Lecture Notes in Artificial Intelligence 3763

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Hoon Hong Dongming Wang (Eds.)

Automated Deduction in Geometry

5th International Workshop, ADG 2004
Gainesville, FL, USA, September 16-18, 2004
Revised Papers

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Hoon Hong
North Carolina State University, Department of Mathematics
Box 8205, Raleigh, NC 27695, USA
E-mail: hong@math.ncsu.edu

Dongming Wang
Beihang University, School of Science
37 Xueyuan Road, Beijing 100083, China
E-mail: Dongming.Wang@lip6.fr

Library of Congress Control Number: 2005938552

CR Subject Classification (1998): I.2.3, I.3.5, F.4.1, I.5, G.2

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-540-31332-X Springer Berlin Heidelberg New York
ISBN-13 978-3-540-31332-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11615798 06/3142 5 4 3 2 1 0

Preface

Automated Deduction in Geometry (ADG) is a series of international workshops where active researchers exchange ideas and views, present research results and progress, and demonstrate software tools on the *intersection* between geometry and automated deduction. This volume contains several excellent papers (selected via peer review) based on the talks given at the ADG 2004 meeting hosted by the University of Florida, USA, during September 16–18, 2004 . The previous four meetings were held in Linz (2002), Zurich (2000), Beijing (1998), and Toulouse (1996).

This volume consists of 12 papers. The paper by *Laura I. Meikle* and *Jacques D. Fleuriot* shows how to prove the correctness of an algorithm for computing convex hulls, by using Hoare logic and Isabelle. The paper by *Judit Robu*, *Tetsuo Ida*, *Dorin Ţepeneu*, *Hidekazu Takahashi*, and *Bruno Buchberger* shows how to prove the correctness of an origami construction (heptagon), by using the Theorema system and Gröbner bases. The paper by *Xuefeng Chen*, *Peng Li*, *Long Lin*, and *Dingkang Wang* shows how to treat degenerate cases in geometric theorems rigorously, by introducing partitioned-parametric Gröbner bases. The paper by *Pavel Pech* shows how to derive formulas for the area and radius of cyclic polygons, by using Gröbner bases. The paper by *Lu Yang* and *Zhenbing Zeng* shows how to solve certain piano movers' problems, by using a specialized real quantifier elimination method (discriminant chains). The paper by *Daniel Lichtblau* shows how to compute curves bounding trigonometric planar maps, by using Gröbner bases and some numerical methods. The paper by *Francisco Botana* and *Tomás Recio* tackles several non-trivial problems (continuity, locus generation, proving, and discovering) arising in dynamic geometry, by using Gröbner bases and other symbolic ideas and methods. The paper by *Britta Denner-Broser* tackles other non-trivial problems (tracing and reachability) arising in dynamic geometry, by introducing an alternative method (to the standard purely algebraic method). The paper by *Tielin Liang* and *Dongming Wang* describes the design and a prototype for an object-oriented language suitable for (parametrically) computing, reasoning about, and visualizing geometric objects. The paper by *Dmytro Chibisov*, *Ernst W. Mayr*, and *Sergey Pankratov* shows how to solve the motion planning problem, by using real quantifier elimination and R-functions. The paper by *Hongbo Li* shows how to reconstruct an nD polyhedral scene from a single 2D line drawing, by using Grassmann–Cayley algebra and various other tools along with carefully chosen heuristics. The paper by *Gui-Fang Zhang* and *Xiao-Shan Gao* introduces planar generalized Stewart platforms and provides a complete characterization.

We, the editors, on behalf of the organizers, thank the speakers and the authors for their excellent talks and papers. On behalf of the speakers and the authors, we would like to thank Neil White, the General Chair of ADG 2004, for organizing the wonderful meeting, and Manfred Minimair, the Publicity Chair,

for making this emerging field known to wider communities. We would also like to thank the Program Committee members (listed on the next page) for lending all their time and expertise in ensuring the high quality of the talks and the papers. Due to all their tireless effort, the meeting was highly successful, fostering lively and insightful discussions, which certainly inspired the papers published in this volume. We eagerly look forward to meeting again in 2006 to share all the new exciting progress being made!

Hoon Hong
November 2005 Dongming Wang

Organization

Invited Speakers

Doron Zeilberger (Rutgers University, USA)
Ileana Streinu (Smith College, USA)

General Organization

Neil White (Gainesville, USA), Chair
Manfred Minimair (South Orange, USA), Publicity

Program Committee

Hoon Hong (Raleigh, USA), Chair
Andreas Dress (Bielefeld, Germany)
Christopher Brown (Annapolis, USA)
Deepak Kapur (Albuquerque, USA)
Dongming Wang (Beijing, China/Paris, France)
Franz Winkler (Linz, Austria)
Giuseppa Carrà Ferro (Catania, Italy)
Hongbo Li (Beijing, China)
Jacques D. Fleuriot (Edinburgh, UK)
Jürgen Richter-Gebert (Munich, Germany)
Laureano González-Vega (Santander, Spain)
Lu Yang (Chengdu, China)
Luis Fariñas del Cerro (Toulouse, France)
Meera Sitharam (Gainesville, USA)
Neil White (Gainesville, USA)
Quoc-Nam Tran (Beaumont, USA)
Rafael Sendra (Madrid, Spain)
Shang-Ching Chou (Wichita, USA)
Thierry Boy de la Tour (Grenoble, France)
Thomas Sturm (Passau, Germany)
Tomás Recio (Santander, Spain)
Volker Weispfenning (Passau, Germany)
Xiao-Shan Gao (Beijing, China)

Table of Contents

Mechanical Theorem Proving in Computational Geometry

Laura I. Meikle and Jacques D. Fleuriot

School of Informatics, University of Edinburgh, Appleton Tower, Crichton Street, Edinburgh, EH8 9LE, UK
{lauram, jdf}@dai.ed.ac.uk

Abstract. Algorithms for solving geometric problems are widely used in many scientific disciplines. Applications range from computer vision and robotics to molecular biology and astrophysics. Proving the correctness of these algorithms is vital in order to boost confidence in them. By specifying the algorithms formally in a theorem prover such as Isabelle, it is hoped that rigorous proofs showing their correctness will be obtained. This paper outlines our current framework for reasoning about geometric algorithms in Isabelle. It focuses on our case study of the convex hull problem and shows how Hoare logic can be used to prove the correctness of such algorithms.

1 Introduction

Computational geometry is the branch of computer science that studies algorithms for solving geometric problems [14]. It has applications in, among other fields, computer graphics, robotics, molecular biology, astrophysics and statistics. Verifying that these algorithms do indeed produce the correct output is important, particularly where they are used in mission-critical instances. Formal verification by computer would boost confidence in the algorithms and would also provide a valuable insight into how they work. However, little has been achieved in this field to date. Our aim is to build a framework for reasoning about geometric algorithms in the theorem prover Isabelle.

As convex hulls are used widely in computational geometry, we look at an algorithm for computing them in two dimensions, known as Graham's Scan [5]. We chose to carry out our formal verification in Hoare logic [7], as it provides a mechanisable formal system on which one can reason about imperative programs.

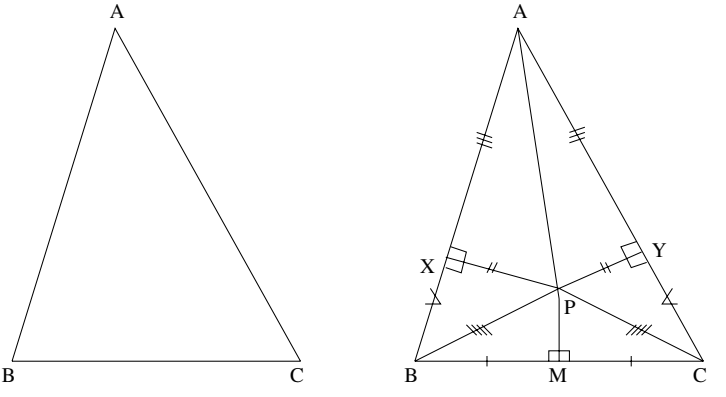
The next section outlines a few of the issues with geometric reasoning. After that we introduce the theorem prover Isabelle and describe the logic and calculus we use. Next we focus on our case study, Graham's Scan, and in particular the formal specification and verification. Finally, we conclude by discussing some related work and our goals for the future.

2 All Triangles Are Equilateral

Are all triangles equilateral? The following diagram and steps of reasoning give quite a convincing argument in favour of this statement: take any arbitrary

H. Hong and D. Wang (Eds.): ADG 2004, LNAI 3763, pp. 1–18, 2006.

triangle $\triangle ABC$ and let the perpendicular bisector of BC and the internal angle bisector of $\angle A$ meet at some point P. If we then construct points X and Y such that $PX \perp AB$ and $PY \perp AC$, then $AX = AY$ (by the angle-side-angle congruence test). Using the hypotenuse-side congruence test it can be shown that $\triangle PXB \cong \triangle PYC$, since $PX = PY$, $PB = PC$, and $\angle PXB$ is a right angle. Thus, $XB = YC$. Therefore, $AX + XB = AY + YC$. This means $AB = AC$. If we now rotate the triangle and apply the same argument on sides AC and BC, we get the conclusion that $AC = BC$. Therefore, all 3 sides of the triangle must be equal and we have proved that all triangles are equilateral.



What is the flaw in this proof? Our intuition tells us the proof is wrong, but if the result were not so obviously false we might be convinced by this spurious proof. Intuition is an important sanity check for proofs that appear logically sound.

On the flip side, if our intuition agrees with a result it may allow us to overlook missing steps or even flaws in proofs. Even Hilbert's rigorous axiomatisation of Euclidean geometry suffered from this human tendency. He argued that his proofs were free of intuition and only required the rules of logic and formal reasoning [6]. However, after formalising his work in Isabelle, we noted that Hilbert's proofs do in fact rely on intuition, through the use of diagrams and the exclusion of certain case splits [9]. We believe that if the commonly accepted "formalisation" of Euclidean geometry relies so unwittingly on intuition, then our confidence in geometric algorithms is suspect.

Diagrams in particular appeal to our intuition. They give a quasi-formal reassurance for geometric reasoning and geometric algorithms, but they can be misleading. Even if our hypotheses are tested through the use of diagrams, we still do not have complete validation of our results. In fact diagrams can be a minefield for making mistakes with undue confidence. The diagrammatic proof that all triangles are equilateral illustrates this. Although the design and verification of geometric algorithms can be aided through diagrams, clearly more rigorous reasoning is required.

A common approach to verification is simulating the algorithm on certain examples. Typically these examples are either human-selected or randomly generated. Although this method is useful, it is often subject to developer bias and can rarely verify every possible case. We believe formal verification is the surest way to have confidence in theorems, proofs, and algorithms. Furthermore, our understanding of a geometric algorithm is increased by formal verification, as it reveals inconsistencies, ambiguities and incompleteness that might otherwise go undetected.

3 The Theorem Prover Isabelle

Isabelle is a generic theorem prover, written in ML, which can be used as a specification and verification system [12]. There are a number of logics in which Isabelle allows the user to encode particular problems. Of specific interest to this work is the capacity for proofs in higher order logic (HOL). This provides a framework powerful enough to reason about algorithms and sophisticated mathematical notions. Isabelle/HOL is influenced by Gordon's HOL theorem prover [4] which itself originates from a classic paper by Church [2]. It provides an extensive library of theories and some automatic proof methods which combine simplification and classical reasoning. These tools greatly help mechanisation.

In particular, the development of Floyd-Hoare logic within Isabelle/HOL is highly relevant [10]. With this logic, the formal specifications of geometric algorithms can closely resemble their implementations.

4 Floyd-Hoare Logic

Floyd-Hoare logic is widely viewed as a way of reasoning mathematically about imperative programs [7]. The logic can not only allow verification of programs but can also aid their constructions from their specifications. Hence, it should be beneficial to reasoning about geometric algorithms in a sound and rigorous manner.

Hoare introduced a notation, called a *partial correctness specification*, for specifying what a program does: $\{P\}\ C\ \{Q\}$ is said to be true if whenever C is executed in a state satisfying P and C terminates, then the state in which C terminates satisfies Q. Total correctness is what ultimately needs to be proved when verifying a program. Informally *total correctness* = *termination* + *partial correctness.*

Although Hoare developed this logic as a means of reasoning about imperative programs, he never fully mechanised it. This was first done by Gordon [3] in the theorem prover HOL using an embedding of an annotated WHILE language in higher order logic and a verification conditions generator. In Gordon's approach, a program can be annotated to show the relationships between the variables by inserting statements called assertions. These assertions express conditions that are meant to hold at various intermediate points.

In particular, in order to formally verify programs involving loops, the partial correctness specification is annotated with mathematical statements known as invariants. An invariant R must satisfy the following conditions: it must hold initially; it must establish the result with the negated test; and the body of the program must leave it unchanged. In Gordon's system, a set of purely mathematical statements called *verification conditions* (or VCs) are then generated. A program is partially correct if the following VCs can be proven:

1. $P \rightarrow R$
2. $R \wedge$ Loop Test $\rightarrow$ body of loop preserves R
3. $R \wedge$ Negated Loop Test $\rightarrow Q$

In Isabelle, work on Hoare logic has been formalised by Nipkow [10] and will be of interest to us. In this work, the notation for representing a theorem about an imperative program with a while loop is of the general form:

```
theorem
 "|- .{ P }.
     variable assignments;;
     WHILE Loop test
     INV .{ Loop invariant }.
     DO
        program
     OD
     .{ Q }."
```

5 Geometric Preliminaries

Our mathematical framework for reasoning formally about geometric algorithms builds upon a 2D real vector theory developed in Isabelle. In this theory a real vector is a pair of real numbers, represented by (a, b). This can be interpreted in two ways; either as the point with coordinates (a, b) or as the position vector $\overrightarrow{(0,0)(a,b)}$. The vector theory defines the notions of vector addition, subtraction, dot product and scalar product. It also formalises the outer product ($\times$), defined in terms of the coordinates of the points A and B:

$$A \times B \equiv A_x * B_y \text{ - } A_y * B_x$$

Here, A_x and A_y denote the x and y-coordinates of point A respectively. Using the outer product definition, it is possible to capture the notion of the signed area of three points A, B and C:

$$\texttt{area}\ A\ B\ C \equiv (B \text{ - } A) \times (C \text{ - } A)$$

Our theory uses signed areas to identify where three points lie in relation to each other. The property of collinearity can be captured as follows:

$$\texttt{coll}\ A\ B\ C \equiv \texttt{area}\ A\ B\ C = 0$$

Another property which can be formalised is the notion of a *left turn*. If a point C lies to the left of the directed line from A to B, we write:

$$\texttt{Left\_turn}\ A\ B\ C \equiv \texttt{area}\ A\ B\ C > 0$$

As shall be seen in the following sections, testing for left turns is very useful in formalising and constructing convex hulls. Another useful property is whether a point lies between two others. We say that B lies between A and C if the following holds:

$$\begin{aligned} B\ \texttt{isBetween}\ A\ C \equiv \texttt{coll}\ A\ B\ C \quad \wedge \quad A \neq C \quad \wedge \\ \big(\forall\, D.\quad \texttt{area}\ A\ C\ D \neq 0 \quad \longrightarrow \\ (0 < (\texttt{area}\ A\ B\ D\ /\ \texttt{area}\ A\ C\ D) < 1)\ \big) \end{aligned}$$

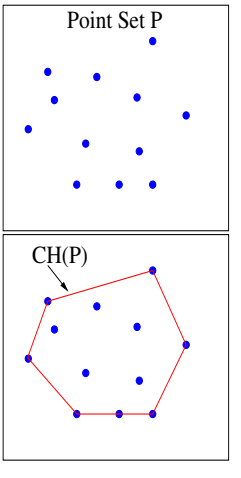
Note that we do not need to state explicitly that all the points are distinct because the *area* tests would not hold simultaneously if the points were identical.

6 Defining Convex Hulls

In this section we describe what the convex hull of a set of points is and show how we formally define it in the theorem prover Isabelle.

6.1 What Is a Convex Hull?

Many definitions exist for *convex hulls*. Intuitively, one may think of a set of points P in 2D as being nails sticking upwards from a board. Now imagine stretching a rubber band around them and letting go so its length is minimised. The region enclosed by the rubber band is known as the convex hull of P. The convex hull of P can also be described as the smallest convex set containing P. Despite the simplicity of this definition, it is not conducive to algorithm development as it is not constructive. For our formal specification we were inspired by a definition Knuth gives in his book *Axioms and Hulls* [8]. This is described in the next section.



6.2 Formal Specification of Convex Hulls

In his book, Knuth describes an orientation predicate which is equivalent to our previously defined `Left_turn`. Using this, he defines a counter-clockwise system (CC) as one which satisfies five axioms that capture the minimal properties of the orientation predicate:

1 (cyclic symmetry). $\texttt{Left\_turn}\ p\ q\ r \Longrightarrow \texttt{Left\_turn}\ q\ r\ p$

2 (antisymmetry). $\texttt{Left\_turn}\ p\ q\ r \Longrightarrow \neg\ \texttt{Left\_turn}\ p\ r\ q$

3 (nondegeneracy). $\texttt{Left\_turn}\ p\ q\ r \vee \texttt{Left\_turn}\ p\ r\ q$

4 (interiority). $\texttt{Left\_turn}\ t\ q\ r \wedge \texttt{Left\_turn}\ p\ t\ r \wedge \texttt{Left\_turn}\ p\ q\ t \Longrightarrow \texttt{Left\_turn}\ p\ q\ r$

5 (transitivity). $\texttt{Left\_turn}\ s\ t\ p \wedge \texttt{Left\_turn}\ s\ t\ q \wedge \texttt{Left\_turn}\ s\ t\ r \wedge \texttt{Left\_turn}\ s\ p\ q \wedge \texttt{Left\_turn}\ s\ q\ r \Longrightarrow \texttt{Left\_turn}\ s\ p\ r$

Although our system bears much resemblance to Knuth's, we have not adopted his axiomatic approach. We have followed Isabelle's methodology of maintaining consistency by developing new theories on top of old ones through conservative extensions only; in our case we have built upon our theory of vectors.

One drawback of Knuth's CC system for our purposes is that it disallows collinear points. This leads to many elegant results in his framework, but for real-world applications this restriction is not practical. In our formalisation, which permits collinear points, four of Knuth's axioms remain true and have been proven from first principles in Isabelle. The remaining axiom, Knuth's third, is altered and proven in our system with the obvious modification as the following theorem:

$$p \not= q \wedge p \not= r \wedge q \not= r \Longrightarrow \texttt{Left\_turn}\ p\ q\ r \vee \texttt{Left\_turn}\ p\ r\ q \vee \texttt{coll}\ p\ q\ r$$

Knuth presents an alternate version of Axiom 5:

5b. $\texttt{Left\_turn}\ s\ t\ p \wedge \texttt{Left\_turn}\ s\ t\ q \wedge \texttt{Left\_turn}\ s\ t\ r \wedge \texttt{Left\_turn}\ t\ p\ q \wedge \texttt{Left\_turn}\ t\ q\ r \Longrightarrow \texttt{Left\_turn}\ t\ p\ r$

Axiom 5b is illustrated in Figure 1 and has also been proven from first principles in Isabelle. Our mechanical proof of Axiom 5b follows the outline Knuth sketches using three applications of Axiom 5. However, allowing collinear points in our system dramatically increases the number of cases which needed to be considered. In fact we had 13 additional configurations to reason about.

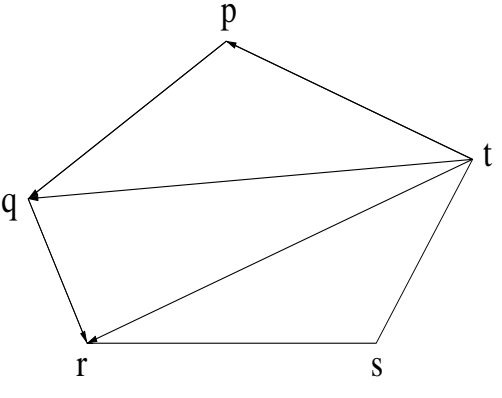


Fig. 1. Knuth's Axiom 5b

Knuth then describes the convex hull C of a set of points P, satisfying his CC system, as the set of all consecutive points ts such that `Left_turn` $t\ s\ p$ holds for all $p \notin \{s, t\}$ (see Figure 2). Clearly this definition only holds when we are traversing the hull in a counter-clockwise direction.

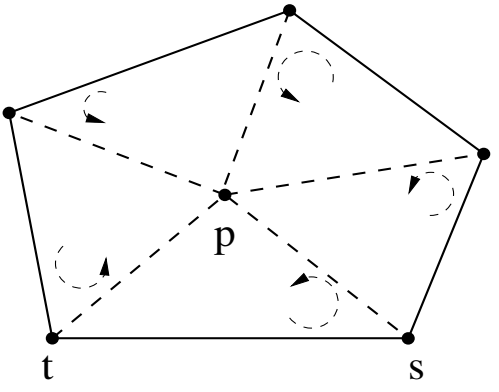


Fig. 2. Formal description of convex hull

Our formalisation of a convex hull also has to allow the possibility that any point in P could lie between two consecutive vertices. In Isabelle, we assume that the points in C are ordered clockwise and check that C is the convex hull of the points P using the infix predicate `isConvexHull`.

$$
\begin{aligned}
C\ \texttt{isConvexHull}\ P \equiv\ & \neg\texttt{all\_collinear}\ P \wedge \texttt{distinct}\ C \wedge \texttt{set}\ C \subseteq \texttt{set}\ P\ \wedge \\
& (\forall\ n < \texttt{length}\ P.\ \forall\ i < (\texttt{length}\ C\ -\ 1). \\
& \ (\texttt{Left\_turn}\ C_{i+1}\ C_i\ P_n\ \vee \\
& \ \ P_n\ \texttt{mem}\ [C_{i+1}, C_i]\ \vee \\
& \ \ P_n\ \texttt{isBetween}\ C_{i+1}\ C_i)\ \wedge \\
& \ (\texttt{Left\_turn}\ (\texttt{hd}\ C)\ (\texttt{last}\ C)\ P_n\ \vee \\
& \ \ P_n\ \texttt{mem}\ [\texttt{hd}\ C, \texttt{last}\ C]\ \vee \\
& \ \ P_n\ \texttt{isBetween}\ (\texttt{hd}\ C)\ (\texttt{last}\ C)))
\end{aligned}
$$

Note that we have represented the point sets C and P using lists and the nth member of a list C is denoted by C_n. The predicate `isConvexHull` ensures that every point in C is distinct and belongs to the original, non-collinear point set P. We then check that every point in P either makes a left turn with respect to consecutive vertices in C, or is a vertex of the hull, or lies between consecutive vertices in C. The last case in the definition simply closes the convex polygon and ensures that the first and last vertices in C satisfy the same tests carried on consecutive vertices.

In the next section we describe Graham's Scan algorithm for computing convex hulls, which was chosen for formalisation in Isabelle.

7 Graham's Scan Algorithm

Computing the convex hull of a set of points is a problem which has been greatly studied. As a result there exists an abundance of algorithms. Graham's Scan is just one which computes 2D convex hulls. We chose to formally verify this algorithm as it is familiar to researchers in the field and easily understood. It is described in the following section, followed by its formalisation in Isabelle's Hoare logic.

