{j+1}, \cdots, v_k^{j+1})^\top = S_k(v_0^j, v_1^j, \cdots, v_k^j)^\top$. Then, we define a matrix Δ as

$$\Delta = \begin{pmatrix} 1 & 0 & \cdots & & \\ -1 & 1 & 0 & \cdots & \\ -1 & 0 & 1 & 0 & \cdots \\ \vdots & & & \ddots & \end{pmatrix}.$$

Using a matrix $D'_k = \Delta S_k \Delta^{-1}$, we get

$$\begin{pmatrix} v_0^{j+1} \\ v_1^{j+1} - v_0^{j+1} \\ \vdots \\ v_k^{j+1} - v_0^{j+1} \end{pmatrix} = D'_k \begin{pmatrix} v_0^j \\ v_1^j - v_0^j \\ \vdots \\ v_k^j - v_0^j \end{pmatrix}.$$

Note that the sum of each row of S_k is 1 from affine invariance. Therefore, the first element v_0^j does not affect elements $v_1^{j+1} - v_0^{j+1}, v_2^{j+1} - v_0^{j+1}, \cdots, v_k^{j+1} - v_0^{j+1}$. Here, we denote the vector consisting of the elements $v_1^j - v_0^j, v_2^j - v_0^j, \cdots, v_k^j - v_0^j$ as d^j (see Fig. 2) and the associated submatrix of D'_k as D_k:

$$D'_k = \left(\begin{array}{c|c} a & * \\ \hline 0 & \\ \vdots & D_k \\ 0 & \end{array}\right).$$

Then, $d^{j+1} = D_k d^j$. We call this the difference scheme, and a row of d^j difference vector.

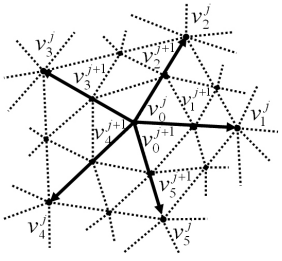


Fig. 2. Difference vectors. A row of d^j is a difference vector $v_i^j - v_0^j$. Difference vectors converge to first derivatives at v_0^∞.

Here, we denote the column vector which is a set of x elements of d^j as d_x^j. Similarly, we denote the column vectors corresponding to y elements, z elements as d_y^j, d_z^j.

Here, using $u_1, u_2 \in \mathbf{R}^k$, we define a matrix ΛD_k as

$$\Lambda D_k(u_1 \wedge u_2) = D_k u_1 \wedge D_k u_2,$$

where $\wedge$ is the wedge product. Then,

$$\Lambda D_k(d_y^j \wedge d_z^j) = D_k d_y^j \wedge D_k d_z^j.$$

Now, we define $N^j = (d_y^j \wedge d_z^j, d_z^j \wedge d_x^j, d_x^j \wedge d_y^j)$. Then,

$$N^{j+1} = \Lambda D_k \; N^j,$$

where a row of N^j is a cross product of between $v_i^j - v_0^j$ and $v_l^j - v_0^j$, that is, normal on the neighborhood of v_0^j. So, we see that the matrix ΛD_k subdivides normals of faces which connects a vertex whose degree is k.

Note that N^j contains normals of unreal faces in the mesh (Real face can be written as $(v_i^j - v_0^j) \times (v_{i+1}^j - v_0^j)$ or $(v_k^j - v_0^j) \times (v_1^j - v_0^j)$. Otherwise, the row of N^j is unreal face.).

Therefore, we call this matrix the "normal subdivision matrix".

3.2 Tangent Plane Continuity

Now, we easily see that difference vectors converge to first derivatives at v_0^∞. So, the limit surface of subdivision is tangent plane continuous at v_0^∞ if and only if all rows of N^∞ (these are normals) point to the same direction (This is direction of normal at v_0^∞. Here, normal n and $-n$ are called same direction).

Here, let $\Lambda D_k = V^{-1} A V$, where A is the Jordan normal form, V is a regular matrix. Let $\Lambda_i,\ i = 1, 2, \cdots$ be eigen values of ΛD_k. Let Λ be the set of Λ_i such that the magnitude of a element of Λ is no less than that of any other eigen value of ΛD_k. Let l_m be maximal size of Jordan cells of $\Lambda_i \in \Lambda$ and q be the number of Jordan cells whose size is l_m.

Then, we get following theorem:

Theorem 1 (Tangent Plane Continuity)
The subdivision surface is tangent plane continuous at v_0^∞ if and only if $q = 1$ and the Λ_i corresponding to the maximal Jordan cell is real positive.

The proof of this theorem is in [1,2].

3.3 C^1-Continuity

In the previous subsection, we presented the condition of tangent plane continuity. Thus, here, we assume tangent plane continuity, and we present the condition of C^1-continuity.

Here, let a be a positive integer such that the first row of the maximal Jordan cell is a-th row of A^n and V_a^{-1} be a-th column of V^{-1}. Let Λ_i be the eigen value of ΛD_k corresponding to the maximal Jordan cell. Then,

$$\begin{aligned} N^\infty &= \lim_{n\to\infty} (\Lambda D_k)^n N^0 = V^{-1} \lim_{n\to\infty} (A)^n V N^0 \\ &= \lim_{n\to\infty} ({}_n\mathrm{C}_{l_m-1} \Lambda_i^{n-l_m+1}) \cdot \left(0 \middle| \cdots \middle| 0 \middle| V_a^{-1} \middle| 0 \middle| \cdots \middle| 0\right) V N^0 \\ &= \lim_{n\to\infty} ({}_n\mathrm{C}_{l_m-1} \Lambda_i^{n-l_m+1}) \begin{pmatrix} V_{1a}^{-1} \cdot vn_a^0 \\ V_{2a}^{-1} \cdot vn_a^0 \\ \vdots \end{pmatrix}, \end{aligned}$$

where V_{ia} is i-th element of V_a^{-1}, vn_a^0 is a-th row of VN^0, and ${}_n\mathrm{C}_k$ denotes the number of different choises of k elements from n elements.

Here, let $\mathcal{R}^1 = \{i | i\text{-th row of } N^\infty \text{ is normal of real face}\}$ (Remember the definition of N^0. N^0 includes unreal faces.). $\forall i \in \mathcal{R}^1$, N_i^∞ point to the same direction including sign if and only if the limit surface is C^1-continuous, where N_i^∞ is i-th row of N^∞.

Note that there is a row of N^∞ which is $(v_1^\infty - v_0^\infty) \times (v_k^\infty - v_0^\infty)$. Here, normal of real face is $(v_k^\infty - v_0^\infty) \times (v_1^\infty - v_0^\infty)$.

So, if the element of V_a^{-1} corresponding to $(v_1^\infty - v_0^\infty) \times (v_2^\infty - v_0^\infty)$ is positive, then the element of V_a^{-1} corresponding to $(v_1^\infty - v_0^\infty) \times (v_k^\infty - v_0^\infty)$ must be negative.

As above, we define "proper sign" of N^∞ as same sign of normal of real face.

Moreover, we must consider the number of sheets of neighborhood of v_0^∞. Let $\mathcal{C}^1$ be an ordered index set corresponding to $(v_1^\infty - v_0^\infty) \times (v_2^\infty - v_0^\infty), (v_1^\infty - v_0^\infty) \times (v_3^\infty - v_0^\infty), \cdots, (v_1^\infty - v_0^\infty) \times (v_k^\infty - v_0^\infty)$. Here, for example, if ordered signs of V_{ia}^{-1}, $\forall i \in \mathcal{C}^1 = (+, +, +, \cdots, +, 0, -, -, \cdots, -)$, then there is one sheet. If this ordered signs have only one shift between + and −, we call this signs "1-cyclical sign". If this signs have two or more shifts, then the subdivision surface is not C^1-continuous at v_0^∞ (See Fig. 3).

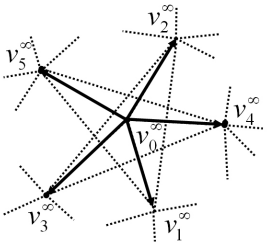


Fig. 3. Two Sheets. The neighborhood of v_0^∞ has two sheets. Then, the surface is not C^1-continuous at v_0^∞.

Then, we get following theorem:

Theorem 2 (C^1-continuity)
Assume that the subdivision surface is tangent plane continuous at v_0^∞. The subdivision surface is C^1-continuous at v_0^∞ if and only if $\forall i \in \mathcal{R}^1$, all V_{ia}^{-1} are proper sign and non-zero, and an ordered signs V_{ia}^{-1}, $\forall i \in \mathcal{C}^1$ is 1-cyclical sign.

The proof of this theorem is in [1,2].

3.4 Limit Normal

If the subdivision surface is C^1-continuous at v_0^∞, we can compute the limit normal N^∞:

$$N^\infty = \lim_{n\to\infty} ({}_n\mathrm{C}_{l_m-1} \Lambda_i^{n-l_m+1}) \begin{pmatrix} V_{1a}^{-1} \cdot vn_a^0 \\ V_{2a}^{-1} \cdot vn_a^0 \\ \vdots \end{pmatrix}.$$

Here, the length of normal corresponds to the determinant of Jacobi matrix at v_0^∞. Now, the subdivision scheme reduces the parameter on the mesh to half. So, by normalizing, we get $d^{j+1} = 2D_k d^j, N^{j+1} = 4\Lambda D_k N^j$. Thus, if $\Lambda_i = \frac{1}{4}$ and $l_m = 1$, then the subdivision surface is regular at v_0^∞.

However, even if the subdivision surface is not regular, $V_{1a}^{-1} \cdot vn_a^0$ denotes the exact normal at v_0^∞.

Therefore, we can compute the exact normal of the subdivision surface at the limit position of a vertex of the original mesh or of the j-th subdivided mesh for any finite j.

At other points on the subdivision surface, we can not compute the exact normal even if the subdivision scheme is C^1-continuous. In fact, the exact normal of such point is infinite sum of linear combinations of cross products of difference vectors.

4 Computation of Limit Normal

In this section, we look C^1-continuity of Loop subdivision scheme at v_0^∞ with degree 6 and compute limit normal N^∞ at the point.

The subdivision matrix S_6 of Loop subdivision and its D_6 are :

$$S_6 = \frac{1}{16}\begin{pmatrix} 10 & 1 & 1 & 1 & 1 & 1 & 1 \\ 6 & 6 & 2 & 0 & 0 & 0 & 2 \\ 6 & 2 & 6 & 2 & 0 & 0 & 0 \\ 6 & 0 & 2 & 6 & 2 & 0 & 0 \\ 6 & 0 & 0 & 2 & 6 & 2 & 0 \\ 6 & 0 & 0 & 0 & 2 & 6 & 2 \\ 6 & 2 & 0 & 0 & 0 & 2 & 6 \end{pmatrix},\ D_6 = \frac{1}{16}\begin{pmatrix} 5 & 1 & -1 & -1 & -1 & 1 \\ 1 & 5 & 1 & -1 & -1 & -1 \\ -1 & 1 & 5 & 1 & -1 & -1 \\ -1 & -1 & 1 & 5 & 1 & -1 \\ -1 & -1 & -1 & 1 & 5 & 1 \\ 1 & -1 & -1 & -1 & 1 & 5 \end{pmatrix}.$$

Then, its normal subdivision matrix ΛD_6 is :

$$\Lambda D_6 = \frac{1}{128}\begin{pmatrix} 12 & 3 & -2 & -2 & -3 & 3 & 2 & 2 & -3 & 1 & 1 & 0 & 0 & 1 & 1 \\ 3 & 12 & 2 & -3 & -2 & 3 & 1 & 0 & -1 & 2 & 3 & -2 & 1 & 0 & 1 \\ -2 & 2 & 12 & 2 & -2 & 0 & 2 & 0 & 0 & -2 & 0 & 0 & 2 & -2 & 0 \\ -2 & -3 & 2 & 12 & 3 & -1 & 0 & 2 & 1 & -1 & -3 & 0 & -2 & -1 & -3 \\ -3 & -2 & -2 & 3 & 12 & -1 & -1 & 0 & 3 & 0 & -1 & -2 & -1 & -2 & -3 \\ 3 & 3 & 0 & -1 & -1 & 12 & 3 & -2 & -2 & 3 & 2 & 2 & 1 & 1 & 0 \\ 2 & 1 & 2 & 0 & -1 & 3 & 12 & 2 & -3 & 3 & 1 & 0 & 2 & 3 & 1 \\ 2 & 0 & 0 & 2 & 0 & -2 & 2 & 12 & 2 & 0 & 2 & 0 & -2 & 0 & 2 \\ -3 & -1 & 0 & 1 & 3 & -2 & -3 & 2 & 12 & -1 & 0 & 2 & -1 & -3 & -2 \\ 1 & 2 & -2 & -1 & 0 & 3 & 3 & 0 & -1 & 12 & 3 & -2 & 3 & 2 & 1 \\ 1 & 3 & 0 & -3 & -1 & 2 & 1 & 2 & 0 & 3 & 12 & 2 & 3 & 1 & 2 \\ 0 & -2 & 0 & 0 & -2 & 2 & 0 & 0 & 2 & -2 & 2 & 12 & 0 & 2 & -2 \\ 0 & 1 & 2 & -2 & -1 & 1 & 2 & -2 & -1 & 3 & 3 & 0 & 12 & 3 & 3 \\ 1 & 0 & -2 & -1 & -2 & 1 & 3 & 0 & -3 & 2 & 1 & 2 & 3 & 12 & 3 \\ 1 & 1 & 0 & -3 & -3 & 0 & 1 & 2 & -2 & 1 & 2 & -2 & 3 & 3 & 12 \end{pmatrix},$$

where $N^0 = (d_1^0 \times d_2^0, d_1^0 \times d_3^0, d_1^0 \times d_4^0, d_1^0 \times d_5^0, d_1^0 \times d_6^0, d_2^0 \times d_3^0, d_2^0 \times d_4^0, d_2^0 \times d_5^0, d_2^0 \times d_6^0, d_3^0 \times d_4^0, d_3^0 \times d_5^0, d_3^0 \times d_6^0, d_4^0 \times d_5^0, d_4^0 \times d_6^0, d_5^0 \times d_6^0)^\top$ $(d_i^0 = v_i^0 - v_0^0)$.

Let $\Lambda D_6 = V_6^{-1} A_6 V_6$, where A_6 is the Jordan normal form. Then,

$$V_6^{-1} = \begin{pmatrix} 1 & 1 & -1 & 0 & -2 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & -1 & 1 & 1 & 0 & 1 & -2 & 0 & 1 & -1 & 0 & 0 & 1 \\ 0 & -2 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ -1 & -1 & -1 & 0 & 2 & 0 & 1 & 0 & -1 & 1 & 0 & -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & -1 & -1 & -1 & 0 & 1 & 2 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & -1 & -1 \\ 1 & -1 & -1 & 0 & -2 & 0 & 1 & 0 & 1 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 \\ -1 & 0 & 0 & -1 & -1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 1 & 1 & -1 \\ 1 & 0 & 0 & -1 & 1 & -1 & 0 & 1 & -2 & 0 & -1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & -1 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 1 & -1 & 1 & 0 & -2 & 0 & -1 & 0 & 1 & 1 & 0 & -1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 1 & 1 & -1 & 0 & -1 & -2 & 0 & 1 & 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & 1 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

$$A_6 = \frac{1}{32}\mathbf{diag}(8, 4, 4, 4, 4, 4, 4, 2, 2, 2, 2, 2, 1, 1, 1)$$

$$V_6 = \frac{1}{12}\begin{pmatrix} 1 & 1 & 0 & -1 & -1 & 1 & 1 & 0 & -1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 2 & 1 & -2 & -2 & -1 & 0 & -2 & 0 & 0 & -1 & 0 & 0 & -2 & -1 & 0 \\ -1 & 0 & 0 & -1 & 0 & 1 & -1 & -4 & -1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 0 & -1 & -2 & 0 & 0 & -2 & -1 & -2 & -2 & 0 & 1 & 2 \\ -1 & 0 & 0 & 1 & 0 & 1 & -1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 \\ 1 & 2 & 2 & -1 & -2 & 0 & -1 & 0 & 0 & -2 & 0 & 0 & -1 & -2 & 0 \\ 1 & 0 & 0 & 1 & 0 & 2 & 1 & -2 & -2 & 0 & -2 & 0 & -1 & 0 & -2 \\ 0 & 1 & 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 & -1 & -4 & 0 & -1 & 1 \\ 0 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & -1 & -1 & 1 & 0 & 0 & -1 & 1 \\ 3 & 0 & 0 & 3 & 0 & 0 & -3 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 3 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & -3 & 0 & 0 & 0 & 3 & 0 \\ 1 & -1 & 2 & -1 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 3 & 0 & -3 & 0 & 0 & 0 & 3 \\ 1 & 0 & 0 & 1 & 0 & -1 & 1 & -2 & 1 & 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 & -1 & 2 & 0 & -1 & 1 \end{pmatrix}.$$

Here, $\Lambda = \{0.25\}$ (Λ is the set of eigen values, whose magnitude is maximal, of ΛD_6). So, the Jordan cell of Λ is unique. Thus, $q = 1$. Moreover, $l_m = 1$ and Λ is real positive. Therefore, by the theorem 1, the Loop subdivision surface is tangent plane continuous at the limit position of a vertex with degree 6.

Now, the first row of this Jordan cell is first row of A_6. So, $a = 1$. Here, $N^0 = (d_1^0 \times d_2^0, d_1^0 \times d_3^0, d_1^0 \times d_4^0, d_1^0 \times d_5^0, d_1^0 \times d_6^0, d_2^0 \times d_3^0, d_2^0 \times d_4^0, d_2^0 \times d_5^0, d_2^0 \times d_6^0, d_3^0 \times d_4^0, d_3^0 \times d_5^0, d_3^0 \times d_6^0, d_4^0 \times d_5^0, d_4^0 \times d_6^0, d_5^0 \times d_6^0)^\top$ $(d_i^0 = v_i^0 - v_0^0)$. So, $\mathcal{R}^1 = \{1, 5, 6, 10, 13, 15\}$.

Thus, $\forall i \in \mathcal{R}^1, V_{ia}^{-1} = \{V_{11}^{-1}, V_{51}^{-1}, V_{61}^{-1}, V_{10,1}^{-1}, V_{13,1}^{-1}, V_{15,1}^{-1}\}$. $V_{11}^{-1} = 1, V_{51}^{-1} = -1, V_{61}^{-1} = 1, V_{10,1}^{-1} = 1, V_{13,1}^{-1} = 1, V_{15,1}^{-1} = 1$. So, $\forall i \in \mathcal{R}^1, V_{ia}^{-1}$ are proper sign and non-zero.

Here, the ordered index set $\mathcal{C}^1 = (1, 2, \cdots, 5)$. V_{ia}^{-1}, $\forall i \in \mathcal{C}^1 = (1, 1, 0, -1, -1)$. So, the ordered signs is $(+, +, 0, -, -)$. Thus, the ordered signs of V_{ia}^{-1}, $\forall i \in \mathcal{C}^1$ is 1-cyclical sign.

Therefore, by the theorem 2, the Loop subdivision surface is C^1-continuous at the limit position of a vertex with degree 6. Moreover, since $\Lambda = \frac{1}{4}$ and $l_m = 1$, the Loop subdivision surface is regular at the point.

Here, we can compute the exact normal N^∞:

$$\begin{aligned} N_1^\infty &= \lim_{n\to\infty} 4^n \cdot \lim_{n\to\infty} ({}_n\mathrm{C}_{l_m-1} \Lambda_i^{n-l_m+1}) V_{1a}^{-1} \cdot vn_a^0 \\ &= \lim_{n\to\infty} 4^n \cdot \lim_{n\to\infty} ({}_n\mathrm{C}_0 (\frac{1}{4})^n) V_{11}^{-1} \cdot vn_1^0 \\ &= \lim_{n\to\infty} 4^n \cdot \lim_{n\to\infty} (\frac{1}{4})^n \cdot 1 \cdot vn_1^0 = vn_1^0, \end{aligned}$$

where N_1^∞ is the first row of N^∞. So, vn_1^0 denotes the exact normal (vn_1^0 is the first row of VN^0).

$$\begin{aligned} vn_1^0 = {} & d_1^0 \times d_2^0 + d_1^0 \times d_3^0 - d_1^0 \times d_5^0 - d_1^0 \times d_6^0 + d_2^0 \times d_3^0 + d_2^0 \times d_4^0 - d_2^0 \times d_6^0 \\ & + d_3^0 \times d_4^0 + d_3^0 \times d_5^0 + d_4^0 \times d_5^0 + d_4^0 \times d_6^0 + d_5^0 \times d_6^0, \end{aligned}$$

where $d_i^0 = v_i^0 - v_0^0$.

Like this, we can easily compute the exact normal of subdivision surfaces at the limit position of a vertex of the original mesh or a the j-th subdivided mesh for any finite j.

Note that if D_k is diagonalizable, then ΛD_k is diagonalizable, because ΛD_k $(u_1^{-1} \wedge u_2^{-1}) = D_k u_1^{-1} \wedge D_k u_2^{-1} = \lambda_1 \lambda_2 (u_1^{-1} \wedge u_2^{-1})$, where λ_1, λ_2 are first and second eigen values of D_k and u_1^{-1}, u_2^{-1} are their right eigen vectors. So, $l_m = 1, \Lambda_1 = \lambda_1 \lambda_2$. In order to be C^1-continuous, a must be 1. Thus, $V_a^{-1} = u_1^{-1} \wedge u_2^{-1}$. So, we can check C^1-continuity easily without the Jordan Decomposition of ΛD_k. Here, the first left eigen vector of ΛD_k is $u_1 \wedge u_2$, where u_1, u_2 are first and second left eigen vectors of D_k. Then, the limit normal is $V_{11}^{-1}(u_1 \wedge u_2) \cdot N^0$.

5 Conclusions

In this paper, we presented a general method of computation of the exact normal for stationary subdivision surfaces. In [1,2], we derived a necessary and sufficient condition for C^k-continuity of subdivision surfaces. Our condition for C^1-continuity was described in terms of "normal subdivision matrix" instead of subdivision matrix. Normal subdivision matrix subdivides cross products of difference vectors. That is, normals have a subdivision scheme.

Using this idea, we could see that at the limit position of a vertex of the original mesh or a the j-th subdivided mesh for any finite j, we could easily

compute the exact normal as linear combination of cross products of difference vectors. This method is valid for any stationary subdivision schemes.

Moreover, we applied our method to the Loop subdivision scheme for a degree-6 vertex. Then, we could easily understand that the Loop subdivision surface is C^1-continuous at the limit position of a degree-6 vertex and compute the exact normal.

Acknowledgments

This work is supported by the 21st Century COE Program on Information Science and Technology Strategic Core at the university of Tokyo, and the Grant-in-Aid for Scientific Research of the Japanese Society for the Promotion of Science.

References

1. Kawaharada, H., Sugihara, K.: c^k-continuity of stationary subdivision schemes. METR 2006-01, The University of Tokyo (2006)
2. Kawaharada, H., Sugihara, K.: c^k-continuity for stationary subdivisions at exraordinary points. Computer Aided Geometric Design **XX** (2006) submitted
3. Kawaharada, H., Sugihara, K.: Dual subdivision a new class of subdivision schemes using projective duality. METR 2005-01, The University of Tokyo (2005)
4. Loop, C.T.: Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, Department of Mathematics (1987)
5. Warren, J., Weimer, H.: Subdivision Methods for Geometric Design: A Constructive Approach. Morgan Kaufmann Publishers (1995)
6. Zorin, D.: Smoothness of stationary subdivision on irregular meshes. Constructive Approximation **16**(3) (2000) 359–397
7. Reif, U.: A unified approach to subdivision algorithms near extraordinary points. Computer Aided Geometric Design **12** (1995) 153–174
8. Prautzsch, H.: Analysis of c^k-subdivision surfaces at extraordinary points. Preprint. Presented at Oberwolfach (1995)
9. Cavaretta, A.S., Dahmen, W., Micchell, C.A.: Stationary subdivision. Memoirs Amer. Math. Soc. **93(453)** (1991)
10. Goodman, T.N.T., Micchell, C.A., D., W.J.: Spectral radius formulas for subdivision operators. In: L. L. Schumaker and G. Webb, editors, Recent Advances in Wavelet Analysis, Academic Press (1994) 335–360
11. Zorin, D.: Subdivision and Multiresolution Surface Representations. PhD thesis, University of California Institute of Technology (1997)
12. Reif, U.: A degree estimate for polynomial subdivision surfaces of higher regularity. Proc. Amer. Math. Soc. **124:2167–2174** (1996)
13. Doo, D., Sabin, M.A.: Behaviour of recursive subdivision surfaces near extraordinary points. Computer Aided Geometric Design **10** (1978) 356–360
14. Meyer, M., Desbrum, M., Schroder, P., Barr, A.: Discrete differential geometry operators for triangulated 2-manifolds. In: Visualization and Mathematics III. (2003) 35–57
15. Stam, J.: Evaluation of loop subdivision surfaces. In: SIGGRAPH 98 Conference Proceedings on CDROM, ACM (1998)
16. Stam, J.: Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In: SIGGRAPH 98 Conference Proceedings, ACM (1998) 395–404

Voxelization of Free-Form Solids Represented by Catmull-Clark Subdivision Surfaces

Shuhua Lai and Fuhua (Frank) Cheng

Graphics & Geometric Modeling Lab, Department of Computer Science,
University of Kentucky, Lexington, Kentucky 40506-0046
{slai2, cheng}@cs.uky.edu
www.cs.uky.edu/∼cheng

Abstract. A voxelization technique and its applications for objects with arbitrary topology are presented. It converts a free-form object from its continuous geometric representation into a set of voxels that best approximates the geometry of the object. Unlike traditional 3D scan-conversion based methods, our voxelization method is performed by recursively subdividing the 2D parameter space and sampling 3D points from selected 2D parameter space points. Moreover, our voxelization of 3D closed objects is guaranteed to be leak-free when a 3D flooding operation is performed. This is ensured by proving that our voxelization results satisfy the properties of separability, accuracy and minimality.

1 Introduction

Volume graphics [5] represents a set of techniques aimed at modeling, manipulating and rendering of geometric objects, which have proven to be, in many aspects, superior to traditional computer graphics approaches. The main drawbacks of volume graphics techniques are their high memory and processing time demands. However, with the progress in both computers and specialized volume rendering hardware, these drawbacks are gradually losing their significance.

Subdivision surfaces have become popular recently in graphical modeling, visualization and animation because of their capability in modeling complex shape of arbitrary topology [1], their relatively high visual quality, and their stability and efficiency in numerical computation. Subdivision surfaces can model/ represent complex shape of arbitrary topology because there is no limit on the shape and topology of the control mesh of a subdivision surface. In this paper we propose a voxelization method for free-form solids represented by Catmull-Clark subdivision surfaces. Instead of direct sampling of 3D points, the new method is based on recursive sampling of 2D parameter space points of a surface patch. Hence the new method is more efficient and less sensitive to numerical error.

A 3D *discrete space* is a set of integral grid points in 3D Euclidean space defined by their Cartesian coordinates (x, y, z), with $x, y, z \in Z$. A voxel is a unit cube centered at the integral grid point. Usually a voxel is assigned a value of 0 or 1. The voxels assigned an '1', called the 'black' voxels, represent opaque objects. Those assigned a '0', called the 'white' voxels, represent the transparent

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 595–601, 2006.
© Springer-Verlag Berlin Heidelberg 2006

background. Two voxels are said to be *26-adjacent* [4] if they share a vertex, an edge, or a face. Every given voxel has 26 such adjacent voxels: eight share a vertex (corner) with the given voxel, twelve share an edge, and six share a face. Accordingly, face-sharing voxels are said to be *6-adjacent* [4], and edge-sharing and face-sharing voxels are said to be *18-adjacent* [4].

Given a control mesh, a *Catmull-Clark subdivision surface* (CCSS) is generated by iteratively refining (subdividing) the control mesh to form new and finer control meshes [1]. The number of faces in the uniformly refined meshes increases exponentially with respect to subdivision depth. Hence it is not practical to sample 3D points directly on subdivided surfaces. Fortunately, parametrization techniques have become available recently [2,3]. Therefore efficient and accurate sampling for voxelization is not a problem any more. Recent parametrization techniques show that every 3D point (its position, normal and partial derivatives) on the limit surface can be explicitly and accurately calculated [2,3].

2 Related Voxelization Techniques

Voxelization techniques can be classified into two major categories. The first category consists of methods that extend the standard 2D scan-line algorithm and employ numerical considerations to guarantee that no gaps appear in the resulting discretization. Most work on voxelization focused on voxelizing 3D polygon meshes [6,7,8,9] by using 3D scan-conversion algorithm. Although this type of methods can be extended to voxelize parametric curves, surfaces and volumes, it is difficult to deal with freefrom surfaces of arbitrary topology.

The other widely used approach for voxelizing free-form solids is to use spatial enumeration algorithms which employ point or cell classification methods in either an exhaustive fashion or by recursive subdivision [10]. However, 3D space subdivision techniques for models decomposed into cubic subspaces are computationally expensive and thus inappropriate for medium or high resolution grids. The voxelization technique that we will be presenting uses recursive subdivision. The difference is the new method performs recursive subdivision on 2D parameter space, not on the 3D object. Hence expensive distance computation between 3D points is avoided.

3 Voxelization of Solids Represented by CCSSs

Given a free-form object represented by a CCSS and a cubic frame buffer of resolution $M_1 \times M_2 \times M_3$, the goal is to convert the CCSS represented free-form object (i.e. continuous geometric representation) into a set of voxels that best approximates the geometry of the object.

With parametrization techniques for subdivision surfaces becoming available, it is possible now to model and represent any continuous but topologically complex object with an analytical representation [2,3]. Consequently, any point in the surface can be explicitly calculated. On the other hand, for any given parameter space point (u, v), a surface point $S(u, v)$ corresponding to this parameter

space point can be exactly computed as well. Therefore, voxelization does not have to be performed in 3D object space, as the previous recursive voxelization methods did, one can do voxelization in 2D space by performing recursive subdivision and testing on the 2D parameter space.

We first consider the voxelization process of a subpatch, which is a small portion of a patch. Given a subpatch of $\mathbf{S}(u, v)$ defined on $[u_1, u_2] \times [v_1, v_2]$, we voxelize it by assuming this given subpatch is small enough (hence, flat enough) so that all the voxels generated from it are the same as the voxels generated using its four corners:

$$\mathbf{V}_1 = \mathbf{S}(u_1, v_1), \mathbf{V}_2 = \mathbf{S}(u_2, v_1), \mathbf{V}_3 = \mathbf{S}(u_2, v_2), \mathbf{V}_4 = \mathbf{S}(u_1, v_2).$$

Usually this assumption does not hold. Hence a test must be performed before the patch or subpatch is voxelized. It is easy to see that if the voxels generated using its four corners are not N-adjacent ($N \in \{6, 18, 26\}$) to each other, then there exist holes between them. In this case, the patch or subpatch is still not small enough. To make it smaller, we perform a *midpoint subdivision* on the corresponding parameter space by setting $u_{12} = (u_1+u_2)/2$ and $v_{12} = (v_1+v_2)/2$ to get four smaller subpatches:

$$\begin{array}{ll} \mathbf{S}([u_1, u_{12}] \times [v_1, v_{12}]), & \mathbf{S}([u_{12}, u_2] \times [v_1, v_{12}]), \\ \mathbf{S}([u_{12}, u_2] \times [v_{12}, v_2]), & \mathbf{S}([u_1, u_{12}] \times [v_{12}, v_2]), \end{array}$$

and repeat the testing process on each of the subpatches. The process is recursively repeated until all the subpatches are small enough and can be voxelized using only their four corners. For simplicity, we first normalize the input mesh to be of dimension $[0, M_1 - 1] \times [0, M_2 - 1] \times [0, M_3 - 1]$. Then for any 2D parameter space point (u, v) generated from the recursive testing process, direct and exact evaluation is performed to get its 3D surface position and normal vector at $S(u, v)$. To get the voxelized coordinates (i, j, k) from $S(u, v)$, simply set

$$i = \lfloor \mathbf{S}(u, v).x + 0.5 \rfloor, j = \lfloor \mathbf{S}(u, v).y + 0.5 \rfloor, k = \lfloor \mathbf{S}(u, v).z + 0.5 \rfloor. \quad (1)$$

Once every single point marked in the recursive testing process is voxelized, the process for voxelizing the given patch is finished. The proof of the correctness of our voxelization results will be discussed in the next section.

Since the above process guarantees that shared boundary or vertex of patches or subpatches will be voxelized to the same voxel, we can perform voxelization of free-form objects represented by a CCSS on patch basis.

The above voxelization method, based on recursive subdivision of the parameter space, is summarized into the following algorithms: *Voxelization* and *VoxelizeSubPatch*. The parameters of these algorithms are defined below. S: control mesh of a CCSS which represents the given object; N: an integer that specifies the N-adjacent relationship between adjacent voxels; M_1, M_2, and M_3: resolution of the Cubic Frame Buffer; k: an integer that specifies the number of subpatches ($k \times k$) that should be generated before fed to the recursive voxelization process.

Voxelization(Mesh S, int N, int M_1, int M_2, int M_3, int k)
1. normalize S so that S is bounded by $[0, M_1 - 1] \times [0, M_2 - 1] \times [0, M_3 - 1]$
2. for each patch *pid* in S
3. for $u = \frac{1}{k} : 1$, step size $\frac{1}{k}$
4. for $v = \frac{1}{k} : 1$, step size $\frac{1}{k}$
5. VoxelizeSubPatch(N, *pid*, $u - \frac{1}{k}$, u, $v - \frac{1}{k}$, v);

VoxelizeSubPatch(int N, int *pid*, float u_1, float u_2, float v_1, float v_2)
1. $(i_1, j_1, k_1) = \text{Voxelize}(S(pid, u_1, v_1))$; $(i_2, j_2, k_2) = \text{Voxelize}(S(pid, u_2, v_1))$;
2. $(i_3, j_3, k_3) = \text{Voxelize}(S(pid, u_2, v_2))$; $(i_4, j_4, k_4) = \text{Voxelize}(S(pid, u_1, v_2))$;
3. if(the size of this subpatch is smaller than a voxel) return;
4. $\Delta_i = \max\{|i_a - i_b|\}$, with a and $b \in \{1, 2, 3, 4\}$;
5. $\Delta_j = \max\{|j_a - j_b|\}$, with a and $b \in \{1, 2, 3, 4\}$;
6. $\Delta_k = \max\{|k_a - k_b|\}$, with a and $b \in \{1, 2, 3, 4\}$;
7. if($N = 6$ & $\Delta_i + \Delta_j + \Delta_k \leq 1$) return;
8. if($N = 18$ & $\Delta_i \leq 1$ & $\Delta_j \leq 1$ & $\Delta_k \leq 1$ & $\Delta_i + \Delta_j + \Delta_k \leq 2$) return;
9. if($N = 26$ & $\Delta_i \leq 1$ & $\Delta_j \leq 1$ & $\Delta_k \leq 1$) return;
10. $u_{12} = (u_1 + u_2)/2$; $v_{12} = (v_1 + v_2)/2$;
11. VoxelizeSubPatch($N, pid, u_1, u_{12}, v_1, v_{12}$);
12. VoxelizeSubPatch($N, pid, u_{12}, u_2, v_1, v_{12}$);
13. VoxelizeSubPatch($N, pid, u_{12}, u_2, v_{12}, v_2$);
14. VoxelizeSubPatch($N, pid, u_1, u_{12}, v_{12}, v_2$);

In algorithm 'VoxelizeSubPatch', corresponding surface points for the four corners are directly evaluated using parametrization techniques in [2,3], where *pid* tells us which patch we are currently working on. The routine 'Voxelize' voxelizes points by using eq. (1). Lines 7, 8 and 9 are used to test if voxelizing the four corners of a subpatch is enough to generate a 6-, 18- and 26-adjacent voxelization, respectively.

4 Separability, Accuracy and Minimality

Let $\mathbf{S}$ be a C^1 continuous surface in R^3. We denote by $\bar{\mathbf{S}}$ the discrete representation of $\mathbf{S}$. $\bar{\mathbf{S}}$ is a set of black voxels generated by some digitalization method. There are three major requirements that $\bar{\mathbf{S}}$ should meet in the voxelization process. First, *separability* [4,9], which requires preservation of the analogy between continuous and discrete space and to guarantee that $\bar{\mathbf{S}}$ is not penetrable since $\mathbf{S}$ is C^1 continuous. Second, *accuracy*. This requirement ensures that $\bar{\mathbf{S}}$ is the most accurate discrete representation of $\mathbf{S}$ according to some appropriate error metric. Third, *minimality* [4,9], which requires the voxelization does not contain voxels that, if removed, make no difference in terms of separability and accuracy. The mathematical definitions for these requirements can be found in [4,9].

First we can see that the voxelization results generated using our recursive subdivision method satisfy the minimality property. The reason is that voxels

are sampled directly from the object surface. The termination condition of our recursive sampling process (i.e., Line 8, 9, 10 in algorithm 'VoxelizeSubPatch') and the coordinates transformation in eq. (1) guarantee that every point in the surface has one and only one image in the resulting voxelization. In other words,

$$\forall\ \mathbf{P} \in \mathbf{S},\ \exists\ \mathbf{Q} \in \bar{\mathbf{S}},\ \text{ such that }\ \mathbf{P} \in \mathbf{Q}. \tag{2}$$

Note that here $\mathbf{P}$ is a 3D point and $\mathbf{Q}$ is a voxel, which is a unit cube. On the other hand, because all voxels are mapped directly from the object surface using eq. (1), we have

$$\forall\ \mathbf{Q} \in \bar{\mathbf{S}},\ \exists\ \mathbf{P} \in \mathbf{S},\ \text{ such that }\ \mathbf{P} \in \mathbf{Q}. \tag{3}$$

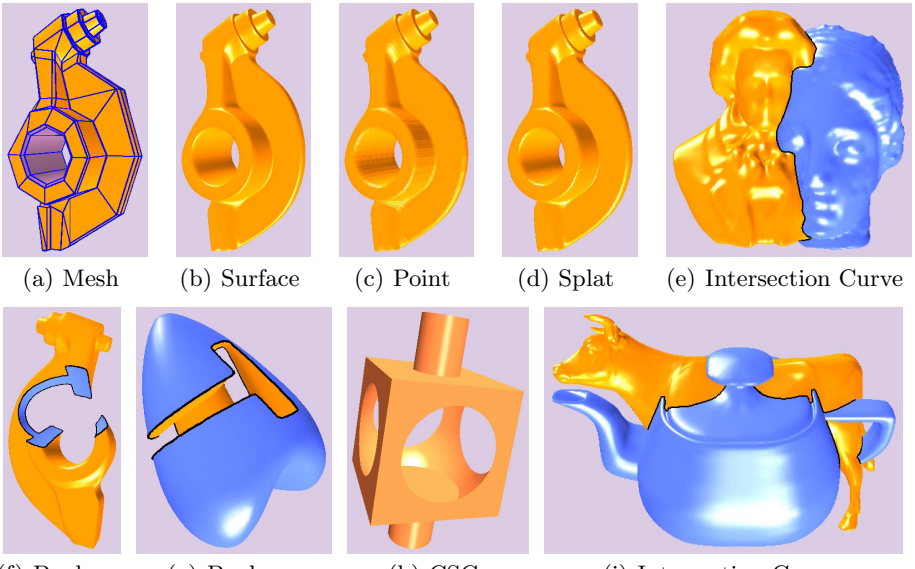
Hence no voxel can be removed from the resulting voxelization, i.e., the property of minimality is satisfied. In addition, from eq. (2) and eq. (3) we can also conclude that the resulting voxelization is the most accurate one with respect to the given resolution. Hence the property of accuracy is satisfied as well.

To prove that our voxelization results satisfy the separability property, we only need to show that there is no holes in the resulting voxelization. For simplicity, here we only consider 6-separability, i.e., there does not exist a ray from a voxel inside the free-form solid object to the outside of the free-form solid object in x, y or z direction that can penetrate our resulting voxelization without intersecting any of the black voxels. We prove the separability property by contradiction. As we know violating separability means there exists at least a hole (voxel) $\mathbf{Q}$ in the resulting voxelization that is not included int $\bar{\mathbf{S}}$ but is intersected by $\mathbf{S}$ and, there must also exist two 6-adjacent neighbors of $\mathbf{Q}$ that are not included in $\bar{\mathbf{S}}$ either and are on opposite sides of $\mathbf{S}$. Because $\mathbf{S}$ intersects with $\mathbf{Q}$, there exist at least one point $\mathbf{P}$ on the surface that intersects with $\mathbf{Q}$. But the image of $\mathbf{P}$ after voxelization is not $\mathbf{Q}$ because $\mathbf{Q}$ is a hole. However, the image of $\mathbf{P}$ after voxelization must exist because of the termination condition of our recursive sampling process (i.e., Line 8, 9, 10 in algorithm 'VoxelizeSubPatch'). Moreover, according to our voxelization method, $\mathbf{P}$ can only be voxelized into voxel $\mathbf{Q}$ because of eq. (1). Hence $\mathbf{Q}$ cannot be a hole, contradicting our assumption. Therefore, we conclude that $\bar{\mathbf{S}}$ is 6-separating.

5 Applications

5.1 Visualization of Complex Scenes

Ray tracing is a commonly used method in the field of visualization of volume graphics [6]. However, ray tracing is very slow due to its large computational demands. Recently, surface splatting technique for point based rendering has become popular [11]. Surface splatting requires the position and normal of every point to be known, but not their connectivity. With explicit position and exact normal information for each voxel in our voxelization results being available, now it is quite easy for us to render discrete voxels using surface splatting



(a) Mesh (b) Surface (c) Point (d) Splat (e) Intersection Curve

(f) Boolean (g) Boolean (h) CSG (i) Intersection Curve

Fig. 1. Applications of Voxelization

techniques. The rendering is fast and high quality results can be obtained. For example, Fig. 1(a) is the given mesh, Fig. 1(b) is the corresponding limit surface. After the voxelization process, Fig. 1(c) is generated only using basic point based rendering techniques with explicitly known normals to each voxel. While Fig. 1(d) is rendered using splatting based techniques. The size of cubic frame buffer used for Fig. 1(c) is $512 \times 512 \times 512$. The voxelization resolution used for Fig. 1(d) is $256 \times 256 \times 256$. Although the resolution is much lower, we can tell from Fig. 1, that the one using splatting techniques is smoother and closer to the corresponding object surface given in Fig. 1(b).

5.2 Integral Properties Measurement

Another application of voxelization is that it can be used to measure integral properties of solid objects such as mass, volume and surface area. Without discretization, these integral properties are very difficult to measure, especially for free-form solids with arbitrary topology.

Volume can be measured simply by counting all the voxels inside or on the surface boundary because each voxel is a unit cube. With efficient flooding algorithm, voxels inside or on the boundary can be precisely counted. But the resulting measurement may not be accurate because boundary voxels do not occupy all the corresponding unit cubes. Hence for higher accuracy, higher voxelization resolution is needed. Once the volume is known, it is easy to measure the mass simply by multiplying the volume with density. Surface area can be measured similarly as well.

5.3 Performing Boolean and CSG Operations

The most important application of voxelization is to perform Boolean and CSG operations on free-form objects. In solid modeling, an object is formed by performing Boolean operations on simpler objects or primitives. A CSG tree is used in recording the construction history of the object and is also used in the ray-casting process of the object. Surface-surface intersection (including the in-on-out test) and ray-surface intersection are the core operations in performing the Boolean and CSG operations. With voxelization, all of these problems simply become easier set operations. Examples of performing Boolean operations on two objects are presented in Fig. 1(f) and Fig. 1(g), respectively. Fig. 1(f) is the difference of a rocker arm shown in Fig. 1(b) and a heart shown in Fig. 1(g), while Fig. 1(g) is the difference of a heart and a rocker arm shown in Fig. 1(b). A mechanical part is also generated in Fig. 1(h) using CSG operations. Intersection curves can be similarly generated by searching for common voxels of objects. The black curves shown in Fig. 1(f), Fig. 1(g), Fig. 1(i) and Fig. 1(e) are the intersection curves generated from two different objects, respectively.

Acknowledgement. Research work of the authors is supported by NSF under grants DMS-0310645 and DMI-0422126. Data sets for Figs. 1(e) and the cow model are downloaded from the web site: http://research.microsoft.com/∼hoppe.

References

1. Catmull E, Clark J. Recursively generated B-spline surfaces on arbitrary topological meshes, *Computer-Aided Design*, 1978, 10(6):350-355.
2. Stam J, Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values, *Proceedings of SIGGRAPH* 1998:395-404.
3. Shuhua Lai, Fuhua (Frank) Cheng, Parametrization of General CCSSs and its Application, *Computer Aided Design & Applications*, 3, 1-4, 2006.
4. Cohen Or, D., Kaufman, A., Fundamentals of Surface Voxelization, *Graphical Models and Image Processing*, 57, 6 (November 1995), 453-461.
5. A. Kaufman, D. Cohen. Volume Graphics. *IEEE Computer*, Vol. 26, No. 7, July 1993, pp. 51-64.
6. D. Haumont and N. Warzee. Complete Polygonal Scene Voxelization, *Journal of Graphics Tools*, Volume 7, Number 3, pp. 27-41, 2002.
7. M.W. Jones. The production of volume data from triangular meshes using voxelisation, *Computer Graphics Forum*, vol. 15, no 5, pp. 311-318, 1996.
8. S. Thon, G. Gesquiere, R. Raffin, A low Cost Antialiased Space Filled Voxelization Of Polygonal Objects, *GraphiCon 2004*, pp. 71-78, Moscou, Septembre 2004.
9. Jian Huang, Roni Yagel, V. Fillipov and Yair Kurzion, An Accurate Method to Voxelize Polygonal Meshes, *IEEE Volume Visualization'98*, October, 1998.
10. Nilo Stolte, Graphics using Implicit Surfaces with Interval Arithmetic based Recursive Voxelization, *Computer Graphics and Imaging*, pp. 200-205, 2003.
11. MatthiasZwicker,HanspeterPfister,JeroenvanBaar,MarkusGross,SurfaceSplatting, *SIGGRAPH 2001*, pp. 371-378.

Interactive Face-Replacements for Modeling Detailed Shapes

Eric Landreneau[1], Ergun Akleman[2], and John Keyser[1]

[1] Texas A&M University, Computer Science Department,
College Station Texas 77843-3112, USA
elan@tamu.edu, keyser@cs.tamu.edu
[2] Texas A&M University, Visualization Sciences Program,
Department of Architecture,
C418 Langford Center, College Station, Texas 77843-3137 USA
ergun@viz.tamu.edu
www-viz.tamu.edu/faculty/ergun

Abstract. In this paper, we present a method that allows novice users to interactively create partially self-similar manifold surfaces without relying on shape grammars or fractal methods. Moreover, the surfaces created using our method are connected. The modelers that are based on traditional fractal methods or shape grammars usually create disconnected surfaces and restrict the creative freedom of users. In most cases, the shapes are defined by hard-coded schemes that provide only a few parameters that can be adjusted by the users. We present a new approach for modeling such shapes. With this approach, novice users can interactively create a variety of unusual and interesting partially self-similar manifold surfaces.

1 Motivation

There exists a strong interest in contemporary sculpture and architecture to extend the limits of conceptual design by using rule-based generative systems. Several architectural design studios experiment with rule-based approaches. Designers in these studios use a wide variety of rule-based techniques such as L-systems, or cellular automata.

For designers, it is important to easily develop new generative procedures to have a variety of alternatives. However, for them it is hard to identify the rules. Shape construction with rules-based methods is generally a rigid and unpredictable process. Users input formulae with little or no interactive control over the resulting shape. Manipulating the end result is often costly and difficult.

Our goal in this paper is to blend rule based techniques such as fractals or L-systems with increased interactivity for designers. Although our motivation comes from fractals and L-systems, we want to give designers interactive control of modifiers to achieve conceptual shapes. We also want the resulting shapes to be physically constructible using 3D printers.

In this paper we present a simple approach that is related to geometric texture modeling [1,2]. Our approach allows interactive and 3D extensions to Fractal and

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 602–608, 2006.
© Springer-Verlag Berlin Heidelberg 2006

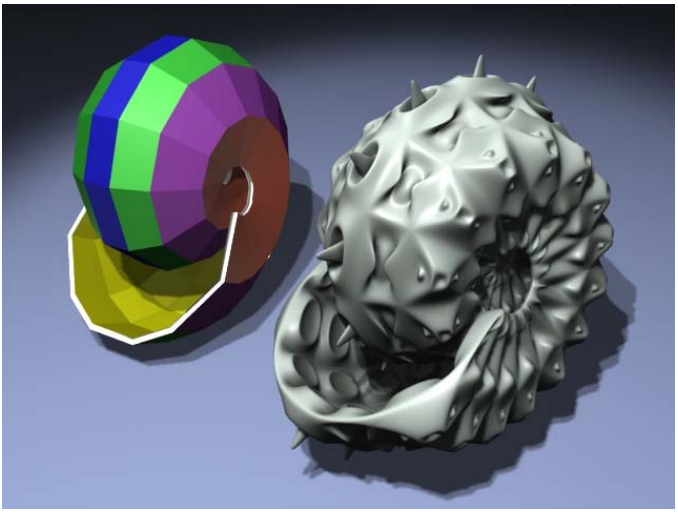


Fig. 1. Using our face coloring and group extrusion methods, users can add finer details with greater control

L-system methods. Using this approach, we have developed a system that enables designers to control each step of the shape generation process with a high level of interactivity.

With the new approach, novice users can easily create a large set of connected "self-similar" manifold surfaces. Disconnected surfaces are acceptable for "virtual" computer graphics applications in which the objects are used for only display purposes. However, in Architecture we usually want to physically construct the resulting shapes. To be able to construct the shapes, the shapes need to be connected and manifold surfaces. Figure 1 shows one example of how users can add finer details with our method.

Disconnected manifold surfaces (if individual surfaces are manifold) can be printed but it is not possible to guarantee that the resulting physical object will stay together. If individual surfaces are not manifold they will not even be suitable for 3D printing. For instance, two methods based on Iterated Function Systems (IFS) [3] create a set of disconnected points or shapes, which can never be printed. In contrast, our method allows us to construct connected manifold surfaces which can be realized a 3D printer.

Our approach is based on face replacements, which are a generalization of the line replacements of 2D fractal geometry. Face replacements are created by using local mesh operators. These operators can be applied to one face of the mesh without affecting the rest. They replace the face with multiple faces. A local operator can be defined by a set of insert edge and create vertex operations [4]. The local mesh operations such as extrusions guarantee that the resulting shape continues to be connected and manifold.

Landreneau et al. recently introduced Platonic extrusions as local mesh operators [5]. These extrusions, except tetrahedral extrusion, are generalized pipes in which bottom and top polygons have the same number of sides [5]. For this paper, we have extended Platonic extrusions to certain Archimedean extrusions. Having a large variety of extrusions provides novice users a simple way to make face replacements. In addition to using these general extrusions, we introduce four new concepts for interactive modeling of connected and self-similar manifold surfaces: (1) *Face Grouping* using colors; (see Figure 1)(2) *Group extrusions*; (3) *Automatic Face Regrouping* and (4) *Remeshing Schemes*. Our method based on these concepts is guaranteed to create connected manifold surfaces.

This modeling approach moves towards a more hands-on approach to grammar based surface modeling. A user can assert much more control over the surface beyond the traditional *plug-in-a-formula-and-wait* method of generating grammar based models. Our approach will particularly be useful in Architectural concept modeling, in which users can quickly determine the effects of various approaches by simply recoloring the faces.

With this approach, users can rapidly learn how to create a wide variety of polygonal meshes that resemble grammar based shapes. The approach is not only limited to fractal looking shapes, however, it can also be used for creating a wide variety of shapes, such as the simple cityscape shown in Figure 2.

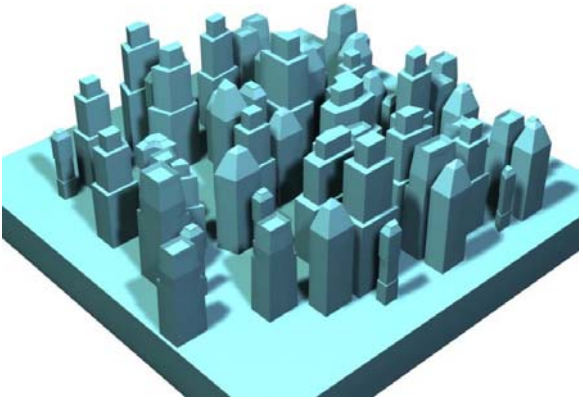


Fig. 2. A simple cityscape that is created by using group extrusions: Face grouping and group extrusions allow us to create similar looking but different buildings

2 Methodology

Our goal in this paper is to achieve the grammar based power of L-systems for constructing connected and manifold surfaces by combining face replacements with face grouping. In this section, we discuss four concepts introduced together in this paper: (1) *Face Grouping* using colors; (2) *Group extrusions*; (3) *Automatic Face Regrouping* and (4) *Remeshing Schemes*. Our method based on these concepts is guaranteed to create connected manifold surfaces.

2.1 Face Grouping Using Colors

Face grouping using colors allows users to easily group the faces in any modeling software. In face grouping, users classify faces by assigning a color to each. Faces are classified by colors, with identically colored faces belonging to a common group. Note that this stage is completely under the user's control. The faces do not have to be geometrically or topologically similar, so there are no restrictions for assigning a color to a face.

2.2 Group Extrusions

Group extrusions simplify multiple extrusions. The users can apply the same extrusion operation to all identically colored faces by selecting only one face (see Figure 3).

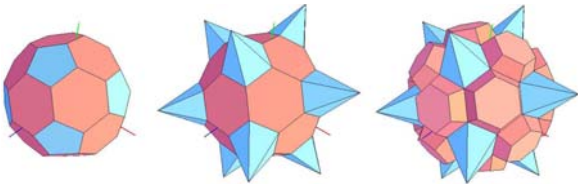


Fig. 3. Face coloring and group extrusions

Extrusions (except tetrahedral) produce a "top" face similar to the parent face, which is connected to the parent edges by "side" faces. The top face can inherent the group of the parent face. However, after a few iterations of remeshing or grouped extrusions, the number of side faces increases exponentially. We have provided automatic face regrouping to simplify the users' job regroup newly created side faces (see Figure 4).

2.3 Automatic Face Regrouping with Modulus Coloring

To regroup side faces, we introduce the modulus coloring concept. The side faces are regrouped according to a modulus scheme. Starting from a randomly chosen side face, new groups are generated using a user supplied modulus. With a modulus of 1, every side face would share the same group. A modulus of 2 would generate two alternating groups. A modulus of 3 will make every third side face the same group, and so on. Using a modulus equal to the number of side faces of the extrusion, equal to the number of edges in the parent face, will assign a unique group identity for each side face.

The modulus ensures that side faces will exhibit radial symmetry, due to side faces sharing colors. The modulus operation is not unique since the regrouping can be different based on the choices of initial side faces. Because of this, regrouping can introduce slight irregularities all allow to break overall symmetry of the object as seen in Figure 4.

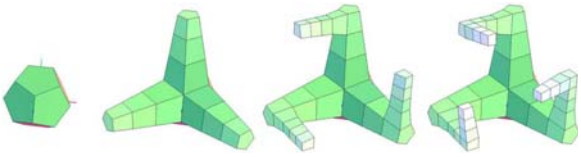


Fig. 4. Automatic coloring newly created faces and group extrusions based on automatic coloring

2.4 Remeshing Schemes with Group Extrusions

Based on the recent research on subdivision surfaces, there now exists a wide selection of remeshing schemes that can be used in interactive applications. It is possible to view these remeshing operations as face replacements that are applied in parallel. By combining these schemes with group extrusions we provide additional flexibility. In fact, Loop style remeshing is particularly common in fractal algorithms [6]. Using Loop style remeshing, it is possible to create generalizations of Koch islands and Sierpinsky tetrahedra [7]. Loop style remeshing with random vertex displacements is widely used for terrain generation. The most apparent reason behind the popularity of Loop style remeshing among Fractal algorithms is that Loop preserves initial faces in every iteration. This property is particularly useful for face replacements since some of the newly created faces inherit the properties of initial faces. However, Loop is not the only one that can provide this property. All dual conversion schemes such as Corner Cutting, Simplest or Honeycomb,and all preservation schemes (Loop belongs to this group) preserve initial faces in every iteration [8]. They all can easily be used for interactive face replacement applications.

3 Implementation and Results

The concepts that is discussed in methodology section are implemented and included in our existing 2-manifold mesh modeling system [4]. Our system is

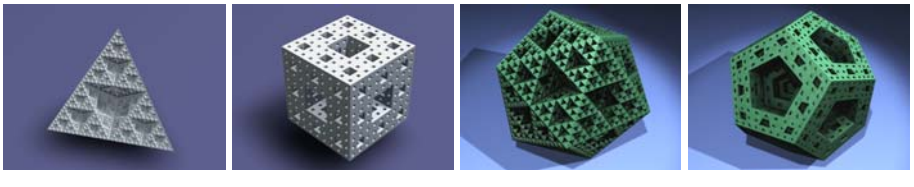


Fig. 5. Four Sierpinsky tetrahedra and Menger Sponge look-a-likes look-a-likes that are created using our method. Since this method does not change the topology, we cannot have holes. On the other hand, the two shapes on the right are generalized versions that cannot be created by fractal methods. For generalized and truly high genus Sierpinsky polyhedra see [7].

Fig. 6. Interactively created fractal looking shapes

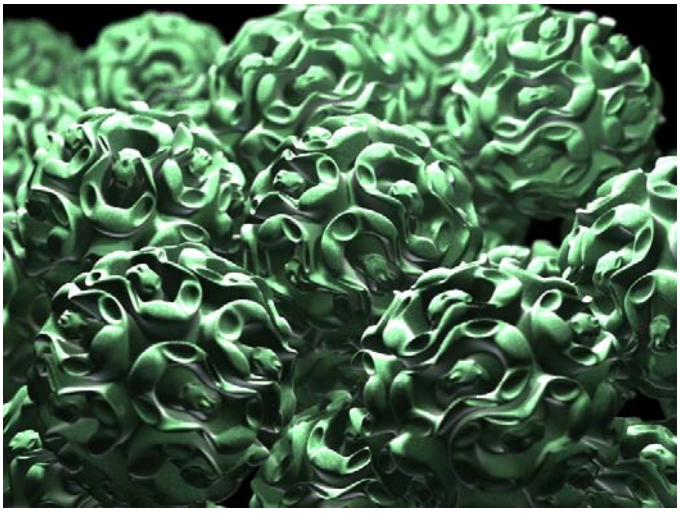


Fig. 7. A detailed shape that is interactively constructed using our approach

implemented in C++ and OpenGL. All the examples in this paper were created using this system. See Figures 5 and 6 for more examples.

4 Conclusion and Future Work

In this paper, we have presented an approach that allows designer to interactively create partially self-similar manifold surfaces without relying on shape grammars or fractal methods. Using this approach, we have developed a system that enables designers to control each step of the shape generation process with a high level of interactivity. With the new approach, designers can easily create a large set of connected "self-similar" manifold surfaces. Our method allows us to construct connected manifold surfaces which can be realized by a 3D printer.

References

1. Elber, G.: Geometric texture modeling. IEEE Computer Graphics and Applications ((25))
2. Bhat, P., Ingram, S., Turk, G.: Geometric texture synthesis by example. In: Proc. Eurographics Symposium on Geometry Processing. (2004) 8–10
3. Barnsley, M.: Fractals Everywhere. Academic Press, Inc. San Diego Ca. (1988)
4. Akleman, E., Chen, J., Srinivasan, V.: A minimal and complete set of operators for the development of robust manifold mesh modelers. Graphical Models Journal, Special issue on International Conference on Shape Modeling and Applications 2002 **65**(2) (2003) 286–304
5. Landreneau, E., Akleman, E., Srinivasan, V.: Local mesh operations,. In: Proceedings of the International Conference on Shape Modeling and Applications. (2005) 351 – 356
6. Fournier, A., Fussel, D., Carpenter, L.: Computer rendering of stochastic models. In: Proceedings of Computer Graphics, Siggraph. (1982) 97–110
7. Srinivasan, V., Akleman, E.: Connected and manifold sierpinski polyhedra. In: Proceedings of Solid Modeling and Applications. (2004) 261–266
8. Akleman, E., Srinivasan, V., Melek, Z., Edmundson, P.: Semi-regular pentagonal subdivision. In: Proceedings of the International Conference on Shape Modeling and Applications. (2004) 110–118
9. Mandelbrot, B.: The Fractal Geometry of Nature. W. H. Freeman and Co., New York (1980)
10. Loop, C.: Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah (1987)
11. Thornton, G., Sterling, V.: Xenodream software,. http://www.xenodream.com (2005)

Straightest Paths on Meshes by Cutting Planes

Sungyeol Lee, Joonhee Han, and Haeyoung Lee

Hongik University, Dept. of Computer Engineering,
72-1 Sangsoodong Mapogu, Seoul Korea 121-791
{leesy, hanj, leeh}@cs.hongik.ac.kr

Abstract. Geodesic paths and distances on meshes are used for many applications such as parameterization, remeshing, mesh segmentation, and simulations of natural phenomena. Noble works to compute shortest geodesic paths have been published. In this paper, we present a new approach to compute the straightest path from a source to one or more vertices on a manifold mesh with a boundary. A cutting plane with a source and a destination vertex is first defined. Then the straightest path between these two vertices is created by intersecting the cutting plane with faces on the mesh. We demonstrate that our straightest path algorithm contributes to reducing distortion in a shape-preserving linear parameterization by generating a measured boundary.

1 Introduction

Calculating geodesic paths and distances on meshes is a fundamental problem in various graphics applications such as parameterization [10], remeshing [6,13,18], mesh segmentation [18], and simulations of natural phenomena [15,9].

Noble works to compute shortest geodesic paths and distances have been presented [12,1,7,19]. An algorithm for straightest geodesic paths [14] was first proposed by Polthier and Schmies. Their straightest geodesics are well defined with the initial condition (a source and direction) but not with boundary conditions (a source and a destination). The straightest path may not be the same as the shortest path on meshes and may be more appropriate for some applications like wave propagation [15] or parameterization [10] for texture mapping.

In this paper, we present a new and simple algorithm to compute straightest paths and distances between two vertices on manifold meshes with a boundary as shown in Figure 1. Our straightest path algorithm enables us to create a measured boundary for a linear parameterization and hence reduces distortion more than the parameterization with a fixed boundary as shown in Figure 3 and 4.

1.1 Related Work

There are several algorithms for geodesic computations on meshes, mostly based on shortest paths with boundary conditions i.e., source and destination vertices.

Derived from Dijkstra's algorithm, The MMP algorithm [12] is introduced to compute an exact shortest geodesic path. The CH algorithm [1] also allows the user to compute an exact shortest path with faster processing. Their implementation and extension are also proposed by [5,6,19] but they are still considered

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 609–615, 2006.
© Springer-Verlag Berlin Heidelberg 2006

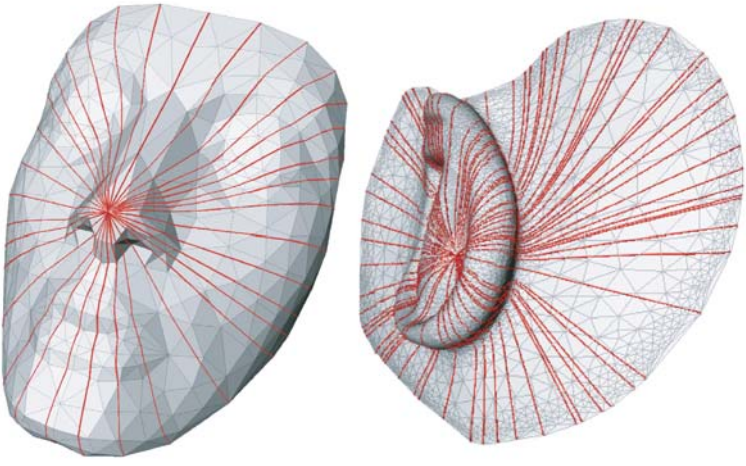


Fig. 1. Straightest paths from a source to every vertex on the boundary as determined by our method. Models are Nefertiti on the left and an Ear on the right.

hard to implement. The Fast Marching algorithm by [7] computes an approximate geodesic path on meshes and has been used for remeshing and parameterization [18,13]. However, it requires special processing for triangles with obtuse angles. A detailed overview of this approach can be seen in [11].

Another approach is to compute the straightest geodesic path. Polthier and Schimes [14] presented an algorithm to compute a straightest geodesic path on a mesh. They extended the notion of straight lines from the Euclidean plane onto the surface. It is uniquely defined with a source and direction. However it is not always defined between a source and a destination and also requires special handling of the swallow tails created by conjugate vertices [15] and triangles with obtuse angles [9].

Our method in this paper is designed to compute straightest paths between a source and one or more vertices on manifold meshes with a boundary.

2 Our Straightest Path Algorithm with Cutting Planes

We present a new and simple algorithm to compute a straightest path from a source to one or more destinations on a genus-0 surface patch. Figure 2 explains our simple steps to compute a straightest path between a source S and a destination D.

1. Specify a center S which is the point of origin and determine the vertex normal at S as follows:
   - Make virtual edges from S to every boundary vertex of the mesh.
   - Make virtual faces around S with these virtual edges.
   - Average normal vectors of virtual faces around S and assign it to the vertex normal vector at S.

   Then design the base plane B at S with the above vertex normal.

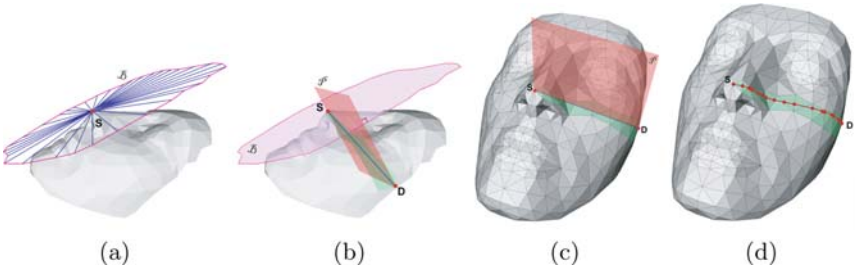


Fig. 2. Our straightest path algorithm: (a) The base plane B at the the source S is generated. (b) A plane P with S and D, which is vertical to B, is generated. (c) P cuts the mesh at intersection faces. Each green face is a result of cutting the mesh by the plane P. Red intersection vertices on green faces are then calculated and connected for our straightest path as shown in (d).

2. Find a plane P (vertical to B) where the line from S to D is.
3. Repeat the following two sub-steps until it reaches D.
   - Find a face by intersecting the mesh with the cutting plane P at the starting vertex.
   - Find a line segment by calculating an intersection vertex on the face and connecting it to the starting vertex.

Two planes always intersect in a line as long as they are not parallel. Our cutting plane P pierces each green face on the mesh. Therefore there is a unique line segment which is the straightest path by our method. The tangent a for a line segment can be easily calculated from the normal N_1 of the green face and the normal N_2 of the cutting plane P as follows:

$$a = N_1 X N_2 \tag{1}$$

The first green face intersecting with the cutting plane P can be easily found by searching only the 1-ring neighbor faces of the source S , the point of origin. Once you find the tangent a on the green face, the red intersection vertex is calculated and connected to S to produce the line segment. Then the red intersection vertex plays the role of a starting vertex to find the next green face. These processes should be repeated until the last red intersection vertex is D. The straightest distance from S to D is the sum of Euclidean distances of red line segments on the green faces.

For multiple destinations from a source, the 2nd and 3rd steps in the above procedure are to be repeated for each destination.

Figure 1 shows straightest paths by our new method from a source to boundary vertices on the models, Nefertiti on the left and the Ear on the right.

Discussion. The tangents of the previous straightest geodesics by Polthier and Schimes [14] are determined by gaussian curvatures at vertices and may be changed into random directions especially when the gaussian curvature is not

2π. As a result their straightest path from source may not be reached a destination. Our straightest path stays both on the cutting plane and on the mesh from the source to the destination. Therefore our straightest path can reach the destination.

Our algorithm is output-sensitive to the number of intersection vertices to form a straightest path, which is a lot less than the total number of vertices on the mesh. Our method is linear to the total number of vertices, i.e., *O(V)* and does not require preprocessing of triangles with obtuse angles.

3 Parameterization with a Measured Boundary

A 3D mesh parameterization provides a piecewise linear mapping between a 3D surface patch and an isomorphic 2D patch. It is a widely used or required operation for texture-mapping, remeshing, morphing or geometry imaging. Guaranteed one-to-one mappings that only requires a linear solver have been sought and many algorithms [3,4,8,10] were proposed. To reduce inevitable distortions when flattening, a whole object is usually partitioned into several genus 0 surface patches. Generally the first step for parameterization is mapping boundary vertices to a fixed position. Usually the boundary is mapped to a square, a circle, or any convex shape while respecting the 3D-to-2D length ratio between adjacent boundary vertices. As long as the boundary vertices are mapped to a convex shape, the resulting mapping will be guaranteed to be one-to-one. The 2D embedded positions of the interior vertices are then found by solving a linear system. The linear system is generated with coefficients in a convex

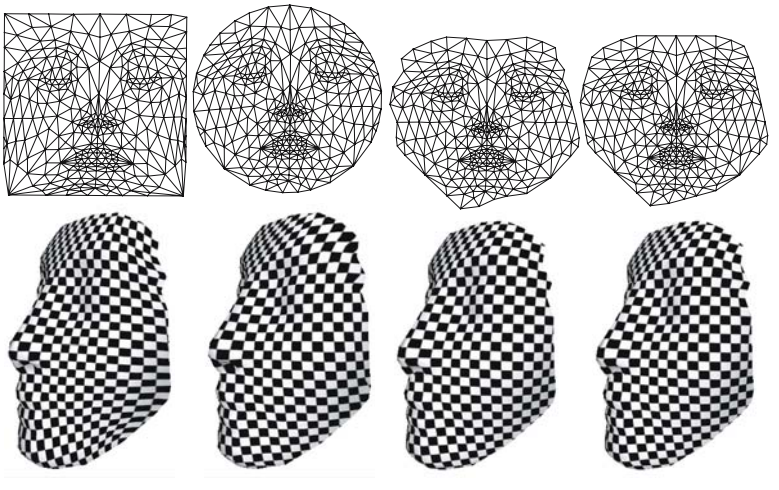


Fig. 3. Parameterization for Nefertiti with different boundaries: From the left, a square, a circle, and a measured using our straightest distances are used for the boundary. The final boundary is modified to a convex from the measured boundary. Notice less distortion near the measured boundary by our method.

combination of 1-ring neighbors for each interior vertex. These coefficients characterize a shape-preserving property. This approach primarily concentrates on how to determine these coefficients.

However as shown in Figure 3 and 4, high distortion occurs near the boundary. To reduce it, a free boundary [2] with a non-linear functional and a virtual boundary [8] with layers of additional vertices have been proposed.

In this paper, we attempt to derive a measured boundary, linearly and without additional vertices, by our straightest path algorithm to reduce distortions. Straightest paths and distances from a center S to every boundary vertex of the mesh can be measured as follows:

1. Make virtual edges from S to every boundary vertex of the mesh.
2. Map each virtual edge onto the base plane B by a polar map, which preserves angles between virtual edges such as [3] (see for Figure 2 (a)).
3. Measure the straightest distance for each virtual edge on B from S to each boundary vertices with corresponding cutting planes.
4. Position each boundary vertex at the corresponding distance from S on B.

Any previous parameterization can be applied and we choose to use [10]. Our straightest path contributes to deriving measured boundaries, reducing distortion, and much better texture-mapping as shown in Figure 3 and 4. The measured boundary is however dependent on the user-specified source S because a different choice of the source S generates different base plane B and different distances to boundary vertices. The center of the mesh is chosen for S by our experiments.

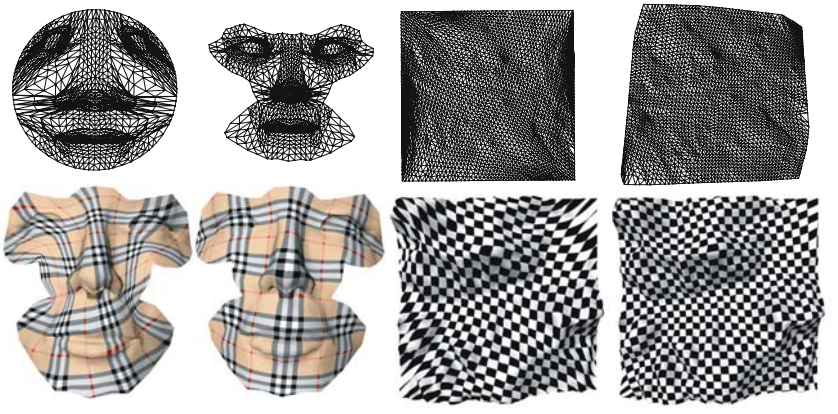


Fig. 4. More parameterization with different boundaries. Models are Face in the two left columns and the Mount on the two right columns. Notice less distortion near the measured boundary of each model.

4 Results

The visual results by our method are shown in Figure 1 for straightest paths from a source to boundary vertices. Figure 3 and 4 also demonstrate visual results by using our straightest paths. The distortion with the texture-stretch metric in [17] is also measured and shown in Table 1. Notice that a parameterization [10] with a measured boundary reduces distortion.

Table 1. Distortion measured by the texture stretch metric [17]: [10] is used for parameterization. The boundary is fixed to a circle. Our straightest path algorithm is used for measured boundaries and reduces distortion.

Models	No. of Vertices	LTD's [10] fixed bound.	LTD's [10] measured bound.
Nefertiti	299	1.165	1.152
Face	1547	1.591	1.341
Mountain	2500	1.552	1.436

The performance complexity of our algorithm is all linear to the number of vertices, i.e., $O(V)$. The longest processing time among our models in Table 1 is 0.53 sec, required for the Mountain having the highest number of vertices. The processing time is measured on a laptop with a Pentium M 2.0GHz 1GB RAM.

5 Conclusion and Future Work

In this paper, we introduce a new and simple algorithm to compute straightest paths between two vertices on manifold meshes with a boundary. To our knowledge our work is the first algorithm to compute the straightest path with boundary conditions. A cutting plane with a source and a destination vertex is first defined. Then the straightest path between these two vertices is created by intersecting the cutting plane with faces on the mesh. We demonstrate the utility of our straightest path algorithm to derive a measured boundary for parameterizations.

In the future, we will study to find straightest geodesic paths between two vertices on closed manifold meshes. Also we will extend the utility of our straightest path algorithm by applying it to other mesh processing techniques such as remeshing, subdivision, or simplification.