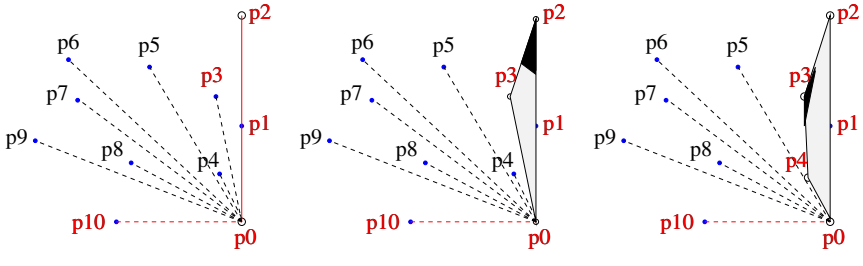
7.1 How It Works

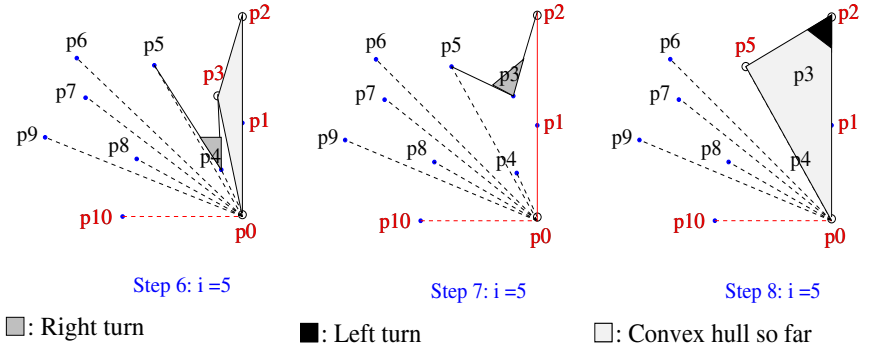
Graham's Scan uses a method known as rotational sweep and solves the problem by maintaining a stack C of candidate points. Each point of the input set P is pushed once onto the stack, and the points that are not vertices of the convex hull are eventually popped. When the algorithm terminates, the stack C contains exactly the vertices of the hull, in counterclockwise order of their appearance on the boundary (see pseudo-code below, from [11]).

GRAHAM'S SCAN ALGORITHM

```
Find rightmost lowest point; label it P_0.
Sort all other points angularly about P_0,
     break ties in favour of closeness to P_0;
     label P_1,...,P_{n-1}.
Stack C=(P_{n-1},P_0)=(P_{t-1},P_t); t indexes top.
i = 1
while i < n do
    if P_i is strictly left of (P_{t-1},P_t)
          then Push(C,i) and set i ← i + 1
          else Pop(C)
```

Notice that point P_{n-1} ends up twice on C, so a final pop is required in the algorithm. The following diagrams illustrate some of the behaviour of Graham's Scan. The hollow points are the vertices of the hull (shaded region) C at each stage.





7.2 Formal Specification of Graham's Scan

This section describes how we formally specify Graham's Scan using Hoare logic in Isabelle. We use lists to represent the point sets P and C, as Isabelle already has this data structure defined and many of its properties proved. Due to the way C is updated, the vertices of the final hull are returned in clockwise order, not counter-clockwise as in the pseudo-code of Section 7.1. The theorem that needs to be proven is specified as follows:

```
theorem
 "|- .{ ordered P & 3 ≤ length P & distinct P & ¬all_collinear P }.
     'i := 0;;
     'C := [hd P, last P];;
     WHILE 'i < length P
     INV .{ Loop invariant }.
     DO
        IF Left_turn C1 C0 Pi
        THEN 'C := Pi # 'C;;
             'i := 'i+1
        ELSE 'C := tl 'C
        FI
     OD
     .{(butlast 'C) isConvexHull P }."
```

We have not shown the loop invariant of Graham's Scan here as it is made up of numerous components which will be explained in Section 8. Note that the variables C and i are written with $'$ before them. This is to signify that they change as the program evolves. Also note that new vertices, P_i, are added to the front of the list of existing vertices, $'C$, using list concatenation ($\#$). The post-condition has to check that all the points, excluding the last one in C (hence the use of the `butlast` function), are indeed vertices of the convex hull of P. We exclude the last point of C as it would appear twice in the list otherwise.

The last three preconditions, in the statement of Graham's Scan in Isabelle, prevent us from trying to prove the algorithm is correct for degenerate cases: we must start with at least three points, they must all be distinct and the points

cannot all lie in the same line. The preconditions also state that the point set P must be *ordered*. Next, we review briefly how this notion is mechanised in Isabelle.

7.3 Ordering Points

In our formalisation of an ordered point set, we refer to a predicate `Lowest_pt`, which represents the point with the minimum y-coordinate in P. Similarly to the algorithm, we take the rightmost point in case of a tie. Instead of using trigonometric functions to sort the points by polar angle, we decided to take advantage of the properties of signed areas. We can say the points in P are ordered if:

$$\begin{aligned}&\texttt{ordered}\ P \equiv P_0 = \texttt{Lowest\_pt}\ P \wedge \\ &\qquad \forall m{<}\texttt{length}\ P.\ \forall n{<}\texttt{length}\ P.\ 0 < m \wedge m < n \longrightarrow \\ &\qquad\qquad \texttt{before}\ P_0\ P_m\ P_n\end{aligned}$$

where the predicate `before` is defined as:

$$\texttt{before}\ L\ u\ v \equiv (u\ \texttt{isBetween}\ L\ v) \vee (\texttt{Left\_turn}\ L\ u\ v)$$

This can be read as: "if L is the lowest point in P and u and v are any other two points in P, then u comes before v if it lies strictly between L and v or if the point v lies to the left of the directed line from L to u".

Many properties of the ordered predicate have been proven in Isabelle, one of which is:

$$\begin{aligned}&\texttt{ordered}\ P \wedge \texttt{distinct}\ P \wedge P \not= [] \wedge \\ &A\ \texttt{mem (take}\ i\ P) \wedge i < \texttt{length}\ P \wedge A \not= \texttt{hd}\ P \\ &\qquad\qquad \Longrightarrow \texttt{before (hd}\ P)\ A\ P_i\end{aligned}$$

where (`take` i P) represents a list of the first i elements in P.

8 Loop Invariants for Graham's Scan

Formulating the correct loop invariant for Graham's Scan is the most difficult task in this particular verification problem. This is because it requires identifying multiple components, which are sufficient to give the desired result on termination, and which are all true each time the loop is iterated. Discovering the correct components is an iterative process, where we first start with the facts we think are necessary in the proof. We then attempt to prove the VCs using these components. If our attempt is unsuccessful it will usually suggest additional information which should be added to the loop invariant.

Our first attempt at formalising the loop invariant of Graham's Scan was drawn from the written proof given by O'Rourke [11]. This written proof states that there must always be two points on the stack in order to determine whether a new point is a vertex of the hull. More specifically, the two points which the stack is initialised with, (`hd` P) and (`last` P), must be vertices of the final hull. In our formalisation, we have the corresponding loop invariant components:

1. $2 \leq \texttt{length}\ C$
2. $\texttt{last}\ C = \texttt{last}\ P$
3. $\texttt{last}\ (\texttt{butlast}\ C) = \texttt{hd}\ P$

The written proof assumes that whenever multiple points are collinear with P_0, all but the furthest are removed in a pre-processing step. With this assumption, and the ordering by polar angle, P_1 must be on the hull. Since the algorithm does not strictly require the removal of these collinear points, and because this removal in fact complicates the verification process, we have not made this assumption. As a result, we introduce a new predicate called `furthest_fst_pol_ang`. This predicate is defined as:

$$\begin{aligned}\texttt{furthest\_fst\_pol\_ang}\ P\ f \equiv\ & \texttt{ordered}\ P \wedge 2 \leq \texttt{length}\ P \wedge \\ & f < \texttt{length}\ P \wedge \texttt{collinear}\ P_0\ P_1\ P_f \wedge \\ & \neg\texttt{collinear}\ P_0\ P_1\ P_{f+1}\end{aligned}$$

In words, the point P_f is the *furthest first polar angle* point in P if P is ordered and P_f is the point furthest from P_0 on the directed line from P_0 through P_1.[1] In the mechanical proof, we also need to keep track of when this vertex appears on the hull. The written proof's claim that P_1 is on the hull becomes, in our system, the following component:

4. $\forall f.\ (\texttt{furthest\_fst\_pol\_ang}\ P\ f \wedge f < i) \longrightarrow P_f\ \texttt{mem}\ C$

The key step in the algorithm, according to the written proof, is the *strict* left turn test, where C_0 is removed from C if it is collinear with P_i (the point under consideration) and C_1. The written proof neglects the important case where C_0 is removed because of a *right* turn with respect to P_i. Our corresponding loop invariant component is:

5. $\forall\ k.\ \neg\ P_k\ \texttt{mem}\ (\texttt{butlast}\ C) \wedge k < i \longrightarrow$
   $(\forall\ j < \texttt{length}\ C - 2.\ \neg\texttt{Left\_turn}\ C_{j+1}\ P_k\ C_j) \wedge$
   $(\forall\ m.\ C_0 = P_m \wedge m < k \longrightarrow \neg\texttt{Left\_turn}\ P_m\ P_k\ P_i)$

This component notes that if we have examined a point P_k which does not belong to the list of vertices C, then it must have been removed from C at some time. Thus it will not be a vertex of the final hull. It must therefore lie to the left of all directed lines from C_{j+1} through C_j for all $j < \texttt{length}\ C - 2$ or between two adjacent vertices in the hull so far. It also states that if P_k is considered after C_0 then the current point under examination, P_i, cannot lie to the left of the directed line from C_0 through P_k. This is because if C_0, P_k, P_i did make a left turn then we would not have popped P_k.

With only these five components drawn from the written proof, it is impossible to mechanically verify Graham's Scan. The following facts are also needed:

[1] In many cases P_f will be P_1. The predicate is introduced to simplify the case where $P_0, P_1, ..., P_f$ are collinear and $1 < f$.

6. $i \leq \texttt{length}\ P$
7. $1 \leq i$
8. $3 \leq \texttt{length}\ P$
9. $\neg\texttt{all\_collinear}\ P$
10. $\texttt{ordered}\ P$
11. $\texttt{distinct}\ P$
12. $\texttt{distinct}\ (\texttt{butlast}\ C)$
13. $i = \texttt{length}\ P \longrightarrow \texttt{last}\ P = \texttt{hd}\ C$
14. $\forall\, V.\ V\ \texttt{mem}\ (\texttt{butlast}\ C) \longrightarrow V\ \texttt{mem}\ (\texttt{take}\ (i + 1)\ P)$
15. $\forall j\ k\ l.\ (j < \texttt{length}\ C - 1 \wedge l < k \wedge k < j) \longrightarrow \texttt{Left\_turn}\ C_j\ C_k\ C_l$
16. $\forall\, k < \texttt{length}\ C - 1.\ \exists\, n < \texttt{length}\ P.\ C_k = P_n \wedge$
    $((\texttt{drop}\ k\ (\texttt{butlast}\ C))\ \texttt{isConvexHull}\ (\texttt{take}\ (n + 1)\ P) \vee$
    $(\texttt{all\_collinear}\ (\texttt{take}\ (n + 1)\ P) \wedge$
    $(\texttt{length}\ C - k = 2 \vee \texttt{length}\ C - k = 3)))$

Here i corresponds to the ith point in the ordered list for which the convex hull is being constructed.

It is fairly clear that components 6 to 12 hold true on each iteration of the loop. Note that component 6 states that $i \leq \texttt{length}\ P$ and not $i < \texttt{length}\ P$, as it has to hold true on the termination of the loop.

Component 13 states that on termination of the loop, the first vertex in C is the last point in P, and component 14 states that the points in C must belong to the set of points we have already examined in P.

The final two components are a little harder to understand. The fact that the list C has a clockwise ordering is contained in component 15. It states that if we travel along say C_j to C_k, then we must make a left turn with respect to vertices of the hull added after C_k.

The final component of the loop invariant reasons about the construction of the convex hull. It says that during every iteration of the loop, the first vertex in C must be one of the points in P, say P_k. Up to P_k either we have constructed a convex hull or all the points are collinear. In the case where all the points up to P_k are collinear, C must have length 2 or 3.

It is understandable why a written proof may omit some of the above components, as they seem obvious. However, the process of formal verification highlights the assumptions people make in written proofs. Several of the above components are important and non-trivial and the mechanical verification makes these explicit.

9 Proving the Verification Conditions

Three verification conditions have to be proven in order to show the partial correctness of Graham's Scan. Next, we briefly discuss their proofs.

VC 1: The first verification condition shows that the preconditions imply the loop invariant holds true at the beginning, with the initialisations $i = 0$ and

C being [hd P, last P]. This is a fairly trivial proof. Components 1-15 of the loop invariant are proved easily from the pre-conditions. To prove component 16 holds true at first, we must show:

$$\begin{aligned}&\texttt{ordered } P \wedge 3 \leq \texttt{length } P \wedge \texttt{distinct } P \wedge \neg\texttt{all\_collinear } P\\ &\quad\Longrightarrow \exists n{<}\ \texttt{length } P.\ P_0 = P_n \wedge\\ &\qquad\quad (\texttt{[} P_0\texttt{] isConvexHull (take (Suc } n\texttt{) } P\texttt{)} \vee\\ &\qquad\quad \texttt{all\_collinear (take (Suc } n\texttt{) } P\texttt{))}\end{aligned}$$

To prove this we eliminate the existential quantifier and instantiate n to be 0. It is then trivial to show all_collinear (take 1 P) by merely expanding the definition of collinear.

VC 2: The second verification condition to prove is:

$$\texttt{Loop invariant} \wedge \neg\ i < \texttt{length } P \Longrightarrow \texttt{(butlast } C\texttt{) isConvexHull } P$$

The proof of this VC involves deriving the fact $i = \texttt{length } P$ from the assumptions. We then instantiate k to be 0 in component 16 of the loop invariant. From this and components 3 and 13 of the loop invariant we know the following facts:

a) last $P = P_{\texttt{length } P-1} = P_n = \texttt{hd } C$
b) (butlast C) isConvexHull (take $(n+1)$ P) $\vee$
(all_collinear (take $(n+1)$ P) $\wedge$
(length C - $n = 2 \vee$ length C - $n = 3$))

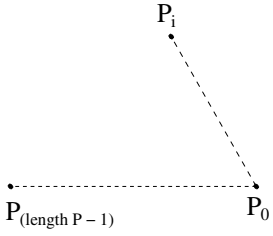
From the assumption that all points in P are distinct and fact (a) above, we show $n = \texttt{length } P - 1$. This allows ($b$) to reduce to (butlast C) isConvexHull $P \vee$ (all_collinear $P \wedge$ ...). From the loop invariant we know that the list of points in P are not all collinear, hence we are left with the desired result.

VC 3: The third verification condition is the most difficult to prove. It involves showing that the loop invariant and loop test are preserved after executing the body of the loop. The proof splits into two cases: one where we turn left with respect to the new point being examined in P, and one where we do not turn left. Both cases require several lemmas to be proven. The crux of the proof involves showing that component 16 of the invariant is preserved if we make a left turn with the respect to the new point P_i:

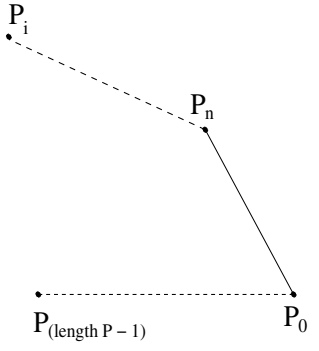
$$\begin{aligned}&\texttt{Loop Invariant} \wedge i < \texttt{length } P \wedge \texttt{Left\_turn } C_1\ C_0\ P_i \Longrightarrow\\ &\quad \forall\ k{<}\texttt{length (}P_i\ \texttt{\#}\ C\texttt{)} - 1.\\ &\qquad \exists\ n{<}\texttt{length } P.\ (P_i\ \texttt{\#}\ \texttt{C})_k = P_n \wedge\\ &\qquad\ \texttt{((drop } k\ \texttt{(butlast (}P_i\ \texttt{\#}\ C\texttt{))) isConvexHull (take (}n\ \texttt{+ 1) } P\texttt{)} \vee\\ &\qquad\quad \texttt{all\_collinear (take (}n\ \texttt{+ 1) } P\texttt{)} \wedge\\ &\qquad\quad\ \texttt{(length (}P_i\ \texttt{\#}\ C\texttt{) - } k\ \texttt{= 2} \vee \texttt{length (}P_i\ \texttt{\#}\ C\texttt{) - } k\ \texttt{= 3))"}\end{aligned}$$

This lemma is basically showing that for every update of C we maintain the property that we have constructed a convex hull or all the points are collinear upto the last vertex added. For the case where k<0, we have to prove that every C before P_i was added meets this property. From our assumptions, specifically

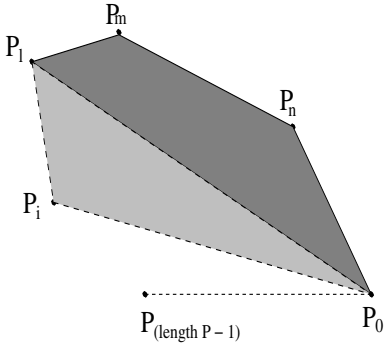
component 16 of the loop invariant, this is easily shown. The complicated part of the proof deals with the case where k=0. Here we have just pushed a new vertex P_i onto the hull and must prove that we maintain the property of convexity or collinearity. The proof splits into three cases; the hull before P_i was added could have contained 2, 3 or more vertices on it. These are illustrated below with a brief description of how each case is tackled (recall that the current hull C is ordered clockwise).



Case One: C=[P_0, $P_{\texttt{length}\ P-1}$]
Clearly once P_i is added to the hull so far, our updated C will contain 3 points. We have to show that all points in P upto P_i are collinear. Now, the points which we considered before P_i must have been popped by the algorithm, as they cannot be vertices. We show that these popped points lie between P_0 and P_i using components 10 and 5 of our invariant. Thus our new hull, P_i # C, meets the desired property of collinearity.



Case Two: C=[P_n, P_0, $P_{\texttt{length}\ P-1}$]
Once P_i gets added to our hull, our updated C is going to contain 4 points. We must show that `(butlast (`P_i `# C)))` is the convex hull of all points in P upto P_i. We first deduce that P_n is the `furthest_fst_pol_ang` point. Thus all points considered after P_0 and before P_n must lie between these two vertices. We must then reason about the points popped after P_n and before P_i using component 5 of the invariant. Together these facts give us the desired property of a convex hull.



Case Three: C=[P_l, P_m, P_n, P_0, $P_{\texttt{length}\ P-1}$]
This case is similar to Case Two. Before P_i is added as a vertex, our C contains the vertices of the convex hull up to P_l. We need to prove that adding P_i maintains the convexity property, this time for all points in P up to P_i. We use component 5 again to reason about points popped after P_l and before P_i. This, together with the fact we had a convex hull after the previous iteration of the loop, helps us derive our result.

Notice that both Cases Two and Three reason about popped points using component 5 of the loop invariant. Proving that this component holds every time we enter and leave the body of the loop was challenging, especially for the case where we do not turn left with respect to the new point P_i. The difficult part of the proof revolves around showing the following lemma:

```
Loop invariant ∧ i < length P ∧ ¬ Left_turn C1 C0 Pi ∧
 ¬ Pk mem butlast (tl C) ∧ k < i ∧ j < length (tl C) - 2 ∧
 (tl C)0 = Pm ∧ m < k ⟹
                    ¬ Left_turn Pm Pk Pi"
```

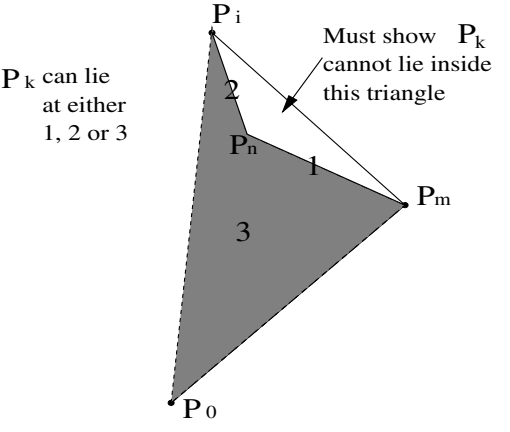


Fig. 3. Three possible cases in proof

The proof splits into many cases, due to the fact that collinear points are allowed in our system. Three of the non-collinear cases are illustrated in Figure 3. Here the point P_k could lie at either positions 1, 2 or, in the general case, 3. If P_k is lying at 3 then we need to use Knuth's property *5b* to deduce the fact `Left_turn` P_k P_m P_i from the following instantiations: $s = P_0$, $t = P_m$, $p = P_i$, $q = P_n$, $r = P_k$. In order to apply this lemma with these instantiations the following five facts need to hold:

1. `Left_turn` P_0 P_m P_i 2. `Left_turn` P_0 P_m P_n 3. `Left_turn` P_0 P_m P_k
4. `Left_turn` P_m P_i P_n 5. `Left_turn` P_m P_n P_k

Although components 5 and 15 of the loop invariant help derive many of these facts easily, the process was fairly tedious to work out by hand first and then mechanise. If more of the geometry theorem proving could be automated in Isabelle, the verification task would be much easier.

10 On Automation

Based on our experience, automating the proof of Graham's Scan in Hoare logic is a challenging task. This is, in part, due to the formulation of the loop invariant,

which is a complex process. Although several researchers have attempted to automate the proofs of imperative programs using Hoare logic, they have not had great success. The most promising results have come out of the work done by Stark and Ireland, who investigated the automatic discovery of loop invariants in the CLAM proof planning system [15]. Despite this work showing a good foundation to build upon, it is worth bearing in mind that Stark and Ireland did not test their approach on geometric problems. It is therefore unclear how applicable their approach would be in this domain.

Incorporating automation into the mechanisation process has wider scope than just discovering the correct loop invariant. In our case, the verification conditions generated in Isabelle are merely statements in higher order logic. If we could automate the proofs of these statements, our task would be greatly simplified. As our proof reasoned so much about signed areas we wondered about incorporating the Signed Area method [1] into Isabelle. However, it turned out not to serve our purposes well as collinearities are not dealt with. We also found restrictions with the existing algebraic techniques used in geometry theorem proving; either they did not deal with collinearites sufficiently or they could not reason about inequalities and hence ordered geometry. We have noted that it is possible to translate our geometric lemmas into algebraic form and use some existing system, like Maple or QEPCAD's cylindrical algebraic decomposition, to solve them. We recognise that there are drawbacks to this approach; we must have trust in the external system, and understanding the proofs intuitively would be difficult. Often a desirable characteristic of a proof is that it be surveyable. We believe the same should apply for program verification, as ideally we want an insight into the algorithm being verified. We are currently investigating how to obtain the correct level of abstraction in a proof so that it can be understood on an intuitive geometric level before being translated into algebraic form.

It is interesting to note one typical subgoal we encountered:

$$\texttt{Left\_turn s t q} \wedge \texttt{Left\_turn s t r} \wedge \texttt{Left\_turn t q r} \wedge \texttt{Left\_turn s r q} \wedge \neg\texttt{Left\_turn p s q} \Longrightarrow \texttt{Left\_turn p q s}$$

Notice that the first four assumptions could never simultaneously hold here, so the subgoal is trivially true. Our proof contained many such subgoals, all of which were tedious to discharge. We plan to try and automatically generate counterexamples for such cases, thus simplifying the proof process.

11 Related Work

Although our work is the first to formally verify Graham's Scan using Hoare logic, it is not the first to mechanise the correctness proofs of convex hull algorithms. Similar to our research, Pichardie and Bertot were inspired by the work of Knuth [13]. However, they used the theorem prover Coq and proved the correctness of two different algorithms: an incremental algorithm and a package wrapping algorithm. They adopted Knuth's method of disallowing collinear points in their CC system and then modified this using two different approaches. Their first

approach was to add axioms into their system. It is not clear from their paper if these axioms were proven for the planar case in Coq[2]. Their second approach at dealing with degeneracies is more interesting as they formalised a perturbation technique for dealing with degenerate cases. This technique was flawed however, as points which lay between two adjacent vertices were sometimes returned as legitimate vertices. Their verification also differed from our work as it focused more on the mechanisation aspects rather than the subtleties overlooked by humans in written proofs.

12 Conclusion

In conclusion, we believe that formally proving geometric algorithms in a theorem prover like Isabelle adds confidence in their correctness, and consequently we believe it should become an important stage in the development process of such algorithms. Despite the difficulty of proving these algorithms at present, we believe that by building libraries of useful theories, and by gaining a better understanding of the field, this task will get easier. Our work demonstrates how successful Hoare logic can be for formalising geometric algorithms. Not only does it allow the formal specifications to resemble the algorithm, but it forces one to think carefully about algorithms involving loops. By formalising the facts that never change on each iteration of the loop, it is often possible to get a better understanding of the algorithm and reveal unforeseen situations that might otherwise go undetected.