Acknowledgement

This work was supported by grant No. R01-2005-000-10120-0 from Korea Science and Engineering Foundation in Ministry of Science & Technology. Thanks to Caltech's Applied Geometry Lab. and Postech's Computer Graphics Lab. for Models, Nefertiti, Ear, Mount, and Face.

References

1. Chen J., Han Y.: "Shortest Paths on a Polyhedron; Part I: Computing Shortest Paths", Int. J. Comp. Geom. & Appl. 6(2), 1996.
2. Desbrun M., Meyer M., Alliez P.: "Intrinsic Parameterizations of Surface Meshes", Eurographics 2002 Conference Proceeding, 2002.
3. Floater M.: "Parametrization and smooth approximation of surface triangulations", Computer Aided Geometric Design, 1997.
4. Floater M.: "Mean Value Coordinates", Comput. Aided Geom. Des., 2003.
5. Kaneva B. O'Rourke J.: "An implementation of Chen and Han's sortest paths algorithm", Proc. of the 12th Canadian Conf. on Computational Geometry, 2000.
6. Kanai T., Suzuki H.: "Approximate Shortest Path on a Polyhedral Surface Based on Selective Refinement of the Discrete Graph and Its Applications", Proc. Geometric Modeling and Processing 2000 (HongKong) 2000.
7. Kimmel R., Sethian J.A.: "Computing Geodesic Paths on Manifolds", Proc. Natl. Acad. Sci. USA Vol.95 1998, 1998.
8. Lee Y., Kim H., Lee S.: "Mesh Parameterization with a Virtual Boundary", Computer and Graphics 26 (2002), 2002.
9. Lee H., Kim L., Meyer M., Desbrun M.: "Meshes on Fire", Computer Animation and Simulation 2001, Eurographics, 2001.
10. Lee H., Tong Y. Desbrun M.: "Geodesics-Based One-to-One Parameterization of 3D Triangle Meshes", IEEE Multimedia January/March (Vol. 12 No. 1), 2005.
11. Mitchell J.S.B.: "Geometric Shortest Paths and network optimization", In Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, Eds. Elsevier Science 2000.
12. Mitchell J.S.B., Mount D.M., Papadimitriou C.H.: "The Discrete Geodesic Problem", SIAM J. of Computing 16(4), 1987.
13. Peyré G., Cohen L.: "Geodesic Re-meshing and Parameterization Using Front Propagation", In Proceedings of VLSM'03, 2003.
14. Polthier K., Schmies M.: "Straightest Geodesics on Polyhedral Surfaces", Mathematical Visualization, 1998.
15. Polthier K., Schmies M.: "Geodesic Flow on Polyhedral Surfaces", Proceedings of Eurographics-IEEE Symposium on Scientific Visualization '99, 1999.
16. Riken T., Suzuki H.: "Approximate Shortest Path on a Polyhedral Surface Based on Selective Refinement of the Discrete Graph and Its Applications", Geometric Modeling and Processing 2000 (Hongkong), 2000.
17. Sander P.V., Snyder J., Gortler S.J., Hoppe H.: "Texture Mapping Progressive Meshes", Proceedings of SIGGRAPH 2001, 2001.
18. Sifri O., Sheffer A., Gotsman C. : "Geodesic-based Surface Remeshing", In Proceedings of 12th Intnl. Meshing Roundtable, 2003.
19. Surazhsky V., Surazhsky T., Kirsanov D., Gortler S., Hoppe H.: "Fast Exact and Approximate Geodesics on Meshes", ACM SIGGRAPH 2005 Conference Proceedings, 2005.

3D Facial Image Recognition Using a Nose Volume and Curvature Based Eigenface

Yeunghak Lee[1], Ikdong Kim[2], Jaechang Shim[2], and David Marshall[1]

[1] Communitcation Research Centre, Cardiff University, Queen's Buildings
5 The Parade, Roath, Cardiff, CF24 3AA, Wales, UK
{Y.Lee, Dave.Marshall}@cs.cardiff.ac.uk
[2] Dept. of Computer Engineering, Andong National University,
388 Songcheon-Dong, Andong, Kyungpook, Korea, 760-749
{kid7, jcshim}@andong.ac.kr

Abstract. The depth information in the face represents personal features in detail. In this study, the important personal facial information was presented by the surface curvatures and the features of vertical and horizontal of nose volume extracted from the face. The approach works by the depth of nose, the area of nose and the volume of nose based both on a vertical and horizontal are calculated. And the principal components analysis (PCA), which is calculated using the curvature data, was presented different features for each person. To classify the faces, the cascade architectures of fuzzy neural networks (CAFNNs), which can guarantee a high recognition rate as well as parsimonious knowledge base, are considered. In the experimental results, 3D images demonstrate the effectiveness of the proposed methods.

1 Introduction

The ability to recognize a person is a task that humans risibly perform but one that computers to date have been unable to perform robustly. Humans employ many visual cues to recognize a face – a process that has been fine tuned by many years through evolution. There are many applications for computers to recognize face ranging from security, tracking through to intelligent multimedia interfaces. To recognize a person's face automatically, many recognition methods have been researched using many a variety of biometric features [1]. Broadly speaking the two ways to establish recognition employs the face feature based approach and the area or statistical based approach. A new set of data processing challenges exist for 3D data which is now being more readily researched [2-5]. Many researchers have used 3D face recognition using differential geometry tools for the computation of curvature [2]. Hiromi et al. [3] treated 3D shape recognition problem of rigid free-form surfaces. Each face in the input images and model database is represented as an Extended Gaussian Image (EGI), constructed by mapping principal curvatures and their directions. Gordon [4] presented a study of face recognition based on depth and curvature features. To find face specific descriptors, he used the curvatures of the face. Comparison of the two faces was made based on the relationship between the spacing of the features. Lee and

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 616 – 622, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Milios [6] extracted the convex regions of the face by segmenting the range of the images based on the sign of the mean and Gaussian curvature at each point.

In this paper, we introduce novel 3D face recognition using nose volume and eigenfaces using the curvature of the 3D face data. We show that this approach well preserves personal characteristics of the face as well as reducing the inherent high dimensional face feature spaces. And we use NNs to classify features in our reduced dimension PCA [7] space. To overcome inherent curse of dimensionality in NNs, cascade architectures of fuzzy neural networks (CAFNNs) is constructed by the use of memetic algorithms (hybrid genetic algorithms) [8].

2 Surface Curvatures

For each data point on the facial surface, the principal, Gaussian and mean curvatures are calculated and the signs of those (positive, negative and zero) are used to determine the surface type at every point. The $z(x, y)$ image represents a surface where the individual Z-values are surface depth information. Here, x and y is the two spatial coordinates. We now closely follow the formalism introduced by Peet and Sahota [9], and specify any point on the surface by its position vector:

$$R(x, y) = xi + yj + z(x, y)k \tag{1}$$

For the simple facet model of second order polynomial of the form, i.e. a 3x3 window implementation in our range images, the local region around the surface is approximated by a quadric

$$z(x, y) = a_{00} + a_{10}x + a_{01}y + a_{01}y + a_{20}x^2 + a_{02}y^2 + a_{11}xy \tag{2}$$

Using the first fundamental form and the second fundamental form of the surface, k_1 (the minimum) and k_2 (the maximum) curvatures, are presented by the following expressions for, respectively. Here we have ignored the directional information related to k_1 and k_2, and chosen k_2 to be the larger of the two. The two quantities, k_1 and k_2, are invariant under rigid motions of the surface. Two other curvatures, Gaussian and the mean curvature, are defined by

$$K = k_1 k_2 \quad M = (k_1 k_2)/2 \tag{3}$$

It turns out that the principal curvatures, k_1 and k_2, and K and M are best suited to the detailed characterization for the facial surface, as illustrated in Fig. 1 and the practical calculation of principal and Gaussian curvatures is extremely simple.

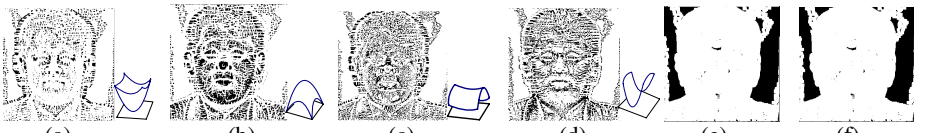
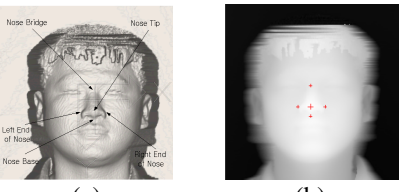


Fig. 1. Six possible surface type according to the sign of principal curvatures for the face surface; (a) concave (pit), (b) convex (peak), (c) convex saddle, (d) concave saddle, (e) minimal surface, (f) plane

3 Feature Extraction and Scalar Features from 3D Image

We now consider the novel processing of 3D facial features and the adaption of the above classification techniques for 3D face recognition. For the range image, the scalar features are extracted from concavities that exist on curves, except range image's nose. We can easily find that the smallest value is feature point, as shown in Fig. 2. And we adopted 12 features values.



(a) (b)

Fig. 2. The feature point definition and the result of the feature point extraction (a) the definition of feature points (b) the result of feature point

3.1 Depth, Area and Angle of Vertical Section

The depth (f_1) of normal line is straight line b-c, which is perpendicular to normal line, as shown Fig. 3 (a). Equation (16) and (17), present relation between a point and straight line,

$$kz + ly + m = 0 \tag{4}$$

$$(y_2 - y_1)z_0 + (z_1 - z_2)y_0 + \{y_1(z_2 - z_1) - z_1(y_2 - y_1)\} = 0 \tag{5}$$

and the f_1 is calculated by

$$f_1 = \frac{|k * z_0 + l * y_0 + m|}{\sqrt{k^2 + l^2}} \tag{6}$$

where k, l and m are constant. If strait line b-c is D_3, the area of longitudinal section (f_2) calculated by

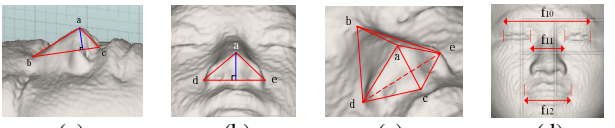
$$kz + ly + m = 0 \tag{7}$$

$$f_2 = \frac{1}{2} * D_3 * f_1 = \frac{1}{2} * \left| k * z_0 + l * y_0 + m \right|$$

$$= \frac{1}{2} \left| (y_2 - y_1)z_0 + (z_1 - z_2)y_0 + \{y_1(z_2 - z_1) - z_1(y_2 - y_1)\} \right| \tag{8}$$

The angle of longitudinal section (f_3) and $\angle bac$ are calculated by

$$f_3 = \sin^{-1}\left(\frac{2 * f_2}{t_1 * t_2} \right) \tag{9}$$



(a) (b) (c) (d)

Fig. 3. 3D expression of feature points for longitudinal section and transection and other scalar features (a) the depth of longitudinal section (b) the depth of transection (c) the volume of nose (d) other scalar features

3.2 Depth, Area and Angle of Horizontal Section

The depth of normal line is from point a to strait line d-e, which is perpendicular to normal line, as shown Fig. 3 (b). The depth of transection(f_4) is calculated by

$$f_4 = \frac{|k * x_0 + l * z_0 + m|}{\sqrt{k^2 + l^2}} \tag{10}$$

where k, l and m are constant. If strait line b-c is D_4, the area (f_5) of transection calculated by

$$f_5 = \frac{1}{2} * D_4 * f_3 = \frac{1}{2} * |k * x_0 + l * z_0 + m|$$
$$= \frac{1}{2} |(z_4 - z_3)x_0 + (x_3 - x_4)z_0 + \{z_3(x_4 - x_3) - x_3(z_3 - z_3)\}| \tag{11}$$

And the angle (f_6) of longitudinal section, $\angle dae$, is calculated by

$$f_6 = \sin^{-1}\left(\frac{2 * f_5}{t_1 * t_2}\right) \tag{12}$$

where t_1 is straight line *a-d* and t_2 is straight line *a-e*.

3.3 The Volume of Nose

As shown in Fig. 3 (c), the volume of nose is composed of the tetrahedral *abde* and *acde*. For the volume of each tetrahedral v_1 and v_2, firstly, the distance of between point *b* and point *c* and plane which includes Δade is calculated. The volume of tetrahedral (v_1) using the equation (9) and (10) which is plane equation is calculated. The plane equation which includes three points, $a(x_0, y_0, z_0)$, $d(x_4, y_0, z_3)$ and $e(x_4, y_0, z_4)$ is given by

$$kx + ly + mz + n = 0 \tag{13}$$

where k, l and m and are constant [10]. The distance, D_5, from the point $b(x_0, y_1, z_1)$ to plane, and D_6, the distance from the point $c(x_0, y_2, z_2)$ to plane are calculated by

$$D_5 = \frac{|k * x_0 + l * y_1 + m * z_1 + n|}{\sqrt{k^2 + l^2 + m^2}} \quad D_6 = \frac{|k * x_0 + l * y_2 + m * z_2 + n|}{\sqrt{k^2 + l^2 + m^2}} \tag{14}$$

Finally, the volume (f_7) of nose is given by

$$f_7 = v_1 + v_2 = \frac{1}{3} * D_5 * f_5 + \frac{1}{3} * D_6 * f_5 = \frac{1}{3} * f_5 * (D_5 + D_6) \tag{15}$$

As shown in Fig. 3 (c), the angle (f_8) of nose bridge, $\angle dce$, and the angle (f_9) of nose base, $\angle dce$, are calculated by equation (12). Additionally the distance of the eye cavity (outside corner), f_{10}, and the distance of the eye cavity (inside corner), f_{11}, and the length of mouth, f_{12} are calculated. For these parameters, maximum curvature is used in this paper. The process of erosion and dilation was conducted to extract the shape of mouth and eye area.

4 Experimental Results

In this study, we used a 3D laser scanner system made by a 4D culture [11] to obtain a 3D face image, 320 by 320 pixels. A database is used to compare scalar features and the different NN recognition strategies, and is composed of 92 images (two images of 46 persons). Of the two pictures available, the second photos were taken at a time interval of 30 minutes.

To perform recognition experiments for extracted area we first need to create two sets of images, i.e. training and testing. For each of the two views, 46 normal-expression images were used as the training set. Training images were used to generate an orthogonal basis, PCA curvature based eigenface. Testing images are a set of 3D images extracted local area we wish to identify.

4.1 Nose Volume

Table 1 represents the result of recognition rate for features, f_1, f_2, f_4, f_5, and f_7. The depth (f_4) and the area (f_5) of longitudinal section is represented higher recognition rate and the area (f_2) of transection is represented the lowest recognition rate. To improve the recognition rate, we adapted the weighted value for each features. The result w_1 and w_2 represent more improved recognition rate in Table 1. The w_1 results that each features were applied same weighted value, and the w_2 results that each features were applied different weighted values according to the each recognition rate.

$$diff = \sum_{i=1}^{5} \left| (w_i * f_i)_{Original_img} - (w_i * f_i)_{DB_img} \right| \tag{16}$$

For the second experiment, the database added 6 images included a little noise. The depth (f_4) of longitudinal section was the primary recognition rate when rank threshold is 5, and if ranked threshold is 10, the area (f_5) of longitudinal section was the best recognition rate, 82.14%.

4.2 Curvature Based Eigenface Using Neural Network

Once the data sets have been extracted using the eigenface approach, to compare training set with data set, the CAFNNs have been followed for the face recognition. The used parameter values are the same as [8]. Since a genetic algorithm is a stochastic

Table 1. The comparison of each features recognition rate

	Ranked Threshold		
	5	10	15
f_1	54%	74%	84%
f_2	30%	64%	86%
f_4	66%	90%	96%
f_5	64%	96%	100%
f_7	46%	82%	100%
w_1	86%	100%	100%
w_2	94%	100%	100%

optimization method, ten times independent simulations were performed to compare the results with the conventional classification method, i.e., neural networks (NNs) [12, 13]. The learning of NNs has been continued until the performance index is not changed. In Fig. 4, the results of the CAFNN are averaged over ten times independent simulations, and subsequently compared with the results of the NNs. Also, the normalized facial images were considered to generate the curvature-based data set. As can be seen from Fig. 4, the recognition rate is improved by using normalized facial images. Moreover, the CAFNN is superior to NNs because the CAFNN uses the most relevant input subsets to construct a parsimonious knowledge base (simple structure to optimize), while the NNs use all the inputs (very complex structure to optimize).

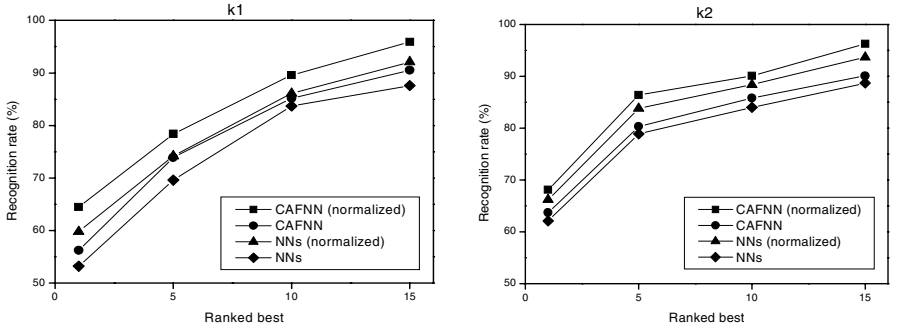


Fig. 4. The recognition results using eigenfaces for each algorithm: (a) k_1 and (b) k_2

5 Conclusions

Nose volume and the surface curvatures extracted from the face contain the most important personal facial information. We have introduced, in this paper, a new practical implementation of a person verification system using the local shape of 3D face images based on PCA and nose features. The underlying motivations for our approach originate from the observation that nose volume and the curvature of face has

different characteristic for each person. The CAFNNs have reduced the dimensionality problem by selecting the most relevant input subspaces too. Experimental results on a group of face images (92 images) demonstrated that our approach produces excellent recognition results for the local eigenfaces than using scalar features. From the experimental results, we proved that the process of face recognition may use low dimension, less parameters, calculations and less same person images (used only two) than earlier suggested. We consider that there are many future experiments, combination of scalar and vectors that could be done to extend this study.

References

1. Jain, L. C., Halici, U., Hayashi, I., Lee, S. B.: Intelligent biometric techniques in fingerprint and face recognition. CRC Press (1999)
2. Chua, C. S., Han, F., Ho, Y. K.: 3D Human Face Recognition Using Point Signature. Proc. of the 4th ICAFGR (2000)
3. Tanaka, H. T., Ikeda, M., Chiaki, H.: Curvature-based face surface recognition using spherial correlation. Proc. of the 3rd IEEE Int. Conf. on Automatic Face and Gesture Recognition (1998) 372-377
4. Gordon, G. G.: Face Recognition based on depth and curvature feature. Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition (1992) 808-810
5. Chellapa, R., Wilson, C. L., Sirohey, S.: Human and machine recognition of faces: A survey. Proceedings of the IEEE, Vol. 83, No. 5 (1995) 705-740
6. Lee, J. C., Milios, E.: Matching range image of human faces. Proc. of the 3rd Int. Conf. on Computer Vision (1990) 722-726
7. Turk, M., Pentland, A.: Eigenfaces for Recognition. Journal of Cognitive Neuroscience, Vol. 3, No. 1 (1991) 71-86
8. Han, C. W., Pedrycz, W.: A new genetic optimization method and its applications. submitted to International Journal of Approximate Reasoning
9. Peet, F. G., Sahota, T. S.: Surface Curvature as a Measure of Image Texture. IEEE Trans. PAMI, Vol. 7, No. 6 (1985) 734-738
10. Mathematics Book Publishing Committee: Linear algebra and Geometry. Hyungseul Publishing Co. (1992)
11. 4D Culture. http://www.4dculture.com
12. Zhao, Z. Q., Huang, D. S., Sun, B. Y.: Human face recognition based on multi-features using neural networks committee. Pattern Recognition Letters, Vol. 25 (2004) 1351-1358
13. Pedrycz, W., Reformat, M., Han, C. W.: Cascade architectures of fuzzy neural networks. Fuzzy Optimization and Decision Making, Vol. 3 (2004) 5-37

Surface Reconstruction for Efficient Colon Unfolding

Sukhyun Lim, Hye-Jin Lee, and Byeong-Seok Shin

Inha University, Dept. Computer Science and Information Engineering
253 Yonghyun-dong, Nam-gu, Inchon, 402-751, Rep. of Korea
{slim, jinofstar}@inhaian.net, bsshin@inha.ac.kr

Abstract. Unfolding is a new visualization method for colorectal disease and polyp detection. Compared with a virtual endoscopy method, it is more suitable for medical applications because it gives us unfold images of the inner surface in an organ. However, since conventional unfold methods generate only 2D images, it is difficult to show the surface at a first glance and to manage unfolding results with diverse viewing controls such as rotation and magnification. To solve it, we propose an efficient unfolding method using surface reconstruction. Firstly, we generate a 2D unfold image using volume ray casting. At the same time, we store distance values from each sample point on a central path to colon surface, for all rays. After making a height field from the distance values, we reconstruct a 3D surface model. Lastly, a 3D unfold image is acquired by mapping the 2D unfold image into the 3D surface model. Since our method offers the overall shape of an organ surface, the problematic areas can be identified quickly and inspected afterwards in more detail.

1 Introduction

The inspection of organ cavities using medical imaging and computer visualization techniques is called virtual endoscopy [1], [2]. An optical endoscopy is a less-invasive procedure that involves a certain degree of risk for the patient. In some diagnostic procedures, a virtual endoscopy has the potential to be used in clinical routine to avoid the inconvenience of an optical endoscopy. Although the virtual endoscopy provides a less-invasive inspection of inner structure in human cavities using tomographic images, it cannot provide the entire view of an organ surface due to limited field-of-view [3]. In addition, because some polyps may be hidden due to complex structures of organs and folds [3], it is difficult to detect them.

Recently, an unfold rendering method is regarded as one of the new techniques to visualize human cavities [4], [5], [6], [7], [8], [9], [10], [11], [12]. Because the virtual dissection of organs is analogous to the real anatomical dissection, we can easily and intuitively recognize special features and pathologies of the organs [4], [5], [8], [9]. The general step is as follows: for each sample point on a central path, rays are radially cast orthogonal to the tangent vector of a sample point. After the rays leaps over colon cavity (transparent region), a 2D unfold image is generated.

However, because conventional unfolding approaches including our previous method [12] only generate 2D rendering results, it is difficult to detect polyps efficiently. To solve it, we propose an efficient unfolding method more similar to the real

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 623 – 629, 2006.
© Springer-Verlag Berlin Heidelberg 2006

dissection of organs. When a 2D unfold image is generated using volume ray casting method [13], [14], [15], [16], [17], we store the distance values between a sample point and colon surface for all rays in entire sample points, in a distancemap. Since the distance values are already computed in volume ray casting step, additional costs are not required. Then, after generating a height field from the values, we generate a 3D unfold model. Lastly, we map the 2D unfold image to the 3D model. As a result, we reconstruct a 3D unfold image. By our method, we can identify problematic areas quickly. In addition, our approach can be applied to any variant of unfold rendering methods.

In Sect. 2, we review our method in detail. Experimental results are presented in Sect. 3, and we conclude our method in last section.

2 Colon Unfolding Using Surface Reconstruction

We assume that the central path of an organ cavity is already computed. It is represented as a set of sample points $\boldsymbol{S_k}$ ($0 \leq k < N$, where N is a total number of sample points). Our method cast rays $\boldsymbol{R_i^k}$ ($0 \leq i < 360$) in the $\boldsymbol{S_k}$. In a 2D unfolding image, therefore, the number of sample points reflects the pixel resolution of x-axis, and the number of rays means that of y-axis (that is, the image size is $N \times i$). The θ is an angle between two consecutive rays in the $\boldsymbol{S_k}$.

To obtain a result similar to a real dissected organ, we reconstruct a 3D surface model. Our method can be divided into two main steps: generating 2D unfold image and 3D unfold model. Fig. 1 shows an overall process of our method. Since those steps are performed sequentially, any variant of unfolding method based on volume ray casting incorporate with our method easily. The 2D image generation step is almost identical to our previous work [12]. Only difference is that we store distance values for all rays in entire sample points.

The steps are as follows: when we compute color values using volume ray casting in 2D image generation, we store the distance (height) $\boldsymbol{h_i^k}$ for the $\boldsymbol{R_i^k}$ to colon surface (see Fig. 2). If we exploit the volume ray casting, color values are computed after leaping colon cavity (transparent region). Therefore, we can acquire them without additional costs. Then, we generate a height field from the distance values, and we determine scaling factors along y- and z-axis for all rays in entire sample points. By applying the two scaling factors, we generate a 3D surface model. Lastly, a 2D image texture is mapped to the 3D surface model.

In next sub-section, we present the method to scale y- and z-axis in detail.

2.1 Scaling Along Y-axis

Due to complicated structures including folds on colon wall, dissected colon cannot be represented as a cylindrical shape. Therefore, we have to make virtually dissected images similar to real organs. Since the distance values in a distancemap are estimated as a height-field data, we can determine a scaling factor along y-axis by Eq. (1).

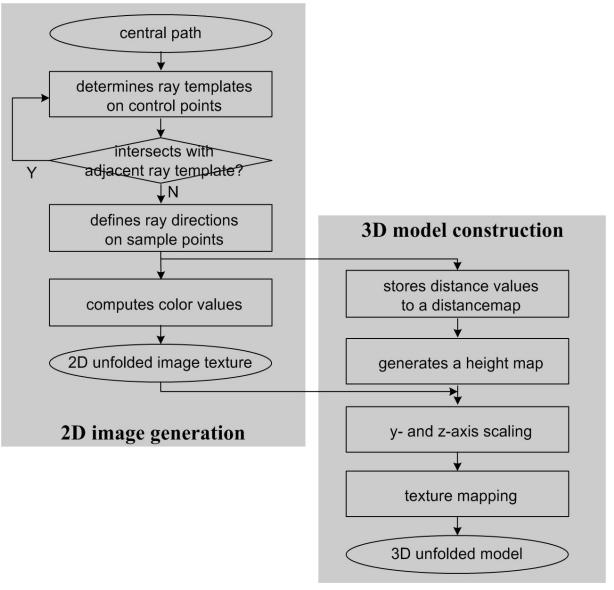


Fig. 1. A block diagram of our method; 2D image generation and 3D model construction

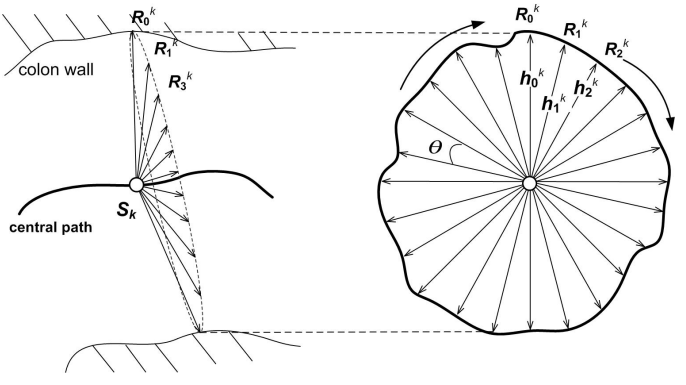


Fig. 2. Compared with our previous method, we store the distance values $\boldsymbol{h_i^k}$ of all rays from a sample point $\boldsymbol{S_k}$ to colon surface in a distancemap

This equation is widely used to compute a height field, and it also exploited in conventional method [9]. However, although the method performs lots of iterations to reduce distortions, our method do not perform them because we focus on reduction of the rendering time. In Eq. (1), assume that $\boldsymbol{d_i^k}$ is an estimated distance for the $\boldsymbol{R_i^k}$, and $\boldsymbol{rd_i^k}$ is a scaled distance.

$$d_i^k = \tan(\theta/2) \times h_i^k \,. \qquad (1)$$
$$sd_i^k = d_i^k + d_{i+1}^k \,.$$

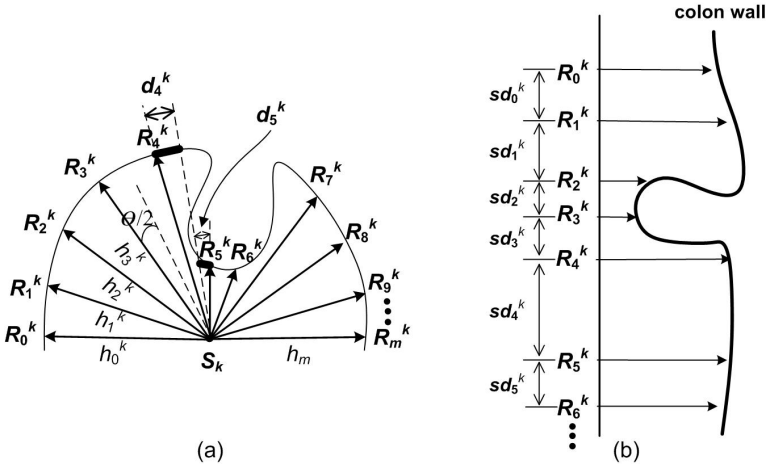


Fig. 3. (a) We compute y-axis scaling factors for all rays. For example, sd_4^k is computed by adding d_4^k and d_5^k. (c) After applying scaling factors to all sample points, we acquire a scaled 2D image.

2.2 Scaling Along Z-axis

We generate a 3D unfold image by determining scaling factors for z-axis. Since we already have the height field data (h_i^k), we can generate a 3D surface model easily through Eq. (2), where **min** is the minimum height, and z_i^k is estimated height. Because our concern is relative height, the computation of **min** is required.

$$z_i^{\,k} = h_i^{\,k} - \min_{j,p=0}^{N,360} h_p^{\,j} \,. \tag{2}$$

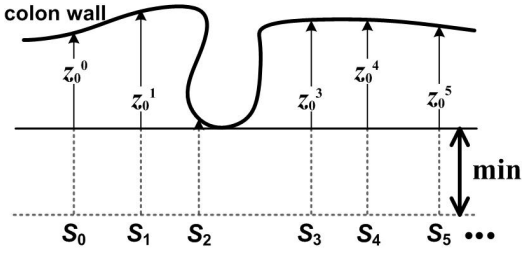


Fig. 4. Computing z-axis scaling factors from a height field. For simplicity, we depict the figure when the R_0^k.

3 Experimental Results

Our method is implemented on a PC equipped with Pentium™ IV 3.0GHz CPU and 1GB main memory. The volume dataset is obtained by scanning abdomens of five

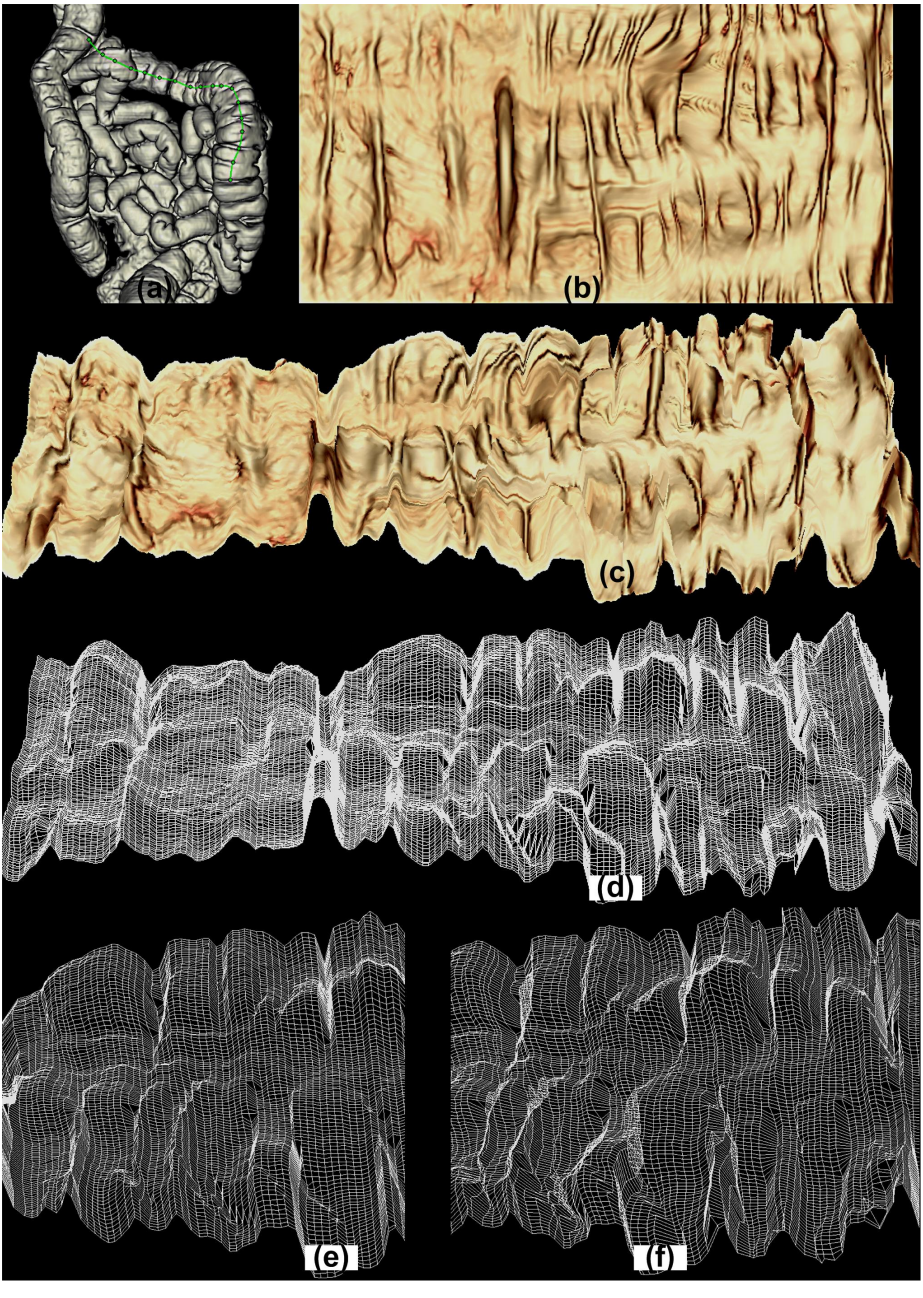


Fig. 5. Unfolding results produced by our method. (a) Outside view. (b) Before applying axis scaling. (c) After applying y- and z-axis scalings simultaneously. (d) Wire-frame model of (c). (e) and (f) Magnification of (d).

patients with a multi-detector CT. The number of sample points is 1972, 1320, 960, and 620, respectively. We call the datasets as *data A*, *data B*, *data C*, and *data D*, respectively. We exploit 360 rays with interval of one degree for one sample point. That is, the size of y-axis of a unfold image is 360. Table 1 shows the rendering time to get a final image when we apply our scaling scheme to our previous method [12]. In this case, additional time to scale y- and z-axis is below 70 milliseconds. In conventional method [9], although they scaled only 2D space without considering z-axis, about one minute or more is required because their approach performs lots of iterations. Therefore, interactive and intuitive rendering is difficult compared with our method. If we apply our z-axis scaling method to the Vilanova et al. method [9], the additional time to scale the z-axis is below 10 milliseconds. Therefore, we confirm that our method can be applied any variant of unfolding methods based on the volume ray casting.

Table 1. Comparison of the rendering time for generating unfold images along y- and z-scalings. The time for the scalings is included in the rendering time.

datasets	# of sample points	rendering time (sec)
data A	1972	0.99
data B	1320	0.95
data C	960	0.95
data D	620	1.55

Fig. 5 shows final rendering results using our previous method [12], y- and z-axis scaling method, and its wire-frame model, respectively, for the data C. The image resolution is 960×360 pixels. From wire-frame views, we can confirm that 3D surface models are normally reconstructed.

4 Conclusion

We propose an efficient unfolding method to inspect the inner surface of an organ. When a 2D unfold image is generated using volume ray casting, we store the distance values between a sample point on a central path and colon surface, for all rays in entire sample points. After generating a height field from the distance values, we reconstruct a 3D unfold model. Lastly, we map the 2D unfold image into the 3D model. Therefore, we make a virtual dissection model analogous to the real anatomical dissection. Our method is helpful for a quick diagnosis because it offers diverse viewing controls such as movement, rotation and magnification. In addition, our approach can be applied to any variant of 2D unfold rendering methods using volume ray casting.

Acknowledgment

This work was supported by INHA UNIVERSITY Research Grant.

References

1. Hong, L., Kaufman, A., Wei, Y., Viswambharan, A., Wax, M., Liang, Z.: 3D Virtual Colonoscopy. IEEE Symposium on Biomedical Visualization (1995) 26-32
2. Hong, L., Muraki, S., Kaufman, A., Bartz, D., He, T.: Virtual Voyage: Interactive Navigation in the Human Colon. Proc. SIGGRAPH 97 (1997) 27-34
3. Vos, F., Gelder, R., Serlie, I., Florie, J., Nio, C., Glas, A., Post, F., Truyen, R., Gerritsen, F., Stoker, J.: Three-dimensional Display Modes for CT Colonography: Conventional 3D Virtual Colonoscopy versus Unfolded Cube Projection, Journal of Radiology, Vol. 228 (2003) 875-885
4. Wang, G., McFarland, E.G., Brown, B.P., Vannier, M.W.: GI Tract Unraveling with Curved Cross-Sections. IEEE Transactions on Medical Imaging, Vol. 17, No. 2 (1998) 318-322
5. Wang, G., Dave, S.B., Brown, B.P., Zhang, Z., McFarland, E.G., Haller, J.W., Vannier, M.W.: Colon Unraveling based on Electrical Field: Recent Progress and Further Work. Proc. SPIE, Vol. 3660 (1999) 125-132
6. Haker, S., Angenent, S., Tannenbaum, A., Kikinis, R.: Nondistorting Flattening Maps and the 3-D Visualization of Colon CT Images. IEEE Transactions on Bio-medical Engineering, Vol.19, No.7 (2000) 665-671
7. Haker, S., Angenent, S., Tannenbaum, A., and Kikinis, R.: Nondistorting Flattening for Virtual Colonoscopy. Proc. MICCAI 2000 (2000) 358-366
8. Vilanova, A., Wegenkittl, R., König, A., Gröller, E., Sorantin, E.: Virtual Colon Flattening. Proc. VisSym 2001 Joint Eurographics – IEEE TCVG Symposium on Visualization (2001) 127-136
9. Vilanova, A., Wegenkittl, R., König, A., Gröller, E.: Nonlinear Virtual Colon Unfolding. Proc. IEEE Visualization 2001 (2001) 411-579
10. Balogh, E., Sorantin, E., Nyúl, L.G., Palágyi, K., Kuba, A., Werkgartner, G., Spuller, E.: Virtual Dissection of the Colon: Technique and First Experiments with Artificial and Cadaveric Phantoms. Proc. SPIE, Vol. 4681 (2002) 713-721
11. Zhu, L., Haker, S., Tannenbaum, A.: Flattening Maps for the Visualization of Multibranched Vessels. IEEE Transactions on Medical Imaging, Vol. 24, No. 2 (2005) 191-206
12. Lee, H., Lim, S., Shin, B.: Unfolding of Virtual Endoscopy using Ray-Template. Lecture Notes in Bioinformatics, Vol. 3745 (2005) 72-81
13. Levoy, M.: Display of Surface from Volume Data. IEEE Computer Graphics and Applications, Vol. 8, No. 3 (1988) 29-37
14. Kaufman, A.: Volume Visualization. 1st ed., Ed. IEEE Computer Society Press (1991)
15. Krishnamurthy, B., Bajaj, C.: Data Visualization Techniques. 1st ed., Ed. John Wiley & Sons (1999)
16. Lim, S., Shin, B.S.: RPO: A Reverse-Phased Hierarchical Min-Max Octree for Efficient Space-Leaping. Proc. Pacific Graphics 2005 (2005) 145-147
17. Lim, S., Shin, B.S.: Efficient Space-Leaping Using Optimal Block Sets. IEICE Trans. on Information and Systems, Vol. E88-D (2005) 2864-2870

Spectral Sequencing Based on Graph Distance

Rong Liu, Hao Zhang, and Oliver van Kaick

GrUVi Lab, School of Computing Sciences,
Simon Fraser University, Burnaby, British Columbia, Canada
{lrong, haoz, ovankaic}@cs.sfu.ca

Abstract. The construction of linear mesh layouts has found various applications, such as implicit mesh filtering and mesh streaming, where a variety of layout quality criteria, e.g., span and width, can be considered. While spectral sequencing, derived from the Fiedler vector, is one of the best-known heuristics for minimizing width, it does not perform as well as the Cuthill-Mckee (CM) scheme in terms of span. In this paper, we treat optimal mesh layout generation as a problem of preserving graph distances and propose to use the subdominant eigenvector of a kernel (affinity) matrix for sequencing. Despite the non-sparsity of the affinity operators we use, the layouts can be computed efficiently for large meshes through subsampling and eigenvector extrapolation. Our experiments show that the new sequences obtained outperform those derived from the Fiedler vector, in terms of spans, and those obtained from CM, in terms of widths and other important quality criteria. Therefore, in applications where several such quality criteria can influence algorithm performance simultaneously, e.g., mesh streaming and implicit mesh filtering, the new mesh layouts could potentially provide a better trade-off.

1 Introduction

Computing linear mesh layouts is an instance of the graph layout problem [1], where an optimal labeling of the vertices of a given graph is sought. Many optimization problems, including sparse matrix reordering [2,3,4], circuit layout [5], DNA sequencing [6], and ranking [7] , are formulated as graph layout problems.

Consider a weighted graph $G = (V, E, w)$ with $V = \{v_1, \ldots, v_n\}$ the set of vertices, E the set of edges, and $w : E \rightarrow \mathbb{R}$ the edge weights. A (linear) *layout* of G is a labeling π of its vertices, $\pi : V \rightarrow \{1, 2, \ldots, n\}$. For a real number $0 < p < \infty$, the *p-discrepancy* [7] of G with respect to a layout π is defined as $\sigma_p(G, \pi) = \left(\sum_{uv \in E} w_{uv} |\pi(u) - \pi(v)|^p\right)^{1/p}$. If $p = \infty$, then $\sigma_\infty(G, \pi) = \max_{uv \in E} |\pi(v) - \pi(v)|$, and is also called the *bandwidth* of the layout. The minimum value $\sigma_p(G) = \min_\pi \sigma_p(G, \pi), 0 < p \leq \infty$, is called the *min-p-sum* of the graph G. Another important layout cost measure is *vertex separation* [1], defined as $\max_{1 \leq i \leq n} |\{\pi(u) \leq i : \exists \pi(v) > i, uv \in E\}|$. Intuitively, it measures, at a certain point of the linear layout, the number of edges for which only one end vertex has been encountered. In the field of numerical analysis, various measures, such as bandwidth [2], *profile* or *envelope size* [3] and *workbound* [4], are

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 630–636, 2006.
© Springer-Verlag Berlin Heidelberg 2006

considered for sparse matrix reordering. It turns out that these measures are related to different p-discrepancies of a graph layout, where the matrix of interest can be considered as the adjacency matrix of the graph G.

Several problems in geometry processing [8,9] benefit from having an optimized mesh layout. A good example is mesh streaming [10], where the *span* and *width*, corresponding to bandwidth and vertex separation, respectively, heavily influence the mesh streamability. Profile, workbound, and bandwidth all affect the cost of Cholesky factorization [11], which becomes necessary when large sparse linear systems for the implicit mesh filtering problem need to be solved [9].

The optimization problems associated with most of the criteria mentioned so far are NP-hard [1]. In practice, one resorts to efficient heuristics. Two best known heuristics for minimizing span (bandwidth) and having comparable performances are the Cuthill-Mckee (CM) [2] and minimum degree schemes [11], both essentially conduct a degree-oriented breadth-first search. For the other costs mentioned so far, spectral sequencing using the Fiedler vector, which is the eigenvector corresponding to the smallest non-zero eigenvalues of the graph Laplacian matrix, has been quite successful empirically [3,7]. Note that the graph Laplacian is derived from the graph's adjacency matrix.

In this paper, we propose a new spectral sequencing operator for mesh layout generation based on *graph distance*. The sequences obtained using our algorithm outperform those derived from the Fiedler vector, in terms of spans, and sequences obtained from CM, in terms of widths, profiles, workbounds, and a few other important layout quality criteria. Thus in applications where several such criteria can influence algorithm performance simultaneously, e.g., mesh streaming and implicit mesh filtering, the new mesh layouts could potentially provide a better trade-off. Although the matrix we eigendecompose is in general non-sparse, approximate eigenvector computation via subsampling and Nyström method [12] allows us to compute layouts for large meshes efficiently.

The rest of the paper is organized as follows. In Section 2, we describe our spectral sequencing algorithm. Various practical issues, such as subsampling for efficient layout computation, are addressed in Section 3. Experimental results are given in Section 4. Finally, we conclude and suggest possible future work.

2 Spectral Sequencing for Graph Layout

Although we focus on mesh layouts, our discussions are cast in the general context of graph layout. To produce a sequence of mesh vertices or faces, the graph of interest can be chosen as the primal or dual graph of the mesh, respectively. For the mesh layout problem we currently consider, edge weights will be assumed to be unit. Our approach however can trivially adapt to weighted graphs.

Independent of any specific layout costs, observe that if two graph vertices are close to each other based on connectivity, they should also be close to each other in the sequence. Take the *min-1-sum* problem for example. If all the adjacent vertices in the graph were consecutive in the sequence, the closest possible in 1-D, an optimal layout would be obtained. This is of course not possible in 1-D

in general, but we can look for an embedding ϕ of the graph vertices in a high dimensional space, in which adjacent vertices are always close to each other. To obtain a sequence, the vertices are projected onto a properly chosen vector, so that their mutual distances are preserved as mush as possible. Given a graph $G = (V, E)$ with n vertices, the following is the overview of our algorithm:

1. Calculate the graph distance, $g(v_i, v_j)$, between each pair of vertices.
2. Compute an embedding $\phi(V) = \{\phi(v_i)\}$ of V, such that given any vertex v, $\forall(i, j), ||\phi(v) - \phi(v_i)||^2 < ||\phi(v) - \phi(v_j)||^2$, if and only if $g(v, v_i) < g(v, v_j)$.
3. Project points in $\phi(V)$ onto a vector $\mathbf{p}^*$, along which their relative positions are preserved as much as possible.
4. Sort the projections of $\phi(V)$ on $\mathbf{p}^*$ to obtain the sequencing of the vertices.

2.1 Graph Distance and the Kernel Matrix

To find the embedding $\phi(V)$, we resort to relevant concepts from Kernel PCA [13]. Imagine a mapping $\phi : V \to \mathcal{F}, v \mapsto \phi(v)$, where $\mathcal{F}$ is typically called the *feature space* which may have a very high, and possibly infinite, dimensionality. For this reason, ϕ is never explicitly used and instead, it is implicitly defined by a kernel (affinity) matrix K, with $K_{ij} = k(v_i, v_j) = \langle\phi(v_i), \phi(v_j)\rangle$. Thus if K is known, the embedding $\phi(V)$ is implicitly induced. To define K, we first form a distance matrix W, where $W_{ij} = g(v_i, v_j)$ is the graph distance between the vertices.

The next step is to convert W into the kernel matrix K, using a certain kernel function. One of the most popular kernels is the Gaussian radial basis function,

$$k(v_i, v_j) = \exp\left(W_{ij}^2/2\delta^2\right), \text{where } \delta \text{ is the kernel width.} \tag{1}$$

In this case, it can be shown [14] that the square distance between two vertices i and j in the feature space $\mathcal{F}$ is $\mathcal{W}_{ij}^2 = ||\phi(v_i) - \phi(v_j)||^2 = 2 - 2k(v_i, v_j)$. Thus distances in $\mathcal{F}$ are seen to be proportional to those in the graph, in that the order among distances between vertices in the graph is preserved in the feature space. Specifically, since the graph edges have unit length, neighboring vertices in the graph are always closest to each other in the feature space $\mathcal{F}$.

2.2 Sequencing Via Spectral Embedding

The positioning of the embeddings $\phi(V)$ provides a good start to extract a vertex sequence since the distances between pairs of vertices in the graph are relatively preserved in $\mathcal{F}$. We simply project each $\phi(v)$ onto a vector $\mathbf{p}$. Afterwards, the projections are sorted to obtain the sequence. Since after the projection the dimensionality of the embedding space is reduced to 1, to obtain a smaller layout cost for the resulting sequence, it is desirable to preserve their mutual distances, hence the relative positions, as much as possible. To this end, we choose the optimal direction $\mathbf{p}^*$ subject to the following objective function

$$\mathbf{p}^* = \underset{\mathbf{p}\in\mathbb{R}^{dim(\mathcal{F})}, ||\mathbf{p}||=1}{\operatorname{argmax}} \sum_{i<j} ||\mathbf{p}^T(\phi(v_i) - \phi(v_j))||^2,$$

To motivate this approach, let us note that as the projection of distances onto $\mathbf{p}$ is always smaller than the corresponding original distances, maximizing the

sum of projected distances tends to preserve the original distances. Assume that $\phi(V)$ is centered around the origin, it can be proven that $\mathbf{p}^*$ is the first principal component of the point set $\phi(V)$. Denote by $\phi(V)_{\mathbf{p}^*}$ the projection of $\phi(V)$ onto $\mathbf{p}^*$, it can be demonstrated that the sequence obtained from $\phi(V)_{\mathbf{p}^*}$ is identical to that obtained from $\mathbf{U}_1$, the largest eigenvector of the kernel matrix K. Therefore the algorithm simply sorts the elements of $\mathbf{U}_1$ to produce the sequence for V for the layout problem. More details on these and our subsequent discussions can be found in an extended version of this paper in [14].

3 Some Practical Issues

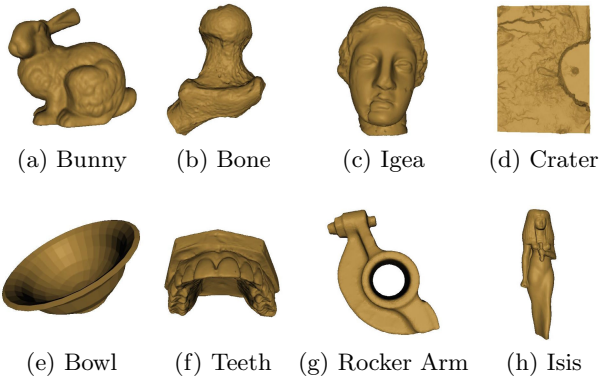
Subsampling: The algorithm described so far is unable to handle large meshes due to overhead caused by pair-wise distance computation required by W and the eigenvalue decomposition of K, giving rise to a time complexity of at least $O(n^2 \log n)$. To overcome this complexity, we adopt subsampling and the Nyström method [15], which allows for computing only the distances between m sampled vertices to the remaining vertices and approximating the leading eigenvectors of K by knowing only m rows of it. Throughout this paper, we fix the number of samples to 10, reducing the overall complexity to $O(n \log n)$. With farthest point sampling [15], our experiments demonstrate that Nyström approximation works remarkably well, even at an extremely low sampling rate.

Centering of K: Till now we have assumed that the set $\phi(V)$ is centered at the origin, which is not true in general. Thus, the kernel K has to be first centered [13] to obtain $\bar{K}$, where $\bar{K} := (I - \frac{1}{n}\mathbf{1}\mathbf{1}^T)K(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T)$ and $\mathbf{1}$ is the column vector of 1's. However, this is inefficient for large K. At the same time, due to subsampling, the full K is unavailable for centering. We thus propose to use the *subdominant eigenvector* of the un-centered K, which can be both formally and experimentally shown [14] to be a good approximation to the dominant eigenvector of $\bar{K}$, as long as the kernel width δ used in (1) is sufficiently large. In our experiments, we choose δ as the average of all the sampled graph distances.

Choice of kernel function: To convert a distance matrix to a kernel matrix, we use the Gaussian kernel, given in equation (1). Note that other kernels, e.g., step function, exponential, polynomial and rational polynomial kernels, are also possible. In fact, the use of the Fiedler vector can be seen as a special case of our general paradigm, when applied to regular graphs, since the eigenvectors of the graph Laplacian coincide with the eigenvectors of K derived from distance matrix W with a step function kernel of width 1. We have experimented with other kernels and have not found particular reasons to prefer one over the other, but this issue requires further investigation.

4 Experimental Results

This section presents an experimental comparison which evaluates the quality of the sequence generated by the Fiedler vector, the Cuthill-Mckee scheme and



(a) Bunny (b) Bone (c) Igea (d) Crater

(e) Bowl (f) Teeth (g) Rocker Arm (h) Isis

Fig. 1. Models used in the experiments

Table 1. Characteristics of models and timing in seconds, I/O excluded

Model	Bunny	Bone	Igea	Crater	Bowl	Teeth	Rocker Arm	Isis
Vertex #	34,834	50,002	60,002	100,000	102,402	100,002	160,704	187,644
Time	2.00	3.14	4.20	7.31	7.45	7.58	15.50	23.80

our approach, referred to as Laplacian, CM, and Affinity, respectively. Given a mesh, its vertex or face sequence is generated by considering the primal or the dual graph, respectively. In our experiments, we consider six quality measures of the sequence: span, width, profile, workbound, 1-discrepancy and 2-discrepancy, where the first two are of particular interest to mesh streaming [10].

Figure 1 shows the triangle meshes used in our experiments, which are performed on a Pentium 1.7GHz processor with 1GB RAM. Note that models with boundary (Crater), elongated aspect ratio (Isis) and non-zero genus (Rocker Arm) are all tested. Table 1 presents the characteristics of these models in conjunction with the timing of our algorithm. Note that CM works the fastest among the three, while Affinity and Laplacian perform similarly in speed.

Figure 2 shows the comparison between the six layout measures for the primal graphs; more experimental results can be found in [14]. It can be seen that CM obtains the best results in terms of span, while the Laplacian operator generally provides the best results in terms of the other measures. However, since the Affinity operator outperforms the Laplacian in span and CM in width, it achieves a trade-off between these two sets of measures, thus providing potential benefits to applications where all these measures influence performance. Since the models we use have different structural properties, we believe our result is model independent. We have also experimented on dual graphs (for face sequencing) and found that Affinity tends to outperform both CM and Laplacian on regular graphs [14]. This issue is currently under investigation.

Note that in all our experiments, the sample size is fixed at 10, which is as small as 0.0053% of the mesh size. We also conduct experiments to compare between using and not using subsampling and report the results in [14].

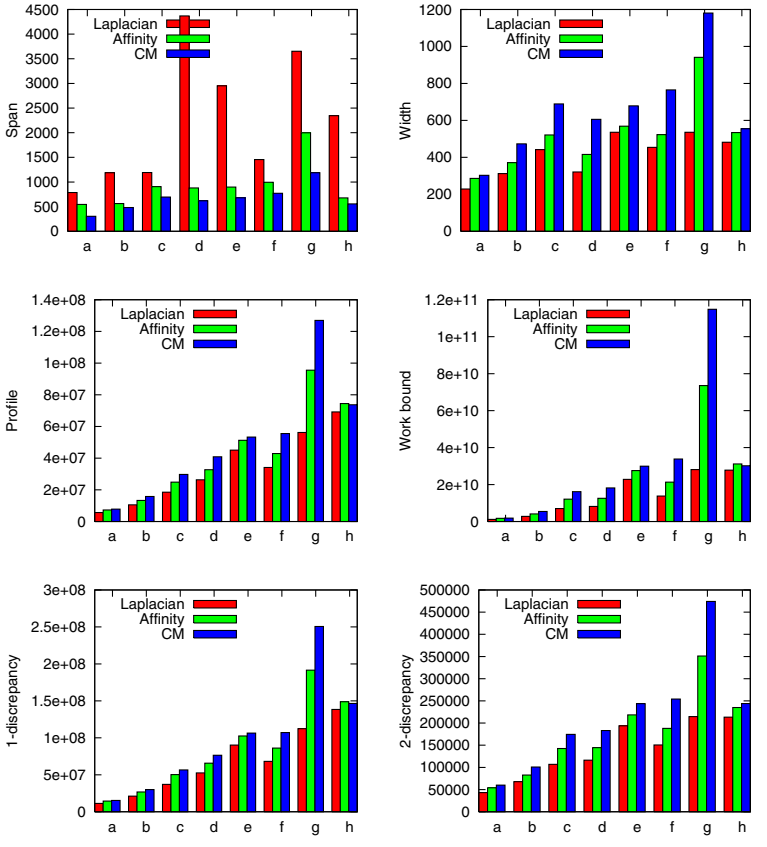


Fig. 2. Comparison of layout quality measures for primal graph of different models: (a) Bunny (b) Bone, (c) Igea, (d) Crater, (e) Bowl, (f) Teeth, (g) Rocker Arm, (h) Isis

5 Conclusion and Future Work

In this paper, we present a spectral sequencing algorithm for linear mesh layout. We study this problem in the more general context of linear graph layout. Independent of any specific cost measure, we abstract that a desirable vertex sequence should be one in which close-by vertices in the graph are also close-by in the sequence. To this end, a high dimensional distance preserving embedding and 1D projection are applied to extract the sequence, with the help of Kernel PCA. To make the algorithm work efficiently for large meshes, several practical issues are addressed, including subsampling, eigenvector extrapolation, and the usage of the subdominant eigenvector of the un-centered kernel matrix.

Our experiments demonstrate that for span and width, the two principal quality measures for mesh streaming, as well as other graph layout cost criteria, e.g., profile and workbound, our algorithm potentially provides a better trade-off compared to ordering schemes based on localized graph traversal, e.g., Cuthill-Mckee, and spectral sequencing using the Fiedler vector.

The geometric appeal of our framework and analysis based on Kernel PCA has shed some light on possible further improvement of layout qualities. One particularly intriguing problem is to investigate the distribution properties of the vertex embedding in the feature space, so as to achieve better results in both span and width. It is also interesting to investigate the role of kernel functions in the layout result, as well as more sophisticated distance measures between graph vertices that take into consideration more global mesh connectivity information.

References

1. Díaz, J., Petit, J., Serna, M.: A survey of graph layout problems. ACM Computing Survey **34**(3) (2002) 313–356
2. Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: Proc. 24th Nat. Conf. ACM. (1969) 157–172
3. Barnard, S.T., Pothen, A., Simon, H.D.: A spectral algorithm for envelope reduction of sparse matrices. In: Proc. ACM/IEEE Conference on Supercomputing. (1993) 493–502
4. Corso, G.M.D., Romani, F.: Heuristic spectral techniques for the reduction of bandwidth and work-bound of sparse matrices. Technical report, Universitá di Pisa, Dipartimento di Informatica (2001)
5. Hur, S.W., Willis, J.: Relaxation and clustering in a local search framework: application to linear placement. In: Proc. ACM/IEEE Conference on Design Automation. (1999) 360–366
6. Karp, R.M.: Mapping the genome: some combinatorial problems arising in molecular biology. In: Proc. ACM Symposium on Theory of Computing. (1993) 278–285
7. Mohar, B., Poljak, S.: Eigenvalues in combinatorial optimization. In Brualdi, R.A., Friedland, S., Klee, V., eds.: IMA Volumes in Mathematics and Its Applications. Volume 50. Springer-Verlag (1993) 107–151
8. Isenburg, M., Lindstrom, P., Gumhold, S., Snoeyink, J.: Large mesh simplification using processing sequences. In: Proc. of the 14th IEEE Visualization. (2003) 61–68
9. Zhang, H., Fiume, E.: Butterworth filtering and implicit fairing of irregular meshes. In: Proceedings of Pacific Graphics. (2003) 502–506
10. Isenburg, M., Lindstrom, P.: Streaming meshes. In: IEEE Visualization. (2005)
11. George, A., Liu, W.H.: Computer Solution of Large Sparse Positive Definite Systems. Prentice-Hall (1981)
12. Press, W., Tekolsky, S., Vetterling, W., Flannery, B.: Numerical Recipies in C. Cambridge Univ. Press (1992)
13. Schölkopf, B., Smola, A., Muller, K.R.: Nonlinear component analysis as a kernel eigenvalue problem. Neural Computation **10** (1998) 1299–1319
14. Liu, R., Zhang, H., van Kaick, O.: An investigation on spectral sequencing using graph distance. Technical Report TR 2006-08, School of Computing Science, Simon Fraser University (2006)
15. Liu, R., Jain, V., Zhang, H.: Subsampling for efficient spectral mesh processing. In: Proceedings of Computer Graphics International 2006 (to appear). (2006)

An Efficient Implementation of RBF-Based Progressive Point-Sampled Geometry

Yong-Jin Liu[1], Kai Tang[2], and Joneja Ajay[2]

[1] Tsinghua University, Beijing, P.R. China
[2] The Hong Kong University of Science and Technology, Hong Kong

Abstract. In this paper we address practical implementation issues of generating a progressive representation of point-sampled geometry. Fast and stable algorithms are proposed to efficiently implement a progressive version of the modified RBF Shepard's method. Comparisons with several well-known methods are presented, showing that the proposed algorithms can achieve good balance between geometric error reduction and time efficiency.

1 Introduction

Dense point sets sampled from surfaces of physical objects have become a new representation of geometric models [2,10,14]. To sufficiently describe the geometric details, the sampled point data is very large, usually among $10^4 \sim 10^8$. Inspired by the successful applications of level-of-details techniques in computer graphics [5], arranging dense point data into a progressive version is desired in many downstream graphics and computer-aided design applications. In this paper we report an implementation of a progressive model with the radial basis function (RBF) method [6]. We also compare our RBF version of progressive representation with several well-known methods including the moving least squares (MLS) methods. Experimental results are presented with conclusions.

2 Preliminaries

A radial function is $\Phi(x) = \phi(\|x\|_2), x \in \mathbb{R}^d$. Some Φ are conditionally positive definite of order M on $\mathbb{R}^d$ and the overall interpolation function f is

$$f(x) = \sum_{j=1}^{n} u_j \Phi(x - p_j) + \sum_{l=1}^{m} \lambda_l b_l(x) \tag{1}$$

with coefficients $u_1, \cdots, u_n, \lambda_1, \cdots, \lambda_m, m = \dim \mathbb{P}_M^d$ which can be determined by satisfying the interpolating constraints $f(p_i) = c_i, \forall p_i \in \mathcal{P}, 1 \le i \le n$; i.e.,

$$\begin{pmatrix} Q_{n\times n} & B_{n\times m} \\ B^T & \mathbf{0}_{m\times m} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{0} \end{pmatrix} \tag{2}$$

where $Q_{ij} = \Phi(p_i - p_j), B_{ij} = b_j(p_i), \mathbf{u} = (u_1, \cdots, u_n)^T, \boldsymbol{\lambda} = (\lambda_1, \cdots, \lambda_m)^T$ and $\mathbf{f} = (c_1, \cdots, c_n)^T$.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 637–643, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Theorem 1. [8] *The solution to the* $(n+m)\times(n+m)$ *linear system (2) can be obtained by solving a smaller* $(n-m)\times(n-m)$ *linear system.*

The computational cost with RBF can be reduced by introducing the partition of unity method; this leads to the modified RBF Shepard's method [6]. Let

$$f_k(x) = \sum_{j\in I_k} u_j^k \Phi(x-p_j) + \sum_{l=1}^{m} \lambda_l^k b_l(x) \tag{3}$$

be a RBF interpolant relative to the subset $\mathcal{P}_k = \{p_j \in \mathcal{P},\ j \in I_k\}$, I_k being a set of indexes of N_R neighbors of p_k, the modified RBF Shepard's function is

$$F(x) = \sum_{k=1}^{n} \widetilde{W}_k(x) f_k(x), \quad x \in \mathbb{R}^d, \tag{4}$$

$$W_k(x) = \left[\frac{(r_{W_k} - r_k)_+}{r_{W_k} r_k}\right]^2 \quad \text{and} \quad \widetilde{W}_k(x) = W_k(x) / \sum_{i=1}^{n} W_k(x) \tag{5}$$

where $r_k = \|x - p_k\|$ and r_{W_k} is the radius of influence about the point p_k chosen just large enough to include N_W neighboring points.

3 Progressive Model

Given a point data $\mathcal{P} = \{p_1, \cdots, p_n\}$, where each point p_i is associated with a normal n_i, a continuous surface $S(\mathcal{P})$ can be inferred from $F(x) = 0$ (ref. eq.(4)), where $x \in S(\mathcal{P})$. Given $\mathcal{P}$ and a subset $\mathcal{P}' \subset \mathcal{P}$, two surfaces $S(\mathcal{P})$ and $S(\mathcal{P}')$ can be inferred. Since the point data $\mathcal{P}$ is dense and sufficiently sampled, we measure the difference between $S(\mathcal{P})$ and $S(\mathcal{P}')$ with the following q-norm:

$$\|S(\mathcal{P}) - S(\mathcal{P}')\|_q = \|\mathcal{P}'\|_q = \left(\sum_{p_i \in \mathcal{P}\setminus\mathcal{P}'} |dist(p_i, S(\mathcal{P}'))|^q\right)^{\frac{1}{q}} \tag{6}$$

where $dist(p_i, S(\mathcal{P}'))$ is the shortest Euclidean distance between p_i and $S(\mathcal{P}')$. To quickly compute $dist(p_i, S(\mathcal{P}'))$, we find the k nearest neighbors $\Xi_{KNN}(x,k)$ of x in $\mathcal{P}$. $\forall p_j \in \Xi_{KNN}(x,k)$, the signed geometric distance $geom_dist(x,p_j) = (x-p_j)\cdot n_j$ is calculated. The first order approximation can be written as

$$\texttt{geom_dist}(x, S(\mathcal{P})) = \min_j \{geom_dist(x,p_j) \mid p_j \in \Xi_{KNN}(x,k)\} \tag{7}$$

Theorem 2. [1] *If* $\|\cdot\|_\alpha$ *and* $\|\cdot\|_\beta$, $\alpha, \beta \geq 1$, *are two arbitrary norms defined by eq.(6), then* $\forall \mathcal{P}' \subseteq \mathcal{P}$, *we have*

$$\frac{1}{\sqrt[\alpha]{m}}\|\mathcal{P}'\|_\alpha \leq \|\mathcal{P}'\|_\beta \leq \sqrt[\beta]{m}\|\mathcal{P}'\|_\alpha$$

where $m = \#(\mathcal{P}\setminus\mathcal{P}')$ *is the number of elements in the set* $\mathcal{P}\setminus\mathcal{P}'$.

[1] Due to page limit, all the proofs in this paper are omitted; they are available by sending request to the first author.

Since by Theorem 2 all the norms are equivalent in a topological sense, we are interested in the *uniform* (or *Chebyshev*) norm $p = \infty$:

$$\|S(\mathcal{P}) - S(\mathcal{P}')\|_\infty = \max_{p_i \in \mathcal{P} \setminus \mathcal{P}'} |geom_dist(p_i, S(\mathcal{P}'))| \tag{8}$$

Given a data $\mathcal{P}$, a subset $\mathcal{P}'$ and the metric (8), a point $p_k \in \mathcal{P}'$ is called the *finest detail point* in $\mathcal{P}'$ if it satisfies

$$p_k = \arg\min_{p_i} \{\|S(\mathcal{P}) - S(\mathcal{P}' \setminus p_i)\|_\infty,\ \forall p_i \in \mathcal{P}'\}$$

If we start from the original data $\mathcal{P}$ ($\mathcal{P}' = \mathcal{P}$) and recursively extract the finest detail points d_i from $\mathcal{P}'$, a sorted set $\mathcal{D} = \{\cdots, d_i, d_{i+1}, \cdots\}$ is obtained. Let the process be terminated at certain subset $\mathcal{P}_0$. Then $\mathcal{P} = (\mathcal{P}_0, \mathcal{D})$, where $\mathcal{P}_0$ encapsulates a basic shape of $S(\mathcal{P})$ and $\mathcal{D}$ encodes the detail parts of $S(\mathcal{P})$.

4 The Main Algorithms

The following operations are frequently used in point data manipulation:

1. Building a multi-attribute index $\mathcal{I}$ in the main memory;
2. Output elements in $\mathcal{I}$ within a given range;
3. Insert a new element in $\mathcal{I}$;
4. Delete an existing element in $\mathcal{I}$;
5. Update the position of an element in $\mathcal{I}$ with one or more changed attributes.

The second operation is necessary since most of the existing practical solutions for k-nearest-neighbors (KNN) queries are built systematically over range searching techniques [3]. To solve range searching problems, two well-known data structures are examined in literature [4]: kd-trees and range-trees. We choose range tree as our fundamental data structure due to its faster query time.

Lemma 1. *The operations,* `insert`, `delete` *and* `update`*, on an element in a d-dimensional range tree, can be performed in $O(\log^{d-1} n)$ amortized time, where n is the maximum number of elements that have been in the tree.*

The pseudo-codes of all proposed algorithms are listed in Fig. 1. By using Algorithms 1-3, a well defined continuous surface that interpolates $\mathcal{P}$ can be determined by tracing all points in space that have the same scalar field value.

Theorem 3. *Given a point set $\mathcal{P} \in \mathbb{R}^d$ with n elements, the set $\mathcal{P}$ can be preprocessed in $O(n \log^{d-1} n)$ time using $O(n \log^{d-1} n)$ storage and, given an arbitrary location $x \in \mathbb{R}^d$, its scalar field value determined by Algorithm 3 can be reported in $O(\log^{d-1} n + k)$ time, where k is the number of points output from a range search in $\mathcal{P}$ with radius $maxr_W$.*

The progressive model presented in Section 3 recursively extracts the finest detail points from $\mathcal{P}$ to obtain a sorted set $\mathcal{D}$. To efficiently determine the finest detail points from $\mathcal{P}$, $\forall p_i \in \mathcal{P}', \mathcal{P}' \subset \mathcal{P}$, we calculate the difference $\|S(\mathcal{P}) - S(\mathcal{P}' \setminus p_i)\|_\infty$

Algorithm 1. k_nearest_neighbors_query(p, k, $I(P)$)
Input. An integer k>0, a point p and a range tree $I(P)$ storing a point set P.
Output. The k nearest neighbors Ξ_{KNN} of p in P stored in ascending order keyed on the value of distance to p, and the minimal radius r with which the sphere centered at p bounds the k nearest neighbors.
1. $r \leftarrow$ 1.5 times minimum distance among any two points in P (pre-computed);
2. $a \leftarrow$ 1.1;
3. $\Gamma \leftarrow$ **range_query**(p, r, $I(P)$);
4. **while** the number of elements in $\Gamma(p, r, I(P))$ is less than k **do**
4.1. $\Gamma \leftarrow$ **range_query**(p, $r = ar$, $I(P)$);
5. Sort the set $\Gamma(p, r, I(P))$ in ascending order keyed on the distance to p;
6. $r \leftarrow d(p, p_k)$, where p_k is the kth elements in the sorted set Γ;
7. Report r and the first k elements in the sorted set Γ;

Algorithm 2. point_data_preprocess(P)
Input. A set P of points.
Output. The maximal radii $maxr$ and $maxr_W$, and $\forall p_k \in P$, the radii r_{Rk} and r_{Wk}, the subset index I_k and the coefficients u^k, λ^k as indicated in eqs. (3-5).
1. $maxr_W = maxr = 0$;
2. Arrange P into a range tree $I(P)$;
3. **for** every point p_k in P **do**
3.1. **if** $N_W > N_R$
3.1.1. $(r_{Wk}, \Xi_{KNN}) \leftarrow$ **k_nearest_neighbors_query**(p_k, N_W, $I(P)$);
3.1.2. Truncate Ξ_{KNN} with the first N_R elements;
3.1.3. $r_{Rk} = \|p_k - \Xi(N_R)\|$, where $\Xi(N_R)$ is the N_R-th elements in Ξ_{KNN};
3.2. **else**
3.2.1. $(r_{Rk}, \Xi_{KNN}) \leftarrow$ **k_nearest_neighbors_query**(p_k, N_R, $I(P)$);
3.2.2. $r_{Wk} = \|p_k - \Xi(N_W)\|$, where $\Xi(N_W)$ is the N_W-th elements in Ξ_{KNN};
3.2.3. **if** $r_{Rk} > maxr$
3.2.3.1. $maxr = r_{Rk}$;
3.3. **if** $r_{Wk} > maxr_W$
3.3.1. $maxr_W = r_{Wk}$;
3.4. **for** every point p_k in Ξ_{KNN} **do**
3.4.1. Generate two off-surface points $p_{ki1,2} = p_{ki} \pm an_{ki}$;
3.5. Calculate the coefficients u^k_i, λ^k_i in equation (3);

Algorithm 3. scalar_field_value(x, P)
Input. An arbitrary location $x \in R^d$ and a set P after **point_data_preprocess(P)**.
Output. The scalar field value $F(x)$ determined by equation (4).
1. $\Gamma \leftarrow$ **range_query**(x, $maxr_W$, $I(P)$);
2. **if** the set $\Gamma(x, maxr_W, I(P))$ is null
2.1. return ∞;
3. **else**
3.1. $S_R = S_W = 0$;
3.2. flag = false;
3.3. **for** each $p_k \in \Gamma(x, maxr_W, I(P))$ **do**
3.3.1. $r_k = \|x - p_k\|$;
3.3.2. **if** $r_k = 0$
3.3.2.1. return c_k; // since $F(x) = f_k(x) = f_k(p_k) = c_k$ by eqs. (3, 4)
3.3.3. **if** $r_{Wk} > r_k$
3.3.3.1. flag = true;
3.3.3.2. $S_W = S_W + W_k(x)$; // implementing equation (5)
3.3.3.3. $S_R = S_R + W_k(x) f_k(x)$; // implementing equation (4)
3.3.4. **if** flag = true
3.3.4.1. return S_R/S_W; // implementing equation (4)
3.3.5. **else**
3.3.5.1. return ∞;

Algorithm 4. key_value_evaluation(p, $I(P')$, $I(D)$, *diff*)
Input. A point p in a subset $P' \subset P$, a complement subset $D = P \backslash P'$ and the difference value *diff* of $\|S(P) - S(P')\|_\infty$.
Output. The value of the difference $\|S(P) - S(P' \backslash p_i)\|_\infty$.
1. Delete p from $I(P')$;
2. Insert p into $I(D)$;
3. $D' \leftarrow \varnothing$;
4. $\Gamma \leftarrow$ **range_query**(p, $maxr$, $I(P')$);
5. **for** each $p_k \in \Gamma(p, maxr, I(P'))$ **do**
5.1. **if** $\|p - p_k\| < max(r_{Rk}, r_{Wk})$
5.1.1. $\Gamma \leftarrow$ **range_query**(p, $maxr_W$, $I(D)$);
5.1.2. $D' \leftarrow D' \cup \Gamma$;
5.1.3. Backup the original RBF data associated to p_k;
5.1.4. **if** $N_W < N_R$
5.1.4.1. $(r_{Wk}, \Xi_{KNN}) \leftarrow$ **k_nearest_neighbors_query**(p_k, N_W, $I(P')$);
5.1.4.2. $r_{Rk} = \|p_k - \Xi(N_R)\|$;
5.1.5. **else**
5.1.5.1. $(r_{Rk}, \Xi_{KNN}) \leftarrow$ **k_nearest_neighbors_query**(p_k, N_R, $I(P')$);
5.1.5.2. $r_{Wk} = \|p_k - \Xi(N_W)\|$;
5.1.6. **for** every point p_{ki} in Ξ_{KNN} **do**
5.1.6.1. Generate two off-surface points $p_{ki1,2} = p_{ki} \pm an_{ki}$;
5.1.7. Calculate the coefficients u^k_i, λ^k_i in equation (3);
6. $key_value(p) = diff$;
7. **for** every point p' in D' **do**
7.1. $c = \text{geom_dist}(p', S(P'))$;
7.2. **if** $key_value(p) < c$
7.2.1. $key_value(p) = c$;
8. **for** each $p_k \in \Gamma(p, maxr, I(P'))$ **do**
8.1. **if** $\|p - p_k\| < max(r_{Rk}, r_{Wk})$
8.1.1. restore the original RBF data associated to p_k;
9. Delete p from D and insert it back to P';
10. Report $key_value(p)$;

Algorithm 5. progressive_representation(P)
Input. A point set P.
Output. The progressive representation $PR = (P_0, D)$;
1. **point_data_preprocess(P)**;
2. $D \leftarrow \varnothing$;
3. $diff = 0$;
4. **for** every point p in P **do**
4.1. **key_value_evaluation(p, $I(P)$, $I(D)$, *diff*)**;
5. Sort P into a queue $Q(P)$ using the key values;
6. **while** P is not empty **do**
6.1. Extract the point tp (with the minimum key value) from the top of $Q(P)$;
6.2. $diff \leftarrow$ the key value associated with tp;
6.3. **if** $diff \leftarrow max_error_dist$ // max_error_dist is predefined
6.3.1. Delete tp from $Q(P)$ and $I(P)$, and insert it into $I(D)$;
6.3.2. $\Gamma \leftarrow$ **range_query**(tp, $maxr$, $I(P)$);
6.3.3. **for** every point p_k in $\Gamma(tp, maxr, I(P))$ **do**
6.3.3.1. **if** $\|tp - p_k\| < r_{Rk}$ or $\|tp - p_k\| < r_{Wk}$
6.3.3.1.1. Update r_{Rk}, r_{Wk}, $maxr$, $maxr_W$ and the RBF data with p_k;
6.3.4. **for** every point p_k in $\Gamma(tp, maxr, I(P))$ **do**
6.3.4.1. **if** $\|tp - p_k\| < r_{Rk}$ or $\|tp - p_k\| < r_{Wk}$
6.3.4.1.1. **key_value_evaluation(p_k, $I(P)$, $I(D)$, *diff*)**;
6.3.4.1.2. Update the position of p_k in $Q(P)$ with the new key value;
6.3.4.1.3. $\Gamma_k \leftarrow$ **range_query**(p_k, $maxr$, $I(P)$);
6.3.4.1.4. **for** every point p_k in $\Gamma_k(p_k, maxr, I(P))$ **do**
6.3.4.1.4.1. **if** $\|p_{ki} - p_k\| < r_{Rki}$ or $\|p_{ki} - p_k\| < r_{Wki}$
6.3.4.1.4.1.1. **key_value_evaluation(p_{ki}, $I(P)$, $I(D)$, *diff*)**;
6.3.4.1.4.1.2. Update the position of p_k in $Q(P)$;
6.4. **else**
6.4.1. return;

Fig. 1. The pseudo-codes of all proposed algorithms in this paper

using equations (7) and (8) and associate it to p_i as a key value. By sorting $\mathcal{P}'$ into a queue with the key values, the finest detail point in $\mathcal{P}'$ is readily obtained at the top of the queue with the minimal key value. Due to the local fitting property inherent in the partition of unity method, extracting one point p_i from the queue only locally affects key values in a small number of points that are neighbors of p_i. By reassigning the key values and updating the positions of these points in the queue, the recursive extraction of finest detail points is efficiently done.