Although confidence in geometric algorithms can be boosted by proofs such as ours, it is important to highlight that these proofs do not guarantee complete correctness. In some instances appropriate input data can unfortunately cause a seemingly correct program to fail. This is due to the fact that geometric algorithms commonly suffer from the problem of nonrobustness, which is caused by two factors: the use of real-world data, which may be degenerate, and the substitution of floating-point arithmetic for real arithmetic. For future work, we plan to incorporate these issues of nonrobustness into the mechanical proof.

References

1. S. C. Chou, X. S. Gao, and J. Z. Zhang. Automated generation of readable proofs with geometric invariants, I. multiple and shortest proof generation. *Journal of Automated Reasoning*, 17:325-347, 1996.
2. A. Church. A formulation of the simple theory of type. *Journal of Symbolic Logic*, 5:56-68, 1940.
3. M. Gordon. Mechanizing Programming Logics in Higher Order Logic. *Current Trends in Hardware Verification and Automated Theorem Proving*, G. Birtwistle and P. A. Subrahmanyam, Springer, 1989.

[2] Axiom 8 in their paper is in fact inconsistent; this may be a typo but it is not clear what is intended.

4. M. Gordon and T. Melham. *Introduction to HOL: A theorem proving environment for Higher Order Logic*. Cambridge University Press, 1993.
5. R. L. Graham. *An efficient algorithm for determining the convex hull of a finite planar set*. Info. Proc. Lett. 1, pages 132-133, 1972.
6. D. Hilbert. *The Foundations of Geometry*. The Open Court Company, 2001, 11th edition. Translation by Leo Unger.
7. C. A. R. Hoare. *An axiomatic basis for computer programming.* Communications of the ACM, v. 12 n. 10, pages 576-580, 1969.
8. D. E. Knuth. *Axioms and Hulls.* Lecture Notes in Computer Science, Volume 606. Springer, 1992.
9. L. Meikle and J. Fleuriot. *Formalizing Hilbert's Grundlagen in Isabelle/Isar*. TPHOLs, vol. 2758, pages 319-334, Springer, 2003.
10. T. Nipkow. *Hoare Logics in Isabelle/HOL*. Proof and System Reliability, Kluwer, 2002.
11. J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.
12. L. C. Paulson. Isabelle: A Generic Theorem Prover. *Lecture Notes in Computer Science*, Volume 828. Springer, 1994.
13. D. Pichardie and Y. Bertot. *Formalizing Convex Hull Algorithms.* TPHOLs, 346-361, Springer, 2001.
14. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer, 1985.
15. J. Stark and A. Ireland. *Invariant Discovery via Failed Proof Attempts.* Lecture Notes in Computer Science, Volume 1559, page 271. Springer, 1998.

Computational Origami Construction of a Regular Heptagon with Automated Proof of Its Correctness

Judit Robu[1], Tetsuo Ida[2], Dorin Ţepeneu[2],
Hidekazu Takahashi[3], and Bruno Buchberger[4,⋆]

[1] Babeş-Bolyai University, M. Kogalniceanu No.1, Cluj-Napoca 400084, Romania
robu@cs.ubbcluj.ro
[2] University of Tsukuba, Tennoudai 1-1-1, Tsukuba 305-8573, Japan
{ida, dorinte}@score.cs.tsukuba.ac.jp
[3] Yokaichi High School, Shiga, Japan
[4] Research Institute for Symbolic Computation,
Johannes Kepler University, A 4232 Schloss Hagenberg, Austria
Bruno.Buchberger@risc.uni-linz.ac.at

Abstract. Construction of geometrical objects by origami, the Japanese traditional art of paper folding, is enjoyable and intriguing. It attracted the minds of artists, mathematicians and computer scientists for many centuries. Origami will become a more rigorous, effective and enjoyable art if the origami constructions can be visualized on the computer and the correctness of the constructions can be automatically proved by an algorithm. We call the methodology of visualizing and automatically proving origami constructions computational origami. As a non-trivial example, in this paper, we visualize a construction of a regular heptagon by origami and automatically prove the correctness of the construction.

1 Introduction

1.1 Origami

Origami, being an art since the 10th century in Japan, became an object of extensive scientific study around the end of 1980's as mathematicians became interested in the mathematical principles of origami. A seminal work in the history of mathematical origami is that of Huzita, who proposed six axioms of origami [9]. It is known that Huzita's origami axiom set is more powerful than the ruler-and-compass method in Euclidean geometry [7]. In [1] Alperin studied the algebraic structure of the set of Origami constructible points. He identifies

⋆ Sponsored by Austrian FWF (Österreichischer Fonds zur Förderung der Wissenschaftlichen Forschung), Project 1302, in the frame of the SFB (Special Research Area) 013 "Scientific Computing", RICAM (Radon Institute for Applied and Computational Mathematics, Austrian Academy of Science, Linz) and JSPS Grant-in-Aid for Scientific Research (B) 17300004, 2005 (Chief Investigator: Tetsuo Ida).

H. Hong and D. Wang (Eds.): ADG 2004, LNAI 3763, pp. 19–33, 2006.

the set of marked points by the application of Huzita's Axioms 1−3, Axioms 1−5, and Axioms 1−6, as some fields.

Origami constructions are expressed concisely by the following axiom set:

(FoldThru). Given two points P and Q, we can make a fold along the crease passing through them.

(FoldBring). Given two points P and Q, we can make a fold to bring one of the points onto the other.

(FoldBringLine). Given two lines m and n, we can make a fold to superpose the two lines.

(FoldPerTh). Given a point P and a line m, we can make a fold along the crease that is perpendicular to m and passes through P.

(FoldBrTh). Given two points P and Q and a line m, either we can make a fold along the crease that passes through Q, such that the fold superposes P onto m, or we can determine that the fold is impossible.

(FoldBrBr). Given two points P and Q and two lines m and n, either we can make a fold along the crease, such that the fold superposes P and m, and Q and n, simultaneously, or we can determine that the fold is impossible.

A rigorous presentation of origami constructions is in [11]. The axiom set provides the basis of computational origami. Namely, by implementing the axiom set by a computer, we can construct sophisticated origami works. Although the notion of completeness is unclear as we do not yet identify a class of origami constructible geometrical objects, by the subsequent works of several mathematicians, we know that origami is more powerful than classical Euclidean construction by a ruler and a compass. Huzita's 6th axiom plainly states that we can make a fold that brings two points on two lines. The statement is more profound than we might think. From the computational point of view, we see that folding involves solving third degree polynomial equations, and from operational point of view we observe that we need a kind of sliding of one point along a fold line to bring the other point onto the other line. It was shown that this operation can not be performed by the ruler-and-compass method. One of the simplest geometric constructions that verifies this remark is trisecting an angle. A more sophisticated example is the construction of a regular heptagon. However, Gleason [8], who developed the theory of the angle trisector gave a construction using ruler, compass and the angle trisector. Huzita already showed the origami method of constructing a regular heptagon together with the correctness proof [10], possibly without imagining fully automated origami solving and proving.

1.2 Novelties

This paper presents, on the non-trivial example of the heptagon construction, a convenient tool for computing and visualizing the intermediate steps of origami constructions and an automated proof of the correctness of the construction. The algorithmic method used for the correctness proof is completely general and can be applied to any origami construction whose conclusion can be formulated as

a polynomial equality in the coordinates of the points involved. We use *Theorema* and the geometrical theorem prover constructed on top of *Theorema* for proving, and a computational origami system for solving geometrical constraints and visualizing origami. The proof performed by *Theorema* uses Buchberger's Gröbner bases method, see e.g. [2]. The computational origami system consists of a human-friendly web interface and of solving and computing engines situated on remote servers.

1.3 Motivation

We are interested in computing, solving and proving (and the interaction of these activities) in mathematical problem solving. Indeed, most of the sophisticated problems that mathematicians and computer scientists face in their research life involve these three intellectual activities. Origami, which is tangible as we use concrete material, i.e. a piece of paper, is in fact very abstract and requires the interplay of these three activities. Furthermore by computational origami we give origami construction yet another level of sophistication. More concretely, by the system of computational origami we aim to provide

1. a tool for creating artworks of origami,
2. a pedagogical tool for teaching mathematics, in particular, geometry,
3. an environment for doing research in geometry and in geometrical theorem proving.

1.4 Our Contribution

After illustrating the visualization of the construction of a regular heptagon by the computational origami system, we will focus on proving the correctness of the construction. At the time of writing, the computational origami system, Theorema and the geometrical theorem prover are not entirely integrated, they communicate through files. It is straightforward to put them together since all of the systems are written in Mathematica. However, we are more interested in the interaction of the three system in a distributed computer environment. Rather than combining the three systems into one, we are trying to make the three systems interact over the Internet. Our vision is to realize a symbolic grid computing framework, under which our three systems are able to interact with each other [15].

2 The Regular Heptagon Problem

2.1 Constructing a Heptagon

We give an example of constructing a heptagon in the origami system. This example also shows a nontrivial use of Axiom 6 (FoldBrBr). Huzita in [10] gave a construction sequence for this problem. In our implementation all the operations are performed by *Mathematica* function calls.

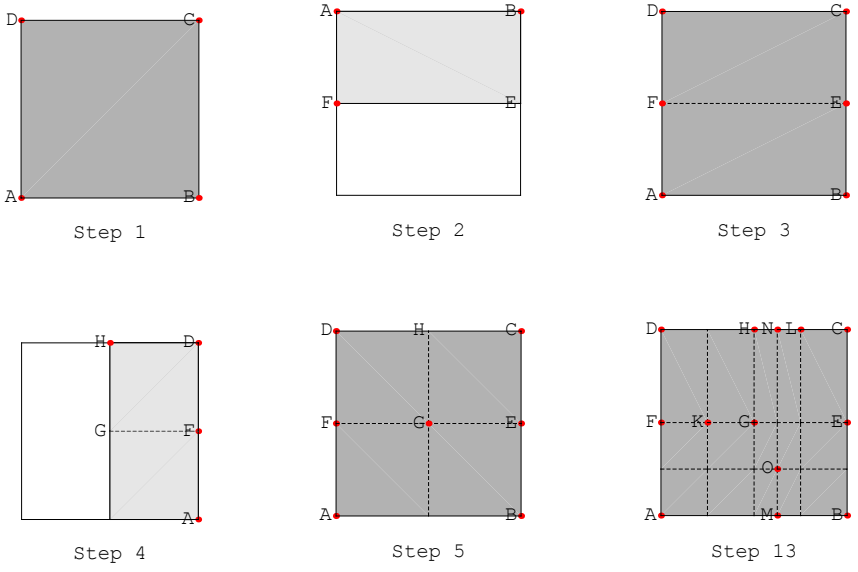


Fig. 1. Heptagon construction steps 1-13

Step 1: First, we define a square origami paper, whose corners are designated by the points A, B, C and D. The size may be arbitrary, but for our example, let us fix it to 100 by 100. The new origami figure is created with two differently colored surfaces: a light-gray front and a dark-gray back.

```
NewOrigami[Square[100, MarkPoints → {'A','B','C','D'}],
    FigureCaption → 'Step '];
```

Our problem is to construct a heptagon in the origami space. The method consists of the following 38 steps (steps 2-39) of folds and unfolds.

Steps 2 and 3: We make a fold to bring point A to point D, to obtain the perpendicular bisector of segment $\overline{AD}$. This is the application of (FoldBring). The points E and F are automatically generated by the system. We unfold the origami and obtain the crease $\overline{EF}$.

```
FoldBring[A, D];
Unfold[];
```

Steps 4 and 5: Likewise we obtain the crease $\overline{HG}$, points H and G being on the segments $\overline{CD}$ and $\overline{EF}$.

```
FoldBring[A, B,MarkCrease → {CD, EF}];
Unfold[];
```

Steps 6-13: Applying four more times axiom (FoldBring) we obtain in order points K, L, crease $\overline{MN}$ and point O.

```
FoldBring[D, H,MarkCrease → {FE}];
Unfold[];
FoldBring[C, H,MarkCrease → {CD}];
Unfold[];
FoldBring[L, H,MarkCrease → {AB, CD}];
Unfold[];
FoldBring[A, F,MarkCrease → {MN}];
Unfold[];
```

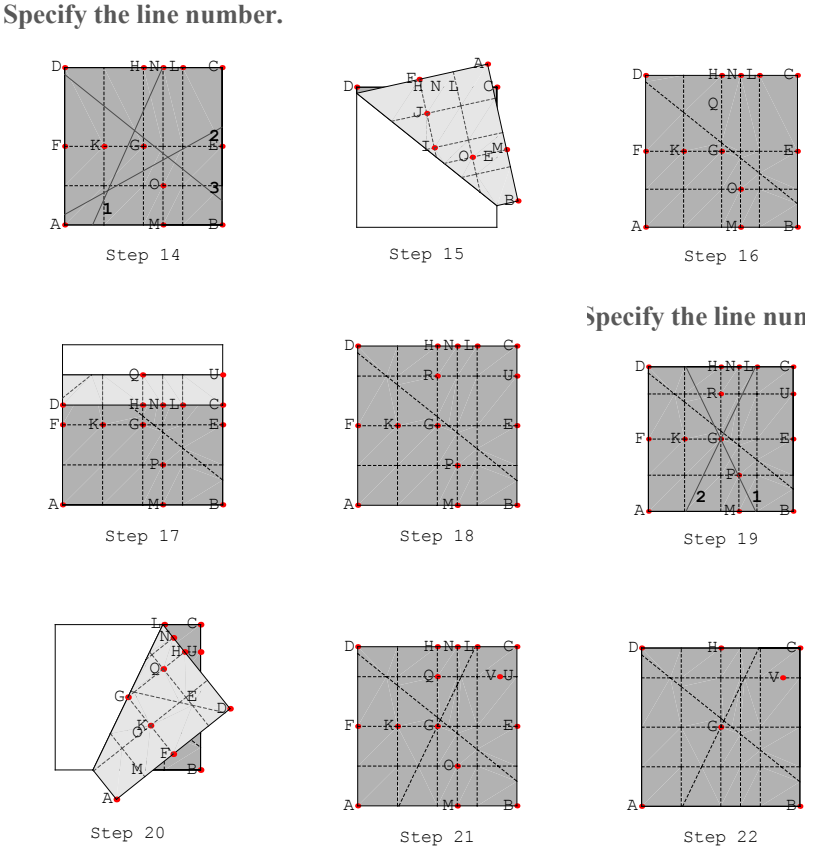


Fig. 2. Heptagon construction steps 14-22

Steps 14 and 15: Step 14 is the crucial step of the construction. We will superpose point K and the line that is the extension of the segment $\overline{HG}$, and superpose point O and the line that is the extension of the segment $\overline{EG}$, simultaneously. This is possible by (FoldBrBr) and is realized by the call of function `FoldBrBr`. There are three candidate fold lines to make these superpositions possible. The system responds with the query "Specify the line

number" together with the fold lines on the origami image. We reply with the call of `FoldBrBr` with the additional parameter 3, which tells the system that we choose line number 3. This is the fold line that we are primarily interested in. However, the other two fold lines are also solutions (then we do not obtain the vertices of the heptagon in order).

```
FoldBrBr[K, HG, O, EG];
FoldBrBr[K, HG, O, EG, 3];
```

Step 16: We will duplicate point K on the face that is below the one that K is on, and unfold the origami. The duplicated point appears as Q. Duplication of a point is not counted as a new step by the system. The names of the points are automatically generated.

```
DupPoint['K'];
Unfold[];
```

Steps 17 and 18: We obtain point U as being on the crease obtained folding along a line passing through point Q and perpendicular to $\overline{HG}$ moving point H. Then we unfold the origami.

```
FoldPerTh[ HG, Q, H, MarkCrease → {BC}];
Unfold[];
```

Steps 19 and 20: In step 19 we use the other interesting axiom (FoldBrTh), superposing point H and the line that is the extension of the segment $\overline{RU}$ folding along a crease that passes through point G. There are two candidate fold lines to make this superposition possible. The system responds with the query "Specify the line number" together with the fold lines on the origami image. We reply with the call of `FoldBrTh` with the additional parameter 2, which tells the system that we choose the line number 2. This is the fold line that we are primarily interested in. However, the other line is also solution (then we don't obtain the vertices of the heptagon in order).

```
FoldBrTh[H, RU, G];
FoldBrTh[H, RU, G,2];
```

Steps 21 and 22: We will duplicate point H on the other face that is below the face that H is on, and unfold the origami. The duplicated point appears as V. We delete the labels of the points that are not any more interesting. At this point, the main part of the construction is achieved, as we obtained the angle $\angle HGV = 2\pi/7$.

```
DupPoint['H'];
Unfold[];
```

Steps 23 and 24: We obtain the next vertex by mirroring point H versus the line passing through points G and V, duplicating point H on the face below obtaining point E and unfolding the origami. We observe that the deleted

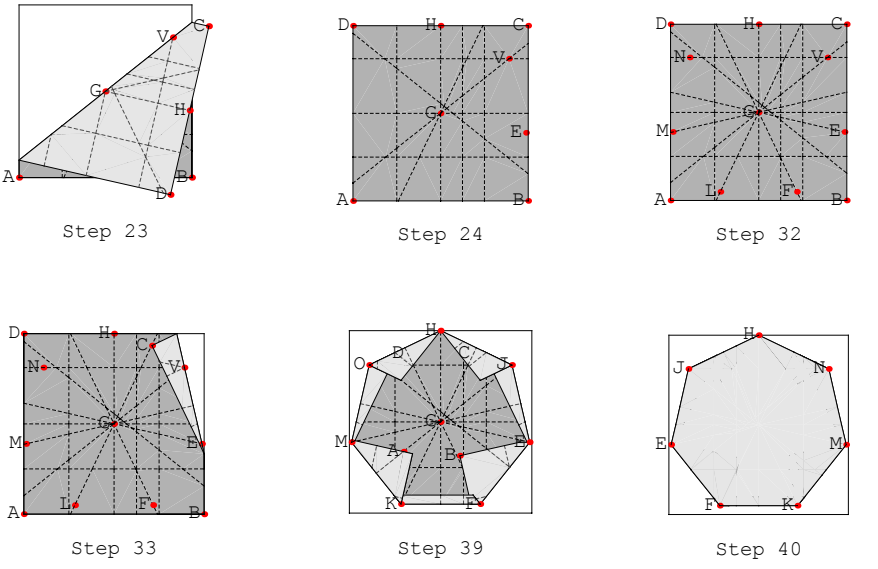


Fig. 3. Heptagon construction steps 23-40

labels are reused by the system when automatically allocating names to the constructed points.

```
FoldThru[G, V, H];
DupPoint['H']
Unfold[];
```

Steps 25-32: Repeating the previous steps we obtain the other four vertices of the heptagon, namely points F, K, M and N.

```
FoldThru[G, V, H];
DupPoint['H']
Unfold[];
FoldThru[G, E, V];
DupPoint['V']
Unfold[];
FoldThru[G, F, E];
DupPoint['E']
Unfold[];
FoldThru[G, K, F];
DupPoint['F']
Unfold[];
```

Steps 33-40: To obtain the heptagon we fold along the edges and finally turn the origami to its other side (as if it were a real piece of paper) to hide the folds.

```
FoldThru[V, E, C];
FoldThru[E, F, B];
FoldThru[F, L, A];
FoldThru[L, M, A];
FoldThru[M, N, D];
FoldThru[N, H, D];
FoldThru[H, V, C];
TurnOver[]
```

2.2 Proving the Correctness

How can we be sure that the construction really gives a regular heptagon? We have to prove the following:

Theorem 1. *The origami construction in section 2.1 produces a regular heptagon.*

Huzita [10] proved the correctness of the construction making use of geometric intuition and high school mathematics.

We want one algorithm for all such proofs. This can be done by reducing, in an algorithmic way, the proof problem to a problem in computer algebra, e.g. Gröbner basis computation [11]. This is possible because:

- each origami step is described by polynomial equalities;
- the sequence of steps and the final assertion, together, form a universally quantified boolean combination of polynomial equalities;
- this universally quantified formula can be converted, using predicate logic and Rabinovich's trick [12], into a finite number of decisions about solvability of polynomial equations;
- these solvability decisions can be answered algorithmically by invocation of Buchberger's Gröbner basis algorithm.

For the heptagon problem we may have two different approaches:

- to prove that the obtained seven edges are of equal length (it is enough to prove that $|\overline{HN}| = |\overline{HV}|$).
- to prove that the angle formed by two adjacent corners and the center of the heptagon is $2\pi/7$, that is, $\angle HGV = 2\pi/7$

We used both approaches, the first one with the origami prover built on top of the geometrical theorem prover [14] of *Theorema* [4], the second one with the proving facility built in the origami system.

3 Automated Proof in *Theorema*

Theorema is a mathematical software system implemented in *Mathematica* and, hence, is available on all computer platforms for which *Mathematica* is available.

Theorema aims at providing one uniform logical and software technological frame for automated theorem proving in all areas of mathematics or, in other words and more generally, for formal mathematics, i.e. proving, solving, and simplifying mathematical formulae relative to mathematical knowledge bases, see [3], [4]. *Theorema* is being developed at the RISC Institute by the *Theorema* Group under the direction of Bruno Buchberger.

Theorema offers a user-friendly interface for problem input. It generates fully automatically the proofs that contain all the necessary explanations.

The geometry prover is based on the methods described in [16], [6], [13]. The input for the geometry prover, i.e. the algebraic formulation of all the construction steps and of the property the final configuration should satisfy is generated automatically from the geometric description of the origami construction and the conclusion specified by the user. This information is then sent, as a *Theorema* Proposition to the automated prover for proving / disproving whether, for all possible input configurations, after applying the construction steps specified, the final configuration always satisfies the desired property. Executing

```
$origami = << D:\TheoremaPrivate\heptagon;
$origamiTh = TransOrigami[$origami, H4V18 = H4N28]
```

we obtain a *Theorema* Proposition to be proved (see the output of the prover).

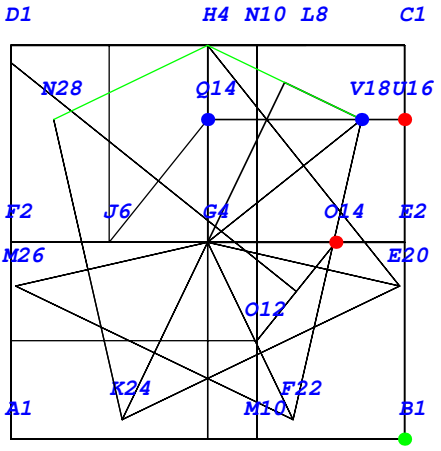
To display graphically the geometrical constraints among the involved points and lines we call function `Simplify` that uses the `KnowledgeBase C1` to specify the coordinates of the free points

```
KnowledgeBase['C1',any[A,B],{{A,{0,0}},{B,{100,0}}}]
Simplify[$origamiTh,
   by → GraphicSimplifier, using → KnowledgeBase['C1']]
```

and obtain the output presented in Fig. 4.