Theorem 4. *The progressive representation of a point set $\mathcal{P}$ in $\mathbb{R}^d$ with the surface* `scalar_field_value`$(x, \mathcal{P}) = 0$ *can be generated by Algorithms 4-5 in $O(mn \log^{d-1} n + k_1 mn)$ time using $O(n \log^{d-1} n)$ storage, where $m = k_1 k_2 k_3 k_4$ and k_1, k_2, k_3, k_4 are strictly less than n and behave like small constants in all the experiments presented in Section 5.*

Fig. 2. Four selected models and their progressive sequences

5 Experimental Results

We implement the proposed algorithms as well as several well-known methods which are summarized below:

I. The proposed RBF implementation with a multiquadrics RBF [11];
II. A modified quadratic Shepard's method [12];
III. A globally supported multiquadratics RBF method [11];
IV. A compactly supported RBF method [13];
V A moving least-squares method [1] with the Levin's projection operator [7].

Parameter settings. In method II, the node function $f_k(x)$ in equation (4) is replaced by a multivariate quadratic function $Q_k(x)$ [12]. By using the progressive model with method II, the resulting sequence is similar to the MPU shape representation proposed in [9]. To set the pair of numbers (N_R, N_W), for method II, Renka [12] recommends $(13, 19)$ for $\mathbb{R}^2$ and $(17, 32)$ for $\mathbb{R}^3$, respectively. For method I, Lazzaro and Montefusco [6] recommend $(13, 10)$ for $\mathbb{R}^2$ and $(18, 7)$ for $\mathbb{R}^3$, respectively, to interpolate a set of functional values (p_i, f_i). In our implementation, to implicitly interpolate a point set with level set zero, our experiments show $(12 \sim 15, 6 \sim 9)$ offers good results for 3D case. To implement method V, let $B = \{x \in \mathbb{R}^3 \mid \|x - p_i\| < r_i, \forall p_i \in \mathcal{P}\}$ be a union of open balls with radius r_i centered at each point $p_i \in \mathcal{P}$ to define a tubular neighborhood of $\mathcal{P}$ in $\mathbb{R}^3$. To calculate a MLS projection operator $P_{MLS} : B \to B$, a local reference domain (a plane if in $\mathbb{R}^3$) is determined by minimizing a non-linear energy functional with a non-negative weight function θ. The widely used weight function $\theta(r) = e^{-r^2/h^2}$ gives C^∞ continuous MLS surfaces. As suggested in [1], the parameter h is not necessarily global and could be adapted to the local feature size. In our setting, we choose h locally to be the value r_i of radius of bounding sphere for each point p_i.

Time and space efficiency. Fig. 3 shows computational time and memory requirements for different surface inference methods. All the tests are performed on an off-the-shelf PC with 512 MB RAM and a Pentium III processor running at 937 MHz.

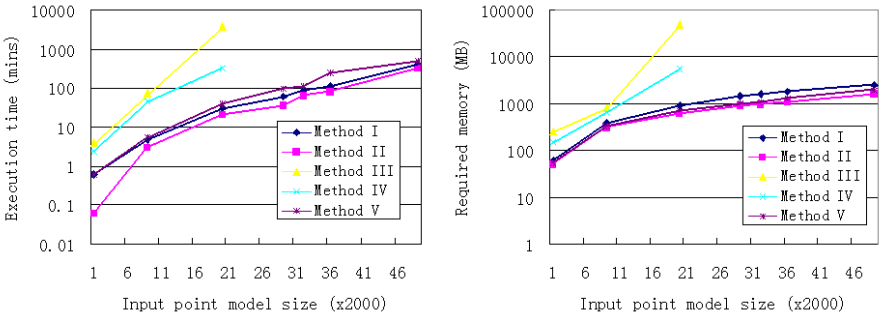


Fig. 3. Execution time and memory requirement for the five methods

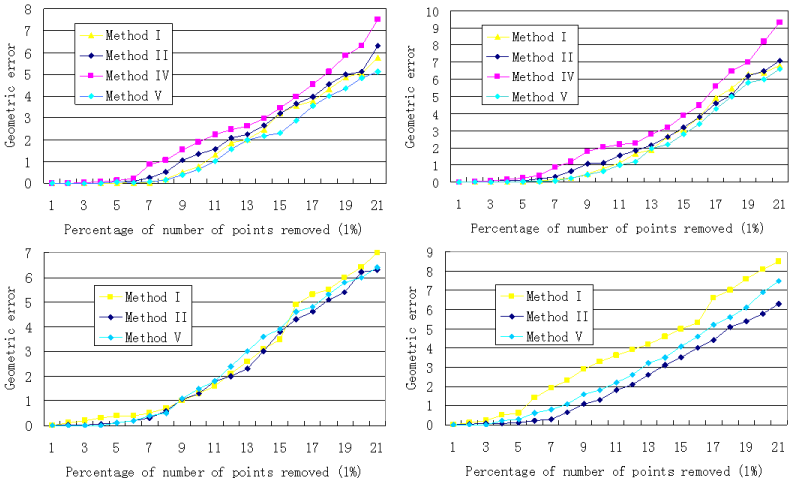


Fig. 4. The geometric approximation properties of four methods (I, II, IV, V). Due to unacceptable memory requirement, method III is not tested.

Geometric error. Fig. 4 shows the quantitative error estimates of four selected models in Fig. 2 by using the proposed first order approximation of geometric distance in equation (7) and the error norm $\|\cdot\|_\infty$ in equation (8).

As indicated by Figs. 3 and 4, we reach the following conclusions:

- Method III consumes the largest memory and take the longest time; with it the system soon run out of resource. So we do not test it further in terms of geometric errors. Method IV is the next time-consuming method and we only perform it with two small models in the geometric error testing;
- For all the tests, method IV has a relatively inferior reproduction property in terms of geometric shape inference. Thus we would not recommend method IV as a candidate method for progressive point-sampled geometry;
- For models that exhibit lots of tiny planar regions (ref. the car model in Fig. 2), method II shows a good reproduction property: this can be interpreted as

that the quadratic surface works well for approximation of near-planar regions. Method II also runs the fastest and is easy of implementation;
- For models that exhibit smooth details everywhere, methods I and V offer good balance between running time and reproduction properties. In the tests with smooth models, method V slightly outperforms method I. This result comes up to our expectation that moving least-squares surfaces should offer good approximation to C^{∞} smooth surfaces. In the tests with models of planar thin structures, method I slightly outperforms method V. Finally methods I and V show different reproduction quality at different percentages of removed points.

6 Conclusions

In this paper we propose practical algorithms to carry out a RBF-based progressive point-sampled geometry. Several well-known methods are also implemented and comparisons are presented. The results show that our proposed algorithms are fast, stable and can achieve good balance between geometric error reduction and time efficiency.

References

1. Alexa, M., et al.: Computing and rendering point set surfaces. IEEE Trans. Vis. Comput. Graph., **9(1)** (2003) 3–15
2. Amenta, N., Kil, Y.J.: Defining point-set surface. ACM Trans. Graph. (SIGGRAPH'04) **23(3)** (2004) 264–270
3. Chavez, E., Navarro, G., Baeza-Yates, R., Marroquin, J.L.: Search in metric spaces. ACM Comput. Surv. **33(3)** (2001) 273–321
4. de Berg, M., van Kreveld, M., Overmas, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Springer (1997)
5. Hoppe, H.: Progressive meshes. Proc. SIGGRAPH'96. 99–108
6. Lazzaro, D., Montefusco, L.B.: Radial basis functions for the multivariate interpolation of large scattered data. J. Comput. Appl. Math., **140(1-2)** (2002) 521–536
7. Levin, D.: Mesh-independent surface interpolation. Geometric Modeling for Scientific Visualization, G. Brunnett et al. eds., Springer-Verlag, (2003) 37–49
8. Liu, Y.J., Tang, K., Yuen, M.M.F.: Efficient and stable numerical algorithms on equilibrium equations for geometric modeling. Proc. GMP'04, 291–300
9. Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., Seidel, H.P.: Multi-level partition of unity implicits. ACM Trans. Graph. (SIGGRAPH'03) **22(3)** (2003) 463–470
10. Pauly, M., Keiser, R., Kobbelt, L.P., Gross, M.: Shape modeling with point-sampled geometry. ACM Trans. Graph. (SIGGRAPH'03) **22(3)** (2003) 641–650
11. Powell, M.J.D: The theory of radial basis function approximation in 1990. Advances in Numerical Analysis, (1992) 105–210
12. Renka, R.J.: Multivariate interpolation of large sets of scattered data. ACM Trans. Math. Software, **14(2)** (1988) 139–148
13. Wendland, H.: Piecewise polynomial, positive definite and compactly supported radial basis functions of minimal degree. Adv. Comput. Math., **4** (1995) 389–396
14. Zwicker, M., Pauly, M., Knoll, O., Gross, M.: Pointshop 3D: an interactive system for point-based surface editing. ACM Trans. Graph. **21(3)** (2002) 322–329

Segmentation of Scanned Mesh into Analytic Surfaces Based on Robust Curvature Estimation and Region Growing

Tomohiro Mizoguchi, Hiroaki Date, Satoshi Kanai, and Takeshi Kishinami

Graduate School of Information Science and Technology, Hokkaido University,
Kita 14, Nishi 9, Kita-ku, Sapporo, 060-0814, Japan
tmizoguchi@minf.coin.eng.hokudai.ac.jp,
{hdate, kanai, kisinami}@ssi.ist.hokudai.ac.jp

Abstract. For effective application of laser or X-ray CT scanned mesh models in design, analysis, and inspection etc, it is preferable that they are segmented into desirable regions as a pre-processing. Engineering parts are commonly covered with analytic surfaces, such as planes, cylinders, spheres, cones, and tori. Therefore, the portions of the part's boundary where each can be represented by a type of analytic surface have to be extracted as regions from the mesh model. In this paper, we propose a new mesh segmentation method for this purpose. We use the mesh curvature estimation with sharp edge recognition, and the non-iterative region growing to extract the regions. The proposed mesh curvature estimation is robust for measurement noise. Moreover, our proposed region growing enables to find more accurate boundaries of underlying surfaces, and to classify extracted analytic surfaces into higher-level classes of surfaces: fillet surface, linear extrusion surface and surface of revolution than those in the existing methods.

1 Introduction

3D laser and X-ray CT scanning systems are widely used in the field of reverse engineering to acquire scanned data of real-world objects. To use the acquired scanned data in today's digital engineering, it is easily converted into a 3D mesh model by a surface reconstruction algorithm such as marching cubes [1]. When we utilize 3D scanned mesh model for repairing, replication, analysis, or inspection of engineering parts, we need to efficiently segment the mesh model into desirable regions depending on its applications.

The surfaces of engineering parts mainly consist of a set of analytic surfaces, such as planes, cylinders, spheres, cones, and tori. Therefore, we need to extract regions each of which can be approximated by a simple analytic surface from a mesh model. However, few methods have been proposed to extract analytic surfaces from a mesh model. Moreover, in these methods, the accuracies of extracting regions from noisy mesh models and the range of extracted analytic surface classes were not necessarily sufficient from the aspect of practical engineering use.

The purpose of this research is to propose a new method that segments a scanned mesh model into regions each of which can be approximated by a simple analytic

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 644 – 654, 2006.
© Springer-Verlag Berlin Heidelberg 2006

surface. In this paper, we only deal with triangular mesh models whose surfaces are completely composed of planes, cylinders, spheres, cones, and tori.

Our algorithm is composed of three steps. The first step accurately estimates mesh principal curvatures based on a modified method of Razdan's [2]. It allows robust estimation for a noisy scanned mesh and ensures more accurate estimation even around sharp edges where the previous methods generated large estimation error (section 3). The second step extracts analytic surfaces based on the modified version of Vieira's region growing algorithm [3]. Our curvature estimation and limiting the fitting surface only to the analytic enable to initially create large seed regions in the region growing. This also enables non-iterative region growing and the efficient linear LSM in analytic surface fitting (section 4). The final step classifies extracted analytic surfaces into higher-level classes of surfaces than those in the existing methods [4][5][6]: fillet surface, linear extrusion surface, and surface of revolution.

2 Related Works

Mesh segmentation is a technique that segments a mesh model into desirable regions depending on applications, and many methods have been proposed for this segmentation in computer graphics(CG)[7][8][9] and engineering field. These works in CG are aiming at decomposing mesh surfaces into visually meaningful sub-meshes. On the other hand, the goal of the segmentation in an engineering field aims at decomposing the mesh surface into functionally meaningful surfaces. Therefore segmentation in CG cannot be directly applied to the engineering purpose. The mesh segmentation in the engineering field is roughly divided into following three groups:

The first group is to extract regions separated by sharp edges on a mesh model. In this group, a *watershed-based* approach has been well studied [10][11][12]. However they cannot extract regions separated by smooth edges (i.e., a region consisting of a plane smoothly connected to a cylinder), and therefore cannot identify the surface geometry of each segmented region.

The second group is to extract regions each of which can be approximated by a simple free form surface. In this group, a *region growing* approach [3][13][14] has been well studied. However the method did not focus on extracting regions approximated by analytic surfaces and their geometric parameters.

The last group is to extract regions each of which can be approximated by a simple analytic surface. Gelfand *et al.*[4] proposed a method based on eigenvalue analysis of a mesh model. Wu *et al.*[5] proposed a method based on Lloyd's clustering algorithm. However, in their method, the range of extracted analytic surface classes was not necessarily sufficient for engineering applications. Benkő *et al.*[6] proposed the *direct segmentation* for reverse engineering of the engineering part. Their algorithm segments a mesh model into regions each of which can be approximated by simple analytic surfaces (planes, cylinders, spheres, cones, and tori), linear extrusion surfaces, and the surface of revolutions. However this algorithm results in poor segmentation near boundaries of surfaces where the indicators may not be properly estimated. And they applied their segmentation only for a mesh model with very simple geometry.

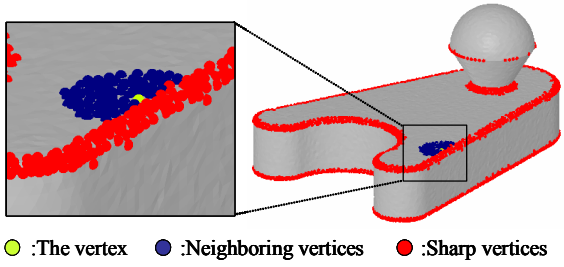
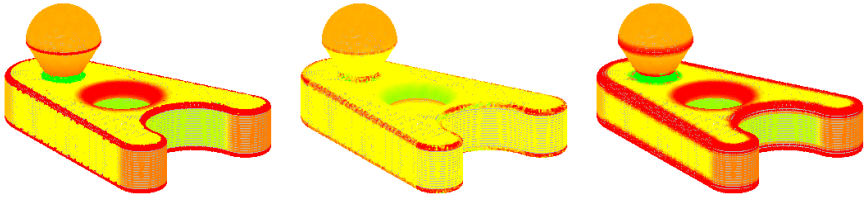


Fig. 1. Curvature estimation with sharp edges recognition



(a) Estimated maximum curvature with sharp edges recognition (b) Estimated minimum curvature with sharp edges recognition (c) Estimated maximum curvature without sharp edges recognition

Fig. 2. Results of curvature estimation

3 Robust Mesh Curvatures Estimation by Recognizing Sharp Edges

To estimate mesh curvatures on a noisy meshes, Razdan proposed the method based on a local biquadratic Bézier surface fitting [2]. This method locally fits the surface for a mesh vertex and a set of vertices included in its 2-ring, and estimates mesh curvatures at the vertex from the fitted surface. We simply modify this method, and fits the surface for each vertex v_i and a set of vertices directly connected to v_i within those which satisfy the condition of eq.(1),

$$\| \mathbf{v}_j - \mathbf{v}_i \| < W \cdot l_{i,avg} \tag{1}$$

where $l_{i,avg}$ is an average length of edge connecting to the vertex v_i, and W is a parameter to specify the size of surface-fit. Our algorithm analyses the fitted surface and estimates two principal curvatures by the method [2] with the parameter α=0.9. Then we define the one whose absolute value is the larger as the maximum principal curvature $\kappa_{i,\max}$, and the other one as the minimum principal curvature $\kappa_{i,\min}$.

To robustly estimate principal curvatures on noisy scanned meshes, a parameter W should be larger. However, a larger parameter setting results in poor curvature estimation around sharp edges. To solve the problem, we propose the following "two-pass" curvature estimations.

In the first pass, principal curvatures are estimated by fitting the relatively small surface on the mesh with W=2. If the maximum curvature $\kappa_{i,\max}$ of the vertex is larger than the user specified parameter th_{se}, the vertex is classified as a *sharp vertex* as shown in Fig.1. This parameter setting is as the same as Vieira's method [3].

In the second pass, principal curvatures are re-calculated for non-sharp vertices with W=5. In this step, sharp vertices and neighboring vertices beyond them are not included for the Bézier surface fitting as shown in Fig.1. This results in better accuracy of principal curvatures of vertices near sharp edges as shown in Fig.2.

Moreover in our method, the normal vector $\mathbf{n}_i'$ for the vertex v_i is also re-calculated as the normal of the fitted Bézier surface. This normal vector is better in accuracy than the one $\mathbf{n}_i$ calculated using connected triangles for noisy mesh models. The normal vectors $\mathbf{n}_i$ are used for region growing described in section 5.2, and the re-calculated normal vectors $\mathbf{n}_i'$ are used for the analytic surface fitting described in section 5.1.

4 Seed Region Creation

Next, seed regions are created on the surface of a mesh by classifying estimated principal curvatures. In this paper, a seed region means a set of topologically connected vertices that are supposed to be on a certain analytic surface. Our seed region creation algorithm is composed of following four steps.

Step1: Allocation of labels for planes, cylinders/cones, and spheres

A label for plane, cylinder/cone, and sphere that discriminates on which surface the vertex v_i lies is allocated to it. Our method allocates the labels for non-sharp vertices classified in sec.3 according to eq.(2) by evaluating two principal curvatures. Three parameters $\varepsilon_1, \varepsilon_2, \varepsilon_3$ are used in eq.(2). Ideally, $\varepsilon_1, \varepsilon_2, \varepsilon_3$ are zero, however they may not be zero due to scanning noise. We found that $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = 0.01$ roughly provided good results for mesh models both with artificially-added and real-scanning noise.

$$label(v_i) = \begin{cases} 1 & (plane) & if\ (|\kappa_{i,\max}| < \varepsilon_1) \\ 2 & (cylinder, cone) & else\ if\ (|\kappa_{i,\min}| < \varepsilon_2) \\ 3 & (sphere) & else\ if\ (|\kappa_{i,\max} - \kappa_{i,\min}| < \varepsilon_3 \\ & & and\ \kappa_{i,\max} \cdot \kappa_{i,\min} > 0) \\ 0 & (others) & otherwise \end{cases} \tag{2}$$

Step2: Classification of cylinders/cones and smoothly connected cylinders

Step 1 allocates the label 2 for both cylinders and cones. In this step, these cylinders and cones are discriminated. The smoothly connected cylinders are also separated into different single cylinders. To achieve this, a similarity value of the principal maximum curvature $f(v_i)$ is calculated for each vertex v_i according to eq.(3).

Table 1. Thresholds for seed region creation

surface type	threshold th_{seed}
plane	2
cylinder	4
sphere	5
cone	4
torus	7

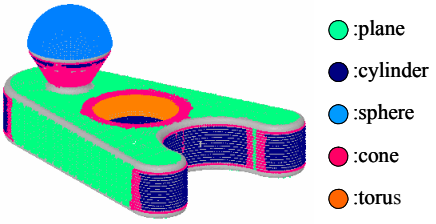


Fig. 3. Result of seed region creation

$$f(v_i) = \frac{1}{n} \sum_{j \in N^{**}(i)} \left| (\kappa_{i,\max} - \kappa_{j,\max}) / \kappa_{i,\max} \right| \tag{3}$$

where $N^{**}(i)$ is a set of vertices in 2-ring of the vertex v_i. Ideally, $f(v_i)$ is zero in cylinders and positive value in cones. Therefore if $f(v_i)$ is larger than the threshold th_{cyl_cone}, the vertex is assumed to belong to a cone, and $label(v_i)$ is changed to 4 and if $f(v_i)$ is smaller than the threshold, $label(v_i)$ is preserved. For our implementation, $th_{cyl_cone} = 0.01 - 0.4$ provides a good result.

Step3: Allocation of labels for tori

Previous two steps allocate labels for plane, cylinder, sphere, and cone, therefore most of vertices with label 0 are assumed to lie on tori. To allocate torus labels to such vertices, principal curvatures of vertices that have label 0 are evaluated.

A torus is the excursion surface where a sphere is rotated along an axis. Therefore one of the principal curvatures corresponds to the constant curvature of the radius r of the sphere. We use this property and create a histogram of discretized principal curvatures for a set of vertices that have the label 0. If the number of vertices that has a particular discretized curvature value is larger than the threshold th_{torus}, label 5 is allocated to their vertices. In our implementation, 0.01 for the step of principle curvature, and 0.1-0.5% of the number of all vertices for th_{torus} provided good results for most mesh models.

Step4: Creation of seed regions

Finally, a seed region is created as a set of topologically connected vertices with the same label that has the number of vertices larger than the threshold th_{seed} shown in table1. In our implementation, th_{seed} corresponds to the minimum number of vertices which enables the least square analytic surface fitting that is proposed in this paper and is described in section 5.1. Fig.3 shows the results of seed region creation.

5 Analytic Surface Extraction

5.1 Analytic Surfaces Fitting to Seed Regions

In this paper, we propose the following efficient analytic surface fitting where we only need to solve the linear least squares problems to find fitted analytic surfaces

instead of the non-linear one by utilizing pre-computed normal vectors $\mathbf{n}_i'$. Our method is less accurate than previous non-linear methods [15][16][17], but faster than them, and provides a practically enough results for analytic surface extraction.

Plane fitting: A plane is defined by its unit normal vector $\mathbf{n} = (n_x, n_y, n_z)$ and a distance d from an origin. Our method calculates $\mathbf{n}$ as the normalized average vector of vertex normals $\mathbf{n}_i'$ in a seed region of the plane. Then the distance d is calculated using linear least squares sense.

Cylinder fitting: A cylinder is defined by its unit axis direction vector $\mathbf{d} = (d_x, d_y, d_z)$, radius r, and an arbitrary point on the axis $\mathbf{p} = (p_x, p_y, p_z)$. First, all of vertex normals $\mathbf{n}_i'$ in a seed region are mapped onto a Gaussian sphere. Then a least square plane is fitted so that the plane passes through the end points of $\mathbf{n}_i'$ in the sphere. The axis direction is derived as a unit normal vector of this plane. Next, all vertices in a seed region are projected onto the plane whose normal vector is $\mathbf{d}$. Then, a circle is fitted to these projected points on the plane in the least squares sense, and the radius r is calculated as the radius of the fitted circle. Finally, the center of the circle is also calculated on the projected plane, and it can be easily transformed to $\mathbf{p}$.

Sphere fitting: A sphere is defined by its center $\mathbf{c} = (c_x, c_y, c_z)$ and radius r. Our method solves a linear least squares problem to find the coefficients (A, B, C, D) defining the sphere of the implicit form $x^2 + y^2 + z^2 + Ax + By + Cz + D = 0$. They are easily converted into the center $\mathbf{c}$ and radius r.

Cone fitting: A cone is defined by its unit axis direction vector $\mathbf{d} = (d_x, d_y, d_z)$, apex $\mathbf{a} = (a_x, a_y, a_z)$, and vertical angle θ. The unit axis direction vector $\mathbf{d}$ is calculated using the same method as for a cylinder. An apex $\mathbf{a}$ is given from the condition that a vector passing through $\mathbf{a}$ and v_i is orthogonal to a normal vector of the vertex $\mathbf{n}_i'$. This is obtained by finding a least squares solution of $\mathbf{a}$ in $\mathbf{n}_i' \cdot (\mathbf{a} - \mathbf{v}_i) = 0$. An apex θ is calculated as the average of angles between $\mathbf{d}$ and a vertex normal $\mathbf{n}_i'$.

Torus Fitting: A torus is defined by its unit axis direction vector $\mathbf{d} = (d_x, d_y, d_z)$, its center $\mathbf{c} = (c_x, c_y, c_z)$, the radius of its body r, and the radius of the centerline of the torus body R. First, to calculate $\mathbf{d}$ and an arbitrary point $\mathbf{p}$ on the axis, we use the same method as Kós *et al.*[15]. Kós first calculated the initial estimates of $\mathbf{d}$ and $\mathbf{p}$ using the generalized eigen analysis, and then the precise solutions of them are calculated by the iterative method from these initial estimates. We first use Kós's initial estimates as our final solutions of $\mathbf{d}$ and $\mathbf{p}$ for fitting torus. Next, all vertices in a seed region are rotated around the calculated axis so as to be placed onto a plane which includes the calculated axis. Then a circle is fitted to the points on the plane in the least squares sense. The radius r is calculated as the radius of that circle. The center $\mathbf{c}$ is calculated according to the condition that the vector toward the center of the torus $\mathbf{c}$ from the center of the circle is orthogonal to the torus axis. The radius R is also calculated as the length between the center of the torus and the center of the circle.

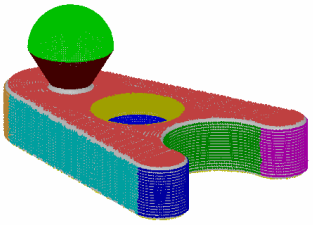


Fig. 4. Results of extracted analytic surfaces using the region growing

5.2 Extraction of Analytic Surfaces Based on Region Growing

Next, our region growing method makes a set of vertices topologically connected to the seed region added to the original seed region if the vertex lies on the fitted surface within a specified tolerance. Therefore our algorithm first calculates the positional error between the mesh vertex v_i and the point $p(v_i)$ on the analytic surface which is the projection of v_i to the surface along $\mathbf{n}_i$, and the directional error of the normal vectors between them. If the vertex is adjacent to the seed region and satisfies the compatibility in eq.(4) and eq.(5), it is added to the seed region.

$$\left|\mathbf{v}_i - \mathbf{p}(v_i)\right| < th_{pos} \cdot l_{avg} \tag{4}$$

$$\left|\cos^{-1}\left(\mathbf{n}_i - \mathbf{n}(p(v_i))\right)\right| < th_{norm} \tag{5}$$

where l_{avg} is an average length of all mesh edges. th_{pos} and th_{norm} are the thresholds of positional and directional errors, and we found in the experiments that $th_{pos} = 0.5$ and $th_{norm} = 8.0$[deg] provided good results for mesh models measured from CT scanning.

This region growing is done for the seed region in the descending order of the number of vertices in the region. This enables to generate a small number of larger regions. If all the vertices adjacent to the region do not satisfy eq.(4) or eq.(5), the region growing stops. The region can be extracted as a set of topologically connected vertices that are approximated by a particular analytic surface. Fig.4 shows the result of analytic surfaces extraction.

6 Recognition of Fillet Surfaces, Linear Extrusion Surfaces, and Surfaces of Revolution

For the effective use of a mesh model in various mesh applications, our method recognizes fillet surfaces, linear extrusion surfaces, and surfaces of revolutions which are often included in most engineering parts from a mesh model.

Recognition of fillet surfaces

We assume that all surfaces in a mesh model are covered with analytical surfaces, and fillet surfaces are also represented by them. This assumption enables to classify fillet

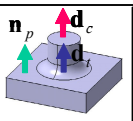
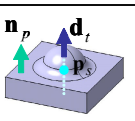
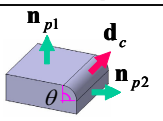
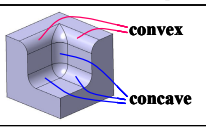
	A torus-type between two surfaces		A cylinder-type between two planes	A sphere/torus-type between three cylinder-type fillets
Recognized fillets				
Neighboring surfaces	A cylinder and a plane	A sphere and a plane	Two planes	Three cylinder-type filletes
Geometric conditions	$\mathbf{n}_p // \mathbf{d}_t // \mathbf{d}_c$	1) $\mathbf{p}_s$ lie on the torus axis 2) $\mathbf{n}_p // \mathbf{d}_t$	1) $\mathbf{n}_{p2} \perp \mathbf{d}_c$ and $\mathbf{n}_{p1} \perp \mathbf{d}_c$ 2) $0^\circ < \theta < 180^\circ$	Sphere-type: only convex or concave are connected Torus-type: convex and concave are connected

Fig. 5. Definition of fillet surfaces

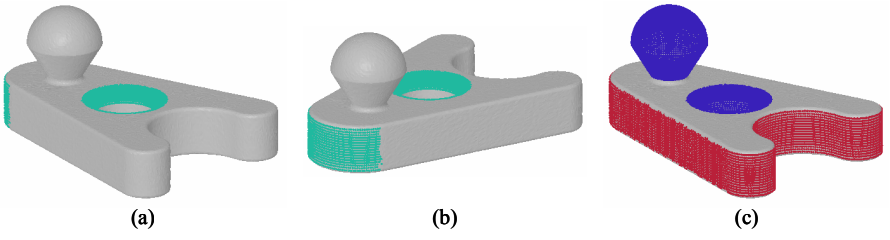


Fig .6. (a)(b)Recognitionof fillet surfaces, (c)alinear extrusion surface(red) and surfaces of revolution(blue)

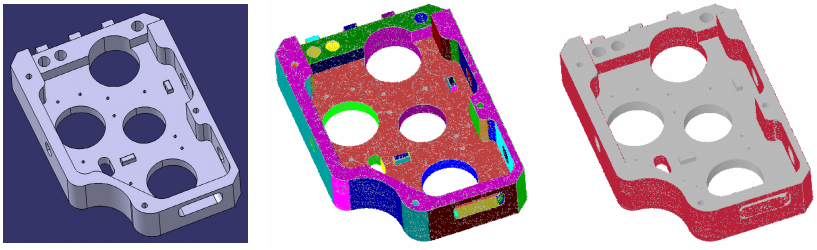
surfaces into three types: cylinders, spheres, and tori [18]. These surfaces can be defined based on their geometric parameters and combinations of neighboring surfaces as shown in Fig.5. Our method recognizes the analytic surface satisfying the definition as a fillet surface. Fig.6(c) shows results of recognizing fillet surfaces.

Recognition of linear extrusion surfaces

The linear extrusion surface is composed of a combination of some planes and cylinders. These surfaces must satisfy the following three conditions: (1) a plane normal and a cylinder axis must be orthogonal, (2) normal vectors of three or more planes must be coplanar, and (3) axes of two or more cylinders must be parallel. Our method recognizes a set of topologically connected analytic surfaces satisfying the above conditions as a linear extrusion surface. Fig.6(c) shows results of recognizing a linear extrusion surface.

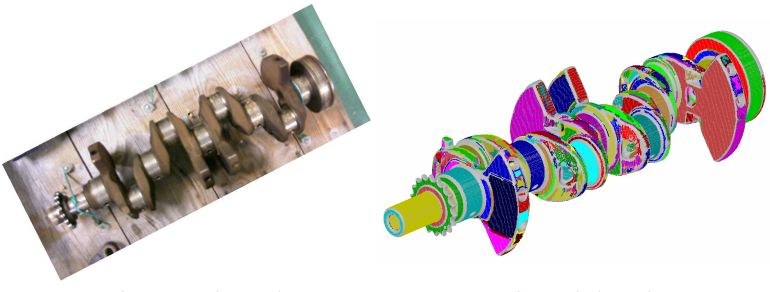
Recognition of surfaces of revolution

The surface of revolution is composed a combination of planes, cylinders, spheres, cone, and tori. These surfaces must satisfy the following two conditions: (1) normal vectors of planes and axis directions must be parallel, and (2) center points of spheres and tori, apexes of cones, and arbitrary points of axes of cylinders must lie in a same line with a direction parallel to their normals or axis. Our method recognizes a set of topologically connected analytic surfaces satisfying the above conditions as a surface of revolution. Fig.6(c) also shows results of recognizing a surface of revolution.



(a) Solid Model (b) Analytic surfaces (c) Linear extrusion Surface

Fig. 7. Results to our mesh model for verification



(a) The automotive engine part (b) Analytic surfaces

Fig. 8. Results to the mesh model created by CT scanning (974,754 tri)

7 Results

Fig.7 shows the results to a mesh model (300,000tri) for verification which was created by FEM meshing of a solid model. Then we added artificial noise on this model by moving the vertex position to its normal direction using a Gaussian distributed random value. Our method can extract regions from a noisy complex shaped model and can find accurate boundaries of underlying analytic surfaces.

Fig.8 shows the results for the CT-scanned mesh model of an automotive engine part. It shows that our method could well extract planes and cylinders with relatively large areas from the model. Especially it could extract all functionally important cylindrical regions (fitted with bearings). The model has about 1,000,000 triangles and our method could extract analytic surfaces in less than 7 minutes.

8 Conclusions and Future Works

In this paper, we proposed a new method of systematically extracting analytic surfaces from a mesh model. From the simulation and the experiment for the various mesh models, we found that our method could produce an accurate and practical geometric model consisting of a set of analytic surfaces, fillet surfaces, linear extrusion surfaces and surfaces of revolution from mesh models.

One limitation of our method is that the thresholds to extract regions cannot be easily set by users. In our research, we experimentally found the appropriate values for all thresholds described in this paper, and they can work well for various mesh models.

As for our future work, we need to impose geometric constraints among fitted surfaces (parallel, orthogonal, continuous, etc). Moreover, feature recognition such as boss, lib, slot, etc., will also be needed to use mesh models as we use feature-based solid models that are commonly used in the commercial 3D CAD systems.