The geometry prover is invoked in the usual *Theorema* manner, specifying the Gröbner basis prover. *Theorema* does the rest of the work:

- finds a convenient coordinate system;
- expresses the origami constructions and the final assertion as a universally quantified boolean combination of polynomial equalities and inequalities, using the cartesian coordinates of the constructed points;
- converts this universally quantified formula, using predicate logic and Rabinovich's trick, into a finite number of decisions about solvability of polynomial equations;
- invokes the *Mathematica* GröbnerBasis function to answer these solvability decisions;
- generates the notebook with all the explained details of the proof.



|H4 V18| ≅ |H4 N28| for this configuration of the points

Fig. 4. *Theorema* output

For the function call

```
Prove[$origamiTh, by → GeometryProver,
   ProverOptions → {Method → "GroebnerProver",
   MyAxis → True, ReverseVars → True,
   Refutation → True}]
```

we obtain the following output from the prover:

Begin of Theorema notebook

We have to prove:

(Proposition(Origami))

$$\forall_{A1,B1,C1,D1,E2,F2,G4,H4,J6,L8,M10,N10,O12,O14,Q14,U16,V18,E20,F22,K24,M26,N28}$$

(neworigami[$A1, B1, C1, D1$]∧
foldBring[$A1, D1,$ crease[$E2$ on line[$B1, C1$], $F2$ on line[$D1, C1$]]]∧
foldBring[$A1, B1,$ crease[$G4$ on line[$E2, F2$], $H4$ on line[$C1, D1$]]]∧
foldBring[$D1, H4,$ crease[$J6$ on line[$E2, F2$]]]∧
foldBring[$C1, H4,$ crease[$L8$ on line[$C1, D1$]]]∧
foldBring[$L8, H4,$ crease[$M10$ on line[$A1, B1$], $N10$ on line[$D1, C1$]]]∧
foldBring[$A1, F2,$ crease[$O12$ on line[$M10, N10$]]]∧
foldBrBr[$Q14, J6,$ line[$H4, G4$], $O14, O12,$ line[$E2, G4$]]∧
foldPerTh[line[$H4, G4$], crease[$Q14, U16$ on line[$B1, C1$]]]∧
foldBrTh[$V18, H4,$ line[$Q14, U16G4$], crease[$G4$]]∧
foldThru[$E20, H4,$ crease[line[$G4, V18$]]]∧
foldThru[$F22, V18,$ crease[line[$G4, E20$]]]∧

$$\begin{array}{l}\text{foldThru}[K24, E20, \text{crease}[\text{line}[G4, F22]]]\wedge\\ \text{foldThru}[M26, F22, \text{crease}[\text{line}[G4, K24]]]\wedge\\ \text{foldThru}[N28, K24, \text{crease}[\text{line}[G4, M26]]] \Rightarrow\\ \quad \text{distequal}[H4, V18, H4, N28]])\end{array}$$

with no assumptions.

To prove the above statement we use the Gröbner bases method. First we have to transform the problem into algebraic form.

To transform the geometric problem into an algebraic form we choose an orthogonal coordinate system.

Let us have the origin at point $A1$, and points $\{B1, M10\}$ and $\{D1, F2\}$ on the two axes.

Using this coordinate system we have the following coordinates:

$$\begin{array}{l}\{\{A1, 0, 0\}, \{B1, 1, 0\}, \{D1, 0, x_1\}, \{\alpha_{E2}, 0, x_2\}, \{F2, 0, x_3\}, \{\alpha_{G4}, x_4, 0\},\\ \{M10, x_5, 0\}, \{\alpha_{O12}, 0, x_6\}, \{\beta_{O14}, 0, x_7\}, \{C1, x_8, x_9\}, \{E2, x_{10}, x_{11}\},\\ \{G4, x_{12}, x_{13}\}, \{H4, x_{14}, x_{15}\}, \{\alpha_{J6}, x_{16}, x_{17}\}, \{J6, x_{18}, x_{19}\},\\ \{\alpha_{L8}, x_{20}, x_{21}\}, \{L8, x_{22}, x_{23}\}, \{\alpha_{M10}, x_{24}, x_{25}\}, \{N10, x_{26}, x_{27}\}\},\\ \{O12, x_{28}, x_{29}\}, \{Q14, x_{30}, x_{31}\}, \{\beta_{Q14}, x_{32}, x_{33}\}, \{\alpha_{O14}, x_{34}, x_{35}\}\},\\ \{O14, x_{36}, x_{37}\}, \{U16, x_{38}, x_{39}\}, \{V18, x_{40}, x_{41}\}, \{\alpha_{V18}, x_{42}, x_{43}\}\},\\ \{\alpha_{E20}, x_{44}, x_{45}\}, \{E20, x_{46}, x_{47}\}, \{\alpha_{F22}, x_{48}, x_{49}\}, \{F22, x_{50}, x_{51}\}\},\\ \{\alpha_{K24}, x_{52}, x_{53}\}, \{K24, x_{54}, x_{55}\}, \{\alpha_{M26}, x_{56}, x_{57}\}, \{M26, x_{58}, x_{59}\}\},\\ \{\alpha_{N28}, x_{60}, x_{61}\}, \{N28, x_{62}, x_{63}\}\end{array}$$

where α_X and/or β_X are variables generated internally to create point X.

The algebraic form[1] of the given construction is:

(1)

$$\begin{array}{l}\underset{x_1,\ldots,x_{63}}{\forall}((-1) + -x_1 == 0 \wedge (-1) + x_8 == 0 \wedge -{x_1}^2 + x_1x_9 == 0\wedge\\ \quad -x_1 + 2x_2 == 0 \wedge -x_9 + x_9x_{10} + x_{11} + -x_8x_{11} == 0\wedge\\ \quad x_1x_2 + -x_1x_{11} == 0 \wedge x_1x_2 + -x_1x_3 == 0\wedge\\ \quad (-1) + 2x_4 == 0 \wedge -x_3x_{10} + x_3x_{12} + -x_{11}x_{12} + x_{10}x_{13} == 0\wedge\\ \quad x_4 + -x_{12} == 0 \wedge -x_1x_8 + x_1x_{14} + -x_9x_{14} + x_8x_{15} == 0\wedge\\ \quad x_4 + -x_{14} == 0 \wedge -x_{14} + 2x_{16} == 0 \wedge -x_1 + -x_{15} + 2x_{17} == 0\wedge\\ \quad -x_3x_{10} + x_3x_{18} + -x_{11}x_{18} + x_{10}x_{19} == 0\wedge\\ \quad x_{14}x_{16} + -x_1x_{17} + x_{15}x_{17} + -x_{14}x_{18} + x_1x_{19} + -x_{15}x_{19} == 0\wedge\\ \quad -x_8 + -x_{14} + 2x_{20} == 0 \wedge -x_9 + -x_{15} + 2x_{21} == 0\wedge\\ \quad -x_1x_8 + x_1x_{22} + -x_9x_{22} + x_8x_{23} == 0\wedge\\ \quad -x_8x_{20} + x_{14}x_{20} + -x_9x_{21} + x_{15}x_{21} + x_8x_{22} +\\ \qquad\qquad -x_{14}x_{22} + x_9x_{23} + -x_{15}x_{23} == 0\wedge\\ \quad -x_{14} + -x_{22} + 2x_{24} == 0 \wedge -x_{15} + -x_{23} + 2x_{25} == 0\wedge\\ \quad -x_5x_{14} + x_5x_{22} + x_{14}x_{24} + -x_{22}x_{24} + x_{15}x_{25} + -x_{23}x_{25} == 0\wedge\\ \quad -x_1x_8 + x_1x_{26} + -x_9x_{26} + x_8x_{27} == 0\wedge\\ \quad x_{14}x_{24} + -x_{22}x_{24} + x_{15}x_{25} + -x_{23}x_{25} + -x_{14}x_{26} +\\ \qquad\qquad x_{22}x_{26} + -x_{15}x_{27} + x_{23}x_{27} == 0 \wedge -x_3 + 2x_6 == 0\wedge\end{array}$$

[1] Notation $x_1, ..., x_{63}$ represents the full sequence of consecutive variables from x_1 to x_{63}.

$$
\begin{aligned}
&x_5x_{27} + x_{27}x_{28} + x_5x_{29} + -x_{26}x_{29} == 0 \wedge x_3x_6 + -x_3x_{29} == 0 \wedge \\
&-x_{13}x_{14} + x_{12}x_{15} + x_{13}x_{30} + -x_{15}x_{30} + -x_{12}x_{31} + x_{14}x_{31} == 0 \wedge \\
&-x_{18} + -x_{30} + 2x_{32} == 0 \wedge -x_{19} + -x_{31} + 2x_{33} == 0 \wedge \\
&-x_7x_{19} + x_7x_{31} + x_{18}x_{32} + -x_{30}x_{32} + x_{19}x_{33} + -x_{31}x_{33} == 0 \wedge \\
&-x_7x_{32} + x_7x_{34} + -x_{33}x_{34} + x_{32}x_{35} == 0 \wedge \\
&x_7x_{29} + -x_{28}x_{32} + -x_{29}x_{33} + x_{32}x_{34} + -x_7x_{35} + x_{33}x_{35} == 0 \wedge \\
&-x_{28} + 2x_{34} + -x_{36} == 0 \wedge -x_{29} + 2x_{35} + -x_{37} == 0 \wedge \\
&-x_{11}x_{12} + x_{10}x_{13} + x_{11}x_{36} + -x_{13}x_{36} + -x_{10}x_{37} + x_{12}x_{37} == 0 \wedge \\
&x_{12}x_{30} + -x_{14}x_{30} + x_{13}x_{31} + -x_{15}x_{31} + \\
&\qquad -x_{12}x_{38} + x_{14}x_{38} + -x_{13}x_{39} + x_{15}x_{39} == 0 \wedge \\
&-x_9 + x_9x_{38} + x_{39} + -x_8x_{39} == 0 \wedge \\
&x_{31}x_{38} + -x_{30}x_{39} + -x_{31}x_{40} + x_{39}x_{40} + x_{30}x_{41} + -x_{38}x_{41} == 0 \wedge \\
&-x_{14} + -x_{40} + 2x_{42} == 0 \wedge -x_{15} + -x_{41} + 2x_{43} == 0 \wedge \\
&2x_{12}x_{40} + 2x_{13}x_{41} + -2x_{12}x_{42} + 2x_{40}x_{42} + 2{x_{42}}^2 + -2x_{13}x_{43} + \\
&\qquad -2x_{41}x_{43} + 2{x_{43}}^2 == 0 \wedge \\
&x_{13}x_{40} + -x_{12}x_{41} + -x_{13}x_{44} + x_{41}x_{44} + x_{12}x_{45} + -x_{40}x_{45} == 0 \wedge \\
&-x_{12}x_{14} + -x_{13}x_{15} + x_{14}x_{40} + x_{15}x_{41} + \\
&\qquad x_{12}x_{44} + -x_{40}x_{44} + x_{13}x_{45} + -x_{41}x_{45} == 0 \wedge \\
&-x_{14} + 2x_{44} + -x_{46} == 0 \wedge -x_{15} + 2x_{45} + -x_{47} == 0 \wedge \\
&x_{13}x_{46} + -x_{12}x_{47} + -x_{13}x_{48} + x_{47}x_{48} + x_{12}x_{49} + -x_{46}x_{49} == 0 \wedge \\
&-x_{12}x_{40} + -x_{13}x_{41} + x_{40}x_{46} + x_{41}x_{47} + \\
&\qquad x_{12}x_{48} + -x_{46}x_{48} + x_{13}x_{49} + -x_{47}x_{49} == 0 \wedge \\
&-x_{40} + 2x_{48} + -x_{50} == 0 \wedge -x_{41} + 2x_{49} + -x_{51} == 0 \wedge \\
&x_{13}x_{50} + -x_{12}x_{51} + -x_{13}x_{52} + x_{51}x_{52} + x_{12}x_{53} + -x_{50}x_{53} == 0 \wedge \\
&-x_{12}x_{46} + -x_{13}x_{47} + x_{46}x_{50} + x_{47}x_{51} + \\
&\qquad x_{12}x_{52} + -x_{50}x_{52} + x_{13}x_{53} + -x_{51}x_{53} == 0 \wedge \\
&-x_{46} + 2x_{52} + -x_{54} == 0 \wedge -x_{47} + 2x_{53} + -x_{55} == 0 \wedge \\
&x_{13}x_{54} + -x_{12}x_{55} + -x_{13}x_{56} + x_{55}x_{56} + x_{12}x_{57} + -x_{54}x_{57} == 0 \wedge \\
&-x_{12}x_{50} + -x_{13}x_{51} + x_{50}x_{54} + x_{51}x_{55} + \\
&\qquad x_{12}x_{56} + -x_{54}x_{56} + x_{13}x_{57} + -x_{55}x_{57} == 0 \wedge \\
&-x_{50} + 2x_{56} + -x_{58} == 0 \wedge -x_{51} + 2x_{57} + -x_{59} == 0 \wedge \\
&x_{13}x_{58} + -x_{12}x_{59} + -x_{13}x_{60} + x_{59}x_{60} + x_{12}x_{61} + -x_{58}x_{61} == 0 \wedge \\
&-x_{12}x_{54} + -x_{13}x_{55} + x_{54}x_{58} + x_{55}x_{59} + \\
&\qquad x_{12}x_{60} + -x_{58}x_{60} + x_{13}x_{61} + -x_{59}x_{61} == 0 \wedge \\
&-x_{54} + 2x_{60} + -x_{62} == 0 \wedge -x_{55} + 2x_{61} + -x_{63} == 0 \Rightarrow \\
&\quad -2x_{14}x_{40} + {x_{40}}^2 + -2x_{15}x_{41} + {x_{41}}^2 + \\
&\qquad 2x_{14}x_{62} + -{x_{62}}^2 + 2x_{15}x_{63} + -{x_{63}}^2 == 0\big)
\end{aligned}
$$

This problem is equivalent to[2]:

$$
\begin{aligned}
&(2) \\
&\neg \exists_{x_1,\ldots,x_{63}} ((-1) + -x_1 == 0 \wedge (-1) + x_8 == 0 \wedge -{x_1}^2 + x_1x_9 == 0 \wedge \\
&\quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&\quad -x_{54} + 2x_{60} + -x_{62} == 0 \wedge -x_{55} + 2x_{61} + -x_{63} == 0 \wedge
\end{aligned}
$$

[2] We shall not repeat all the polynomials.

$$-2x_{14}x_{40} + {x_{40}}^2 + -2x_{15}x_{41} + {x_{41}}^2 + \\ 2x_{14}x_{62} + -{x_{62}}^2 + 2x_{15}x_{63} + -{x_{63}}^2 \neq 0\big)$$

To remove the last inequality, we use the well-known Rabinovich trick. Let v_0 be a new variable. Then the problem becomes:

(3)

$$\neg \exists_{v_0,x_1,\ldots,x_{63}} ((-1) + -x_1 == 0 \wedge (-1) + x_8 == 0 \wedge -{x_1}^2 + x_1x_9 == 0 \wedge$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$-x_{54} + 2x_{60} + -x_{62} == 0 \wedge -x_{55} + 2x_{61} + -x_{63} == 0 \wedge$$
$$1 + -v_0(-2x_{14}x_{40} + {x_{40}}^2 + -2x_{15}x_{41} + {x_{41}}^2 + \\ 2x_{14}x_{62} + -{x_{62}}^2 + 2x_{15}x_{63} + -{x_{63}}^2) == 0\big)$$

To prove this statement we have to compute the Gröbner bases of the above polynomials.

The polynomials of the Gröbner bases are: {1}

As the obtained Gröbner bases is 1 the statement is generically true.

End of Theorema notebook.

4 Conclusions

In this paper we gave an automated correctness proof of an origami construction of a regular heptagon. This illustrates our recent research in a combined technology for algorithmic simplifying, solving, and proving.

In our future research, we will pursue origami computation in four directions:

- Improving the software technology for the interaction of symbolic and graphic systems over the web.
- A systematic investigation of the origami axioms: We will analyze the origami axioms with respect to the existence of creases with real coordinates and, correspondingly, the suitability of algorithmic methods from real algebraic geometry for the correctness proofs of origami constructions. Also, we want to make the sliding operation the basic building block of origami operations and, consequently, we will also introduce generalized origami axioms of higher order, in which points can slide not only on lines but on higher order curves already constructed by origami steps.
- In certain artistic origami constructions it is essential that the construction executed by foldings on the plane paper are, actually, expanded to 3D. Also, for certain constructions, it is necessary that the paper is distorted for a moment (in 3D space) in order to execute certain foldings. The role of 3D space in origami is not yet appropriately reflected by the current origami axioms. We intend to analyze the role of 3D and formalize it in terms of appropriate axioms.

– Origami solving: In the past two years we introduced the new aspect of automated origami proving to origami theory. As the next step, we want to add the aspect of automated origami *solving* which goes beyond automated origami proving: In origami proving, the construction (a sequence of origami steps) is given and we want to answer, by an algorithm, the question whether or not, for all possible initial situations, the resulting object has certain properties. In contrast, in origami solving we formulate desired properties of a configuration to be constructed (for example, the property that a constructed angle is one third of an initial angle or the property that the constructed configuration is a regular heptagon) and ask to find a sequence of admissible origami steps that yields a configuration with the desired property. Of course, it may turn out that, for certain properties, such a sequence cannot exist. In the case of ruler and compass geometry, solving questions are traditionally answered by Galois theory. In the case of origami solving, the corresponding study via Galois theory is open. The origami solving problem can also be considered under the aspect of automated algorithm synthesis. Recent progress in this area has been made within the *Theorema* system based on the use of algorithm schemes and the automated analysis of failing correctness proofs, see [5]. We expect that the combination of this heuristic method with Galois techniques may lead to new insights and practical techniques for the origami solving problem and to similar solving problems in general.

References

1. Alperin,R.C.: A Mathematical Theory of Origami Constructions and Numbers. New York J. Math. **6** 119–133 (2000).
2. Buchberger, B.: Gröbner-Bases: An Algorithmic Method in Polynomial Ideal Theory. Chapter 6 in: N.K. Bose (ed.), Multidimensional Systems Theory - Progress, Directions and Open Problems in Multidimensional Systems, , Dodrecht - Boston - Lancaster:Reidel Publishing Company 1985 (Second edition: Kluwer Academic Publisher 2003.)
3. Buchberger, B., Jebelean, T., Kriftner, F., Marin, M., Tomuta E., Vasaru, D.: An overview on the Theorema project. In: W. Kuechlin (ed.), Procdings of ISSAC'97 (International Symposium on Symbolic and Algebraic Computation, Maui, Hawaii, July 2123, 1997). ACM Press 1997.
4. Buchberger, B., Dupre, C., Jebelean, T., Kriftner, F., Nakagawa, K., Vasaru, D., Windsteiger, W.: The Theorema Project: A Progress Report, In: Kerber, M. and Kohlhase, M. (eds.): Symbolic Computation and Automated Reasoning: The Calculemus-2000 Symposium (Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, August 6-7, 2000, St. Andrews, Scotland). A K Peters Ltd 2001.
5. Buchberger, B.: Towards the Automated Synthesis of a Groebner bases Algorithm. RACSAM, Reviews of the Spanish Royal Academy of Science. Serie A: Mathematicas **98**(1), 65−75 (2004).
6. Chou, S.C.: Mechanical geometry theorem proving. Dordrecht Boston: Reidel 1988
7. Geretschläger, R.: Euclidean constructions and the geometry of origami. Math. Mag. **68**(5), 357−371 (1995).

8. Gleason, A. M.: Angle Trisection, the Heptagon, and the Triskaidecagon. American Mathematical Monthly **95**(3) 185−194 (1998).
9. Huzita, H.: Axiomatic Development of Origami Geometry. Proceedings of the First International Meeting of Origami Science and Technology, 143−158 (1989).
10. Huzita, H.: Drawing the regular heptagon and the regular nonagon by origami (paper folding). Symmetry: Culture and Science **5**(1), 69−84 (1994).
11. Ida T., Ţepeneu D., Buchberger B., Robu J.: Proving and Constraint Solving in Computational Origami. In: Buchberger B., Campbell J. (eds.): Proceedings of AISC 2004 (7 th International Conference on Artificial Intelligence and Symbolic Computation, September 22-24, 2004, RISC, Johannes Kepler University, Austria). Lecture Notes in Artificial Intelligence 3249. Berlin: Springer 2004.
12. Kapur, D.: Using Gröbner bases to reason about geometry problems. J. Symbolic Computation **2**(4), 399−408 (1986).
13. Kutzler, B., Stifter, S.: On the application of Buchberger's algorithm to automated geometry theorem proving. J. Symb. Comput. **2**, 389−397 (1986).
14. Robu, J.: Automated Geometric Theorem Proving (PhD Thesis), RISC-Linz Report Series No. 02-23. Johannes Kepler University Linz, Austria (2002).
15. Ţepeneu, D., Ida, T., MathGridLink - A bridge between Mathematica and "the Grid", The 20th Annual Conference of Japan Society of Software Science and Technology, Nagoya, September 2003.
16. Wu, W.t.: Basic principles of mechanical theorem proving in elementary geometries. J. Automat. Reason. **2**, 221−252 (1986).

Proving Geometric Theorems by Partitioned-Parametric Gröbner Bases*

Xuefeng Chen, Peng Li, Long Lin, and Dingkang Wang

Key Laboratory of Mathematics Mechanization,
Academy of Mathematics and System Sciences,
Chinese Academy of Sciences,
Beijing 100080, P.R. China
{xfchen, pli, llin, dwang}@mmrc.iss.ac.cn

Abstract. The notion of partitioned-parametric Gröbner bases of a polynomial ideal under constraints is introduced and an algorithm for constructing partitioned-parametric Gröbner bases is given; the correctness and the termination of the algorithm are proved. We also present a method based on computing partitioned-parametric Gröbner bases for proving geometric theorems mechanically. By this method, besides proving the generic truth of a geometric theorem, we can give the necessary and sufficient conditions on the free parameters for the theorem to be true. An example for proving geometric theorems by the partitioned-parametric Gröbner bases method is given.

1 Introduction

Many geometric statements can be formulated in terms of polynomial equations, and such algebraic formulations usually involve a number of parameters. An important problem concerning proving geometric theorems is to determine whether a geometric statement is valid under a specialization of parameters.

In detail, a geometric statement of equality-type consists of two parts: hypotheses and conclusion. Both hypotheses and conclusion can be expressed in terms of polynomial equations in a number of free arbitrary coordinates $u_1, \ldots,$ u_m, which we call parameters, and a number of dependent coordinates $x_1, \ldots, x_n$, which we call variables. Typically, the hypotheses are composed of

$$\begin{cases} h_1(u_1, \ldots, u_m, x_1, \ldots, x_n) = 0, \\ \cdots\cdots \\ h_r(u_1, \ldots, u_m, x_1, \ldots, x_n) = 0, \end{cases} \tag{1}$$

where the h's are polynomials over a ground field K. The conclusion is

$$g(u_1, \ldots, u_m, x_1, \ldots, x_n) = 0, \tag{2}$$

where g is a polynomial over K. The problem to be considered is:

* Partially supported by a NKBRPC (2004CB318000).

H. Hong and D. Wang (Eds.): ADG 2004, LNAI 3763, pp. 34–43, 2006.

Find all the constraints, viewed as polynomial equations and inequations in $u_1, \ldots, u_m$, *such that (1) implies (2).*

For most geometric theorems, the conclusion does not strictly follow from the hypotheses; there are some so-called degenerate cases. Wu introduced the characteristic set method to prove geometric theorems and the "non-degenerate" conditions can be given automatically [12]. This method has been successfully used to prove many difficult geometric theorems, and to discover new theorems [3, 4, 11]. The application of the Gröbner bases method to geometric theorem-proving has been investigated in [3, 6, 10]. An algorithm based on Gröbner bases is presented in [10] for deriving simplest degeneracy conditions for geometric theorems.

Inspired by the work in [7, 9], in particular, the notion of parametric Gröbner bases in [7], in this paper we introduce the notion of partitioned-parametric Gröbner bases and based on it a method for analyzing the parameters involved in an algebraic formulation of a geometric statement. This method partitions the parametric space into finitely many subsets defined by polynomial equations and inequations (i.e., parametric constraints), and show clearly on which subsets the statement is valid and on which it is invalid. In other wrods, the necessary and sufficient conditions on the parameters for a geometric statement to be true can be given by this method.

Recently, we found that Montes [8] also presented an algorithm for discussing Gröbner bases with parameters.

In the next section, the method for checking the consistency of a polynomial constraint is described. In Section 3, the notion of parametric partition of a constrained polynomial set is introduced and an algorithm for constructing the parametric partition is described. In Section 4, the notion of partitioned-parametric Gröbner bases of an ideal under a constraint is introduced and an algorithm for computing partitioned-parametric Gröbner bases is presented. In Section 5, a partitioned-parametric Gröbner bases method for proving geometric theorems is proposed and an example is given to show how to use this method to prove geometric theorems mechanically.