Acknowledgement

This work was financially supported by a grant-in-aid of Intelligent Cluster Project (Sapporo IT Carrozzeria) founded by MEXT. We thank for Ichiro Nishigaki and Noriyuki Sadaoka in HITACHI Co.Ltd. for providing CT-scanned mesh model.

References

1. Lorensen, W.E., Harvey, E.C.: Marching cubes: A high resolution 3D surface construction algorithm. ACM SIGGRAPH Computer Graphics, Vol.21, No.4. (1987) 163-169
2. Razdan, A., Bae, M.S.: Curvature estimation scheme for triangle meshes using biquadratic Bézier patches. Computer-Aided Design, Vol.37, No.14. (2005) 1481-1491
3. Vieira, M., Shimada, K.: Surface mesh segmentation and smooth surface extraction through. Computer-Aided Geometric Design, Vol.22, No.8. (2005) 771-792
4. Gelfand, N., Guibas, L.J.: Shape segmentation using local slippage analysis. Proc. of Eurographics/ACM SIGGRAPH symposium on Geometry processing. (2004) 214-223
5. Wu, J., Kobbelt, L.: Structure Recovery via Hybrid Variational Surface Approximation. Proc. of Eurographics. Vol.24, No.3. (2005) 277-284
6. Benkő, P., Várady, T.: Segmentation methods for smooth point regions of conventional engineering objects. Computer-Aided Design, Vol.36, No.6. (2004) 511-523
7. Yamauchi, H., Gumhold, S., Zayer, R., Seidel, H.P.: Mesh Segmentation Driven by Gaussian Curvature. Visual Computer. Vol.21, No.8-10. (2005) 649-658
8. Katz, S., Leifman, G., Tal, A.: Mesh segmentation using feature point and core extraction. Proc. of Pacific Graphics. (2005) 649-658
9. Attene, M., Katz, S., Mortara, M., Patané, G., Spagnuolo, M., Tal, A.: Mesh Segmentation - A comparative study. Proc. of Shape Modeling and Applications. (2006)
10. Mangan, A.P., Whitaker, R.T.: Partitioning 3D Surface Meshes Using Watershed Segmentation. IEEE Trans. on visualization and computer graphics, Vol.5, No.4. (1999) 308-321
11. Sun, Y.D., Page, L., Paik, J. K., Koschan, A., Abidi, M.A.: Triangle mesh-based edge detection and its application to surface segmentation and adaptive surface smoothing. Proc. of the International Conference on Image Processing. Vol.3. (2002) 825-828
12. Razdan, A.: Hybrid approach to feature segmentation of triangle meshes. Computer-Aided Design, Vol.35, No.9. (2003) 783-789
13. Besl, P.J., Jain, R.C.: Segmentation through Variable-Order Surface Fitting. IEEE Trans. on Pattern Analysis and Machine Intelligence. Vol.10, No.2. (1988) 167-192

14. Djebali, M. Melkemi, M., Sapidis, N.: Range-Image segmentation and model reconstruction based on a fit-and-merge strategy. Proc. of ACM symposium on Solid modeling and applications.(2002) 127-138
15. Kós, G., Martin, R., Várady, T.: Methods to recover constant radius rolling blends in reverse engineering. Computer-Aided Geometric Design. Vol.17, No.2. (2000) 127-160
16. Benkő, P., Kós, G., Várady, T., Andor, L., Martin, R.: Constrained fitting in reverse engineering. Computer-Aided Geometric Design. Vol.19, No.3. (2002) 173-205
17. Marshall, D., Lukacs, G., Martin, R.: Robust Segmentation of Primitives from Range Data in the Presence of Geometric Degeneracy. IEEE Trans. on Pattern Analysis and Machine Intelligence. Vol.23, No.3. (2001) 304-314
18. Zhu, H., Menq, C.H.: B-Rep model simplification by automatic fillet/round suppressing for efficient automatic feature recognition. Computer-Aided Design. Vol.34, No.2. (2002) 109-123

Finding Mold-Piece Regions Using Computer Graphics Hardware

Alok K. Priyadarshi[1] and Satyandra K. Gupta[2]

[1] Solidworks Corporation, Concord, MA 01742, USA
apriyadarshi@solidworks.com
[2] University of Maryland, College Park, MD 20742, USA
skgupta@eng.umd.edu

Abstract. An important step in the mold design process that ensures disassembly of mold pieces consists of identifying various regions on the part that will be formed by different mold pieces. This paper presents an efficient and robust algorithm to find and highlight the mold-piece regions on a part. The algorithm can be executed on current-generation computer graphics hardware. The complexity of the algorithm solely depends on the time to render the given part. By using a system that can quickly find various mold-piece regions on a part, designers can easily optimize the part and mold design and if needed make appropriate corrections upfront, streamlining the subsequent design steps.

1 Introduction

While designing injection molds, there are often concerns about disassemblability of the mold as designed. An important step in the mold design process that ensures disassembly of mold pieces consists of identifying various regions on the part that will be formed by different mold pieces. These regions are called *Mold-Piece Regions*.

Most of the literature in mold design is focused on detecting undercuts and finding undercut-free directions. For an overview of mold design literature, the reader is directed to [Priy03, Bane06]. Ahn et al. [Ahn02] presented a provably complete algorithm for finding undercut-free parting directions. Khardekar et al. [Khar05] implemented the algorithm presented by Ahn et al. [Ahn02] on programmable GPUs. They also describe a method to highlight the undercuts.

We use GPUs to find mold-piece regions on a part efficiently and robustly. The basic idea behind the algorithm is similar to shadow mapping. The near-vertical facets are handled by slightly perturbing the vertices on those facets and visibility sampling. We describe an implementation of our algorithm that can be executed on any OpenGL 2.0. The complexity of our algorithm solely depends on the time to render the given part.

2 Finding Mold-Piece Regions

A *Mold-Piece Region* of a part is a set of part facets that can be formed by a single mold piece. Given a polyhedral object P and a parting direction $\boldsymbol{d}$, there are four mold-piece regions with the following property:

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 655–662, 2006.
© Springer-Verlag Berlin Heidelberg 2006

1. *Core* is accessible from $+\boldsymbol{d}$, but not $-\boldsymbol{d}$
2. *Cavity* is accessible from $-\boldsymbol{d}$, but not $+\boldsymbol{d}$
3. *Both* is accessible from both, $+\boldsymbol{d}$ and $-\boldsymbol{d}$
4. *Undercut* is not accessible from either $+\boldsymbol{d}$ or $-\boldsymbol{d}$

Hence, the problem of finding mold-piece regions reduces to performing accessibility analysis [Dhal03] of P along $+\boldsymbol{d}$ and $-\boldsymbol{d}$. Figure 1 shows various mold-piece regions for a part.

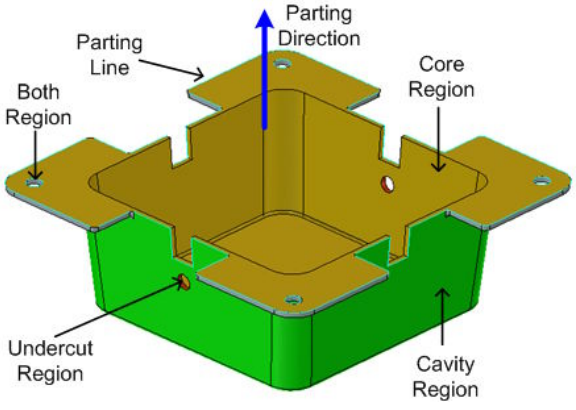


Fig. 1. Mold-Piece Regions

2.1 Overview of Approach

We use programmable GPUs to highlight the mold-piece regions on a part. The basic idea is very similar to hardware shadow mapping [Kilg01]. The given part is illuminated by two directional light sources located at infinity in the positive and negative parting directions. The regions that are lit by the upper and lower lights are marked as 'core' and 'cavity' respectively. The regions lit by both the lights are marked as 'both', while the regions in shadow are marked as 'undercuts'.

For a given parting direction, our approach highlights the mold-piece regions on a part in two steps:

1. *Preprocessing*: We create two shadow maps by performing the following procedure. First the part is rendered with the camera placed above the part and view direction along the negative parting direction. The resulting z-buffer is transferred to a depth texture (shadow map). The current orthogonal view matrix is also stored for the next step. The same procedure is repeated with the view direction along positive parting direction.
2. *Highlighting*: The user can then rotate the camera and examine the mold-piece regions of the part from all directions. A vertex program transforms the incoming vertices using the two model-view matrices stored in the preprocessing stage. The fragment program determines the visibility of each incoming fragment by comparing its depth with the depth texture values stored in the preprocessing stage and colors it accordingly.

If the algorithm is implemented as described, all the vertical facets will be reported as undercuts. Also, like any method based on shadow mapping, it needs to handle aliasing and self-shadowing. Section 2.2 and Section 2.3 describe techniques to handle these issues.

2.2 Handling Near-Vertical Facets

There is a slight difference between the notion of visibility in computer graphics and accessibility. The mathematical conditions for visibility and accessibility of a facet with normal $\boldsymbol{n}$ in direction $\boldsymbol{d}$ are the following:

Visible if: $\boldsymbol{d} \cdot \boldsymbol{n} > 0$
Accessible if: $\boldsymbol{d} \cdot \boldsymbol{n} \geq 0$

In other words, a facet perpendicular to a direction (vertical facet) is not visible, but accessible. This means that all the vertical facets will be reported as undercuts.

In addition to vertical facets, we also need to handle facets whose normals are very close to being perpendicular to the parting direction. These near-vertical facets are usually produced as a result of the approximation introduced by faceting vertical curved surfaces. The robustness problems in geometric computations are usually handled by slightly perturbing the input. But we cannot adopt this approach here as perturbing the vertices of the part will change it's appearance on the computer screen. We solve this problem by visibility sampling. To determine the accessibility of a rasterized fragment, the neighborhood of the corresponding texel in the shadow map is sampled in the image space. If any sample passes the visibility test, the fragment is marked as accessible. Incidentally, percentage closer filtering (PCF) [Reev87] used to produce anti-aliased shadows does just that.

For a given parting direction $\boldsymbol{d}$, we divide the part facets into three categories:

1. Up facets: $\boldsymbol{d}.\boldsymbol{n} \geq \tau$
2. Down facets: $\boldsymbol{d}.\boldsymbol{n} \leq -\tau$
3. Near-vertical facets: $|\boldsymbol{d}.\boldsymbol{n}| < \tau$

where $\boldsymbol{n}$ is the facet normal and τ is normal tolerance whose value is dependent on the surface tolerance introduced by faceting the part. It is usually set between 1-2 degrees.

Up and down facets are tested for accessibility along $-\boldsymbol{d}$ and $+\boldsymbol{d}$ respectively. The near-vertical facets are tested in both the directions with PCF enabled. In our implementation, we used the OpenGL extension ARB_shadow that samples the neighborhood of a fragment and returns the average of all the depth comparisons. If the returned value is greater than zero, we mark the fragment as accessible. The PCF kernel that determines the size of the sampling neighborhood should be adjusted according to the surface tolerance of the given part and resolution of the shadow map. We found that 3x3 kernel (9 samples) worked fine for the parts that we tested.

2.3 Preventing Self-shadowing

Our algorithm, being based on shadow mapping, is prone to self-shadowing due to precision and sampling issues. The focus of the currently available algorithms to prevent self-shadowing is mainly on producing aesthetically pleasing results. They may not be physically correct. We decided not to use the most popular polygon offset technique [Kilg01] after extensive experimentation. We found that it is very difficult to specify an appropriate bias for a part automatically. If the bias is too little, everything begins to shadow. And if it is too much, shadow starts too far back i.e., some of the fragments that should be in shadow are incorrectly lit. We found that this problem is exaggerated in case of mechanical parts with regions of high slope. We developed an adaptation of the second depth technique [Wang94] that prevents self-shadowing and robustly handles the near-vertical facets.

Second depth technique [Wang94] is based on the observation that in case of solid objects there is always a back facet on top of a shadowed front facet. It renders only the back facets into the shadow map and avoids many aliasing problems because there is adequate separation between the front and back facets. But it may show incorrect results when used with PCF for near-vertical facets. As explained in Section 2.2, we use PCF to sample the neighborhood of a point on a near-vertical facet. If any sample passes the visibility test, we mark the point as accessible. Because the shadow map only partially overlaps the PCF kernels for both points A and B, they will be reported as only 50% shadowed and hence accessible. This is the intended result for point B, but incorrect for point A.

To solve this problem, we use a visibility theorem for polyhedral surfaces based on the results published in [Kett99] and [Ahn02].

Definition 1. *An edge is a contour edge if it is incident to a front-facing facet and a back-facing facet for a given viewing direction.*

Theorem 1. *For a given polyhedron and a viewing direction, if the edges and facets of the polyhedron are projected into the viewing plane, the visibility of the projected facets can only change at the intersection with convex contour edges.*

The proof of the above theorem follows from the results presented in [Kett99] and [Ahn02]. We exploit the corollary of the above theorem that the visibility of projected facets cannot change at the intersection with *concave* contour edges. When creating the shadow map, we also render thick concave contour edges along with the back facets. As can be seen in Figure 2(b), now that the shadow map fully overlaps the PCF kernel for point A, it will be correctly reported as fully shadowed and hence marked as inaccessible. It can also be seen that thickening the concave contour edges does not affect the accessibility of point B.

2.4 Transferring Results from the GPU to CPU

The previous sections describe how to find and highlight the mold-piece regions using GPUs. This section describes how the information on mold-piece regions can be transferred back to the CPU for other purposes such as designing molds. We describe a simpler two-pass algorithm to accomplish the same.

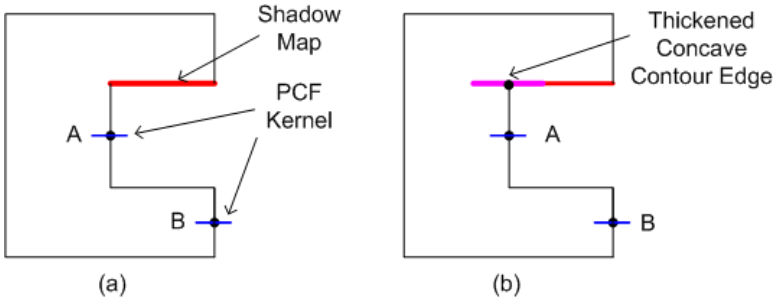


Fig. 2. The problem with the second-depth technique when used with PCF (the PCF kernel and the contour edge has been exaggerated for illustration purposes); (a) Both point A and point B are reported as only 50% shadowed and hence accessible; (b) The problem is solved by rendering thick concave contour edges into the shadow map

We first assign a unique ID (color) to each facet of the given part. Almost all the currently available graphics cards support at least 24-bit color palette that can generate over 16 million unique colors. Then we follow the following procedure to obtain the results on the CPU. The part is first rendered with the camera placed above the part and view direction along the negative parting direction. The resulting frame buffer (image) is read back to the CPU. The facets whose IDs are present in the resulting image constitute the 'core' region. The same procedure is followed with the view direction along the positive parting direction to obtain the 'cavity' region. The facets missing from both the images are undercuts.

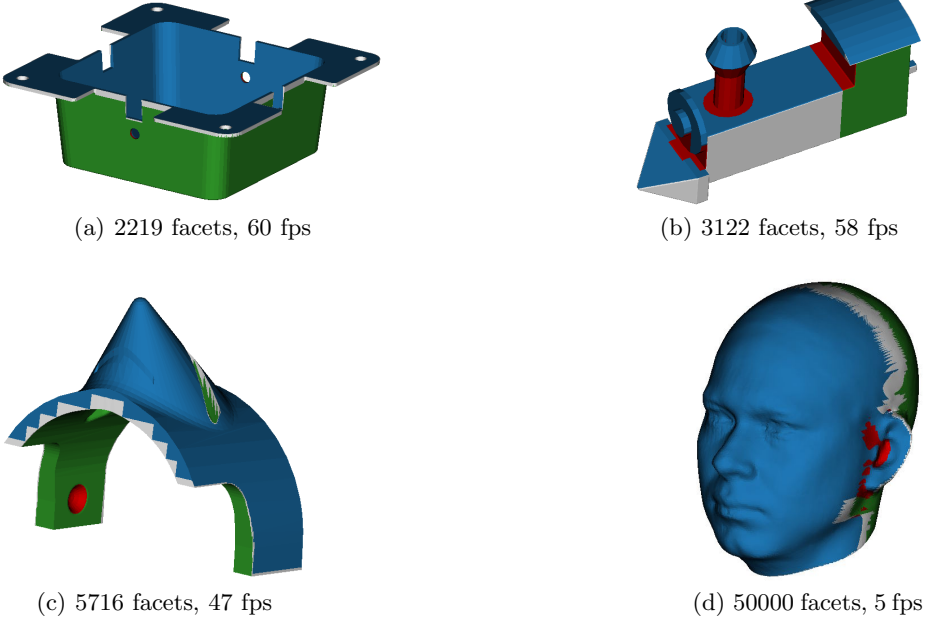
The problem with the above approach is that it cannot find the 'both' region. None of the facets will be present in both the frame buffers and all of the vertical facets will be reported as undercuts because being perpendicular to the viewing direction, they cannot be rendered. But now since the part is not rendered for visualization purposes, we can perturb the vertices of the part. For both the viewing directions (negative and positive parting direction), we slightly perturb the vertices of the near-vertical facets such that it becomes a front-facing facet for that viewing direction and hence an eligible candidate for being rendered. This perturbation is similar to adding a draft to the near-vertical facets and can be done on either the CPU or by a vertex program loaded on the GPU. A reference plane is first located at the top-most vertex with respect to the viewing direction and then each vertex on the near-vertical facets is slightly moved along the surface normal at that point. The perturbation amount is in proportion to the distance of the vertex from the reference plane and is given by $d = z.\tan(\tau)$, where τ is a small user-defined angle, which depends on the average length of facets and resolution of the frame buffer. We found that for a 512x512 buffer, $\tau = 0.5^\circ$ was appropriate for the parts that we tested.

The algorithm for transferring the results from the GPU to CPU is based on the assumption that the each facet belongs to only one mold-piece region. Sometimes a front facet needs to be split into a core and an undercut facet, or a

vertical facet needs to be split into all the four mold-piece regions. A brute-force approach to overcome this limitation could be splitting each facet into very small facets. Another approach could be projecting each facet into the viewing plane and splitting them at the intersection with convex contour edges [Kett99], and performing trapezoidal decomposition of vertical facets [Ahn02].

3 Implementation and Results

The latest GPUs allow users to load their own programs (shaders) to replace some stages of the fixed rendering pipeline. We have implemented our algorithm as shader programs using OpenGL Shading Language (GLSL). The implementation has been successfully tested on more than 50 industrial parts. It currently supports Stereolithography (STL) and Wavefront (OBJ) part files. Figure 3 shows the screenshot of four example parts. Figure 4 shows the performance of our implementation on 128 MB NVIDIA Fx700Go card. It shows the obtained frame rates when simply rendering the part using fixed OpenGL pipeline (without highlighting) and with highlighting. It can be seen that the overhead imposed by the highlighting algorithm does not significantly affect the time taken by the GPU to render a frame. The observed drop in performance



(a) 2219 facets, 60 fps

(b) 3122 facets, 58 fps

(c) 5716 facets, 47 fps

(d) 50000 facets, 5 fps

Fig. 3. Screenshots of four example parts. The color scheme for highlighting is following. Core region is blue, cavity region is green, both region is gray, and the undercuts are red. Number of facets and obtained rendering speed is reported against each subfigure.

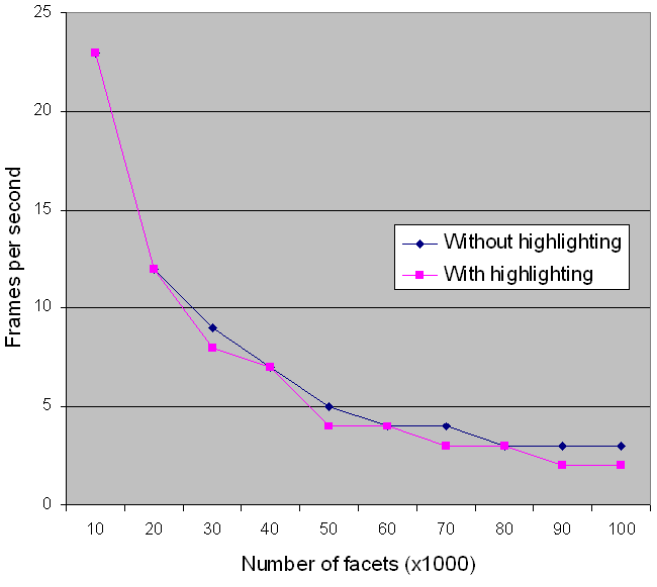


Fig. 4. Performance of the algorithm on 128 MB NVIDIA Fx700Go card. The plot shows the obtained frame rates when simply rendering the part (without highlighting) and those when also highlighting the mold-piece regions.

when highlight is at most one fps. In other words, the complexity of the algorithms solely depends on the time to render the given part.

4 Conclusions

In this paper we describe a method that utilizes current-generation GPUs to find and highlight mold-piece regions on a part. We presented techniques for robustly handling the near-vertical facets by slightly perturbing the vertices on those facets and visibility sampling. We also presented a technique that prevents self-shadowing and robustly handles the near-vertical facets.

Our algorithm exploits the computational power offered by the GPUs. Moreover, an efficient implementation of our algorithm does not impose any significant overhead on the GPU. The mold-piece regions even for parts with more than 50,000 facets can be highlighted at interactive rates. We believe that in the current scenario when the data size is growing at exponential rates because of the advances in scanning technology, such a system that provides real-time information about mold-piece regions will be very useful to the part and mold designers alike. They can easily optimize the part and mold design and if needed make appropriate corrections upfront, streamlining the subsequent design steps.

Acknowledgments

This work has been supported by NSF grant DMI-0093142. However, the opinions expressed here are those of the authors and do not necessarily reflect that of the sponsor. We would also like to thank the reviewers for their comments that improved the exposition.

References

[Ahn02] Ahn, De Berg, Bose, Cheng, Halperin, Matousek, and Schwarzkopf, Separating an object from its cast. *Computer-Aided Design*, 34, 547-59, 2002

[Bane06] A.G. Banerjee, and S.K. Gupta, A step towards automated design of side actions in injection molding of complex parts. In *Proceedings of Geometric Modeling and Processing*, Pittsburgh, PA, 2006

[Dhal03] S. Dhaliwal, S.K. Gupta, J. Huang, and A. Priyadarshi. Algorithms for computing global accessibility cones. *Journal of Computing and Information Science in Engineering*, 3(3):200–209, September 2003

[Kett99] Lutz Kettner. Software Design in Computational Geometry and Contour-Edge Based Polyhedron Visualization. PhD Thesis, ETH Zrich, Institute of Theoretical Computer Science, September 1999

[Kilg01] Mark Kilgard. Shadow Mapping with Today's Hardware. Technical presentation, http://developer.nvidia.com/object/ogl_shadowmap.html

[Khar05] Khardekar, Burton, and McMains. Finding feasible mold parting directions using graphics hardware, *Proceedings of the 2005 ACM symposium on Solid and Physical Modeling*, Cambridge, MA, June 2005, pp. 233-243

[Priy03] A. Priyadarshi, and S.K. Gupta. Geometric algorithms for automated design of multi-piece permanent molds. *Computer Aided Design*, 36(3): 241–260, 2004.

[Reev87] W. T. Reeves, D. H. Salesin, and R. L. Cook. Rendering antialiased shadows with depth maps. In *Computer Graphics (SIGGRAPH 87 Proceedings)*, pages 283-291, July 1987

[Wang94] Y. Wang and S. Molnar. Second-Depth Shadow Mapping. *UNC-CS Technical Report TR94-019,* 1994

A Method for FEA-Based Design of Heterogeneous Objects

Ki-Hoon Shin and Jin-Koo Lee

Department of Mechanical Engineering, Seoul National University of Technology,
172 Gongneung 2-dong, Nowon-gu, Seoul 139-743, Korea
shinkh@snut.ac.kr, ljk@snut.ac.kr

Abstract. This paper introduces an iterative method for FEA-based design of heterogeneous objects. A heterogeneous solid model is first created by referring to the libraries of primary materials and composition functions. The model is then discretized into an object model onto which appropriate material properties are mapped. Discretization converts continuous material variation inside an object into stepwise variation. Next, the object model is adaptively meshed and converted into a finite element model. FEA using ANSYS software is finally performed to estimate stress levels. This FEA-based design cycle is repeated until a satisfactory solution is obtained. An example (FGM pressure vessel) is shown to illustrate the entire FEA-based design cycle.

Keywords: FEA, Composition-based Mesh Generation, Heterogeneous Objects.

1 Introduction

There exists an increasing need for extending conventional CAD/CAM systems beyond geometry to consider material attributes (e.g., material composition and microstructure) inside heterogeneous objects. Heterogeneous objects are mainly classified as multi-materials, which have distinct material regions, and Functionally Graded Materials (FGMs), which possess continuous material variation along with the geometry. In particular, modern FGMs are required in a variety of structural materials, such as high efficiency components, direct metal tools, biomaterials and many more. For these applications, composition control is necessary to achieve multiple functionalities (e.g., improving thermo-mechanical performance, reducing interfacial stresses between dissimilar materials). Fig. 1-(a) shows such an example, pressure vessel with thermal barrier FGM [1]. Suppose that high temperature/pressure fluid flows inside the pressure vessel while the outer surface is exposed to ambient conditions, it is desirable to have ceramic on the inner surface due to its good thermal resistance while having metal away from the inner surface. The composition of the metal is therefore gradually increasing in a controlled manner starting with zero at the inner surface and to unity on the outer surface.

Fig. 1-(b) shows the information flow in the FEA-based design and fabrication system for heterogeneous objects under developing. Based on the given geometry and material information, a heterogeneous solid model is first constructed by Heterogeneous Solid Modeler (HSM). The material information (primary materials and

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 663 – 669, 2006.
© Springer-Verlag Berlin Heidelberg 2006

composition functions) is chosen from the libraries already available in the field of material science. The solid model is then converted into an object model by discretization that changes continuous material variation into stepwise variation. In addition, appropriate material properties are mapped onto the object model by property estimation rules. Next, the object model is adaptively meshed and converted into an FE model. FEA using ANSYS software is finally performed to estimate stress levels. If FEA results are not satisfactory, the coefficients of the composition function chosen as an initial guess are modified and the design cycle is repeated until a satisfactory solution is obtained. Once a satisfactory design is obtained, the object model is fed into the process planner [2] that generates data for driving LM machines.

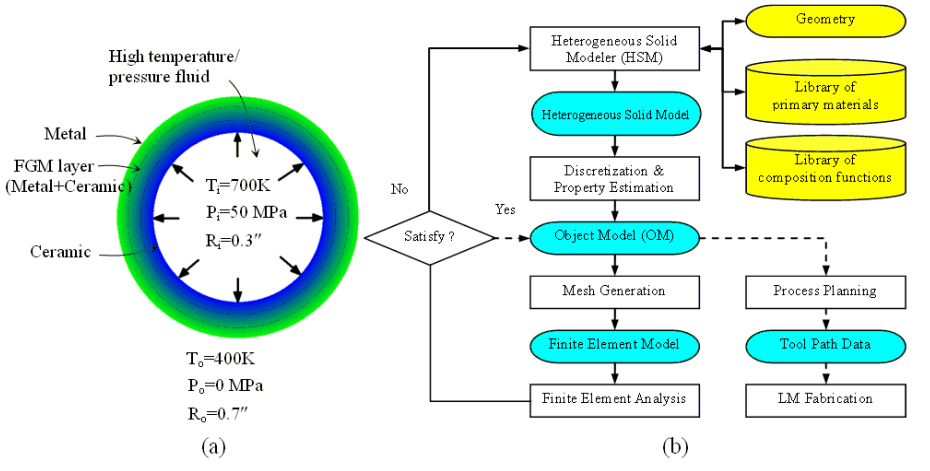


Fig. 1. An integrated CAD/CAM system for heterogeneous objects

As shown in Fig. 1, FEA is an important step for the optimal design of heterogeneous objects. However, the accuracy of the FEA-based design wholly depends on the quality of finite element models. There thus exists an increasing need for generating finite element models adaptive to both geometric complexity and material distributions. In fact, the representation scheme remains central to the proposed system where CSG-based representation [3] is used as the core of Heterogeneous Solid Modeler. The FEA-based design of heterogeneous objects is a popular research topic extensively published. Jackson et al.[4] proposed a tetrahedral mesh model. The shape and material composition over each tetrahedron is evaluated simply as a linear interpolation of the positions and compositions of its nodes. However, this method presented little discussion of FE modeling that varied refinements to reduce or increase resolution as needed in different regions. Bhashyam et al. [1] proposed a prototype CAD system for FEA-based design of heterogeneous objects. However, this method generated an FE input file manually for the geometry and material distribution. On the other hand, Tsukanov and Shapiro [5] proposed a mesh-free method for modeling and analysis of heterogeneous media.

2 Composition-Based Finite Element Modeling

Generation of the meshes is the most important and difficult stage in FE modeling and analysis. In fact, the design, analysis, and fabrication are closely related to the modeling of heterogeneous objects. Hence, the most efficient way to generate FE models is to fully utilize the material information inside an object.

Hereinafter, an algorithm we have implemented to generate composition-based FE models will be described with the pressure vessel shown in Fig. 1-(a). First, a hollow cylinder (R_i = 0.3″, R_o = 0.7″, height = 2″) is created to construct a heterogeneous solid model. Then, two primary materials (Al_2O_3 for ceramic, Al for metal) and a parabolic composition function ($v_{Al} = ((R - R_i)/(R_o - R_i))^{\mu}$) is chosen from the library. The exponent μ will be determined based on analysis results by trial and error.

2.1 Discretization

Discretization converts a heterogeneous solid model, which has gradation, into a multi-material model of homogeneous lumps, each of which has constant volume fractions of *n* primary materials. This discretization is performed with an admissible resolution in terms of volume fractions of primary materials. This is because actual fabricable resolutions are finite (e.g., 0.05-5%, 0.1-10%, etc.). Hence, the design optimization through FEA must be performed with the discretized model.

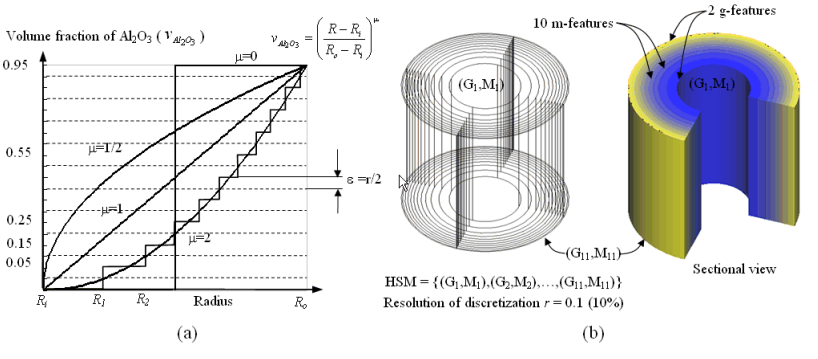


Fig. 2. Discretization of composition functions and a solid model

Each composition function has an effective function domain for variables due to the fact that volume fractions of primary materials are restricted between 0 (0%) and 1(100%), and their sum should be unity. Using this information, the algorithm determines the parameter at which the geometry is to be split. Fig. 2-(a) shows four composition functions f_1 for μ = 1, 2, 1/2, and 0 where a set of composition functions are $F(R) = [f(R), f(R)]^T = [v_{Al}, v_{Al_2O_3}]^T$. These functions are discretized by the resolution of discretization r = 0.1 (10%) in terms of the volume fraction of Al. The maximum composition error created by this intermediate discretization is half of the resolution of discretization (ε= r/2).

Once parameters R_i ($0 \leq i < 10$) are obtained, the actual geometric decomposition is then performed by sequentially applying regularized Boolean operations (differences and intersections). Finally, a single heterogeneous solid model is discretized into eleven individual homogeneous lumps $\{(G_1,M_1),\ldots,(G_{11},M_{11})\}$, each of which has volume fractions of two primary materials such as $[0.0,1.0]^T$, $[0.1,0.9]^T$, …, $[0.9,0.1]^T$ and $[1.0,0.0]^T$ (Fig. 2-(b)). In fact, this discretization converts continuous material distribution inside an object into material features (internal surfaces) like geometric features (external surfaces). Next, an object model is generated by mapping appropriate material properties onto the discretized eleven lumps. For this example, a rule of mixture is applied. More information regarding discretization can be found in [2].

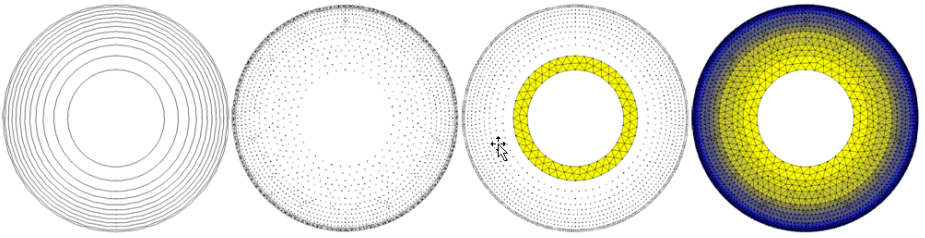


Fig. 3. 2D Mesh generation for pressure vessel

2.2 2D Mesh Generation

Fig. 3 illustrates overall four steps sequentially performed to generate a 2D FE model for the pressure vessel. The first step is the creation of primary nodes adaptively distributed on the boundary curves of the discretized models as shown in Fig. 3-(b). This step is performed by curve faceting. In fact, curve faceting is a process to determine the number of straight line segments that are required to approximate iso-material contours and regions in space with admissible approximation error.

Once the primary nodes are adaptively distributed on the boundary curves (i.e. iso-material contours) of the discretized models, triangular meshes are created by the well-known DT (Delaunay Triangulation) [6, 7] on each iso-material region as shown in Fig. 3-(c). During this triangulation, the secondary nodes are inserted within the boundaries of each iso-material region if necessary. After triangulation, Laplacian smoothing technique is then applied to the meshes not well-shaped and reposition them. This repositioning is also iteratively done. If necessary, more subdivision steps can be applied to the interesting regions where FEA results may be critical.

Repeatedly applying the same algorithm to all discretized homogeneous lumps, eleven regional meshes are sequentially generated. Concatenating all regional meshes without overlapping, a complete mesh model is then obtained as shown in Fig. 3-(d). Finally, material properties, already assigned to each homogeneous lump, are mapped onto each element and an FE model for pressure vessel is created. In particular, the proposed method keeps the iso-material contour before and after mesh generation, and thus more reliable analysis result might be obtained.

2.3 3D Mesh Generation

Based on the proposed 2D mesh generation, the similar algorithm is applied to 3D FGMs by replacing curve faceting with surface faceting, thus generating tetrahedral meshes instead of triangular meshes. Fig. 4 shows the overall procedure to generate a 3D FE model for the pressure vessel. First, boundary curve faceting is performed to the top and bottom circular planes. Piecewise planar polygons are then generated on the side cylindrical surface (Fig. 4-(a)). Next, regional triangular meshes are generated on four surfaces, the top & bottom circular planes and two cylindrical surfaces (Fig. 4-(b)). Tetrahedral meshes are then created on each iso-material region by 3D Delaunay triangulator (Fig. 4-(c)). Finally, concatenating all tetrahedral meshes, a complete 3D FE model is obtained (Fig. 4-(d)).

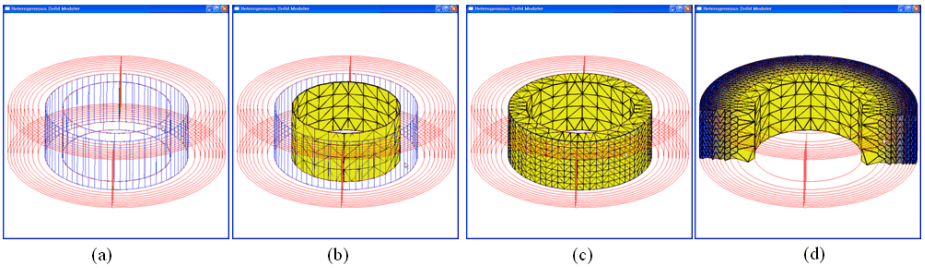


Fig. 4. 3D FE model for the pressure vessel ($\mu = 2$)

3 Examples

As part of the integrated CAD/CAM system for heterogeneous objects shown in Fig. 1-(b), a composition-based FE modeling algorithm is implemented in C++ using ACIS kernel and the GUI is implemented using MFC and OpenGL libraries.

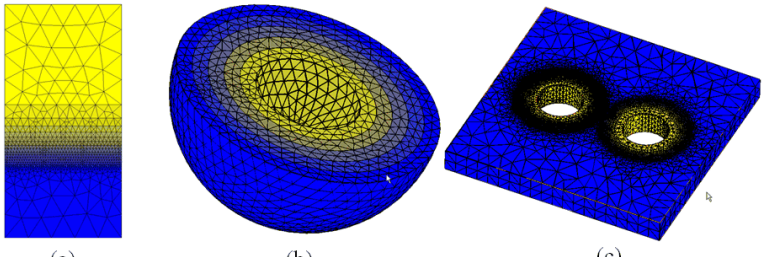


Fig. 5. FE models for freeform heterogeneous objects

Fig. 5 shows three FE models created by the proposed method. Three different types of composition functions are defined over the design domains. Fig. 5-(a) shows an FGM layer where a parabolic composition function is defined in Cartesian coordi-

nates. A ball joint slide part is shown in Fig. 5-(b) where a linear function is defined in spherical coordinates. Fig. 5-(c) shows an FGM block with two holes from which a distance-based function is defined.