2 Constraints over the Parameters

Let K be a computable field and E be an algebraically closed field containing K. For simplification, let $u = (u_1, \ldots, u_m)$, where $u_1, \ldots, u_m$ are parameters. $K[u]$ denotes the polynomial ring $K[u_1, \ldots, u_m]$.

A *constraint* is viewed as a set of polynomial equations and inequations over parameters, denoted by

$$C = \{c_1 = 0, \ldots, c_s = 0, d_1 \neq 0, \ldots, d_t \neq 0\},\ c_i, d_j \in K[u], \tag{3}$$

which is true or false, depending on values in E substituted for parameters appearing in the constraint.

Let C be a constraint over the parameters; a set $S(C) \subset E^m$ is defined as

$$S(C) = \{u' \in E^m | u' \text{ satisfy the constraint } C\}.$$

Especially, $S(C) = E^m$ when C is the empty set.

A constraint C is said to be *consistent* if $S(C)$ is not an empty set.

A Gröbner bases algorithm or a characteristic set algorithm can be used for checking the consistency of a constraint C of form (3).

- **GB method:** by introducing $y_1, \ldots, y_t$, let $d'_j = d_j y_j - 1, j = 1, \ldots, t$, and $C' = \{c_1, \ldots, c_s, d'_1, \ldots, d'_t\} \subset K[u, y]$, where $y = (y_1, \ldots, y_t)$; then C is consistent if and only if $\{1\}$ is not the reduced Gröbner basis of C'.
- **CS method:** $S(C)$ can be considered as a quasi-variety in E^m. Whether $S(C)$ is an empty set can be detected by computing its projection [2]. Moreover, the methods of regular decomposition or irreducible decomposition of $S(C)$ can also be used to detect its consistency [11, 12].

For a polynomial constraint, the following proposition is obvious.

Proposition 1. *If C is a constraint and p is a polynomial in $K[u]$, then one and only one of the following three cases should be satisfied:*

(a) $C \cup \{p \neq 0\}$ is not consistent, which can be equally described as for each $u' \in S(C)$, $p(u') = 0$, i.e., p can be considered as a zero function under $S(C)$.
(b) $C \cup \{p = 0\}$ is not consistent, which can be equally described as for each $u' \in S(C)$, $p(u') \neq 0$, i.e., p as a function is nonzero on $S(C)$.
(c) Both $C \cup \{p = 0\}$ and $C \cup \{p \neq 0\}$ are consistent.

3 Parametric Partition of a Constrained Polynomial Set

Let $u = (u_1, \ldots, u_m)$ and $x = (x_1, \ldots, x_n)$. By $K[u, x]$, we denote the polynomial ring with indeterminates u and x over K. Let f be a polynomial in $K[u, x]$ and u' be a specialization of u, and $f(u', x)$ denotes the polynomial obtained by substituting u' for u. Let F be a set of polynomials in $K[u, x]$, and $F(u', x)$ be the set of polynomials obtained by substituting u' for u into the polynomials in F.

In [5], some terminologies about polynomials have been introduced. We will extend them to polynomials with parameters.

Definition 1. *Let f be a nonzero polynomial in $K[u, x]$, where f can be considered as a polynomial in $K[u][x]$, and $f = \sum_\alpha a_\alpha x^\alpha$, where $a_\alpha \in K[u]$. Let $>$ be a monomial order on x.*

(a) The multidegree *of f is*

$$\text{multideg}(f) = \max(\alpha \in Z^n_{\geq 0} : a_\alpha \neq 0).$$

(b) The leading coefficient *of f is*

$$\text{lc}(f) = a_{\text{multideg}(f)} \in K[u].$$

In particular, for $f \in K[u]$, $\text{lc}(f) = f$.

(c) The leading monomial *of f is*

$$\mathrm{lm}(f) = x^{\mathrm{multideg}(f)}.$$

(d) The leading term *of f is*

$$\mathrm{lt}(f) = \mathrm{lc}(f)\mathrm{lm}(f).$$

In paper [7], Kapur defined a constrained polynomial as a pair (C, f), where C is a consistent constraint and f is a polynomial in $K[u, x]$. The constraint polynomial (C, f) is unambiguous if $\mathrm{lc}(f)(u') \neq 0, \forall u' \in S(C)$.

In the following, we will extend constrained polynomial and unambiguous polynomial to constrained polynomial set and unambiguous polynomial set.

Definition 2. *A* constrained polynomial set *is a pair (C, F), where C is a consistent constraint over the parameters u, and F is a finite set of polynomials in $K[u, x]$. A constrained polynomial set (C, F) is an* unambiguous polynomial set *if for all u' in $S(C)$ and for all f in F, $lc(f)(u') \neq 0$.*

For example, $(\{u_1 - u_2 = 0, u_3 = 0, u_1 \neq 0, u_2 \neq 0\}, \{x_1 - u_1, 2u_1u_2x_1 + 1\})$ and $(\{u_1 = 0, u_2 = 0\}, \{1\})$ are two unambiguous polynomial sets.

Now, we define the parametric partition of a constrained polynomial set.

Definition 3. *A set $\{(C_1, F_1), \ldots, (C_s, F_s)\}$ of unambiguous polynomial sets is a* parametric partition *of a constrained polynomial set (C, F) if it satisfies the following conditions:*

(a) $S(C_1), \ldots, S(C_s)$ is a partition of $S(C)$, i.e., $\bigcup_{i=1}^{s} S(C_i) = S(C)$ and $S(C_i) \cap S(C_j) = \emptyset$, for $1 \leq i \neq j \leq s$;
(b) $\forall u' \in S(C)$, if $u' \in S(C_i)$ then $F_i(u', x)$ and $F(u', x)$ generate the same ideal in $K(u')[x]$, where $K(u')$ is the field generated by u' over K.

Let F be a polynomial set; the parametric partition of $(\emptyset, F)$ will be called the parametric partition of F.

Theorem 1. *For any constrained polynomial set, there is an algorithm to compute its parametric partition in finite steps.*

Proof. Let (C, F) be an arbitrary constraint polynomial set. First we will consider the case where F consists of only one polynomial, i.e., suppose that $F = \{f\}$. According to Proposition 1, we know that:

1. If $C \cup \{\mathrm{lc}(f) \neq 0\}$ is not consistent, then $\mathrm{lc}(f)$ is zero on $S(C)$; let $f' = f - \mathrm{lt}(f)$. It is obvious that the parametric partition of $(C, \{f'\})$ is exactly the one of $(C, \{f\})$.
2. If $C \cup \{\mathrm{lc}(f) = 0\}$ is not consistent, then $\mathrm{lc}(f)$ is nonzero on $S(C)$, and $(C, \{f\})$ is the parametric partition of itself.
3. Otherwise, both $C \cup \{\mathrm{lc}(f) = 0\}$ and $C \cup \{\mathrm{lc}(f) \neq 0\}$ are consistent; then the union of $\{(C \cup \{\mathrm{lc}(f) \neq 0\}, \{f\})\}$ and the parametric partition of $(C \cup \{\mathrm{lc}(f) = 0\}, \{f - \mathrm{lt}(f)\})$ is the parametric partition of $(C, \{f\})$.

Since f has a finite number of terms, the above process will terminate in finite steps and the number of the unambiguous polynomial sets in the parametric partition of $(C, \{f\})$ is also finite. It is easy to check that the above process will give the parametric partition of $(C, \{f\})$.

If F has more than one polynomial, then suppose that $F = \{f_1, \ldots, f_{k-1}, f_k\}$, and that $\{(C_1, F_1), \ldots, (C_t, F_t)\}$ is a parametric partition of $\{f_1, \ldots, f_{k-1}\}$. Let $\{(C_{i1}, F_{i1}), \ldots (C_{ik_i}, F_{i,k_i})\}$ be the parametric partition of $(C_i, \{f_k\})$; it is easy to check that $(C_{ij}, F_{ij} \cup F_i)$ for $i = 1, \ldots, t$, $j = 1, \ldots, k_i$ is the parametric partition of (C, F).

For example, $f = vxy + ux^2 + x$, $g = uy^2 + x^2$, $F = \{f, g\}$, assuming a lexicographic order on terms defined by the variable order $y > x$. First, we will construct the parametric partition of $(\emptyset, \{f\})$, It is $\{(C_1, \{f\}), (C_2, \{ux^2 + x\}), (C_3, \{x\})\}$, $C_1 = \{v \neq 0\}$, $C_2 = \{v = 0, u \neq 0\}$, $C_3 = \{v = 0, u = 0\}$. Then, we will construct the parametric partitions of $(C_1, \{g\})$, $(C_2, \{g\})$ and $(C_3, \{g\})$. The parametric partitions of $(C_1, \{g\})$, $(C_2, \{g\})$ and $(C_3, \{g\})$ are $\{(\{v \neq 0, u \neq 0\}, \{g\}), (\{v \neq 0, u = 0\}, \{x^2\})\}$, $\{(C_2, \{g\})\}$ and $\{(C_3, \{x^2\})\}$ respectively. The parametric partition of $(\emptyset, F)$ will be $\{(\{v \neq 0, u \neq 0\}, \{f, g\}), (\{v \neq 0, u = 0\}, \{f, x^2\}), (C_2, \{ux^2 + x, g\}), (C_3, \{x, x^2\})\}$.

4 Partitioned-Parametric Gröbner Bases

Let F be a polynomial set and u' be an element in E; we use I_F to denote the ideal generated by F in $K[u, x]$. Let

$$I_F(u', x) = \{p \mid p \text{ can be written as } f(u', x)/g(u'), f \in I_F, g \in K[u], g(u') \neq 0\};$$

it is easy to check that $I_F(u', x)$ is an ideal in $K(u')[x]$.

Definition 4. *Let (C, F) be a constrained polynomial set; a parametric partition $(C_1, G_1), \ldots, (C_s, G_s)$ of (C, F) is called the* (reduced) partitioned-parametric Gröbner basis *of the ideal I_F under the constraint C if: $\forall u' \in S(C)$, if $u' \in S(C_i)$ then $G_i(u', x)$ is the (reduced) Gröbner basis of $I_F(u', x)$.*

The partitioned-parametric Gröbner basis of the ideal I_F under constraint $\emptyset$ will be called the partitioned-parametric Gröbner basis of the ideal I_F.

Two important operations in Gröbner bases computation are that of computing an *S-polynomial* of a pair of distinct polynomials and the *remainder* on division of a polynomial by one polynomial list. Below we extend these two notations to polynomials with parameters.

For a polynomial f in $K[u, x]$, for example $f = u^3x + x^2 + 1$, for a lexicographic order with $u > x$, the leading monomial will be u^3x. u^3x becomes 0 by specifying u to 0, which is different from the leading monomial of $f|_{u=0}$. We will define the following *S-polynomial* in $K(u)[x]$.

Definition 5. *Let f, g be two polynomials in $K[u, x]$. Suppose that* multideg$(f) = \alpha$, multideg$(g) = \beta$, *and let* $\gamma = (\gamma_1, \ldots, \gamma_n)$, $\gamma_i = \max(\alpha_i, \beta_i)$. *The* S-polynomial *of f and g is the combination*

$$\text{spoly}(f, g) = \frac{x^\gamma}{\text{lt}(f)} \cdot f - \frac{x^\gamma}{\text{lt}(g)} \cdot g = \frac{x^\gamma \cdot (f \cdot \text{lt}(g) - g \cdot \text{lt}(f))}{\text{lt}(f) \cdot \text{lt}(g)},$$

which is in $K(u)[x]$.

For example, $f = vxy + ux^2 + x$, $g = uy^2 + x^2$, assuming a lexicographic order on terms defined by the variable order $y > x$,

$$\text{spoly}(f, g) = \frac{u^2x^2y + uxy - vx^3}{uv}.$$

Definition 6. *We will write $\bar{f}^F$ for the* remainder *on division of $f \in K[u, x]$ by the ordered s-tuple $F = (f_1, \ldots, f_s) \subset K[u, x]$; $\bar{f}^F$ can be written as*

$$\bar{f}^F = f - \frac{a_1 f_1}{lc(f_1)} - \frac{a_2 f_2}{lc(f_2)} - \cdots - \frac{a_s f_s}{lc(f_s)},$$

where $a_1, \ldots, a_s$ are in $K[u, x]$. $\overline{f}^F$ is a linear combination with coefficients in $K(u)$, of monomials, none of which is divisible by any of $\text{lm}(f_1), \ldots, \text{lm}(f_s)$.

For example, $F = \{vxy + ux^2 + x, uy^2 + x^2\}$ and $f = vy^2 + ux^3y + y$, assuming a lexicographic order on terms defined by the variable order $y > x$. Then

$$\bar{f}^F = \frac{vuy - v^2x^2 - u^3x^4 - u^2x^3}{uv}.$$

Let f be a polynomial in $K(u)[x]$. We use num(f) to denote the numerator of f; num(f) is in $K[u, x]$.

Theorem 2. *The parametric partition $\{(C_1, G_1), \ldots, (C_s, G_s)\}$ of a constrained polynomial set (C, F) is the partitioned-parametric Gröbner basis of I_F under constraint C if and only if for each i, $\forall f, g \in G_i$,* num$(\overline{\text{num}(\text{spoly}(f, g))}^{G_i})$ *is a zero polynomial on $S(C_i)$.*

Proof. Since f and g are in G_i, the denominator of spoly(f, g) is the product of lc(f) and lc(g) by Definitions 5 and 6. The leading coefficients of the polynomials in G_i will be nonzero on $S(C_i)$, so that the denominator of spoly(f, g) will be nonzero on $S(C_i)$. For the same reason, the denominator of $\overline{\text{num}(\text{spoly}(f, g))}^{G_i}$ is also nonzero on $S(C_i)$.

By the definition of parametric partition, we know that for each $u' \in S(C_i)$, $G_i(u', x)$ and $F(u', x)$ generate the same ideal in $K(u')[x]$. By Buchberger's S-polynomial criterion of Gröbner bases [1], we know that for each u' in $S(C_i)$, $G_i(u', x)$ is the Gröbner basis of $I_{F(u',x)}$.

Theorem 3. *For any constrained polynomial set, there is an algorithm to compute out its partitioned-parametric Gröbner basis in finite steps.*

We give the algorithm first.

Algorithm: PPGB(C, F)
Input: (C, F) is a constrained polynomial set;
Output: A partitioned-parametric Gröbner basis of (C, F).

1. If $F = \{1\}$ then return (C, F).
2. Let $(C_1, F_1,), \ldots, (C_s, F_s)$ be the parametric partition of (C, F).
3. For each i, compute the partitioned-parametric Gröbner basis of (C_i, F_i).
   - SP(F_i)=$\{\overline{\text{spoly}(f,g)}^{F_i} \mid$ for each pair $f, g \in F_i\}$.
   - If for each h in SP(F_i), for each $u' \in S(C_i)$, $h(u', x)$ is 0 as a polynomial in $K[u', x]$, then (C_i, F_i) is a partitioned-parametric Gröbner basis of (C_i, F_i).
   - Otherwise, compute the partitioned-parametric Gröbner basis of $(C_i, F_i \cup \text{SP}(F_i))$.
4. Return the union of the partitioned-parametric Gröbner bases of (C_i, F_i).

It should be noticed that the polynomials in SP(F_i) will be in $K(u)[x]$, and their denominators will be nonzero on $S(C_i)$. These polynomials can be replaced by their numerators which are in $K[u, x]$.

Proof. The correctness of the algorithm is guaranteed by Theorem 2. Now we prove that the algorithm terminates in finite steps. It is well known that Buchberger's algorithm for computing Gröbner bases terminates in finite steps and the reason is that during the loop of successive iterations through expanding the original polynomial set with the nonzero remainders of S-polynomials, the leading terms of the ever-increasing polynomial set form an ascending chain of ideals. As for the partitioned-parametric case, it becomes a litter more complicated. On the one hand, it is easy to see that the ascending chain of ideals does also exist for the leading coefficients are certainly nonzero under corresponding constraint. On the other hand, in step 2, $(C_1, F_1), \ldots, (C_s, F_s)$ is the parametric partition of (C, F), and s is a finite number. So the algorithm PPGB forms a tree structure of unambiguous polynomial sets, and the two sides prove that the length of the tree is finite and the node number of the same layer is finite respectively. So the number of leaves, which are unambiguous polynomial sets, is finite too. This proves the termination of the algorithm.

5 Proving Geometric Theorem by Partitioned-Parametric Gröbner Bases

The following theorem can solve the radical ideal membership problem.

Theorem 4 (Radical Ideal Membership). *Let F be a finite set of polynomials in $K[x]$ and g be a polynomial in $K[x]$. Then g is in the radical of the ideal I_F if and only if $\{1\}$ is the reduced Gröbner basis of $(F, gy - 1)$.*

Proof. See [1, 5].

We can extend the above theorem to the case of polynomial ideals involving parameters to establish the following theorem, which can solve the parametric radical ideal membership problem.

Theorem 5 (Parametric Radical Ideal Membership). *Let $h_1, \ldots, h_r, g$ be polynomials in $K[u, x]$, G=$\{(C_1, G_1), \ldots, (C_s, G_s)\}$ be the reduced partitioned-parametric Gröbner basis of the ideal generated by $h_1, \ldots, h_r, gy - 1$ under constraint C. Then $\forall u' \in S(C_i)$, $g(u', x)$ is in the radical of the ideal generated by $h_1(u', x)$, …, $h_r(u', x)$ in $K(u')[x]$ if and only if $G_i = \{1\}$.*

Proof. It is obvious according to the definition of partitioned-parametric Gröbner bases and Theorem 4.

Based on the above theorem, we propose the following method to prove geometric theorems mechanically.

For a geometric theorem, hypotheses can be expressed by a set of polynomial equations: $\{h_1 = 0, \ldots, h_r = 0\}$, and the conclusion can be expressed by a polynomial equation: $g = 0$. The polynomials h_i and g are in $K[u, x]$, where the u's are parameters and x's are variables. Generally the conclusion $g = 0$ does not strictly follow from the hypotheses $\{h_1 = 0, \ldots, h_r = 0\}$.

Let $F = \{h_1, \ldots, h_r, gy - 1\}$, for any term order on x and y, and let

$$\{(C_1, G_1), \ldots, (C_r, G_r), (C_{r+1}, G_{r+1}), \ldots, (C_s, G_s)\}$$

be the reduced partitioned-parametric Gröbner basis of I_F. Assume that $G_i = \{1\}$ for $i = 1, \ldots, r$ and $G_i \neq \{1\}$ for $i = r+1, \ldots, s$; then the geometric theorem is true under the constraints $C_1, \ldots, C_r$ and the geometric theorem is false under the constraints $C_{r+1}, \ldots, C_s$.

Consider the following example.

Example 1. The bisectors of the three angles of an arbitrary triangle, three-to-three, intersect at four points. In other words, let the triangle be ΔABC, the two bisectors of $\angle A$ and $\angle B$ intersect at point D. We need to show that CD is the bisector of $\angle C$.

To simplify calculation, and without loss of generality, we take the coordinates of the points as $A(u_1, 0), B(u_2, 0), C(0, u_3), D(x_1, x_2)$. The hypotheses of the theorem are expressed as:

$$h_1 = u_3[x_2^2 - (x_1 - u_1)^2] - 2u_1x_2(x_1 - u_1) = 0 \quad (DA \text{ is the bisector of } \angle CAB)$$
$$h_2 = u_3[x_2^2 - (x_1 - u_2)^2] - 2u_2x_2(x_1 - u_2) = 0 \quad (DB \text{ is the bisector of } \angle ABC)$$

The conclusion to be proved is

$$\begin{aligned} g = {} & [u_1(x_2 - u_3) + u_3x_1][u_3(x_2 - u_3) - u_2x_1] \\ & + [u_2(x_2 - u_3) + u_3x_1][u_3(x_2 - u_3) - u_1x_1] = 0. \end{aligned}$$

Compute the partitioned-parametric Gröbner basis of $\{h_1, h_2, gy - 1\}$ with respect to the graded lex order with tie broken by $y > x_2 > x_1$. Here u_1, u_2, u_3

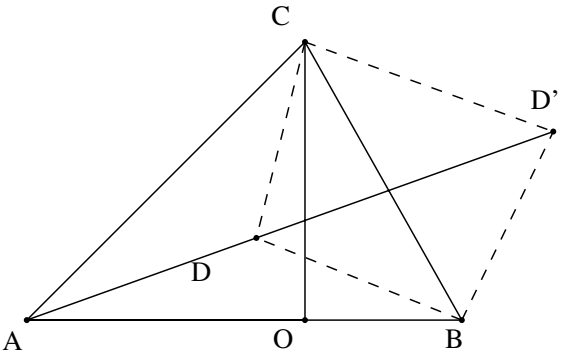


Fig. 1. Three bisectors pass through the same point

are considered as parameters. The partitioned-parametric Gröbner basis is $G = (C_1, G_1), \ldots, (C_7, G_7)$, where

$$
\begin{aligned}
&C_1 = \{u_1 = 0, u_3 = 0\}, \quad G_1 = \{1\};\\
&C_2 = \{-u_2 + u_1 \neq 0, u_3 \neq 0\}, \quad G_2 = \{1\};\\
&C_3 = \{u_3 = 0, -u_2 + u_1 \neq 0, u_1 \neq 0, u_2 \neq 0\}, \quad G_3 = \{1\};\\
&C_4 = \{u_2 = 0, u_3 = 0, u_1 \neq 0\}, \quad G_4 = \{1\};\\
&C_5 = \{u_3^2 + u_1^2 = 0, -u_2 + u_1 = 0, u_3 \neq 0\}, \quad G_5 = \{1\};\\
&C_6 = \{-u_2 + u_1 = 0, u_3 = 0, u_1 \neq 0, u_2 \neq 0\}, \quad G_6 = \{x_1 - u_1, 1 + 2x_2 y u_1^3\};\\
&C_7 = \{-u_2 + u_1 = 0, u_3^2 + u_1^2 \neq 0, u_3 \neq 0\},\\
&G_7 = \{u_3 x_2^2 - u_3 x_1^2 + 2u_3 u_1 x_1 - u_3 u_1^2 - 2u_1 x_2 x_1 + 2u_1^2 x_2,\\
&\quad 2y u_1 u_3^3 + 2y u_3 u_1^3 - 2y u_3^3 x_1 - 4y u_1 x_2 u_3^2 - 2u_3 x_1 y u_1^2 - 4x_2 y u_1^3\\
&\quad + 2y u_3^2 x_1 x_2 + 2x_1 x_2 y u_1^2 - 1, 2u_3^5 y u_1 + 2u_3^3 y u_1^3 - 2u_3^5 y x_1 - 2y u_1 x_2 u_3^4\\
&\quad - 2y u_1^3 x_2 u_3^2 + 2x_1 y u_3 u_1^4 - 4y u_1 u_3^3 x_1^2 - 4y u_1^3 u_3 x_1^2 + 2u_3^3 y x_1^3\\
&\quad + 2u_3 y x_1^3 u_1^2 - u_3^2 - 2u_1^2 - u_3 x_2 + 2u_1 x_1\}.
\end{aligned}
$$

From this partitioned-parametric Gröbner basis G, one can see that the conclusion $g = 0$ can be deduced from the hypotheses $h_1 = 0, h_2 = 0$ if and only if the free parameters u_1, u_2, u_3 satisfy one of the constraints $C_1, \ldots, C_5$. From

$$C_2 = \{-u_2 + u_1 \neq 0, u_3 \neq 0\}, \quad G_2 = \{1\},$$

we know that the theorem is generically true.

If the variety defined by the hypotheses of a geometric statement is reducible, this method for proving the geometric theorem cannot determine if the conclusion of the geometric statement is true on some components of the hypotheses. For example, when the hypothesis is $x^2 - u^2 = 0$ and the conclusion is $x - u = 0$, the variety defined by $x^2 - u^2 = 0$ is reducible and there are 2 components: one

is $x - u = 0$ and the other is $x + u = 0$. We cannot deduce that the conclusion is true on the component $x - u = 0$ by our method.