4 FEA-Based Design of the Pressure Vessel

In this section, we describe the overall design cycle for the pressure vessel shown in Fig. 1-(a). The heterogeneous solid model is first discretized by r = 0.1 (10%) and then gets converted into an object model by mapping material properties onto each iso-material region. The object model is then fed to the mesh generator, thus being converted into an FE model. For FE models, four values of μ = 1(linear), 2, 1/2, and 0(direct bonding) are chosen to estimate the effect of the exponent as shown in Fig. 6. In particular, an FE model for μ = 0 is also created to estimate the stress distribution when two materials are directly bonded.

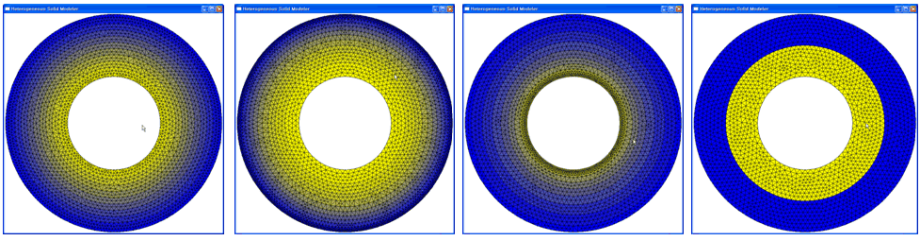


Fig. 6. FE models for pressure vessel (a) μ=1,(b) μ=2,(c) μ=1/2, (d) μ=0

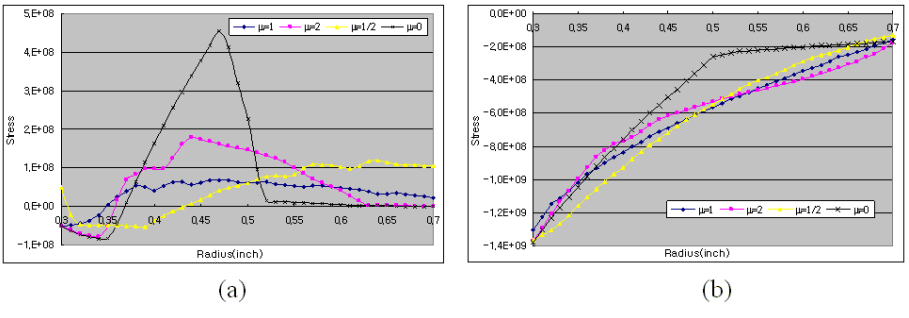


Fig. 7. Stress distribution (a) Circumferential stress, (b) Radial stress

The FE models are then used for a linear elastic analysis using ANSYS. First, FE models are fed to ANSYS in the form of three input files, a node file, an element file, and a material file. These files are automatically generated while an FE model is created. Then, the loading and boundary conditions given in Fig. 1-(a) are applied. Each analysis consists of two steps, thermal analysis and structural analysis. The resulting stress distributions for different values of μ are shown in Fig. 7. The radial stress

distributions are similar to one another, however, a minimum peak of circumferential stresses occurs at $\mu = 1$. Hence, the linear composition gradient (μ=1) will be the conclusion of the design process. The conclusion conforms to our expectation because we use a simple rule of mixture for property estimation. In particular, the circumferential stress for $\mu = 0$ (direct bonding) abruptly changes around R = 0.5″, and thus delamination might occur at the interface between Al_2O_3 and Al as expected.

5 Conclusion

This paper introduces an iterative method for FEA-based design of heterogeneous objects. At the design stage, a heterogeneous solid model is first created by referring to the libraries of primary materials and composition functions. The heterogeneous solid model is then discretized into an object model onto which appropriate material properties are mapped. Discretization converts continuous material variations inside an object into stepwise variations. Next, the object model is adaptively meshed and converted into a finite element model. FEA using ANSYS software is finally performed to estimate stress levels. This FEA-based design cycle is repeated until a satisfactory solution is obtained. An example (FGM pressure vessel) is shown to illustrate the entire FEA-based design cycle. The current implementation considers 2D problems and simple 3D problems, and thus needs extension to include complicated 3D problems.

References

1. Bhashyam, S., Shin, K. H., and Dutta, D.: An Integrated CAD System for Design of Heterogeneous Objects, Rapid Prototyping Journal, Vol. 6, No. 2 (2000) 119-135
2. Shin, K. H., and Dutta D.: Process-Planning for Layered Manufacturing of Heterogeneous Objects Using Direct Metal Deposition, ASME Journal of Computing and Information Science in Engineering (JCISE), Vol. 2, No. 4 (2002) 330-344
3. Shin, K. H. and Dutta, D., 2001, “Constructive representation of heterogeneous objects”, ASME Journal of Computing and Information Science in Engineering (JCISE), Vol. 1, No. 3 (2001) 205-217
4. Jackson, T. R., Liu, H., Patrikalakis, N. M., Sachs, E. M., and Cima, M. J.: Modeling and Designing Functionally Graded Material Components for Fabrication with Local Composition Control, Material & Design, Vol. 20, No. 2/3 (1999) 63-75
5. Tsukanov , I. and Shapiro, V.: Meshfree Modeling and Analysis of Physical Fields in Heterogeneous Media, Journal of Advances in Computational Mathematics, Vol. 23, No. 1-2 (2005) 95-124
6. Lee, D. T. and Schachter B.: Two algorithms for constructing Delaunay triangulations, International Journal of Computing and Information Science, Vol. 9, No. 3 (1980) 219-242
7. Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. T.: The Quickhull algorithm for convex hulls, ACM Trans. on Mathematical Software, Vol. 22, No. 4 (1996) 469-483

Time-Varying Volume Geometry Compression with 4D Lifting Wavelet Transform

Yan Wang and Heba Hamza

NSF Center for e-Design, University of Central Florida, Orlando, FL 32816-2996, USA
{wangyan, hhamza}@mail.ucf.edu

Abstract. Geometry compression is an effective way to distribute high-volume geometry data within limited bandwidth and storage capacity. In this paper, a new time-varying 3D geometry compression method based on 4D lifted wavelet transform is presented. In this hybrid approach, geometric information and animation are compressed based volume grid values. Isosurfaces are reconstructed from decompressed grid values. With rescaling and integer-to-integer lifting, compression ratio is significantly increased without compromising quality of surfaces.

1 Introduction

There has been extensive research on 3D static geometry compression in the past decade [1,2]. In general, there are two approaches, mesh oriented and image oriented. In mesh oriented approach (e.g., [3]), both geometry (vertex coordinates in Euclidean space) and topology (connectivity among vertices) information is compressed. Accurate 3D meshes can be reconstructed, which is ideal for engineering applications. However, connectivity of mesh cannot be changed, which restrains it from general applications. In image oriented approaches (e.g., [4]), topology information is not considered. Volumetric geometry information is represented by voxels or points, and shapes can be compressed based on 2D image alike methods. The dynamics of topology can be captured easily, and the compression methods are general. However, to render a visually recognizable and appealing surface requires a large amount of data. A balanced approach considering these two ends will possibly introduce general solution with acceptable performance.

Visualization technologies tend to merge [5]. In the past three decades, digital signal processing has evolved from 1D to 3D data. However, only a few focus on dynamic geometry change over time. Similar to 2D video that complements 2D image, time-dependent 3D geometry can be looked as 3D video and has great potential in various applications, including communication, entertainment, scientific and medical computing, computer-aided design and engineering, as well as simulation and visualization. One can expect that 3D videos with compressed formats are standardized in the future and used as commonly as today's audio and video.

In general, there are two approaches in 3D animation compression. In the first approach, topology is assumed to be static, and there is no or small change in connectivity. Mesh compression is achieved by clustering and segmentation (e.g., Lengyel [6], Ahn et al. [7]), principal component analysis (e.g., Alexa and

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 670 – 676, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Müller [8], Sattler et al. [9]), motion predication (e.g., Ibarria and Rossignac [10], Karni and Gotsman [11]), parametric coding (e.g., Briceño et al. [12], Guskov and Khodakovsky [13]). In the second approach, topology may change arbitrarily between frames. Geometric data are encoded with voxel (e.g., Ma et al. [14]) and 3D wavelet coding (e.g., Guthe and Strasser [15], Sohn et al. [16]).

Different from the above methods, we propose a 3D animation compression scheme based on 4D lifted wavelet transform (LWT), which considers spatial and temporal coherence simultaneously. This method focuses on dynamic volume geometry compression with isosurface construction. Topology will not be coded as in mesh compression, as in many cases the connectivity exists purely for rendering purpose and it is not as essential as geometry information. Isosurface representation can reduce the data size for surface boundary reconstruction. With compressed geometry as well as related surface information such as normal directions and colors, it is sufficient to reconstruct surfaces for visualization with relatively low density volume information. In the rest of the paper, Section 2 gives the overview of the compression scheme. Sections 4 and 5 compare the floating point and integer lifting based on 4D LWT. Section 6 describes motion compensation issues in this scheme.

2 Overview of Proposed Compression Scheme

The volume data to be compressed is assumed to be regularly sampled, which is commonly used in scientific and medical visualization. Isosurface construction based on volume data can provide a realistic rendering and can also be applied in other visualization environments with artifacts and natural objects. The framework described here is based on the combination of volumetric data and isosurfaces, as illustrated in Fig. 1. 4D volumetric data is divided into groups of frames (GOFs) with no special requirement on GOF boundary. Each GOF is decomposed and compressed with 4D LWT at the server side. When received by client, the compressed data is decoded and decompressed. Isosurface is constructed based on the decompressed 4D volume data. The advantages of wavelet-based approaches include scalability and simultaneous redundancy reduction for spatial and time domains. With the inherent scalable representation, wavelet provides multi-resolution solution without extra costs. 4D LWT captures coherence locality of spatial and temporal domains. There is no need to divide intraframe data into blocks, as in MPEG-2 standards based on discrete cosine transform (DCT).

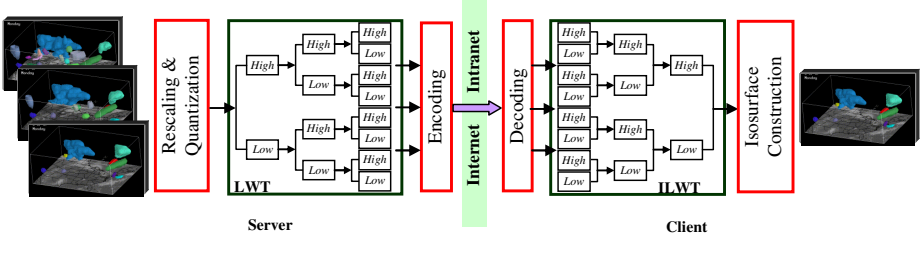


Fig. 1. Volume animation scheme based on 4D lifted wavelet transform

3 4D Lifted Wavelet Transform (LWT)

Lifting scheme [17] is also called second generation wavelet transform. It is a two-step filtering process: prediction and update. In the prediction step, even sequences are used to predict odd sequences. The prediction error forms the corresponding high-pass subband. In the update step, an approximation subband is obtained by updating even sequences with the scaled high-subband samples, generally described as

$$\begin{aligned} h_k[\mathbf{x}] &= f_{2k+1}[\mathbf{x}] + \sum_i p_i f_{2(k-i)}[\mathbf{x}] \\ l_k[\mathbf{x}] &= f_{2k}[\mathbf{x}] + \sum_j u_j h_{k-j}[\mathbf{x}] \end{aligned} \tag{1}$$

where $f_k[\mathbf{x}]$ is the input data to be processed at position $\mathbf{x}$ in frame k, h_k and l_k are resulting high-pass and low-pass sequences respectively, p_i and u_j are prediction and update coefficients of filters respectively. The main advantage of lifting is its memory efficiency in computation. Different from traditional traversal discrete wavelet transform (DWT), wavelet coefficient calculation in lifting can be embedded in-place.

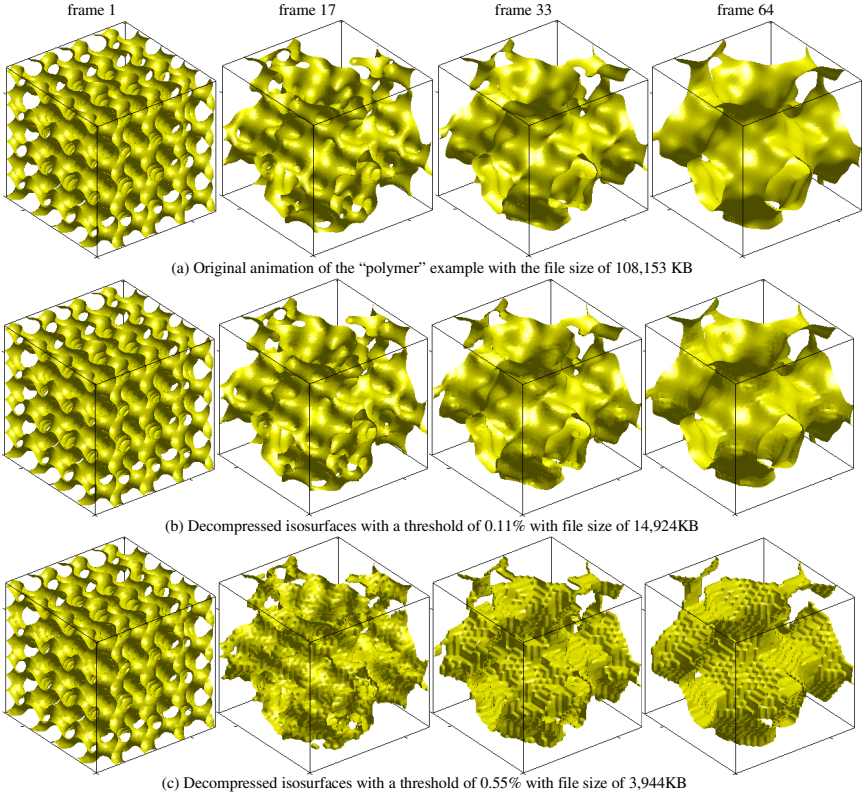


Fig. 2. *Polymer* example with the grid size 64x64x64 in each frame

Another advantage is that backward transform is easy to find and has the same complexity as the forward transform.

An important issue associated with spatio-temporal decomposition is the choice of filters. Different filters exhibit varied signal characteristics in terms of energy compactness in the transform domain and coding gain. Long filters tend to explore coherences of large regions or long period of time. However, they may blur boundary of occupation or movement. A dyadic decomposition approach is taken in this paper since it considers the coherence relation between time and space simultaneously.

An example is used to illustrate the visual effect. Fig. 2(a) shows four frames of the original 3D animation of polymer morphology, during which significant topology change is observed. A 2-level decomposition process is applied with Haar lifting scheme. Compression is achieved by setting the transformed coefficients to zero with the original values less than a threshold. Following the meta information including data type flags and dimensions of arrays, concatenated 4D arrays of coefficients form compressed files. No additional arithmetic coding is applied in the file size comparison. The highest coefficient magnitude in this example is M=89.944. If the threshold T is 0.11%, which is 0.11% of M, the decompressed surfaces are shown in Fig. 2(b) without significant visual difference. The classic marching cubes algorithm is applied in isosurface reconstruction. As T increases to 0.55%, the reconstructed surfaces are shown in Fig. 2(c). An interesting "voxel" effect occurs because the grid neighborhood with same isovalues expands. To avoid the voxel effect, resolution of grid needs to be reduced by either decreasing the density of grids at client side or taking advantage of inherent multi-resolution of wavelet transform thus transmitting only low resolution data from server side, which further increases compression ratio.

4 Rescaling and Integer-to-Integer Transform

Compared to classical wavelet transform, in which transformed wavelet coefficients are floating point numbers even if the original data are integers, lifting scheme supports lossless integer-to-integer transform [18]. It transforms integer data to integer coefficients. With inverse transform, original integer data can be reconstructed.

Another feature of the volume-based isosurface compression scheme is that isosurface construction is not sensitive to the number of bits used in coding if the range of grid values is large. As a result, floating-point grid values can be rounded to integers and integer-to-integer LWT can be used to increase the compression ratio without compromising quality. If the range of grid values is too small, the grid values can be rescaled before the rounding. A good rescaling strategy is to rescale the isosurface values to close to zero and the overall grid values to be evenly distributed between positive and negative sides. This reduces the number of bits to code values. In the example of Fig. 2, the maximum and minimum of grid values are ±7.8959. The grid values are multiplied by 100 before rounding. To reduce distortion, floor operation that rounds towards negative infinity is used. The result of integer lifting is depicted in Fig. 3, where the size of the file containing integer coefficients is 6,318KB compared to the floating coefficients of 61,900KB in Fig. 2(a) with full reconstruction. The quality of the surfaces is very close to the original ones.

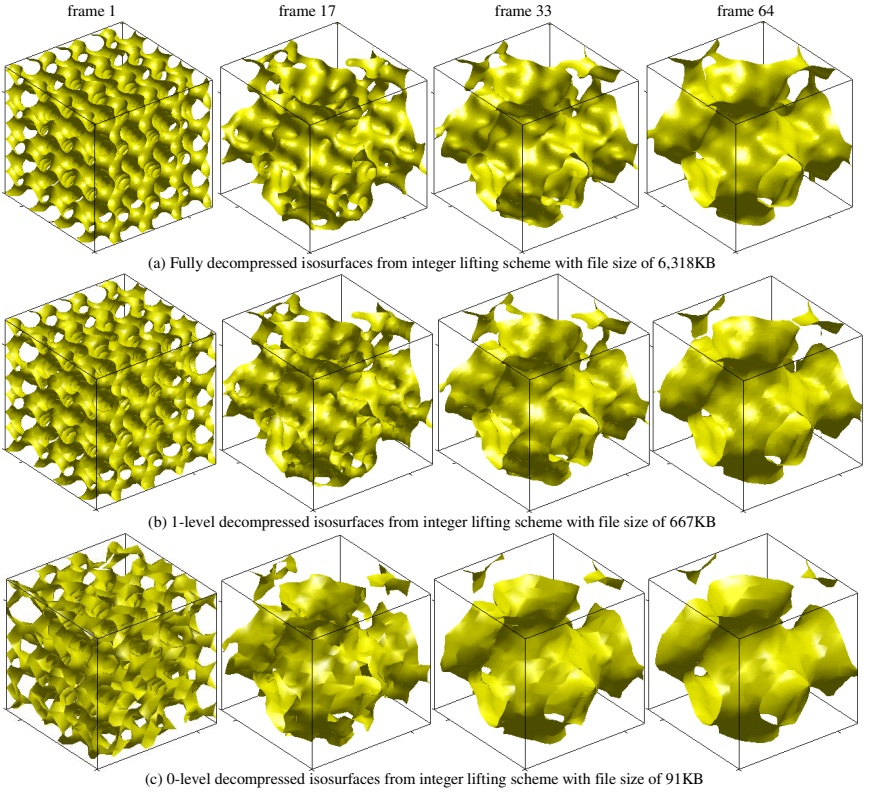


Fig. 3. The *Polymer* example with integer-to-integer 4D LWT

5 Motion Compensation

Motion compensation (MC) is to remove temporal redundancy of video signal further in 4D subband coding. Block-based motion models are predominantly used in traditional motion-compensated MPEG coding. They can accurately represent very smooth motion fields but not complex ones. In contrast, deformable mesh motion model can improve motion compensation by tracking expansions and contractions while sustaining a continuous motion field. With the notation in (1),

$$\begin{aligned} h_k[\mathbf{x}] &= f_{2k+1}[\mathbf{x}] + \sum_i p_i f_{2(k-i)}[MC_{2k\to 2k+1}(\mathbf{x})] \\ l_k[\mathbf{x}] &= f_{2k}[\mathbf{x}] + \sum_j u_j h_{k-j}[MC_{2k+1\to 2k}(\mathbf{x})] \end{aligned} \tag{2}$$

is the MC scheme in spatial domain, where $MC_{i\to j}$ denotes motion-induced transformation from frame i to frame j. In traditional 2D video, MC increases signal-to-noise ratio thus video quality with the reduction of energy in high-pass temporal subbands. Yet it may introduce distortion because of motion inversion errors.

In the context of our approach, motion of surfaces is represented by value change of discrete grids, which is more resilient than direct pixel or voxel representation. We develop a Control Grid MC model for lifting transform to study the effectiveness on our 4D LWT scheme. In each frame, the 3D space is divided into small cells. The average value of within each cell is taken to be elements of motion vectors. For example, the Haar lifting with MC is shown in Fig. 4. Odd frames are predicted by even frames with motion vectors before the lifting process where high-pass temporal subbands are generated. Low-pass temporal subbands are generated with inverse motion vectors and lifting. Final high-pass and low-pass subbands are normalized.

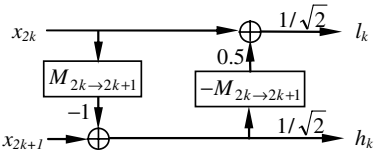


Fig. 4. MC in Haar lifting scheme

A simple "ball" example is used to test the Control Grid MC scheme. As shown in Fig. 5, shape distortion besides motion distortion is observed. As the cell size decreases, the distortion is reduced. The extreme case is that only one grid point is in each cell, which is indeed regular LWT. Compared to integer LWT, no compression gain is obtained with the motion compensation. The result indicates that block or grid based MC in our 3D video approach, in which grids are not as dense as pixel or voxel representation in 2D video, does not show significant advantages.

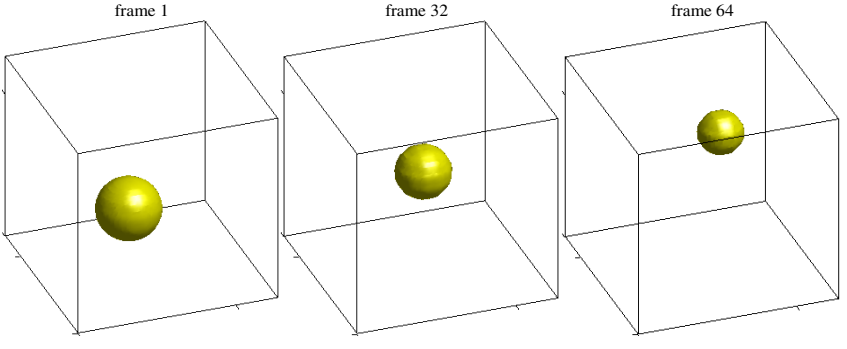


Fig. 5. Motion compensated *Ball* example shows distortion (MC cell size of 4x4x4, compressed file size is 6,238KB with motion vector included)

6 Concluding Remarks and Future Work

In this paper, a new time-varying 3D geometry compression scheme based on 4D lifted wavelet transform is presented. It is demonstrated that a hybrid approach with volume grid values and isosurfaces is feasible for 3D animation compression. Geometric information and animation are compressed based on volume grid values. Surfaces are

reconstructed from grid values and isovalues. Rescaling and integer-to-integer LWT shows significant improvement on compression ratio without compromising quality of surfaces. Blocked based motion compensation appears unnecessary.

The proposed 4D geometry compression can be used in general applications such as scientific computing and visualization, collaborative engineering, modeling and simulation, teleconferencing, and entertainment. To enable application of 4D LWT to general 3D videos, future work may include volume data editing methods to support multiple isosurface construction, content-based motion compensation, and associated motion error, as well as filter selection and comparison.

References

1. Alliez, P. and Gotsman, C.: Recent advances in compression of 3D meshes. In: *Proc. Symp. on Mutiresolution in Geometric Modeling*. 2003
2. Peng, J., Kim, C.-S. and Kuo, C.-C. J.: Technologies for 3D mesh compression: A survey. *J. Visual Communication & Image Representation*. **16** (2005) 688-733
3. Taubin, G., Horn, W., Lazarus, F. and Rossignac, J.: Geometry coding and VRML. In: *Proc. the IEEE*. **96** (1998) 1228-1243
4. Muraki, S.: Volume data and wavelet transforms. *IEEE CG&A*. **13** (1993) 50-56
5. Nielson, G.M., Brunet, P., Gross, M., Hagen, H., Klimenko, S.V.: Research issues in data modeling for scientific visualization. *IEEE CG&A*. 14 (1994) 70-73
6. Lengyel, J.: Compression of time dependent geometry. In: *Proc. 1999 ACM Symp. on Interactive 3D Graphics*. (1999) 89-95
7. Ahn, J.-H., Kim, C.-S., Kuo, C.-C. J., and Ho, Y.-S.: Motion-compensated compression of 3D animation models. *IEE Elec. Lett.*, **37** (2001) 1445-1446
8. Alexa, M. and Müller, W.: Representing animations by principle components. *Computer Graphics Forum*. **19** (2000) 411-418
9. Sattler, M., Sarlette, R., and Klein, R.: Simple and efficient compression of animation sequences. In: *Proc. EUROGRAPHICS 2005*. (2005) 209-217
10. Ibarria, L. and Rossignac, J.: Dynapack: space-time compression of the 3D animations of triangle meshes with fixed connectivity. In: *Proc. EUROGRAPHICS 2003*. (2003) 126-135
11. Karni, Z. and Gotsman, C.: Compression of soft-body animation sequences. *Computers & Graphics*. **28** (2004) 25-34
12. Briceño, H.M., Sander, P.V., McMillan, L., Gortler, S., and Hoppe, H.: Geometry videos: A new representation for 3D animations. In: *Proc. EUROGRAPHICS 2003*. (2003) 136-146
13. Guskov, I. and Khodakovsky, A.: Wavelet compression of parametrically coherent mesh sequences. In: *Proc . EUROGRAPHICS 2004*. (2004) 183-192
14. Ma, K.-L., Smith, D., Shih, M.-Y., and Shen, H.-W.: Efficient encoding and rendering of time-varying volume data. *NASA/CR-1998-208424 ICASE Report No.98-22* (1998)
15. Guthe, S. and Strasser, W.: Real-time decompression and visualization of animated volume data. In: *Proc. IEEE Visualization 2001* (2001) 349-356
16. Sohn, B.-S., Bajaj, C., and Siddavanahalli, V.: Volumetric video compression for interactive playback. *Comp. Vis. & Image Understanding*. **96** (2004) 435-452
17. Sweldens, W.: The lifting scheme: A custom-design construction of biorthogonal wavelets. *Appl. & Comp. Harmonic Analysis*. **3** (1996) 186-200
18. Calderbank, A.R., Daubechies, I., Sweldens, W., and Yeo, B.-L.: Lossless image compression using integer to integer wavelet transforms. In: *Proc. IEEE Int. Conf. Image Processing*. (1997) 596-599

A Surface Displaced from a Manifold

Seung-Hyun Yoon

Computer Science and Engineering, Seoul National University, Korea

Abstract. We present a new surface representation scheme based on a manifold structure and displacement functions. Given a geometric model represented as a point cloud, we construct a domain manifold which is a smooth surface blended from simple local patches. The original points are then projected on to the local patches and their displacements are adjusted so that the final displaced surface approximates the given point data with high precision. We demonstrate the effectiveness of our approach in various applications, including multi-resolution representations and skeleton-driven shape deformation.

1 Introduction

Thanks to recent advances in scanning technologies, it has become easy to obtain accurate and detailed geometric data from real-world objects. However, contemporary scanning systems produce a discrete representation, in the form of unorganized point clouds or polygonal meshes. Such models can have serious problems, including irregularities, discontinuities, huge size and missing areas, making it difficult to use this sort of data for practical applications.

Many mesh processing techniques have been developed to resolve these problems. Representative methods include simplification, smoothing, re-meshing and mesh optimization [4,9,10]. However, most of these techniques operate entirely in the domain of discrete geometry, and do not offer the analytic representations which are so useful in geometric modeling and processing.

Cook [1] introduced the displaced surface as an ingenious way of creating a detailed geometric model by applying a displacement map to a smooth surface. Krishnamurthy and Levoy [15] used tensor product B-spline patches to represent the underlying domain surface and created fine details with displacement vectors; but this leads to a discontinuity problem across patch and displacement map boundaries. More recently, Lee et al. [16] have proposed a framework to unify subdivision surfaces and scalar displacements. Thanks to the properties of subdivision surfaces, there are no discontinuity problems in this approach. However, subdivision surfaces do not offer analytic representations in general.

In this paper, we introduce a new representation scheme for displaced surfaces, based on the manifold technique proposed by Ying and Zorin [19]. We first construct a simple C^2-continuous smooth surface which interpolates the vertices of a given control mesh. Then, we define a scalar displacement function on each chart of an atlas of this domain. A detailed surface can now be constructed by applying scalar displacement functions to the local patches and blending them using the manifold structure of a control mesh.

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 677–686, 2006.
© Springer-Verlag Berlin Heidelberg 2006

In order to be able to address practical applications with our technique, we also propose an algorithm to approximate a detailed geometric model given as a point cloud. This approximation is based on the analytic properties of our representation scheme and the optimization of scalar displacement functions on the local charts of the domain atlas. We demonstrate the effectiveness of our approach by presenting experimental results relating to applications in multi-resolution representations and skeleton-driven shape deformation.

The main contributions of this paper are summarized as follows:

- **Surface representation:** We propose an analytic surface representation scheme which combines a manifold technique with a displacement function.
- **Shape approximation:** We present an algorithm that approximates a detailed geometric model from an unorganized point cloud. Instead of using ray-shooting, we employ a more precise method of point projection and then minimize the approximation error.
- **Applications to significant geometric problems:** The proposed technique can be applied to a variety of applications including multi-resolution representations and skeleton-driven deformation.

The rest of this paper is organized as follows. In Section 2, we review some of the related recent work. In Section 3, the construction of our displaced surface is detailed. We go on to propose an algorithm that can approximate a detailed geometric model in Section 4. Experimental results are presented in Section 5 and we conclude this paper in Section 6.

2 Related Work

We begin with a brief review of some representative approximation schemes that use displaced surfaces and also of techniques for surface modeling that are based on the manifold structure of a model.

Approximation with displaced surfaces. Krishnamurthy and Levoy [15] put forward a technique for approximating an arbitrary mesh using B-spline patches and displacement map, using vector displacements to approximate a target mesh. This technique produces a good surface fit, but there is a discontinuity problem across patch boundaries.

Lee et al. [16] proposed the displaced subdivision surface. They constructed a subdivision surface and a map of scalar displacements along its normals. The resulting detailed surface is represented using a unified subdivision framework. However, this approach does not provide an analytic representation of the displaced surface.

Jeong and Kim [13] proposed a technique for constructing a displaced subdivision surface from an unorganized point cloud. However, their method only works for objects of genus 0. Recently, Kim and Kim [14] proposed a surface reconstruction technique that works for objects of arbitrary topology. They used a butterfly subdivision surface and a moving least-squares approach to approximate an unorganized point cloud.

Manifold surfaces. Grim and Hughes [7] used a manifold concept in constructing surfaces of arbitrary topology. This technique has also been applied to the parametrization of a surface [5] and to fit a surface to point cloud data [6]. Cotrina and Pla [2] proposed a C^k surface construction algorithm which uses a regular star-shaped configuration to represent an irregular vertex and a polynomial transition function with overlapping charts. The resulting surface is of B-spline form at a boundary and may be seen as a generalized B-spline surface. A more generic approach to freeform surface generation was proposed by Cotrina et al. [3]. They can construct three different types of surfaces, but their techniques require complicated transition functions.

A simpler surface construction technique was proposed by Ying and Zorin [19]. They constructed charts in the complex plane with transition functions that have a simple analytic form. Their method provides C^∞ continuity and local support; moreover, the resulting surfaces are visually satisfactory. Recently, a new manifold spline has been proposed by Gu et al. [8], which provides a general theoretical and computational framework for manifold splines defined on a control mesh with arbitrary topology. They have also provided a practical algorithm for the construction of a triangular B-spline surface on a control mesh.

3 Displaced Surface Construction

We employ the simple manifold structure of Ying and Zorin [19]. A simple bi-quadratic local patch is constructed on each chart and a scalar displacement function is added to it. The detailed displaced surface is then constructed by blending the local patches and the scalar displacement functions in a unified way.

A local patch. Let $\mathbf{v}_i$ be the vertex of a control mesh (Figure 1(a)) and let U_i be the corresponding chart. We construct a local patch $P_i(s,t)$ on each chart U_i using the following bi-quadratic polynomial representation, which is constructed by approximating the positions of the neighboring vertices of $\mathbf{v}_i$ in the control mesh:

$$P_i(s,t) = \begin{bmatrix} s^2 & s & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{c}^i_{1,1} & \boldsymbol{c}^i_{1,2} & \boldsymbol{c}^i_{1,3} \\ \boldsymbol{c}^i_{2,1} & \boldsymbol{c}^i_{2,2} & \boldsymbol{c}^i_{2,3} \\ \boldsymbol{c}^i_{3,1} & \boldsymbol{c}^i_{3,2} & \boldsymbol{c}^i_{3,3} \end{bmatrix} \begin{bmatrix} t^2 \\ t \\ 1 \end{bmatrix}, \tag{1}$$

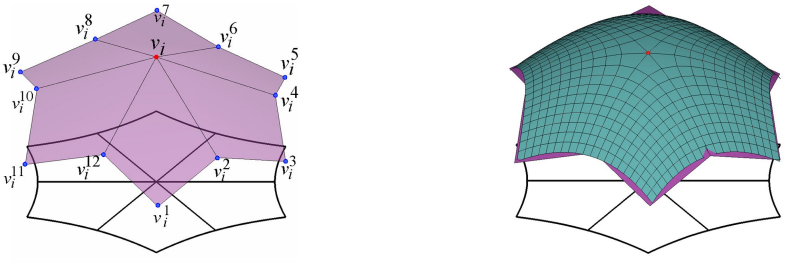
where $\boldsymbol{c}^i_{j,k} = \begin{bmatrix} x^i_{j,k} & y^i_{j,k} & z^i_{j,k} \end{bmatrix}^T$.

The coefficient vectors $\boldsymbol{c}^i_{j,k}$ of Equation (1) can be determined by minimizing the following functional:

$$L(\boldsymbol{c}^i_{1,1}, \cdots, \boldsymbol{c}^i_{3,3}) = \sum_{k=1}^{2d} \|\mathbf{v}^k_i - P_i(s_k, t_k)\|^2, \tag{2}$$

subject to $P_i(0,0) = \mathbf{v}_i$, where d is the valency of a vertex $\mathbf{v}_i$, and $\mathbf{v}^k_i$ is the k^{th} neighboring vertex of $\mathbf{v}_i$. The constraint represented by this equation forces

the coefficient $\boldsymbol{c}^i_{3,3}$ to be $\mathbf{v}_i$ in order to interpolate the position of the vertex $\mathbf{v}_i$, which is assigned to the origin of each chart. Figure 1(a) shows the vertices of a control mesh around the vertex $\mathbf{v}_i$ and Figure 1(b) shows the corresponding local patch, which interpolates the position of the center vertex $\mathbf{v}_i$.



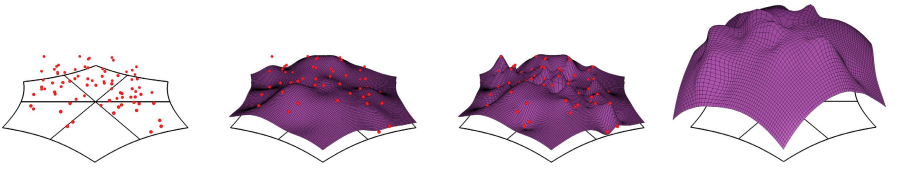
(a) (b)

Fig. 1. A local patch: (a) the vertices of a control mesh, (b) the resulting local patch

Displacement function. Once local patches have been constructed for all the charts, a scalar displacement function $d_i(s,t)$ is created on each chart U_i by interpolating the signed distances from each local patch to the point cloud data. Since these are not regular data, we used a scattered-data interpolation technique based on multi-level B-splines [17]. Figure 2(a) shows random displacements on a chart U_i, and Figures 2(b) and (c) show the corresponding displacement functions $d_i(s,t)$ at different levels of detail. Figure 2(d) shows the displaced local patch $S_i(s,t)$ generated by applying a displacement function $d_i(s,t)$ to a local patch $P_i(s,t)$. The position of a displaced point is computed as follows:

$$S_i(s,t) = P_i(s,t) + d_i(s,t) \cdot N_i(s,t), \tag{3}$$

where $N_i(s,t)$ is the unit normal vector of a local patch $P_i(s,t)$.



(a) (b) (c) (d)

Fig. 2. Displacement functions: (a) randomly generated displacements, (b) a displacement function with a low level of detail, (c) a displacement function with a high level of detail, (d) the displaced local patch $S_i(s,t)$

Displaced Surface Construction. We have now constructed a local patch and a scalar displacement function on each chart. To construct a blended surface, we

use the same transition function and blending function as Ying and Zorin [19]. Since the blending function has unit weight at its origin and zero weight at its boundary, the interpolation property of the local patches are preserved in the final blended domain surface.