6 Conclusion

In this paper, for any geometric theorem expressed as an algebraic formulation which involves both parameters and variables, we present a method of partitioned-parametric Gröbner bases to partition the parametric space to finitely many subsets. We can give all the partitions of the parameter space on which the geometric theorem is true.

Our partitioned-parametric Gröbner bases method comes from Kapur's parametric Gröbner bases and has more advantages in the structure and expression. The partitioned-parametric Gröbner bases can be applied for solving many problems about parametric polynomial systems, such as parametric ideal membership, the number of solutions of a parametric polynomial equation system and elimination of quantifier-blocks in algebraically closed fields.

We should thank Z. M. Li for helpful discussions.

References

1. Buchberger, B.: Gröbner Bases: An Algorithmic Method in Polynomial Ideal theorey. In: Multidimensional Systems Theory (N. K. Bose, ed.), D. Reidel Publishing Co., Dordrecht Boston, 184–232 (1985).
2. Chen, X. F., Wang, D. K.: The Projection of Quasi Variety and Its Application on Geometric Theorem Proving and Formula Deduction, In: Automated Deduction in Geometry (F. Winkler, ed.), LNAI **2930**, Springer-Verlag, Berlin Heidelberg, 21–30 (2004).
3. Chou, S.-C.: Mechanical Geometry Theorem Proving. D. Reidel Publishing Co., Dordrecht Boston (1988).
4. Chou, S.-C., Gao, X.-S.: Methods for Mechanical Geometry Formula Deriving. In: Proc. ISSAC '90, ACM Press, New York, 265–270 (1990).
5. Cox, D., Little, J., O'Shea, D.: Ideal, Varieties, and Algorithems. Second Edition, Springer-Verlag, New York (1997).
6. Kapur, D.: Using Gröbner Bases to Reason About Geometry Problems. J. Symbolic Computation **2**(4), 399–408 (1986).
7. Kapur, D.: An Approach for Solving Systems of Parametric Polynomial Equations, In: Principles and Practice of Constraint Programming (Saraswat and Van Hentenryck, eds.), MIT Press, Cambridge (1995).
8. Montes, A.: A New Algorithm for Discussing Gröbner Bases with Parameters. J. Symbolic Computatio **33**(1-2), 183–208 (2002).
9. Weispfenning, V.: Comprehensive Gröbner Bases. J. Symbolic Computation **14**, 1–29 (1991).
10. Winkler, F.: Gröbner Bases in Geometry Theorem Proving and Simplest Degeneracy Conditions. Mathematica Pannonica **1**(1), 15–32 (1990).
11. Wang, D.: Elimination Methods, Springer-Verlag, Wien New York (2001).
12. Wu, W.-T.: Basic Principles of Mechanical Theorem-proving in Elementary Geometries. J. Sys. Sci. & Math. Scis. **4**, 207–235 (1984).

Computations of the Area and Radius of Cyclic Polygons Given by the Lengths of Sides

Pavel Pech

University of South Bohemia, Jeronýmova 10,
371 15 České Budějovice, Czech Republic
pech@pf.jcu.cz

Abstract. Some properties of inscribed polygons, i.e., such plane polygons whose vertices lie on a circle, are investigated. Given an inscribed polygon with the lengths of its sides, we explore the area and radius of its circumcircle. We start with a triangle and a quadrangle and then we will explore the case of a pentagon. All the computations are based on results of commutative algebra especially on Gröbner bases method and elimination of variables in a given ideal.

1 Introduction

Every student knows the formula of Heron for the area p of a triangle with given lengths of sides a, b, c

$$p = \sqrt{s(s-a)(s-b)(s-c)}, \tag{1}$$

where $s = (a+b+c)/2$. This formula has its name after Heron of Alexandria (cca. 60 B.C.) although it was likely known to Archimedes.

The formula of Brahmagupta (598 – about 665 A.D.) for the area p of a convex quadrilateral with given lengths of sides a, b, c, d which is inscribed in a circle

$$p = \sqrt{(s-a)(s-b)(s-c)(s-d)}, \tag{2}$$

where $s = (a+b+c+d)/2$, is a generalization of the Heron's formula.

Since then no similar formulas for the area of an inscribed (or cyclic) n-gon for $n > 4$, despite a great effort of mathematicians from all over the world, appeared until 1994, when D. P. Robbins published the paper [12]. See also the latest references [8, 16]. For almost 1400 years the formula for a cyclic pentagon was missing. In the meantime some papers for special cyclic polygons appeared, see [1, 5, 14]. The main reason why such a long time elapsed is a complexity of such formulas.

In this paper we will find the formula for the area of cyclic pentagon using purely computational methods based on results of computational commutative algebra like Gröbner bases of ideals, elimination of variables and the theory of automatic theorem proving, especially automatic derivation, see [3, 4, 10, 17]. We will start with well-known cases for $n = 3$ and $n = 4$ and then carry on for $n = 5$. All the computations were done on Intel Pentium 2.00GHz/1572MB RAM.

H. Hong and D. Wang (Eds.): ADG 2004, LNAI 3763, pp. 44–58, 2006.

2 Area of Polygons

If the vertices of a closed polygon have coordinates $A_i = [x_i, y_i]$, then for the signed area p of $A_1, A_2, \ldots, A_n$

$$p = \frac{1}{2} \sum_{i=1}^{n} \begin{vmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{vmatrix}, \tag{3}$$

see [2]. It it easy to verify, that (3) does not depend on choosing the coordinate system.

The formula (3) enables us to express the area of an n-gon knowing the coordinates of its vertices.

We can also express the area p of an n-gon $A_1A_2 \ldots A_n$ by means of all $\binom{n}{2}$ mutual distances of its vertices. We will use the formula (4), which is due to Nagy and Rédey [11].

Denote the square of the distance of vertices A_i, A_j by $d_{ij} = |A_iA_j|^2$; then

$$16p^2 = \sum_{i,j=1}^{n} \begin{vmatrix} d_{i,j} & d_{i,j+1} \\ d_{i+1,j} & d_{i+1,j+1} \end{vmatrix}. \tag{4}$$

2.1 Formula of Heron

The well-known formula of Heron is a special case of (4). We will derive it by computer.

A triangle ABC with sides a, b, c is given. Find the formula for the area p of ABC.

Choose the Cartesian coordinate system so that the coordinates of the vertices of a triangle ABC are $A = [0, 0], B = [c, 0], C = [x, y]$, see Fig. 1. We want to express the area p of the triangle ABC by the lengths of its sides $a = |BC|, b = |CA|, c = |AB|$. It is obvious that

$|AC| = b \Leftrightarrow h_1 : x^2 + y^2 - b^2 = 0,$
$|BC| = a \Leftrightarrow h_2 : (x - c)^2 + y^2 - a^2 = 0,$
$p = \text{area of } ABC \Leftrightarrow h_3 : p - \frac{1}{2}cy = 0.$

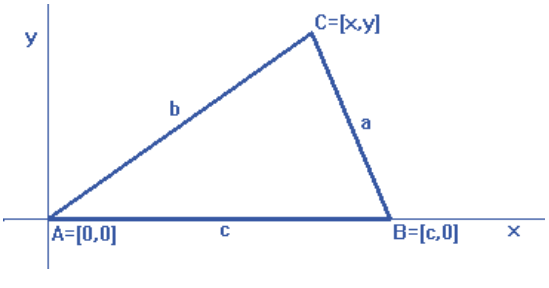


Fig. 1.

Let us construct the ideal $I = \langle x^2 + y^2 - b^2, (x-c)^2 + y^2 - a^2, p - \frac{1}{2}cy \rangle$ in the ring $R[a, b, c, x, y, p]$. We are searching for a formula, which describes a relation between the lengths of sides a, b, c of the triangle ABC and its area p. Such a polynomial should belong into the elimination ideal $I \cap R[a, b, c, p]$. Eliminating the variables x, y we obtain

$$16p^2 = -a^4 - b^4 - c^4 + 2a^2b^2 + 2a^2c^2 + 2b^2c^2, \tag{5}$$

which is the well-known formula of Heron.

We see that the right-hand side of (5) is a symmetric polynomial in a^2, b^2, c^2. Let us denote the elementary symmetric functions of a^2, b^2, c^2 by $k = a^2+b^2+c^2$, $l = a^2b^2 + b^2c^2 + c^2d^2$, $m = a^2b^2c^2$ and let $q = 16p^2$. Then (5) may be written in the form

$$k^2 - 4l + q = 0. \tag{6}$$

To evaluate the circumradius r of a triangle with sides a, b, c we can proceed in a similar way and obtain

$$s(k^2 - 4l) + m = 0, \tag{7}$$

where $s = r^2$. From (6) and (7) we get the formula

$$qs - m = 0, \tag{8}$$

which connects the area p with the circumradius r of ABC.

2.2 Formula of Staudt

We know, that lengths of four sides of a quadrilateral do not determine it. To determine it, we need one more condition, e.g. the length of a diagonal. But neither five conditions (four sides and a diagonal) is enough to determine a quadrilateral uniquely. We will suppose, that in a quadrilateral all 6 distances between its vertices are given and solve the next problem.

Let $ABCD$ be a quadrilateral with the lengths of sides a, b, c, d and diagonals e, f. Find the formula for the area p of the quadrilateral $ABCD$.

Choose the coordinate system so that the vertices of the quadrilateral $ABCD$ be $A = [0, 0], B = [a, 0], C = [x, y], D = [u, v]$ and $a = |AB|, b = |BC|, c = |CD|, d = |DA|, e = |AC|, f = |BD|$, see Fig. 2. We have the following relations:

$|BC| = b \Leftrightarrow h_1 : (x-a)^2 + y^2 - b^2 = 0,$
$|CD| = c \Leftrightarrow h_2 : (u-x)^2 + (v-y)^2 - c^2 = 0,$
$|DA| = d \Leftrightarrow h_3 : u^2 + v^2 - d^2 = 0,$
$|AC| = e \Leftrightarrow h_4 : x^2 + y^2 - e^2 = 0,$
$|BD| = f \Leftrightarrow h_5 : (u-a)^2 + v^2 - f^2 = 0.$

By (3) we get

$p =$ area of $ABCD \Leftrightarrow h_6 : p - 1/2(ay + xv - yu) = 0.$

We will eliminate variables x, y, u, v in the ideal $I = \langle h_1, h_2, \ldots, h_6 \rangle$.

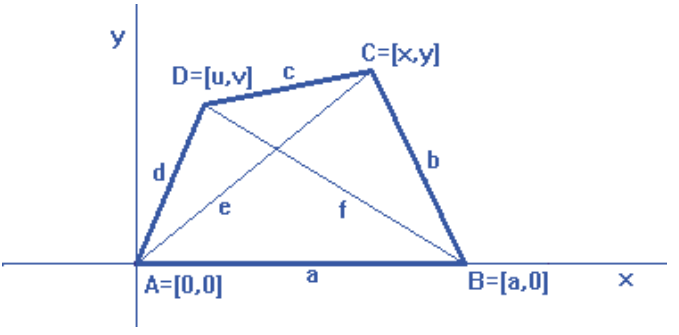


Fig. 2.

In CoCoA[1] we enter

```
Use R::= Q[xyuvpabcdef];
I:=Ideal((x-a)^2+y^2-b^2,(u-x)^2+(v-y)^2-c^2,u^2+v^2-d^2,x^2+y^2-
e^2,(u-a)^2+v^2-f^2,p-1/2(ay+xv-yu));
Elim(x..v,I);
```

We get four polynomials from which the following one is of our interest. After simplification we get

$$16p^2 = 4e^2f^2 - (a^2 - b^2 + c^2 - d^2)^2. \tag{9}$$

This formula, which is a special case of (4), was first published by Staudt [15].

Remark: Instead of the elimination of the variables x, y, u, v we can eliminate variables x, y, u, v, p in the same ideal I. This leads to the so-called *Euler's four points relation*, see [6]:

$$a^4c^2 - a^2b^2c^2 + a^2c^4 - a^2b^2d^2 + b^4d^2 - a^2c^2d^2 - b^2c^2d^2 + b^2d^4 + a^2b^2e^2 - a^2c^2e^2 - b^2d^2e^2 + c^2d^2e^2 - a^2c^2f^2 + b^2c^2f^2 + a^2d^2f^2 - b^2d^2f^2 - a^2e^2f^2 - b^2e^2f^2 - c^2e^2f^2 - d^2e^2f^2 + e^4f^2 + e^2f^4 = 0,$$

which expresses the mutual dependence of all six distances a, b, c, d, e, f between four vertices of a quadrilateral. Euler's four points relation follows from the Cayley–Menger determinant, see [1], for the volume V of a tetrahedron with edges of lengths a, b, c, d, e, f

$$288V^2 = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & b^2 & f^2 & a^2 \\ 1 & b^2 & 0 & c^2 & e^2 \\ 1 & f^2 & c^2 & 0 & d^2 \\ 1 & a^2 & e^2 & d^2 & 0 \end{vmatrix} \tag{10}$$

if we put $V = 0$. A comparison of the equation $V = 0$ with the equality which we received in the elimination process above shows that both of the polynomials are the same up to the constant factor 2.

[1] Software CoCoA is freely distributed at http://cocoa.dima.unige.it

2.3 Area of a Pentagon and Hexagon

Let us discover by computer the Nagy–Rédey formula (4) for $n = 5$ and $n = 6$.

Given a pentagon $ABCDE$ in a plane, denote the sides and diagonals by $a = |AB|$, $b = |BC|$, $c = |CD|$, $d = |DE|$, $e = |EA|$, $i_1 = |AC|$, $i_2 = |AD|$, $i_3 = |BD|$, $i_4 = |BE|$, $i_5 = |CE|$. Choose the Cartesian system of coordinates so that $A = [0, 0]$, $B = [a, 0]$, $C = [x, y]$, $D = [u, v]$, $E = [w, z]$, see Fig. 3. Then the following relations hold:

$$
\begin{aligned}
&|BC| = b \Leftrightarrow h_1 : (x-a)^2 + y^2 - b^2 = 0,\\
&|CD| = c \Leftrightarrow h_2 : (x-u)^2 + (y-v)^2 - c^2 = 0,\\
&|DE| = d \Leftrightarrow h_3 : (u-w)^2 + (v-z)^2 - d^2 = 0,\\
&|EA| = e \Leftrightarrow h_4 : w^2 + z^2 - e^2 = 0,\\
&|AC| = i_1 \Leftrightarrow h_5 : x^2 + y^2 - i_1^2 = 0,\\
&|AD| = i_2 \Leftrightarrow h_6 : u^2 + v^2 - i_2^2 = 0,\\
&|BD| = i_3 \Leftrightarrow h_7 : (u-a)^2 + v^2 - i_3^2 = 0,\\
&|BE| = i_4 \Leftrightarrow h_8 : (w-a)^2 + z^2 - i_4^2 = 0,\\
&|CE| = i_5 \Leftrightarrow h_9 : (x-w)^2 + (y-z)^2 - i_5^2 = 0,
\end{aligned}
$$

area of $ABCDE = p \Leftrightarrow h_{10} : p = 1/2(ay + xv - uy + uz - vw)$.

The elimination of x, y, u, v, w, z in the ideal $I = \langle h_1, h_2, \ldots, h_{10} \rangle$ takes in CoCoA 9m 18s and there appears

$$
\begin{aligned}
16p^2 = &-(a^4 + b^4 + c^4 + d^4 + e^4) + 2(a^2b^2 + b^2c^2 + c^2d^2 + d^2e^2 + e^2a^2) + 2(i_1^2 i_3^2 + \\
&i_2^2 i_4^2 + i_3^2 i_5^2 + i_4^2 i_1^2 + i_5^2 i_2^2) - 2(a^2 i_5^2 + b^2 i_2^2 + c^2 i_4^2 + d^2 i_1^2 + e^2 i_3^2),
\end{aligned}
$$

which is the Nagy–Rédey formula (4) for $n = 5$. In a similar way we get the following formula for a hexagon.

Denote $a = |AB|, b = |BC|, c = |CD|, d = |DE|, e = |EF|, f = |FA|, i_1 = |AC|, i_2 = |AD|, i_3 = |AE|, i_4 = |BD|, i_5 = |BE|, i_6 = |BF|, i_7 = |CE|, i_8 = |CF|, i_9 = |DF|$.

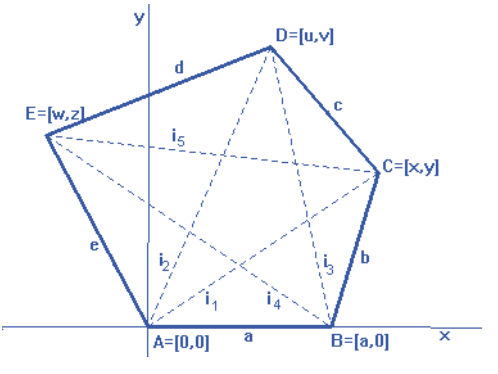


Fig. 3.

Choosing the Cartesian system of coordinates so that $A = [0, 0]$, $B = [a, 0]$, $C = [x, y]$, $D = [u, v]$, $E = [w, z]$, $F = [s, t]$, we express all distances b, c, d, e, f, i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8, i_9 by the coordinates $a, c, y, u, v, w, z, s, t$ with the area p of $ABCDEF$

$$p = 1/2(ay + xv - uy + uz - vw + wt - sz).$$

We will eliminate variables x, y, u, v, w, z, s, t. In 29m we get in Singular[2] (together with 121 further polynomials)

$$16p^2 = -(a^4 + b^4 + c^4 + d^4 + e^4 + f^4) + 2(a^2b^2 + b^2c^2 + c^2d^2 + d^2e^2 + e^2f^2 + f^2a^2) + 2(i_1^2i_4^2 + i_3^2i_4^2 + i_2^2i_5^2 + i_1^2i_6^2 + i_3^2i_6^2 + i_4^2i_7^2 - i_6^2i_7^2 + i_2^2i_8^2 + i_5^2i_8^2 - i_1^2i_9^2 + i_3^2i_9^2 + i_7^2i_9^2) - 2(a^2i_8^2 + b^2i_2^2 + c^2i_5^2 + d^2i_8^2 + e^2i_2^2 + f^2i_5^2),$$

which is the special case of (4) for $n = 6$.

3 Area of Cyclic Polygons

Now consider polygons which are inscribed in a circle. A polygon with vertices on a circle is called *inscribed* or *cyclic*. We will use the results of the previous paragraph to express the area of a cyclic n-gon for $n = 3, 4, 5$, which is given by the lengths of its sides.

For $n = 3$ we get formula of Heron, for $n = 4$ we obtain the well-known formula of Brahmagupta for the area of a convex cyclic quadrilateral. First we will deal with cyclic quadrilaterals to show the essence of the technique. Then we will derive the formula for the area of a cyclic pentagon.

We will use *two* basic methods to obtain the formulas for the area of a cyclic polygon and in the end we compare a time consumption of these two attitudes. Basically we can proceed in a *coordinate* way to introduce a Cartesian system of coordinates and express that the vertices of a cyclic n-gon lie on a circle etc., using the formula (3) for the area of an n-gon.

The other method, say *distance* or *coordinate-free* method, consists in the use of the Nagy–Rédey formula (4) and Ptolemy's type conditions. We shall see that the latter method is more effective.

3.1 Ptolemy's Type Conditions

Given a quadrilateral $ABCD$ with the lengths of sides $a = |AB|, b = |BC|, c = |CD|, d = |AD|$ and diagonals $e = |BD|, f = |AC|$, we will search for a necessary and sufficient condition for the quadrilateral $ABCD$ being cyclic. Denote the coordinates of the circumcenter by $O = [s, t]$ and the circumradius by r. In the position and notation as in Fig. 2 we have:

$r = |AS| \Leftrightarrow h_1 : s^2 + t^2 - r^2 = 0,$
$r = |BS| \Leftrightarrow h_2 : (a - s)^2 + t^2 - r^2 = 0,$
$r = |CS| \Leftrightarrow h_3 : (x - s)^2 + (y - t)^2 - r^2 = 0,$

[2] Software Singular is available at http://www.singular.uni-kl.de

$r = |DS| \Leftrightarrow h_4 : (u-s)^2 + (v-t)^2 - r^2 = 0,$
$b = |BC| \Leftrightarrow h_5 : (a-x)^2 + y^2 - b^2 = 0,$
$c = |CD| \Leftrightarrow h_6 : (u-x)^2 + (v-y)^2 - c^2 = 0,$
$d = |DA| \Leftrightarrow h_7 : u^2 + v^2 - d^2 = 0,$
$e = |AC| \Leftrightarrow h_8 : x^2 + y^2 - e^2 = 0,$
$f = |BD| \Leftrightarrow h_9 : (u-a)^2 + v^2 - f^2 = 0.$

In the ideal $I = \langle h_1, h_2, \ldots, h_9 \rangle$ we eliminate all the variables except a, b, c, d, e to find out the length e as a function of a, b, c, d. Similarly we get the length of f. For given lengths of sides a, b, c, d of a cyclic quadrilateral we obtain *two* lengths of diagonals e, f, see Fig. 4. For a convex cyclic quadrilateral we get

$$e^2(ab+cd) = (ac+bd)(ad+bc), \quad f^2(ad+bc) = (ac+bd)(ab+cd), \tag{11}$$

and for a non-convex case

$$e^2(-ab+cd) = (ac-bd)(ad-bc), \quad f^2(-ad+bc) = (ac-bd)(ab-cd). \tag{12}$$

From (11) the *Ptolemy's condition*

$$ef = ac+bd \tag{13}$$

for a convex case follows. In a non-convex case from (12) we receive Ptolemy's conditions

$$ef = ac-bd \quad \text{or} \quad ef = -ac+bd. \tag{14}$$

From (11) and (12) we get another interesting relations

$$e(ab+cd) = f(ad+bc) \tag{15}$$

in a convex case and

$$e(ab-cd) = f(ad-bc) \quad \text{or} \quad e(ab-cd) = f(-ad+bc) \tag{16}$$

for non-convex case. Later we will need such formulas to compute the area of a cyclic pentagon. See [5], where relations between sides and diagonals of a cyclic quadrilateral are discussed in detail.

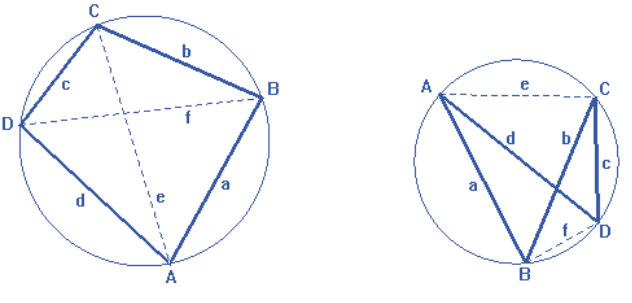


Fig. 4. Cyclic quadrilaterals with the sides a, b, c, d — convex and non-convex cases

It is easy to show that each of the conditions (13) and (14) is sufficient for a quadrilateral $ABCD$ being cyclic. Conversely we will show that the conditions (13) and (14) are necessary as well. We enter

```
Use R::= Q[abcdefxyuvt];
I:=Ideal((x-a)^2+y^2-b^2,(u-x)^2+(v-y)^2-c^2,u^2+v^2-d^2,x^2
+y^2-e^2,(u-a)^2-v^2-f^2,-yu^2+x^2v+y^2v-yv^2+yua-xva,(ac-bd
-ef)(ac+bd-ef)(ac-bd+ef)(ac+bd+ef)t-1);
NF(1,I);
```

and get 0. We discovered and proved the Ptolemy's theorem which reads [7]:

A quadrilateral with lengths of sides a, b, c, d and diagonals e, f is cyclic iff

$$(ac - bd - ef)(ac + bd - ef)(ac - bd + ef)(ac + bd + ef) = 0. \tag{17}$$

Remarks:
1) Note that we had to add (and find) one more condition $ac + bd + ef = 0$.
2) The condition (17) can be expressed as, see [1], cf. (10)

$$\begin{vmatrix} 0 & b^2 & f^2 & a^2 \\ b^2 & 0 & c^2 & e^2 \\ f^2 & c^2 & 0 & d^2 \\ a^2 & e^2 & d^2 & 0 \end{vmatrix} = 0. \tag{18}$$

3) The conditions (15) and (16) are surprisingly also necessary and sufficient for a quadrilateral $ABCD$ being cyclic, see [13, 9].

3.2 Formula of Brahmagupta

Consider a cyclic quadrilateral $ABCD$ with the sides a, b, c, d. Find the formula of $ABCD$.

To discover the formula for the area of $ABCD$ we will use both coordinate and coordinate-free methods as mentioned above. The coordinate method is as follows.

Choose the Cartesian coordinate system so that $A = [r, 0], B = [x, y], C = [u, v], D = [w, z]$ and place the origin into the center of the circumcircle with the radius r, see Fig. 5. Then the following relations hold:

$|AB| = a \Leftrightarrow h_1 : (x - r)^2 + y^2 = a^2,$
$|BC| = b \Leftrightarrow h_2 : (x - u)^2 + (y - v)^2 = b^2,$
$|CD| = c \Leftrightarrow h_3 : (u - w)^2 + (v - z)^2 = c^2,$
$|DA| = d \Leftrightarrow h_4 : (r - w)^2 + z^2 = d^2,$
$|OB| = r \Leftrightarrow h_5 : x^2 + y^2 = r^2,$
$|OC| = r \Leftrightarrow h_6 : u^2 + v^2 = r^2,$
$|OD| = r \Leftrightarrow h_7 : w^2 + z^2 = r^2.$

By (3) for the area p of $ABCD$ we have

$p = \text{area of} ABCD \Leftrightarrow h_8 : p = 1/2(ry + xv - uy + uz - vw - rz).$

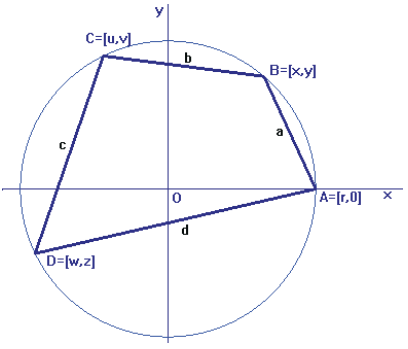


Fig. 5.

The elimination of variables x, y, u, v, w, z, r in the ideal $\langle h_1, h_2, \ldots, h_8 \rangle$ yields one polynomial, which gives after the factorization two equations. The first one

$$16p^2 = -(a^4+b^4+c^4+d^4)+2(a^2b^2+a^2c^2+a^2d^2+b^2c^2+b^2d^2+c^2d^2)+8abcd \quad (19)$$

gives the well-known formula of Brahmagupta (2). The second equation

$$16p'^2 = -(a^4+b^4+c^4+d^4)+2(a^2b^2+a^2c^2+a^2d^2+b^2c^2+b^2d^2+c^2d^2)-8abcd \quad (20)$$

expresses the signed area p' of a *non-convex* cyclic quadrilateral. We can arrive at it from the formula (19) writing for example $-c$ instead of c.

Remark: We should realize that all the derived formulas are *necessary* conditions for the area p. For example for the choice $a = 2, b = 1, c = 1, d = 1$ we get only one real solution for the convex quadrilateral (non-convex quadrilateral obviously does not exist for these values).

By the elimination of variables x, y, u, v, w, z in the ideal $\langle h_1, h_2, \ldots, h_7 \rangle$ we get the circumradii r and r' of a convex and non-convex cyclic quadrilateral $ABCD$

$$16p^2r^2 = (ab+cd)(ac+bd)(ad+bc),\ 16p'^2r'^2 = (ab-cd)(-ac+bd)(ad-bc). \quad (21)$$

Thus for the given lengths a, b, c, d (which fulfil the analog of the triangle inequality $a+b+c > d, \ldots$, etc.) there exist *at most* two classes of cyclic quadrilaterals with *different* radii and areas, see Fig. 4.

The second and more effective way to explore the area of a cyclic quadrilateral consists in the distance method. To express the area p of a quadrangle $ABCD$ we will use the Staudt's formula (9).

Consider both *convex* and *non-convex* cyclic quadrilaterals; then in accordance with Ptolemy's theorem relations $ef = ac + bd$ for convex and $ef = ac - bd$ or $ef = bd - ac$ for non-convex quadrilaterals hold. From (9) and Ptolemy's conditions above we immediately get, eliminating variables e, f, relations (19) and (20) for convex and non-convex cases, see Fig. 4.

```
Use R::=Q[abcdefp];
I:=Ideal(16p^2-4e^2f^2+(a^2-b^2+c^2-d^2)^2,(ac+bd-ef)(ac-bd-ef)
(ac-bd+ef));
Elim(e..f,I);
```

If we replace a^2, b^2, c^2, d^2 in both of the symmetric formulas (19) and (20) by elementary symmetric functions $k = a^2 + b^2 + c^2 + d^2, l = a^2b^2 + a^2c^2 + a^2d^2 + b^2c^2 + b^2d^2 + c^2d^2, m = a^2b^2c^2 + a^2b^2d^2 + a^2c^2d^2 + b^2c^2d^2, n = a^2b^2c^2d^2$, we get the following formula (22), which involves both Brahmagupta's relations (19) and (20)

$$(k^2 - 4l + q)^2 - 64n = 0. \tag{22}$$

Notice that if one of the sides a, b, c, d vanishes then $n = 0$ and (22) gives the formula (6) of Heron.

To calculate the circumradius r we can choose unlike the coordinate method the following way. A cyclic quadrilateral $ABCD$ is divided by the diagonal $e = |AC|$ into two triangles ABC, CDA with the sides a, b, e and c, d, e respectively, see Fig. 4. Both of the triangles have the same circumradius r as the quadrilateral $ABCD$. In accordance with the formula (8) for the circumradius of a triangle we eliminate the variable e from both equations, which leads to the formula for the circumradius of a quadrilateral

$$[s(k^2 - 4l) + m)]^2 - n(8s - k)^2 = 0, \tag{23}$$

where k, l, m, n are elementary symmetric functions and $s = r^2$. The following formula connects s and q

$$(qs - m)^2 - k^2n = 0. \tag{24}$$

Notice again that if $n = 0$ a quadrilateral becomes a triangle and we get the formulas (7) and (8).

4 Area of a Cyclic Pentagon

Now we are ready to investigate the area and circumradius of a cyclic pentagon with the sides a, b, c, d, e. We will proceed in a similar way as in the previous paragraphs. At first we will use the coordinate method to compute the area of a cyclic pentagon (Fig. 6). Choose the Cartesian coordinate system so that the coordinates of the vertices of a cyclic pentagon $ABCDE$ be $A = [0, 0], B = [a, 0], C = [x, y], D = [u, v], E = [w, z]$, and $a = |AB|, b = |BC|, c = |CD|, d = |DE|, e = |EA|$. Let the coordinates of the circumcenter O be $O = [s, t]$ with the radius r. Then the following relations hold:

$|BC| = b \Leftrightarrow h_1 : (x - a)^2 + y^2 - b^2 = 0,$
$|CD| = c \Leftrightarrow h_2 : (u - x)^2 + (v - y)^2 - c^2 = 0,$
$|DE| = d \Leftrightarrow h_3 : (w - u)^2 + (z - v)^2 - d^2 = 0,$
$|EA| = e \Leftrightarrow h_4 : w^2 + z^2 - e^2 = 0,$
$|OA| = r \Leftrightarrow h_5 : s^2 + t^2 - r^2 = 0,$

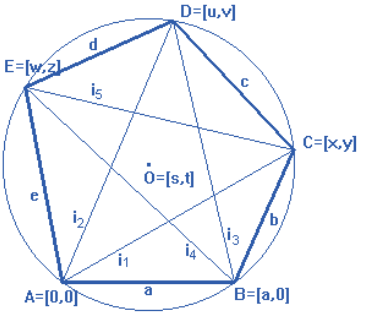


Fig. 6. Area of a cyclic pentagon — convex case

$|OB| = r \Leftrightarrow h_6 : (s-a)^2 + t^2 - r^2 = 0,$
$|OC| = r \Leftrightarrow h_7 : (x-s)^2 + (y-t)^2 - r^2 = 0,$
$|OD| = r \Leftrightarrow h_8 : (u-s)^2 + (v-t)^2 - r^2 = 0,$
$|OE| = r \Leftrightarrow h_9 : (w-s)^2 + (z-t)^2 - r^2 = 0.$

By (3) for the area p of $ABCDE$

$p =$ area of $ABCDE \Leftrightarrow h_{10} : p = 1/2(ay + xv - uy + uz - vw)$ holds.

We have 10 equations and we are to eliminate 9 variables $x, y, u, v, w, z, s, t, r$ in the ideal $\langle h_1, h_2, \ldots, h_{10}\rangle$. In 9 hours and 5 minutes in CoCoA we obtain the polynomial equation of degree 14 in p with 6672 terms. The substitution of elementary symmetric functions

$$k = \sum a^2,\ \ l = \sum a^2b^2,\ \ m = \sum a^2b^2c^2,\ \ n = \sum a^2b^2c^2d^2,\ \ o = a^2b^2c^2d^2e^2$$

with $q = 16p^2$ leads to the equation $h = 0$ which contains 153 terms and is still too long to write. It is as follows

$$h \equiv q^7 + q^6(7k^2 - 24l) + q^5(21k^4 - 144k^2l + 240l^2 + 16km - 192n) + \cdots + = 0.$$

If we denote $A = k^2 - 4l + q, B = kA + 8m, C = A^2 - 64n, D = 128o$ and eliminate k, l, m, n, o from h in $\langle h, A, B, C, D\rangle$ we get the formula in more compact form with only 5 terms. We arrived at

Theorem 1 (Robbins). *A cyclic pentagon with sides a, b, c, d, e and the area p is given. Let $q = 16p^2$ and A, B, C, D be as above. Then q satisfies the equation*

$$B^2C^2 + C^3q - 16B^3D - 18BCDq - 27D^2q^2 = 0. \tag{25}$$

The formula (25) can be considered as a generalization of the formulas of Heron and Brahmagupta.

Now we will derive the formula (25) using *coordinate-free* method. We come out from the Nagy–Rédey formula (4) for the area of a pentagon which is given by the lengths of sides a, b, c, d, e and diagonals i_1, i_2, i_3, i_4, i_5. In the position as in

Fig. 6 we apply Ptolemy's conditions on quadrilaterals $ABCD$, $BCDE$, $CDEA$, $DEAB$ and $EABC$

$$i_1i_3 = ac+bi_2,\ i_3i_5 = bd+ci_4, i_5i_2 = ce+di_1, i_2i_4 = da+ei_3, i_4i_1 = eb+ai_5. \quad (26)$$

The substitution (26) into the Nagy–Rédey formula (4) for a pentagon gives

$$16p^2 = 2\sum a^2b^2 - \sum a^4 + t, \quad (27)$$

where

$$t = 4(abci_2 + bcdi_4 + cdei_1 + deai_3 + eabi_5). \quad (28)$$

Denoting $k = \sum a^2 = a^2+b^2+\cdots$ and $l = \sum a^2b^2 = a^2b^2+a^2c^2+\cdots$ elementary symmetric functions of a^2, b^2, c^2, d^2, e^2 and $q = 16p^2$ we can write (27) in the form

$$k^2 - 4l + q = t. \quad (29)$$

Thus to express the area $16p^2$ of $ABCDE$ by means of a, b, c, d, e it suffices to express t by a, b, c, d, e because the other parts do not depend on i_1, i_2, i_3, i_4, i_5. The elimination of i_1, i_2, i_3, i_4, i_5 from t will need, besides Ptolemy's conditions (26), another relation holding between sides and diagonals in a cyclic pentagon. The conditions (26) are namely dependent, and only three of them are independent. The relations which we need to add are the same as in (15) and are as follows

$$i_1(ab + ci_2) = i_3(bc + ai_2), i_3(bc + di_4) = i_5(cd + bi_4), i_5(cd + ei_1) = i_2(ed + ci_1),$$
$$i_2(ed + ai_3) = i_4(ea + di_3),\ i_4(ea + bi_5) = i_1(ab + ei_5). \quad (30)$$

Elimination of i_1, i_2, i_3, i_4, i_5 from the system of polynomials (26), (30) and (28) is now feasible.

In 16m in CoCoA we obtain the polynomial $F_1(t, a, b, c, d, e)$ with 827 terms. The substitution of elementary symmetric functions k, l, m, n, o into F_1 gives the polynomial $F_2(t, k, l, m, n, o)$ with 37 terms. The substitution $A = t = k^2 - 4l + q, B = kt + 8m, C = t^2 - 64n, D = 128o$ into F_2 gives (25).

Up to now we explored the area of a convex cyclic pentagon. Let us consider a non-convex cyclic pentagon with the same lengths of sides a, b, c, d, e, for example that one in Fig. 7. Then the Ptolemy's conditions give

$$i_1i_3 = ac-bi_2,\ i_3i_5 = -bd+ci_4,\ i_5i_2 = ce+di_1,\ i_2i_4 = da+ei_3,\ i_4i_1 = -eb+ai_5$$

and by (16) the following relations

$$i_1(ab - ci_2) = i_3(bc - ai_2), i_3(bc - di_4) = i_5(-cd + bi_4), i_5(cd + ei_1) = i_2(de + ci_1),$$
$$i_2(de + ai_3) = i_4(ea + di_3),\ i_4(ea - bi_5) = i_1(-ab + ei_5)$$

hold. Instead of (28) we get

$$t = -abci_2 - bcdi_4 + cdei_1 + deai_3 - eabi_5. \quad (31)$$

Replacing b by $-b$ in the conditions above and (31) we get the same system of equations as in the convex case. Thus the elimination of i_1, i_2, i_3, i_4, i_5 from (31) gives the same result. Similarly we proceed in other cases.

Now we summarize the main steps of the previous method.

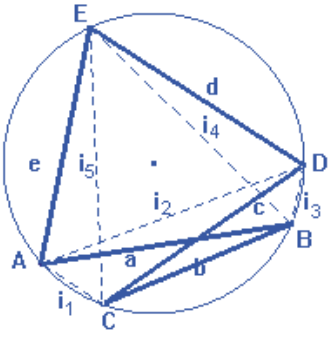


Fig. 7. Area of a cyclic pentagon — non-convex case

Cyclic Pentagon Area Algorithm:

1. Eliminate the lengths of diagonals i_1, i_2, i_3, i_4, i_5 from the system (28), (26), (30) to obtain a symmetric polynomial $F_1(t, a, b, c, d, e)$.

2. Express the polynomial $F_1(t, a, b, c, d, e)$ by the elementary symmetric functions k, l, m, n, o to get a polynomial $F_2(t, k, l, m, n, o)$.

3. Eliminate t, k, l, m, n, o in the ideal $\langle F_2, A, B, C, D\rangle$ to obtain the final polynomial $F_3(q, A, B, C, D)$.

Remarks:
1) The polynomial (25) is of degree 7 in $q = 16p^2$. This means that there exist *at most* 7 cyclic pentagons with the given lengths of sides a, b, c, d, e.

2) The terms A, B, C in (25) have geometric meanings. A is the left-hand side of (6), B fulfils the equation $8qs - B = 0$ which is equivalent to (24) and C is the left-hand side of the formula (22).

3) The formula (25) was published by D. P. Robbins in 1994, see [12]. His discovery is based on the fact that the left-hand side of (25) is the discriminant of the cubic polynomial $x^3 + 2Bx^2 - Cqx + 2Dq^2$. Why it is the case is still a mystery.

5 Radius of a Cyclic Pentagon

We will compute the circumradius r of a cyclic pentagon $ABCDE$ with the given sides a, b, c, d, e.

By the coordinate method, introducing the Cartesian coordinate system so that $A = [r, 0]$, $B = [x, y]$, $C = [u, v]$, $D = [w, z]$, $E = [s, t]$ and placing the origin O into the circumcenter we get in 11m 5s, eliminating 8 variables x, y, u, v, w, z, s, t, a polynomial $G_1(s, a, b, c, d, e)$ of degree 7 in $s = r^2$ with 2992 terms. The substitution of elementary symmetric polynomials

$$k = \sum a^2,\ l = \sum a^2b^2,\ m = \sum a^2b^2c^2,\ n = \sum a^2b^2c^2d^2,\ o = a^2b^2c^2d^2e^2$$

yields the polynomial $G_2(s, k, l, m, n, o)$ with 81 terms.

The distance method is as follows. Divide a cyclic pentagon $ABCDE$ into three triangles ABC, ACD and ADE, see Fig. 6. It is clear that all these triangles have the same circumcircle. By means of (7) we will eliminate common diagonals i_1, i_2.

```
Use R::=Q[abcdei[1..5]s];
I:=Ideal(s((a^2+b^2+i[1]^2)^2-4(a^2b^2+a^2i[1]^2+b^2i[1]^2))+a^2b^2
i[1]^2,s((c^2+i[1]^2+i[2]^2)^2-4(c^2i[1]^2+c^2i[2]^2+i[1]^2i[2]^2))
+c^2i[1]^2i[2]^2,s((d^2+e^2+i[2]^2)^2-4(d^2e^2+d^2i[2]^2+e^2i[2]^2))
+d^2e^2i[2]^2);
Elim(i[1]..i[2],I);
```

In 17s in CoCoA we receive (32). We can state

Theorem 2. *A pentagon with the sides a, b, c, d, e which is inscribed in the circle with the radius r is given. Let s, k, l, m, n, o are as above. Then*

$$s^3[(s(k^2-4l)+m)^2 - n(8s-k)^2]^2 + os^2Q + o^2sP + o^3 = 0 \qquad (32)$$

holds, where P, Q are polynomials in k, l, m, n.

To find the circumradius of a cyclic pentagon we can proceed in the following way, whose main steps are as follows.

Cyclic Pentagon Radius Algorithm:

1. Divide a cyclic pentagon $ABCDE$ into three triangles ABC, ACD, ADE and write the respective formulas (7) for ABC, ACD, ADE.

2. Eliminate i_1, i_2 from the system of equations from the first step to obtain the polynomial $G_1(s, a, b, c, d, e)$.

3. Eliminate a, b, c, d, e from the system of G_1 and elementary symmetric polynomials $k = \sum a^2, l = \sum a^2b^2, m = \sum a^2b^2c^2, n = \sum a^2b^2c^2d^2, o = a^2b^2c^2d^2e^2$ to obtain a polynomial $G_2(s, k, l, m, n, o)$.

Remarks:
1) There exist at most 7 cyclic pentagons with different radii.

2) If we put $o = 0$ we get a quadrilateral and the formula (32) becomes (23).

6 Final Remarks

1. Two given algorithms Cyclic Pentagon Area Algorithm and Cyclic Pentagon Radius Algorithm could serve as a tool for computing of the area and radius of a cyclic n-gons for $n \geq 6$.

2. By using this method two main problems occurred: (a) Still a big CPU time spent for computations; (b) Finding appropriate expressions like A, B, C, D to abbreviate the final polynomial.

3. Consider an arbitrary cyclic $(2n+1)$-gon with the given sides $a_1, a_2, \ldots, a_{2n+1}$. Then it is conjectured by Robbins [12] that there exist at most k_n such $(2n+1)$-gons with different areas and radii, where

$$k_n = \sum_{j=0}^{n-1} (n-j) \binom{2n+1}{j}, \tag{33}$$

i.e., $k_1, k_2, k_3, \ldots = 1, 7, 38, 187, 874, \ldots$. For $(2n+2)$-gons with sides $a_1, a_2, \ldots, a_{2n+2}$ there are at most $2k_n$ such polygons with different areas and radii. This conjecture is still open.

4. The formula which connects the area and radius of a cyclic pentagon of the type (8), (24) is still missing.

References

1. Berger, M.: Geometry I. Springer-Verlag, Berlin Heidelberg, 1987.
2. Blaschke, W.: Kreis und Kugel. Walter de Gruyter & Co, Berlin, 1956.
3. Chou, S.-C.: Mechanical Geometry Theorem Proving. D. Reidel Publishing Company, Dordrecht, 1987.
4. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms. Second Edition, Springer-Verlag, New York, 1997.
5. Coxeter, H. S. M., Greitzer, S. L.: Geometry Revisited. Toronto New York, 1967.
6. Dörrie, B. H.: Triumph der Mathematik. Breslau, 1933.
7. Kowalewski, G.: Einfürung in die Determinantentheorie. Veit & Comp., Leipzig, 1909.
8. Pak, I.: The Area of Cyclic Polygons: Recent Progress on Robbins' Conjectures. Mini Survey, 5 pages to appear in Adv. Applied Math. (special issue in memory of David Robbins).
9. Rashid, M. A., Ajibade, A. O.: Two Conditions for a Quadrilateral to be Cyclic Expressed in Terms of the Lengths of Its Sides. Int. J. Math. Educ. Sci. Techn. **34**, No. 5, 2003, 739–742.
10. Recio, T., Sterk, H., Vélez, M. P.: Project 1. Automatic Geometry Theorem Proving. In: Some Tapas of Computer Algebra (A. Cohen, H. Cuipers, H. Sterk, eds.), Algorithms and Computations in Mathematics, Vol. 4, Springer-Verlag, Berlin Heidelberg, 1998, 276–296.
11. Rédey, L., Nagy, B. Sz.: Eine Verallgemeinerung der Inhaltsformel von Heron. Publ. Math. Debrecen **1**, 1949, 42–50.
12. Robbins, D. P.: Areas of Polygons Inscribed in a Circle. Discrete Comput. Geom. **12**, 1994, 223–236.
13. Sadov, S.: Sadov's Cubic Analog of Ptolemy's Theorem. http://www.math.rutgers.edu/~zeilberg/mamarim/mamarinhtml/sad.
14. Schreiber, P.: On the Existence and Constructibility of Inscribed Polygons. Beiträge zur Algebra und Geometrie **34**, 1993, 195–199.
15. Staudt, Ch. R.: Über die Inhalte der Polygone und Polyeder. Journal für die reine und angewandte Mathematik **24**, 1842, 252–256.
16. Svrtan, D., Veljan, D., Volenec, V.: Geometry of Pentagons: From Gauss to Robbins. http://arxiv.org/abs/math/0403503
17. Wang, D.: Gröbner Bases Applied to Geometric Theorem Proving and Discovering. In: Gröbner Bases and Applications (B. Buchberger, F. Winkler, eds.), Cambridge University Press, Cambridge, 1998, 281–301.

Symbolic Solution of a Piano Movers' Problem with Four Parameters*

Lu Yang[1,2] and Zhenbing Zeng[1]

[1] East China Normal University, Shanghai 200062, China
[2] Guangzhou University, Guangzhou 510405, China
lyang@sei.ecnu.edu.cn, zbzeng@sei.ecnu.edu.cn

Abstract. This paper presents a symbolic solution to a piano movers' problem with four parameters through investigating the positive definiteness of an even polynomial of degree 8 and the feasibility of certain inequality systems.

1 Introduction

Given an open subset U in n-dimensional space and two compact subsets C_0 and C_1 of U, where C_1 is derived from C_0 by a continuous motion, is it possible to move C_0 to C_1 while remaining entirely inside U? This is the general form of the piano movers' problem. It is a basic problem in robot motion planing.

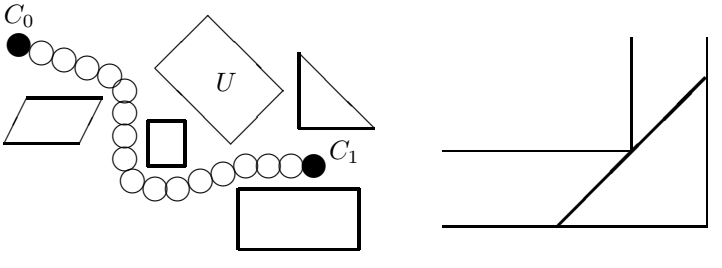


Fig. 1. Robot motion problem and moving ladder problem

The Moving Ladder Problem is a special case of the piano movers' problem. It asks what is the longest ladder that can be moved around a right-angled hallway of unit width. In [2] J. H. Davenport shows that for a straight, rigid ladder, the answer is $2\sqrt{2}$, which allows the ladder to just pivot around the corner at an angle. For a smoothly-shaped ladder, the largest diameter is $\geq 2(1+\sqrt{2})$ (see [4]).