The final displaced surface is achieved by applying the scalar displacement functions to the local patches. For a given parameter value z in a chart U_i, the displaced surface point $D_i(z)$ can be evaluated from Equation (4). First we need to find all the overlapping charts $\{U_j\}$ in the atlas which contains z, and then to compute the corresponding transition parameters on each chart U_j, using the transition functions $\theta_{i,j}(z)$. Then we can determine the displaced points $S_j(z)$ from Equation (3). Finally, all the displaced points are blended together smoothly using the weight functions $w_j(\theta_{i,j}(z))$ on each chart:

$$D_i(z) = \sum_{j \in J_z} w_j(\theta_{i,j}(z)) \cdot S_j((\theta_{i,j}(z)), \tag{4}$$

where z is the value of a parameter in a local chart U_i, $J_z = \{j | z \in U_j\}$ is an index set, and $w_j(\cdot)$ are the smooth blending functions that form a partition of unity.

4 Approximation to a Point Cloud

In this section, we present a new algorithm that approximates a detailed geometric model with our displaced surface. The approximation proceeds in three steps. First we generate a control mesh and the corresponding domain surface from a point cloud. Each data point is then projected on to nearby local patches and initial projection parameters and corresponding displacements are computed. Finally, they are adjusted to minimize the approximation errors and optimal displacement functions are constructed. Our contribution consists of the second and third steps, while the first step is performed using existing methods.

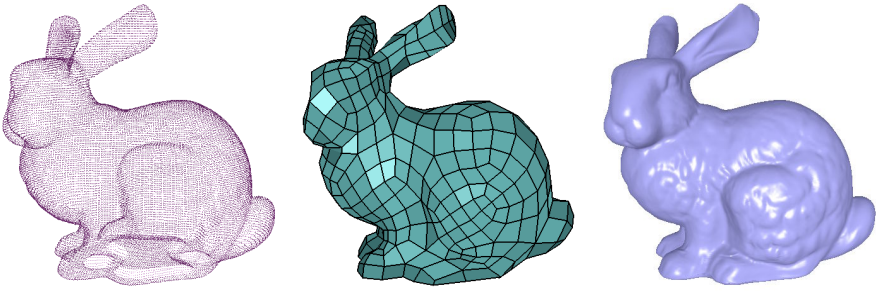
Control mesh. Many advanced techniques [9,13,14,16] exist for the construction of a good control mesh and domain surface from a detailed geometric model. We use a commercial software package [12] which is based on a combination of existing methods. Figures 3(a) and 4(a) show typical point cloud data and Figures 3(b) and 4(b) show the resulting control meshes.

Initial displacements. Once the control mesh has been constructed, a smooth domain surface is generated by blending the local patches. Scalar displacement functions are then constructed on each chart. To construct optimal scalar displacement functions, we compute initial displacements on each chart and adjust them to minimize the approximation errors. The initial displacements are computed in a different way from previous methods [13,14,16], which shoot rays from the domain surface to the detailed geometry. We project each point $\mathbf{p}_i$ on to a nearby local patch, which is considerably easier to implement and produces a more precise result than shooting methods. Assuming that we are using a quadrilateral control mesh, $\mathbf{p}_i$ will be projected on to four local patches

P_j, $j = 0, 1, 2, 3$. The use of a bi-quadratic surface representation for the local patches greatly simplifies projection.

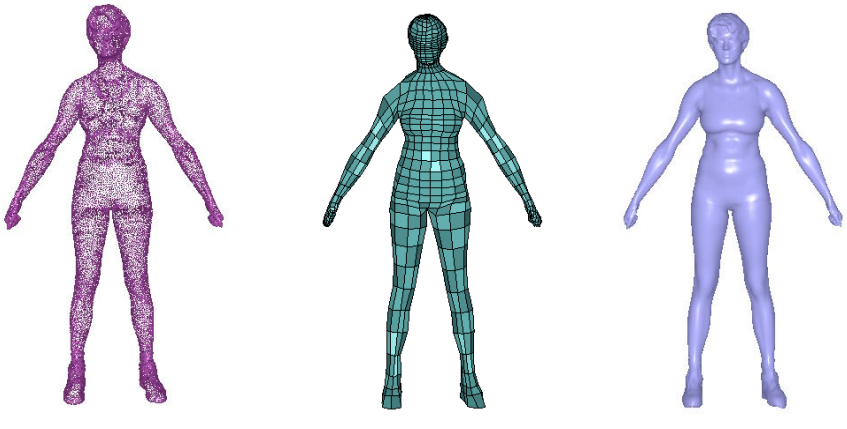
Let $\hat{\mathbf{p}}_i$ denote the projection of $\mathbf{p}_i$ on to the closest face. Then we find the vertex $\mathbf{v}_0$ of that face which is nearest to the projection point $\hat{\mathbf{p}}_i$. The local patch corresponding to $\mathbf{v}_0$ will be denoted as $P_0(s, t)$. An initial guess of the parameters of the projected point, (s_0^0, t_0^0), is made from the coordinates of $\hat{\mathbf{p}}_i$ projected on to the corresponding face of the control mesh. From the initial solution (s_0^0, t_0^0), we compute more precise parameters $(\hat{s}_0, \hat{t}_0)$ using a Newton iteration.

Once the projection parameter $(\hat{s}_0, \hat{t}_0)$ on the first patch has been obtained, the other three projection parameters $(\hat{s}_j, \hat{t}_j)$, $j = 1, 2, 3$ are guessed using the transition functions $\theta_{i,j}$. More precise solutions can be computed in a few additional steps of a Newton iteration. For robustness, we ignore any projection parameters that end up outside their chart.



(a) (b) (c)

Fig. 3. Approximation process: (a) point cloud, (b) control mesh, (c) result



(a) (b) (c)

Fig. 4. Approximation process: (a) point cloud, (b) control mesh, (c) result

The displacements $\hat{d}_j, j = 0, 1, 2, 3$ are then computed as signed distances from $P_j(\hat{s}_j, \hat{t}_j)$ to $\mathbf{p}_i$. These become the initial scalar displacements that correspond to the projection parameters $(\hat{s}_j, \hat{t}_j)$, $j = 0, 1, 2, 3$ on each chart. Using existing techniques [13,14,16] it is difficult to find the projected points since the surface does not have an analytic form; which is why previous authors have used ray-shooting. Our analytic representation greatly simplifies the projection procedure and produces considerably more precise results, by the use of an optimization technique to be discussed below.

Optimal displacement function. In general, the four projection parameters $(\hat{s}_j, \hat{t}_j)$, $j = 0, 1, 2, 3$ on each chart do not satisfy the transition relations. Each initial scalar displacement only represents the exact displacement of the point data from one local patch. When the four displacements are blended together, the resulting surface may not interpolate the original point data exactly. This approximation error is due to a mismatch in the transition relations among the projection parameters $(\hat{s}_j, \hat{t}_j)$, $j = 0, 1, 2, 3$. To minimize the approximation error, we need to compute optimal parameters (s_j, t_j) which satisfy the transition relations and their corresponding displacements d_j, $j = 0, 1, 2, 3$. These optimal parameters and displacements are obtained by minimizing the following error functional:

$$\Psi(s_0, t_0, d_0, d_1, d_2, d_3) = \|\mathbf{p}_i - \sum_{j=0}^{3} w_j(s_j, t_j) \cdot (S_j(s_j, t_j) + N_j(s_j, t_j) \cdot d_j)\|^2, \quad (5)$$

where the parameters (s_0, t_0) are initialized to the projection parameter $(\hat{s}_0, \hat{t}_0)$, and $(s_j, t_j) = \theta_{i,j}(s_0, t_0)$ are its transition parameters. The displacements d_j are initialized to $\hat{d}_j$, $j = 0, 1, 2, 3$. From the initial projection parameters and displacements, we can then find the optimal ones which minimize Equation (5). This is a general non-linear optimization problem in 6-dimensional space, which is solved by a direction-set method [18]. Although the optimal parameters may not be the exact projection parameters on to local patches, they represent the least-square minimization of the approximation error.

The optimal parameters and displacements are assigned to each chart, and optimal scalar displacement functions are then constructed by interpolating the assigned displacements.

5 Experimental Results

We implemented our displaced surface algorithm in C++ on a Pentium-IV (3.2GHz) desktop PC with a 2GB main memory. Figures 3 and 4 illustrate the approximation process and its results on two different models. The processing time can be divided into two parts: at the vertex projection stage all the vertices are projected on to nearby local patches, which involves the calculation of optimal projection parameters and displacements; and at the second stage optimized displacement functions are constructed on each chart. Table 1 itemizes processing times for each stage, together with the number of points in the point cloud data and the control meshes.

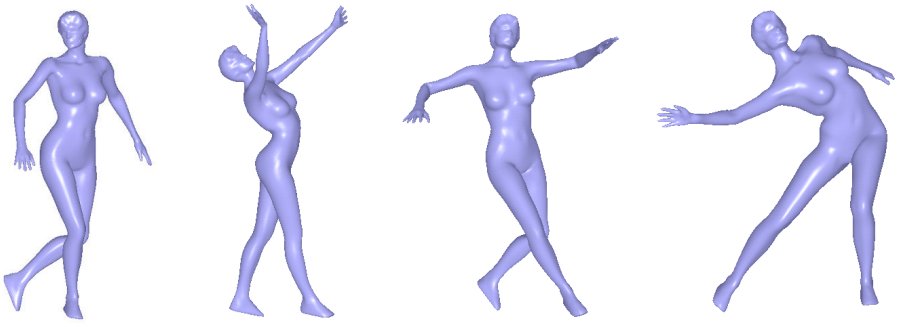


Fig. 5. Skeleton-driven deformation results

Fig. 6. Multi-resolution representations

Table 1. Approximation results

Models	Bunny	Human	Armadillo
Number of data points	34,834	43,817	159,103
Vertices on control mesh	540	1,134	720
Time for projection and optimization (sec)	221.07	318.72	813.72
Time for displacement functions (sec)	1.43	1.95	4.21
Approximation error without optimization	$1.78 * 10^{-3}$	$6.86 * 10^{-4}$	$7.84 * 10^{-3}$
Approximation error with optimization	$1.52 * 10^{-4}$	$1.23 * 10^{-4}$	$2.78 * 10^{-4}$

To demonstrate the effectiveness of our approach in practical applications, we applied the sweep-based deformation technique of Hyun et al. [11] to the approximation of a human model. Figure 5 shows clips from an animation made using this technique. It demonstrates that our displaced surface representation can produce continuous and natural deformations of a human model.

Our technique can also be used to construct multi-resolution representations. Our scalar displacement functions are in the form of a multi-level B-spline representation, which allows us to change their level of detail at runtime. Figure 6 shows an Armadillo model represented by varying levels of the displacement

functions. We analyzed the approximation errors (average distances) for each model (normalized to unit cube size), and they are listed in the last two rows of Table 1.

6 Conclusion

We have introduced a new displaced surface representation based on a manifold structure. Our representation consists of simple local patches and scalar displacement functions on each chart of a control mesh. These two geometries are smoothly blended to represent a detailed surface in a unified way. Moreover, our domain surface has a convenient interpolation property for the vertices of the control mesh and our scalar displacement function supports multi-level representations.

We have also presented an algorithm for approximating a detailed geometric model given as a point cloud. This algorithm is different from existing techniques in that our approach is based on vertex projection and the optimization of displacement functions. The vertex projection technique eliminates the need for connectivity information in the model, and an optimization procedure produces precise results. We present experimental results which demonstrate the effectiveness of our approach in applications such as multi-resolution modeling and skeleton-driven deformation.

Acknowledgements. The authors would like to thank the anonymous reviewers for their invaluable comments. This work was supported by the Korean Ministry of Information and Communication (MIC) under the Program of IT Research Center on CGVR.

References

1. R. Cook. Shade trees. *Proceedings of ACM SIGGRAPH*, 223–231, 1984.
2. J. Cotrina and N. Pla. Modeling surfaces from planar irregular meshes. *Computer Aided Geometric Design 17*, 1, 1–15, 2000.
3. J. Cotrina, N. Pla and M. Vingo. A generic approach to free form surface generation. *Proceedings of the ACM Symposium on Solid Modeling and Applications*, 35–44, 2002.
4. M. Eck, T. DeRose, H. Duchamp, M. Lounsbery and W. Stuetzle. Multiresolution analysis of arbitrary meshes. *Proceedings of ACM SIGGRAPH*, 303–312, 1995.
5. C. Grim. Simple manifolds for surface modeling and parameterization. *Proceedings of Shape Modeling International*, 237, 2002.
6. C. Grim, J. Crisco and D. Laidlaw. Fitting manifold surfaces to 3D point clouds. *Journal of Biomechanical Engineering 124*, 1, 136–140, 2002.
7. C. Grim and J. Hughes. Modeling surfaces of arbitrary topology using manifolds. *Proceedings of ACM SIGGRAPH*, 359–368, 1995.
8. X. Gu, Y. He and H. Qin. Manifold spline. *Proceedings of the ACM Symposium and Solid and Physical Modeling*, 27–38, 2005.

9. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle. Surface reconstruction from unorganized points. *Proceedings of ACM SIGGRAPH*, 71–78, 1992.
10. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle. Mesh optimization. *Proceedings of ACM SIGGRAPH*, 19–26, 1993.
11. D.-E. Hyun, S.-H. Yoon, J.-W. Chang, J.-K. Seong, M.-S. Kim and B. Jüttler. Sweep-based human deformation. *The Visual Computer 21*, 8, 542–550, 2005.
12. INUS TECHNOLOGY. *RapidForm Users Manual*, 2006.
13. W.-K. Jeong and C.-H. Kim. Direct reconstruction of displaced subdivision surface from unorganized points. *Graphical Models 64*, 2, 78–93, 2002.
14. S.-J. Kim and C.-H. Kim. Point cloud approximation by displaced butterfly subdivision surfaces. *Proceedings of Israel-Korea Bi-National Conference*, 5–10, 2005.
15. V. Krishnamurthy and M. Levoy. Fitting smooth surface to dense polygon meshes. *Proceedings of ACM SIGGRAPH*, 313–324, 1996.
16. A. Lee, H. Moreton and H. Hoppe. Displaced subdivision surfaces. *Proceedings of ACM SIGGRAPH*, 85–94, 2000.
17. S.-Y. Lee, G. Wolberg and S.-Y. Shin. Scattered data interpolation with multilevel B-splines. *IEEE Transactions on Visualization and Computer Graphics 3*, 3, 228–244, 1997.
18. W.-H. Press, S.-A. Teukolsky, W.-T. Vetterling and B.-P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.
19. L. Ying and D. Zorin A simple manifold-based construction of surfaces of arbitrary smoothness. *Proceedings of ACM SIGGRAPH*, 271–275, 2004.

Smoothing of Meshes and Point Clouds Using Weighted Geometry-Aware Bases

Tim Volodine[1], Denis Vanderstraeten[2], and Dirk Roose[1]

[1] KULeuven, Department of Computer Science, Celestijnenlaan 200A, B-3001, Leuven, Belgium
{timv, dirkr}@cs.kuleuven.be
[2] Metris N.V., Interleuvenlaan 86, B-3001, Leuven, Belgium

Abstract. In [1] Sorkine et al. proposed a least squares based representation of meshes, which is suitable for compression and modeling. In this paper we look at this representation from the viewpoint of Tikhonov regularization. We show that this viewpoint yields a smoothing algorithm, which can be seen as shape approximation using *weighted* geometry aware bases, where the weighting factor is determined by the algorithm. The algorithm combines the Laplacian smoothing approach with the smoothing spline approach, where a global deviation constraint is imposed on the approximation. We use the generalized Laplacian matrix to measure smoothness and show how it can be modified in order to obtain smoothing behavior similar to that of curvature flow and feature preserving smoothing algorithms. The method is applicable to meshes, polygonal curves and point clouds in arbitrary dimensional spaces.

1 Introduction

Shape smoothing is often a necessary pre-processing step for the manipulation of 3D models. Many 3D models today are obtained by laser scanners, touch probes and structural light range-finding techniques, which produce noisy point clouds and meshes. Obviously, for further processing and visualization it is desirable to reduce the measurement noise. In addition smoothing algorithms often have the effect of data reduction, since it is easier to encode a smooth shape, than a noisy one. Smoothing also greatly improves the reliability of derived shape properties such as normals and curvatures.

An important class of mesh smoothing algorithms is based on the discrete Laplacian smoothing approach [2,3,4]. These algorithms are typically iterative and can be seen as an approximation of a diffusion process, where at each iteration each vertex moves in the direction of the Laplacian vector at that vertex. Since the Laplacian vector at a vertex is usually estimated using a local neighborhood, the resulting algorithms are efficient and simple to implement.

The use of a uniform approximation of the Laplacian operator [2] results in what we call the *classical* Laplacian smoothing algorithm, where each vertex moves towards the center of its neighbors. When the mesh is irregular, weighted combinations of neighbors can be used to obtain better approximations of the

M.-S. Kim and K. Shimada (Eds.): GMP 2006, LNCS 4077, pp. 687–693, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Laplacian vector [4] or to discretize the curvature flow approach [3]. We refer to these methods as *weighted* Laplacian smoothing algorithms.

In this paper we describe an algorithm similar to the weighted Laplacian smoothing approach but with additional features. While the Laplacian smoothing algorithms are iterative and do not have a natural stopping criterion, the proposed algorithm is formulated as an optimization problem and provides a global deviation constraint by means of a user-specified parameter τ, called the *smoothing factor*. This parameter controls the extent of smoothing, such that the *average quadratic deviation* of each smoothed point from its original position is limited to τ/n, with n the number of points. This kind of deviation constraint is used in the literature on smoothing splines and requires the solution of a non-linear problem. Fortunately, the optimal solution can be found efficiently by solving a number of least squares systems [5].

Another improvement is the use of the *generalized Laplacian* operator (section 2.1) with weights chosen such that vertex drift and triangle overlap is minimized. We also provide scaled versions of the generalized Laplacian operator, which make the proposed smoothing algorithm behave like *curvature flow* smoothing and *feature preserving* smoothing methods.

2 Notation and Definitions

2.1 Generalized Discrete Laplacian

We represent the input to the smoothing algorithm as a directed graph $G = (V, E)$, where $V = \{\mathbf{p}_i \in \mathbb{R}^{s\times 1}, i = 1..n\}$ is the set of vertices, and E is the set of directed edges, i.e. $E = \{E_i, i = 1..n\}$, with $E_i = \{\overline{ij} \mid j \in \mathcal{I}_i\}$ and $\mathcal{I}_i = \{j \mid \mathbf{p}_j \text{ is a neighbor of } \mathbf{p}_i\}$. Note that we only need to know the local neighborhoods of each point. A mesh can be used to define neighborhoods, but it is not required. Using the graph G we can construct a *generalized discrete Laplacian* matrix $L \in \mathbb{R}^{n\times n}$, which is defined as

$$L_{ij} = \begin{cases} -\sum_{j\in\mathcal{I}_i} w_{ij} & i = j \\ w_{ij} & i \neq j, \end{cases} \tag{1}$$

with weights $w_{ij} \neq 0$ if points $\mathbf{p}_i$ and $\mathbf{p}_j$ are neighbors and zero otherwise. When all $w_{ij} \geq 0$, the matrix L can be viewed as a coefficient matrix of a system of *convex combination equations*, written as $L\mathbf{x} = 0$. We say that L is *normalized* if its diagonal entries are -1.

We denote by $P = (\mathbf{p}_1, \mathbf{p}_2, ...\mathbf{p}_n)^\top$ the $n \times s$ matrix of all input points coordinates and by $X = (\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n)^\top$ the $n \times s$ matrix of all smoothed points. L_i, P_i and X_i denote the i-th row of L, P and X respectively. By $P^{(k)}$ and $X^{(k)}$ we denote the k-th column of P and X. With the introduced notation the generalized discrete Laplace operator at $\mathbf{p}_i$ can be written as $\Delta\mathbf{p}_i = (L_i P)^\top$.

2.2 Smoothing Using Tikhonov Regularization

Our smoothing approach can be formulated as a linear algebra problem, where we find the solution matrix X^*, such that

$$X^* = \arg\min_X \{\|LX\|_F^2\} \tag{2}$$

$$\text{subject to} \quad \|D(X-P)\|_F^2 \leq \tau, \tag{3}$$

where L is the *generalized discrete Laplacian* as in (1), $D = diag(\omega_1, \omega_2, ..., \omega_n)$ a diagonal matrix of weights and τ is the *smoothing factor*. Equation (2) represents the *smoothness condition* and the inequality (3) defines a constraint on the *average displacement* of the smoothed points. Introducing a Lagrange multiplier λ, we reformulate (2) as

$$X^* = \arg\min_X \{\|LX\|_F^2 + \lambda\|D(X-P)\|_F^2\}, \tag{4}$$

where λ can be determined *a priori*, such that constraint (3) is satisfied.

Equation (4) can be visually represented by a mass-spring system shown in figure 1. The displacement term $\|D(X-P)\|_F^2$ represents the potential energy of the springs between the original points $\mathbf{p}_i$ and the smoothed points $\mathbf{x}_i$. The smoothness term $\|LX\|_F^2$ minimizes some energy derived from the springs between the smoothed points themselves.

The discrete weighted least squares problem (4) is also known as linear *Tikhonov regularization* method in numerical literature [6]. Since the generalized discrete Laplacian is singular, minimizing $\|LX\|_F^2$ as such, is not well-defined. By supplying additional information about the solution, in the form of a λ-weighted displacement term, the solution to (4) will be unique for any $\lambda > 0$.

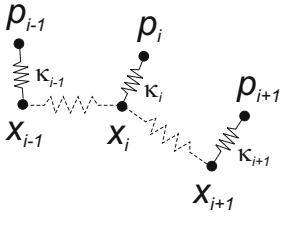


Fig. 1. Physical interpretation of the equation (4) as a mass-spring system, the stiffness of springs $\kappa_i = \omega_i^2$

3 Related Work

There is extensive literature available on both smoothing of meshes and smoothing splines [7]. We do not attempt to give an exhaustive overview of the references, instead we refer to the overview article by Belyaev et al. [8] and the references therein.

A large class of smoothing algorithms can be written using the generalized Laplacian, where in each iteration a new position $\mathbf{p}'_i$ is computed for each vertex $\mathbf{p}_i$,

$$\mathbf{p}'_i = \mathbf{p}_i + \mu(L_i P)^\top, \tag{5}$$

with $\mu \in (0, 1)$. Different weights w_{ij} for the generalized Laplacian matrix L, result in different smoothing algorithms. For example, *equal weights* and *reciprocal weights* were used by Taubin [2] to construct a low-pass filter. In the irregular setting, Desbrun et al. [3] use Fujiwara weights, which give a more accurate approximation for the Laplace operator. Guskov et al. [4] propose a non-uniform generalization of the discrete Laplacian operator using weights based on second order differences. Finally, the so-called *cotangent weights* [9] can be used to discretize the mean curvature flow approach [3].

Recently, several *feature preserving* and *non-shrinking* smoothing algorithms were proposed [10,11,12].

In [1], Sorkine et al. introduces the concept of *geometry aware bases*, as a new means of shape approximation. These bases can be used for smoothing as well, because they are related to the Tikhonov regularization approach [13]. Contrary to the approach presented in [1], the optimal λ is determined automatically during the smoothing algorithm, resulting in a method which can be viewed as an approximation method using λ-*weighted* geometry aware bases.

4 Construction of the Laplacian Matrices

In our smoothing algorithm we use positive weights for the construction of the generalized Laplacian. This choice is related to the aim to preserve the topology of the original mesh, because positive weights result in *convex combinations* of neighboring points. As λ approaches zero we expect the convex combination constraint $\|LX\|_F^2$ to be increasingly satisfied, reducing the possibility of flips.

4.1 Generalized Laplacian Matrix Variants

We introduce three different variants of the generalized Laplacian matrix:

$L^{(n)}$ – denotes the normalized Laplacian (1).
$L^{(\kappa)}$ – is the curvature scaled Laplacian.
$L^{(f)}$ – is the feature preserving Laplacian.

For the construction of $L^{(n)}$ we use the *mean-value* weights [14], because they are always positive. In this section we show how the matrices $L^{(\kappa)}$ and $L^{(f)}$ are derived from $L^{(n)}$. First, let us define the *length normalized* Laplacian matrix, for which the i-th row is given by

$$\overline{L}_i^{(n)} = L_i^{(n)} g\left(\|L_i^{(n)} P\|_2\right). \tag{6}$$

The scalar function $g(x)$ serves the purpose to scale each row $L_i^{(n)}$ such that $(\overline{L}_i^{(n)} P)^\top$ has approximately unit length. In regions of low curvature $\|L_i^{(n)} P\|_2$ can be very small or even zero, so simply taking $g(x) = 1/x$ would lead to very large entries in $\overline{L}_i^{(n)}$, yielding a very ill-conditioned matrix. To avoid numerical problems we use the robust Huber *edge stopping* function, which is defined for positive x as

$$g(x) = \begin{cases} 1/\sigma_h & x \leq \sigma_h \\ 1/x & x > \sigma_h, \end{cases} \tag{7}$$

with some small $\sigma_h \approx 1e-7$.

The normalized Laplacian represents a weighted version of the classical Laplacian smoothing. Since the normalized Laplacian is scale-independent, it does not always properly reflect the frequencies of the mesh. If the sampling is non-uniform, the normalized Laplacian can be the same for small neighborhoods as well as for large ones. In order to solve this problem we introduce the scale-dependent version of the Laplacian, i.e. $L^{(\kappa)}$ with $L_i^{(\kappa)} = \overline{L}_i^{(n)} \widetilde{\kappa}_i$, where $\widetilde{\kappa}_i$ denotes an approximation of the curvature at point $\mathbf{p}_i$. This matrix $L^{(\kappa)}$ actually mimics the *curvature flow* approach [3]. The effect of the curvature dependent smoothing can be seen in figure 3.

In some cases, however, it is desirable not to smooth points of high curvature. This kind of feature preservation can be achieved by using the feature preserving Laplacian $L^{(f)}$, which is obtained from $\overline{L}^{(n)}$ by weighting each row with a feature preserving function, $\psi(\widetilde{\kappa}_i)$, i.e. $L_i^{(f)} = \overline{L}_i^{(n)} \psi(\widetilde{\kappa}_i)$. For the feature preserving function ψ we use the Gaussian $\psi(\widetilde{\kappa}_i) = e^{-\frac{1}{2}\widetilde{\kappa}_i^2/\sigma_f^2}$, because it tends to assign zero weight to points in high curvature regions, which we consider to belong to features. The curvatures are rescaled such that $\widetilde{\kappa}_i \in [0, 1]$, and σ_f is a user-specified parameter.

5 Results

In figure 2 we apply our smoothing method to a real world example, i.e. a laser-scanned point cloud of a sheet metal part with 56K points. Figures 2(e)-2(d) show the deviations from the original after applying different smoothing methods. The light colors represent the largest deviation ($\approx 0.07\%$ of the bounding box diagonal). Note that to have a fair comparison, we have chosen the parameters of the smoothing algorithms such, that the average quadratic deviation is the same for all figures and equals $\approx 0.002\%$ of the bounding box diagonal.

In inspection and manufacturing applications it is important to have a smoothing method which preserves features. In case of figure 2 the important features are the fillets. As can be seen in figure 2(d) the points on the fillets are displaced the most during bilateral smoothing [10]. Smoothing using $L^{(n)}$ (figure 2(e)) is somewhat better and $L^{(f)}$-smoothing (figure 2(f)) preserves fillets the most.

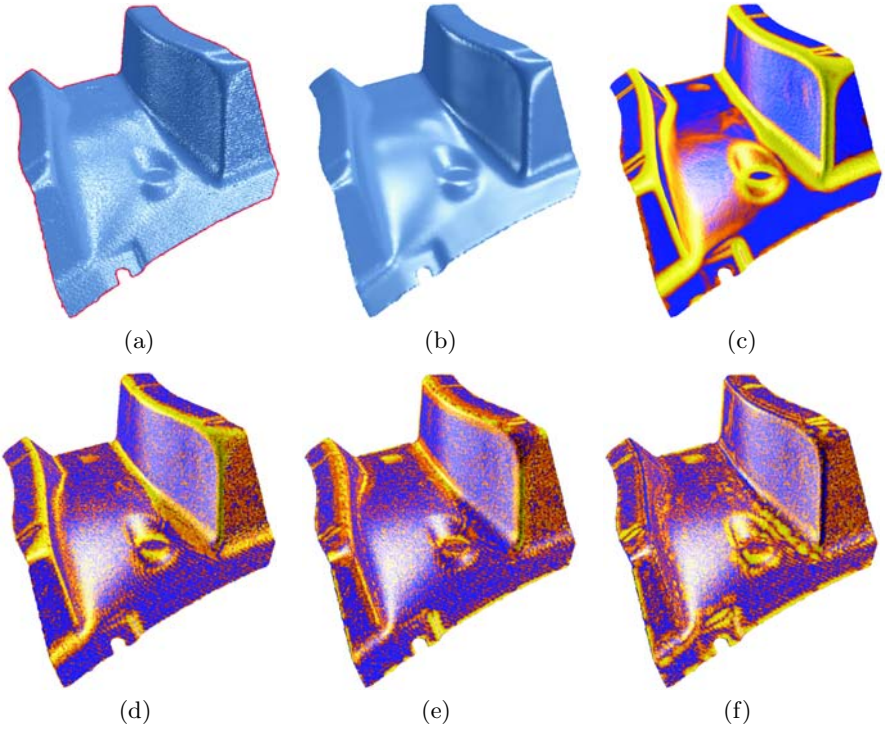


Fig. 2. (a) The original scanned noisy point cloud of a sheet metal part, visualized as a mesh; (b) $L^{(n)}$-smoothed version; (c) curvature plot; (d) difference between the original and smoothed point clouds after applying 10 iterations of bilateral smoothing; (e) difference after $L^{(n)}$-smoothing; (f) difference using $L^{(f)}$-smoothing

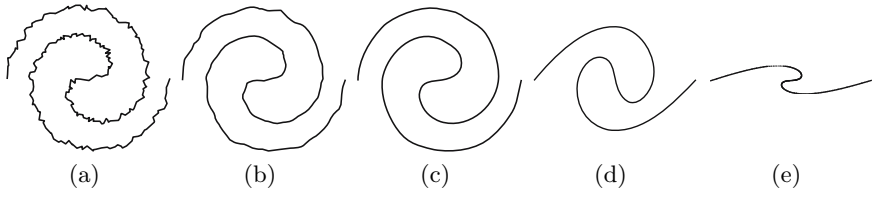


Fig. 3. (a) Non-uniformly sampled Fermat's spiral with polar equation $r^2 = \theta$, with noise; (b) smoothing using $L^{(\kappa)}$, with $\tau = 0.4$; (c) $\tau = 0.7$; (d) $\tau = 100$; (e) $\tau = 500$

6 Conclusion

We presented an algorithm for smoothing of meshes and point clouds inspired on one hand by the *geometry-aware bases*, and on the other hand by the smoothing spline approach. The results indicate that the presented method is suitable both for denoising and for modeling, where we need a smooth approximation, not deviating too much from the original data.

One of the advantages of the presented smoothing algorithm is that, once the Laplacian matrix is available, the solution can be computed very efficiently using present-day, widely available sparse linear solvers, e.g. available in Matlab. Because of its connection to Tikhonov regularization, the method can be viewed as a filter in the space of the right singular vectors corresponding to the generalized singular values. For the description of computational aspects and the analysis of the filtering properties we refer to [5].

References

1. Sorkine, O., Cohen-Or, D., Irony, D., Toledo, S.: Geometry-aware bases for shape approximation. IEEE Transactions on Visualization and Computer Graphics **11**(2) (2005) 171–180
2. Taubin, G.: A signal processing approach to fair surface design. In: SIGGRAPH '95 Conference Proceedings, ACM Press (1995) 351–358
3. Desbrun, M., Meyer, M., Schröder, P., Barr, A.H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In: SIGGRAPH '99 Conference Proceedings, ACM Press (1999) 317–324
4. Guskov, I., Sweldens, W., Schröder, P.: Multiresolution signal processing for meshes. Computer Graphics Proceedings (SIGGRAPH 99) (1999) 325–334
5. Volodine, T., Vanderstraeten, D., Roose, D.: Smoothing of meshes and point clouds using weighted geometry-aware bases. Technical Report TW-451, Dept. of Computer Science, KULeuven (2006)
6. Hansen, P.C.: Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1998)
7. Dierckx, P.: Curve and Surface Fitting with Splines. Oxford University Press (1995)
8. Belyaev, A., Ohtake, Y.: A comparison of mesh smoothing methods. In: Israel-Korea Bi-National Conference on Geometric Modeling and Computer Graphics. (2003) 83–87
9. Pinkall, U., Polthier, K.: Computing discrete minimal surfaces and their conjugates. Experimental Mathematics **2** (1993) 15–36
10. Fleishman, S., Drori, I., Cohen-Or, D.: Bilateral mesh denoising. ACM Trans. Graph. **22**(3) (2003) 950–953
11. Jones, T.R., Durand, F., Desbrun, M.: Non-iterative, feature-preserving mesh smoothing. ACM Trans. Graph. **22**(3) (2003) 943–949
12. Bobenko, A., Schröder, P.: Discrete willmore flow. In: Symposium on Geometry Processing. (2005) 101–110
13. Tikhonov, A.: Regularization of incorrectly posed problems. In: In Sov. Math. Dokl. (1963) 1624–1627
14. Floater, M.S., Hormann, K., Kós, G.: A general construction of barycentric coordinates over convex polygons. Advances in Computational Mathematics (2004)

Author Index

Aigner, Martin 45
Ajay, Joneja 637
Akleman, Ergun 287, 602
Ar, Sigal 485

Baker, Matthew L. 235
Banerjee, Ashis Gopal 500
Belyaev, Alexander 34
Boulanger, Pierre 528
Branch, John Willian 528

Cardoze, David E. 248
Chen, David T. 553
Chen, Falai 175
Chen, Gang 545
Chen, Jianer 287
Chen, Xianming 87, 101
Chen, Xiaorui 514
Cheng, Fuhua (Frank) 545, 595
Chiu, Wah 235
Choi, Yoo-Joo 563
Cohen, Elaine 87, 101, 221, 451

Dai, Junfei 59
Damon, James 101
Daniels II, Joel 221
Date, Hiroaki 644
Demarsin, Kris 571
Deng, Jiansong 175
Dheeravongkit, Arbtip 189

Elber, Gershon 115, 143, 451, 465
Etzion, Michal 325

Facello, Michael A. 1
Fischer, Anath 485
Fishkel, Fabricio 485
Fujimori, Tomoyuki 313
Furukawa, Yoshiyuki 207

Gatzke, Timothy 578
Grimm, Cindy 578
Gu, Xianfeng David 59, 409
Guibas, Leonidas J. 129
Gupta, Satyandra K. 500, 655

Hamza, Heba 670
Han, Joonhee 609
Hanniel, Iddo 115
He, Ying 409
Higashi, Masatake 371

Ivrissimtzis, Ioannis 17

Jain, Varun 299
Ju, Tao 235
Jüttler, Bert 45, 175

Kanai, Satoshi 644
Kaneko, Takanobu 371
Kawaharada, Hiroshi 585
Keyser, John 602
Kim, Ikdong 616
Kim, Young J. 563
Kishinami, Takeshi 644
Kobayashi, Yohei 313

Lai, Shuhua 595
Landreneau, Eric 602
Langbein, F.C. 267, 465
Lee, Byung-Gook 563
Lee, Haeyoung 609
Lee, Hye-Jin 623
Lee, Jin-Koo 663
Lee, Seungyong 17
Lee, Sungyeol 609
Lee, Yeon-Ju 563
Lee, Yeunghak 616
Lee, Yunjin 17
Li, Guiqing 423
Li, M. 267
Lim, Sukhyun 623
Liu, Rong 630
Liu, Yang 73
Liu, Yong-Jin 637
Lowekamp, Bradley C. 553
Luo, Wei 59

Ma, Weiyin 157, 423
Marshall, David 616
Martin, R.R. 267, 465
Masuda, Hiroshi 207

McMains, Sara 514
Mikami, Takenori 371
Miller, Gary L. 248
Mizoguchi, Tomohiro 644
Morse, Bryan S. 553

Nasri, Ahmad H. 441
Ni, Tianyun 441

Oya, Tetsuo 371

Phillips, Todd 248
Prieto, Flavio 528
Priyadarshi, Alok K. 655

Qin, Hong 409
Quinn, J.A. 465

Rappoport, Ari 325
Riesenfeld, Richard F. 87, 101
Rivara, Maria-Cecilia 536
Roose, Dirk 571, 687

Schall, Oliver 34
Seidel, Hans-Peter 17, 34
Seong, Joon-Kyung 451
Shen, Liyong 175
Shim, Jaechang 616
Shimada, Kenji 189
Shin, Byeong-Seok 623
Shin, Ki-Hoon 663
Simpson, Bruce 536
Šír, Zbynek 45
Spitz, Steven 325
Stefanus, L. Yohanes 397
Subag, Jacob 143
Sugihara, Kokichi 585
Suzuki, Hiromasa 313

Tang, Kai 637
Terék, Zsolt 1

van Kaick, Oliver 630
Vanderstraeten, Denis 571, 687
Várady, Tamas 1
Volodine, Tim 571, 687

Wang, Hongyu 409
Wang, Kexiang 409
Wang, Wenping 73
Wang, Yan 343, 670
Wang, Yimin 385
Wang, Yusu 129

Xu, Guoliang 357

Yan, Dong-Ming 73
Yau, Shing-Tung 59
Yoo, Terry S. 553
Yoon, Jungho 563
Yoon, Mincheol 17
Yoon, Seung-Hyun 677
Yoshioka, Yasuhiro 207

Zhang, Hao 299, 630
Zhang, Qin 357
Zhang, Renjiang 157
Zheng, Jianmin 385