There have been many different approaches to the piano movers' problem in the past (see, e.g., [5, 7, 9, 10, 3, 8]). Most worked to find real-time and numeric solutions. In this paper, we discuss the following piano movers' problem with

* This work is supported in part by NKBRSF-2004CB318003 and NNSFC-10471044.

H. Hong and D. Wang (Eds.): ADG 2004, LNAI 3763, pp. 59–69, 2006.

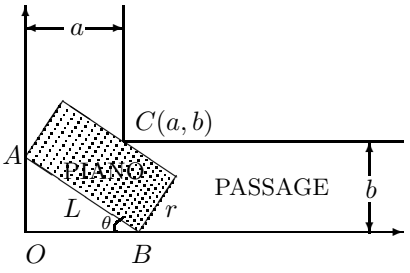


Fig. 2. The piano movers' problem

four parameters: Given a 2D rectangle of length L and width r, and an L-shape passage with a vertical passage of width a and a horizontal passage of width b, as shown in Fig. 2, with $L > a \geq b > r$, under what condition there is a continuous collision-free path to move the rectangle through the L-shape passage?

We will construct a symbolic solution in the simplest form. By "symbolic solution" we mean a system of inequalities in parameters L, r, a, b that performs as a sufficient and necessary condition for existence of a continuous collision-free path moving the piano through the passage. Our result is the following theorem.

Theorem 1. *Let L, r, a, b be positive real numbers such that $L > a \geq b > r$. Then there is a continuous collision-free path for moving a rectangle of $L \times r$ through the L-shape passage with widths a and b if and only if the following inequality holds:*

$$\begin{aligned} p_5 = &-a^6 + 3a^4L^2 - 3a^4b^2 + a^4r^2 + 18a^3bLr - 20a^2L^2r^2 + 2a^2b^2r^2 - 3a^2L^4 \\ &-3a^2b^4 - 21a^2b^2L^2 + 36abL^3r - 16abLr^3 + 18ab^3Lr + 16L^2r^4 \\ &-8L^4r^2 + b^4r^2 + L^6 - b^6 - 20b^2L^2r^2 - 3b^2L^4 + 3b^4L^2 < 0. \end{aligned}$$

The paper is organized as follows: In Section 2, we show that the piano movers' problem is equivalent to finding a sufficient and necessary condition in terms of L, r, a, b such that the following polynomial of degree 8 in x is positive definite:

$$\begin{aligned} g(x) = &4(a-r)x^8 - 4(L-2a-b+2r)x^6 - 2(3L-3a-3b+4r)x^4 \\ &-2(L-a-2b+2r)x^2 + b - r > 0. \end{aligned}$$

In Section 3, we give a quantifier-free solution to the positivity of $g(x)$, in terms of the list of signs of polynomial p_5 mentioned in Theorem 1 and other four polynomials p_1, p_2, p_3, p_4 in variables L, r, a, b. In Section 4 we reduce the obtained sign lists through analyzing on the satisfiability of certain smaller systems of inequalities, and give a proof to Theorem 1.

2 A Simple Model of the Piano Movers' Problem

We choose a Cartesian coordinate system, as shown in Fig. 2, by taking the left-bottom corner O of the L-shape passage as the origin, the related sides of the

horizontal and vertical passages as x and y axes, respectively. Then the another corner C of the passage is (a, b). Let C_r be the circle $(x-a)^2+(y-b)^2 = r^2$. For any point $P(x, y)$ on the circle C_r, draw the tangent line of C_r through P. Let A, B be the intersection points of the tangent line with y and x axes, respectively. If $P(x, y)$ satisfies $x < a, y < b$, then A, B are on the positive parts of y and x axes. It is obvious that a rectangle of length L and width r can be moved through the passage if and only if the length of the segment AB constructed through any point $P(x, y)$ on C_r satisfying $x < a, y < b$ is greater than L.

Suppose that A, B are points on the positive parts of y and x axes. Let $AB = \rho$ and $\angle OBA = \theta$ $(0 < \theta < \pi/2)$. Then the tangent line is

$$\frac{x}{\rho\cos\theta} + \frac{y}{\rho\sin\theta} = 1,$$

and the distance from point $C(a, b)$ to this tangent line is

$$r = \frac{|\frac{a}{\rho\cos\theta} + \frac{b}{\rho\sin\theta} - 1|}{\sqrt{\frac{1}{(\rho\cos\theta)^2} + \frac{1}{(\rho\sin\theta)^2}}}.$$

Since point $C(a, b)$ and the origin O are in the different sides of the tangent line, we have

$$\frac{a}{\rho\cos\theta} + \frac{b}{\rho\sin\theta} > 1,$$

and therefore,

$$r = \frac{\frac{a}{\rho\cos\theta} + \frac{b}{\rho\sin\theta} - 1}{\sqrt{\frac{1}{(\rho\cos\theta)^2} + \frac{1}{(\rho\sin\theta)^2}}} = a\sin\theta + b\cos\theta - \rho\sin\theta\cos\theta.$$

So the relation of ρ and θ can be expressed in the following equation

$$\rho = \frac{a\sin\theta + b\cos\theta - r}{\sin\theta\cos\theta}.$$

This proves the following lemma.

Lemma 1. *A rectangle of length L and width r can be moved through the L-shape passage with a vertical passage of width a and a horizontal passage of width b if and only if the following inequality*

$$\frac{a\sin\theta + b\cos\theta - r}{\sin\theta\cos\theta} - L > 0$$

holds for all θ $(0 < \theta < \pi/2)$.

By substituting

$$\cos\theta = \frac{1-t^2}{1+t^2}, \quad \sin\theta = \frac{2t}{1+t^2}$$

into Lemma 1 we can transform the sufficient and necessary condition into the following quantifier formula

$$\forall t(0 < t < 1 \Longrightarrow 2at(1-t^2) + b(1-t^4) - r(1+t^2)^2 - 2Lt(1-t^2) + 4at^3 > 0).$$

Using variable substitution $t = x^2/(1+x^2)$ this formula can be simplified to the following one:

$$\forall x \ (g(L, r, a, b, x) > 0),$$

where

$$\begin{aligned} g(L, r, a, b, x) = {} & 4(a-r)x^8 - 4(L-2a-b+2r)x^6 - 2(3L-3a-3b+4r)x^4 \\ & -2(L-a-2b+2r)x^2 + b - r. \end{aligned}$$

This leads to the following result.

Lemma 2. *A rectangle of length L and width r can be moved through the L-shape passage with a vertical passage of width a and a horizontal passage of width b if and only if the polynomial $G(x) := g(L, r, a, b, x)$ is positive definite.*

According to the continuous dependence of polynomials upon their coefficients, we know that if $G(x) = g(L, r, a, b, x)$ is positive definite, then there exists a neighborhood $U \subset R^4$ of (L, r, a, b) such that $g(L', r', a', b', x)$ is also positive definite for any parameters $L', r', a', b' \in U$. This meets the requirement of collision-free path.

3 Quantifier Elimination Using Discriminants

Regarding $G(x)$ as a univariate polynomial in x with parameters L, r, a, b, our goal is to establish the necessary and sufficient condition in terms of L, r, a, b such that $G(x)$ is positive for all $x \in \mathbf{R}$. It is clear that any polynomial

$$F(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n$$

is positive definite if and only if $F(x)$ has no real root, with $F(0) = a_n > 0$.

Quantifier elimination methods ([1]) provide a standard tool to build such kind of conditions. D. Wang [11] used the package `QEPCAD` based on quantifier elimination and cylindrical algebraic decomposition to solve Davenport's moving ladder problem in 1990. For the piano movers' problem with four parameters, one of the referees pointed out that the same result as in Theorem 1 can also be obtained by using H. Hong's improved package `QEPCAD B` (version 1.44) [6] with appropriate memory management in Saclib. In this paper we shall use an explicit criterion for root-classification of a polynomial with symbolic coefficients, given in [12], to express the condition into a system of polynomial inequalities in variables $a_0, a_1, \ldots, a_{n-1}, a_n$. While the problem can also be solved by a program in an automatic mode, we take here a much more "readable" argument to explain every step in detail. In order to state the explicit criterion the following definitions and notations are needed.

Definition 1 (Discrimination Matrix). *Given a polynomial*

$$F(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n$$

with general symbolic coefficients, the following $2n \times 2n$ *matrix*

$$\begin{bmatrix}
a_0 & a_1 & a_2 & \cdots & a_n & & & \\
0 & na_0 & (n-1)a_1 & \cdots & a_{n-1} & & & \\
 & a_0 & a_1 & \cdots & a_{n-1} & a_n & & \\
 & 0 & na_0 & \cdots & 2\,a_{n-2} & a_{n-1} & & \\
 & & & \cdots & \cdots & & & \\
 & & & \cdots & \cdots & & & \\
 & & & & a_0 & a_1 & \cdots & a_n \\
 & & & & 0 & na_0 & \cdots & a_{n-1}
\end{bmatrix}$$

is called the **discrimination matrix** *of* $F(x)$*, and denoted by* $\mathrm{Discr}(F)$*.*

Definition 2 (Discriminant Sequence). *Let* $F(x)$ *be a polynomial of degree* n *and* $\mathrm{Discr}(F)$ *its discrimination matrix. By* d_k *or* $d_k(F)$ *denote the determinant of the submatrix of* $\mathrm{Discr}(F)$*, formed by the first* k *rows and the first* k *columns, for* $k = 1, \ldots, 2n$*. Let* $D_k = d_{2k}$ *for* $k = 1, \ldots, n$*. We call the* n*-tuple*

$$\{D_1,\ D_2,\ \ldots,\ D_n\}$$

the **discriminant sequence** *of the polynomial* $F(x)$*.*

Definition 3 (Sign List and Revised Sign List). *Let* $\mathrm{sgn}(\cdot)$ *be the sign function and* $r_1, r_2, \ldots, r_n$ *be any sequence of real numbers with* $r_1 \neq 0$*. Let* $s_i = \mathrm{sgn}(r_i)$*,* $i = 1, 2, \ldots, n$*. Call* $[s_1, s_2, \ldots, s_n]$ *the* **sign list** *of* $r_1, r_2, \ldots, r_n$*. Let* $[\varepsilon_1,\ \varepsilon_2,\ \ldots,\ \varepsilon_n]$ *be a list constructed according to the following rules:*

1. *If* $s_i \neq 0$*, then* $\varepsilon_i = s_i$*;*
2. *If* $s_i \neq 0, s_{i+1} = \cdots = s_{i+j-1} = 0, s_{i+j} \neq 0$*, then* $\varepsilon_{i+r} = (-1)^{(r+1)/2} \cdot s_i$ *for* $r = 1, \ldots, j-1$*, that is, replace the sub-sequence* $s_{j+1}, \ldots, s_{i+j-1}$ *by* $-s_i, -s_i, s_i, s_i, -s_i, -s_i, s_i, s_i, -s_i, \ldots$*, with keeping the number of terms;*
3. *If* $s_i \neq 0, s_{i+1} = 0, \ldots, s_n = 0$*, then* $\varepsilon_k = s_k$ *for* $k = i, i+1, \ldots, n$*.*

Call $[\varepsilon_1,\ \varepsilon_2,\ \ldots,\ \varepsilon_n]$ *the* **revised sign list** *of* $r_1, r_2, \ldots, r_n$*.*

The following result [12] can be used to decide the number of the distinct real or imaginary roots of a univariate polynomial with symbolic coefficients. It is a generalization of the classical approach of Riquier and Sturm for polynomials with real coefficients.

Theorem 2. *Given a polynomial*

$$F(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_n$$

with real coefficients, if the number of the sign changes of the revised sign list of the discriminant sequence $\{D_1, D_2, \ldots, D_n\}$ *of* $F(x)$ *is* ν*, then the number of*

the pairs of distinct conjugate imaginary roots of $F(x)$ equals ν. Furthermore, if the number of non-vanishing terms of the revised sign list is l, then so is the number of the distinct roots of $F(x)$, and hence the number of the distinct real roots of $F(x)$ equals $l - 2\,\nu$.

In particular, for the polynomial $G(x) = g(L, r, a, b, x)$ related to the piano movers' problem, the corresponding discriminant sequence $\{D_1, D_2, \ldots, D_8\}$ can be written in the following form:

$$\begin{aligned}
&D_1 = 128(a-r)^2, && D_2 = 4096(a-r)^3 \cdot p_1,\\
&D_3 = -32768(a-r)^3 \cdot p_0 \cdot p_1, && D_4 = -524288(a-r)^3 \cdot p_0 \cdot p_2,\\
&D_5 = -8388608(a-r)^3 \cdot p_2 \cdot p_3, && D_6 = 67108864(a-r)^3 \cdot p_3 \cdot p_4,\\
&D_7 = -268435456(a-r)^3 \cdot p_4 \cdot p_5, && D_8 = 1073741824(a-r)^3(b-r) \cdot p_5^2,
\end{aligned}$$

where p_5 is as in Theorem 1, and $p_0, p_1, \ldots, p_4$ are as follows

$$\begin{aligned}
p_0 &= -4\,a\,r - 3\,L^2 - 3\,b^2 + 6\,b\,L + 4\,r^2,\\
p_1 &= -2\,a - b + L + 2\,r,\\
p_2 &= 3\,a^2L + 3\,a^2b + 2\,a^2r - 11\,aLr - abr - 6\,ar^2 + 3\,ab^2 - 3\,abL + 4\,r^3\\
&\quad - 2\,br^2 + 8\,Lr^2 - 9\,b^2L + 5\,bLr - L^2r - 4\,b^2r - 3\,L^3 + 9\,bL^2 + 3\,b^3,\\
p_3 &= 2\,a^3r + 3\,a^2b^2 + 12\,a^2bL + 2\,a^2r^2 + 3\,a^2L^2 - 18\,abLr - 16\,aL^2r\\
&\quad - 2\,ab^2r + 3\,b^4 - 6\,b^3L - 2\,b^2r^2 + 8\,bLr^2 + 6\,bL^3 + 12\,L^2r^2 - 3\,L^4,\\
p_4 &= 6\,b^3ra - 13\,L^2ab^2 + 12\,Lab^3 + 4\,r^3Lb - 8\,r^3aL - 4\,r^3ab + 5\,L^2a^2r\\
&\quad - 5\,L^2a^2b - 12\,L^2r^2b - 36\,L^2r^2a - 17\,b^2a^2L + 5\,b^2a^2r - 21\,b^2r^2L\\
&\quad + 19\,L^3rb + 6\,L^3ab - 6\,a^3Lb + 6\,ra^3L + 6\,ra^3b - 5\,r^2a^2L - b^3rL\\
&\quad + b^2r^2a - L^4a - 14\,L^2rb^2 - 6\,L^3ra + 8\,r^4L - 4\,b^4a - 3\,r^2a^3 + 4\,b^3r^2\\
&\quad - 2\,b^2r^3 - 5\,b^2a^3 - 7\,b^3a^2 - a^4L + 2\,L^2a^3 + 2\,L^3a^2 + 2\,r^3a^2 + 2\,L^3r^2\\
&\quad - 2a^4b + 2ra^4 + 28L^2r^3 - L^5 - 5b^5 - 2b^3L^2 + 11b^4L + 3b^4r - 10L^3b^2\\
&\quad + 7\,L^4b - 7\,L^4r - a^5 + 24\,L^2bra + 38\,ra^2Lb - 34\,r^2Lab + 30\,Lb^2ra.
\end{aligned}$$

Observed that from $L > a \geq b > r$, we have $a - r > 0, b - r > 0$ and

$$p_0 = -4\,r(a-r) - 3(L-b)^2 < 0.$$

Thus, the revised sign list of the discriminant $\{D_1, D_2, \ldots, D_8\}$ is equal to that of the following polynomial sequence

$$DS = \{1, p_1, p_1, p_2, -p_2p_3, p_3p_4, -p_4p_5, p_5^2\}.$$

By $V(r_1, r_2, \ldots, r_n)$ denote the sign changes of the revised sign list of $r_1, r_2, \ldots, r_n$, a given sequence of real numbers. Then $V(D_1, D_2, \ldots, D_8) = V(DS)$.

Let $PS = \{p_1, p_2, p_3, p_4, p_5\}$. Then each sign list of PS decides a sign list of DS. Using Theorem 2 we can construct the following solution to the quantifier elimination problem $\forall x\,(G(x) > 0)$.

Theorem 3. *Let $G(x) = g(L, r, a, b, x)$ and $p_1, p_2, \ldots, p_5$ be the polynomials in PS. Then $G(x) > 0$ for all $x \in R$ if and only if the sign list $[s_1, s_2, s_3]$ of p_1, p_2, p_3 and the sign list $[s_4, s_5]$ of p_4, p_5 belong to one of the 116 cases shown in the Table 1.*

Table 1. The sign lists of p_1, p_2, p_3 andp_4, p_5 that satisfy $\forall x\,(G(x) > 0)$ (the meaning of the subscript of the sign lists will be explained later)

$[s_1, s_2, s_3]$	$[s_4, s_5]$
[-1,-1,-1]	[-1,-1]
[-1,-1, 0]	[-1,-1],[0,-1]$_{12}$,[0,1]$_8$, [1,1]$_8$
[-1,-1, 1]	[-1,-1],[-1,0]$_9$,[-1,1]$_7$,[0,-1]$_{12}$,[0,1]$_7$,[1,1]$_7$
[-1,0,-1]$_5$	[-1,-1],[0,-1]$_{12}$,[0,1], [1,-1]$_{11}$,[1,0]$_{10}$,[1,1]
[-1,0, 0]	[-1,-1],[-1,1]$_8$,[0,-1]$_{12}$,[0,1]$_8$,[1,-1]$_{11}$,[1,1]$_8$
[-1,0, 1]	[-1,-1],[-1,0]$_9$,[-1,1]$_7$,[0,-1]$_{12}$,[0,1]$_7$,[1,1]$_7$
[-1,1,-1]$_4$	[-1,-1],[0,-1]$_{12}$,[0,1], [1,-1]$_{11}$,[1,0]$_{10}$,[1,1]
[-1,1, 0]$_6$	[-1,-1],[-1,0],[-1,1]$_8$,[0,-1]$_{12}$,[0,0],[0,1]$_8$, [1,-1]$_{11}$, [1,0]$_{10}$, [1,1]$_8$
[-1,1, 1]	[-1,-1],[-1,0]$_9$,[-1,1]$_7$,[0,-1]$_{12}$,[0,1]$_7$,[1,-1]$_{11}$
[0,-1,-1]	[-1,-1]
[0,-1, 0]	[-1,-1],[0,-1]$_{12}$,[0,1]$_8$, [1,1]$_8$
[0,-1, 1]	[-1,-1],[-1,0]$_9$,[-1,1]$_7$,[0,-1]$_{12}$,[0,1]$_7$,[1,1]$_7$
[0,0,-1]$_5$	[-1,-1],[0,-1]$_{12}$,[0,1], [1,-1]$_{11}$,[1,0]$_{10}$,[1,1]
[0,0, 0]	[-1,-1],[-1,1]$_8$,[0,-1]$_{12}$,[0,1]$_8$,[1,-1]$_{11}$,[1,1]$_8$
[0,0, 1]	[-1,-1],[-1,0]$_9$,[-1,1]$_7$,[0,-1]$_{12}$,[0,1]$_7$,[1,1]$_7$
[0,1,-1]$_{2,4}$	[-1,-1],[0,-1]$_{12}$,[0,1], [1,-1]$_{11}$,[1,0]$_{10}$,[1,1]
[0,1, 0]$_{2,6}$	[-1,-1],[-1,0],[-1,1]$_8$,[0,-1]$_{12}$,[0,0],[0,1]$_8$, [1,-1]$_{11}$, [1,0]$_{10}$, [1,1]$_8$
[0,1, 1]$_2$	[-1,-1],[-1,0]$_9$,[-1,1]$_7$,[0,-1]$_{12}$,[0,1]$_7$,[1,-1]$_{11}$
[1,-1,-1]	[-1,-1]
[1,-1, 0]	[-1,-1],[0,-1]$_{12}$,[0,1]$_8$, [1,1]$_8$
[1,-1, 1]	[-1,-1],[-1,0]$_9$,[-1,1]$_7$,[0,-1]$_{12}$,[0,1]$_7$,[1,1]$_7$
[1,0,-1]$_{3,5}$	[-1,-1]
[1,0, 0]$_3$	[-1,-1],[1,1]$_8$
[1,0, 1]$_3$	[1,1]
[1,1,1]$_1$	[1,1]

Proof. Since $G(x)$ is an even polynomial and $G(0) = b - r > 0$, the discriminant sequence can be classified into the following 4 cases according to Theorem 2.

C1. $D_8 \neq 0$: in this case, $G(x)$ has no real roots if and only if $V(D_1, \ldots, D_8) = 4$.
C2. $D_6 \neq 0$, $D_7 = D_8 = 0$: in this case, $G(x)$ has no real roots if and only if $V(D_1, \ldots, D_8) = 3$.
C3. $D_4 \neq 0$, $D_5 = \cdots = D_8 = 0$: in this case, $G(x)$ has no real roots if and only if $V(D_1, \ldots, D_8) = 2$.

C4. $D_2 \neq 0$, $D_3 = \cdots = D_8 = 0$: in this case, $G(x)$ has no real roots if and only if $V(D_1, \ldots, D_8) = 1$.

Let

$$\Sigma^k = \{[s_1, \ldots, s_k] \mid s_1, \ldots, s_k \in \{-1, 0, 1\}\}, \ \ k \in N,$$

and

$$\mu: \quad \Sigma^5 \to \Sigma^8, \quad [s_1, s_2, s_3, s_4, s_5] \mapsto [\,1, s_1, s_1, s_2, -s_2 s_3, s_3 s_4, -s_4 s_5, s_5^2\,].$$

Then for any sign list $\sigma \in \Sigma^5$, a specification of L, r, a, b with $0 < r < b \leq a < L$ and

$$[\,\mathrm{sgn}(p_1(L, r, a, b)), \ldots, \mathrm{sgn}(p_5(L, r, a, b))\,] = \sigma$$

satisfies $\forall\, x\,(\,G(x) = g(x, L, r, a, b) > 0)$ if and only if the corresponding sign list $\mu(\sigma) = [m_1, m_2, \ldots, m_8]$ satisfies one of the following 4 cases:

$$\begin{aligned}
&(C_1): \quad m_8 \neq 0,\ V(m_1, m_2, \ldots, m_8) = 4;\\
&(C_2): \quad m_6 \neq 0,\ m_7 = m_8 = 0,\ V(m_1, m_2, \ldots, m_8) = 3;\\
&(C_3): \quad m_4 \neq 0,\ m_5 = \cdots = m_8 = 0,\ V(m_1, m_2, \ldots, m_8) = 2;\\
&(C_4): \quad m_2 \neq 0,\ m_3 = \cdots = m_8 = 0,\ V(m_1, m_2, \ldots, m_8) = 1.
\end{aligned}$$

It is easy but tedious to check that among all the $3^5 = 243$ lists contained in Σ^5, there are 116 cases that satisfy condition (C_1) to (C_4), as listed in Table 1.

4 Simplifying the Quantifier-Free Formula

For any sign list $\sigma = [s_1, s_2, \ldots, s_5] \in \Sigma^5$, we define a system of inequalities $Q_1, Q_2, \ldots, Q_5$ as follows:

$$Q_i = \begin{cases} p_i < 0, & \text{if } s_i = -1,\\ p_i = 0, & \text{if } s_i = 0,\\ p_i > 0, & \text{if } s_i = 1, \end{cases}$$

where $i = 1, \ldots, 5$. A list σ is said to be *feasible* if the corresponding system $Q_1, Q_2, \ldots, Q_5$ is satisfiable, i.e., there exist L, r, a, b such that $0 < r < b \leq a < L$ and

$$[\,\mathrm{sgn}(p_1(L, r, a, b)), \ldots, \mathrm{sgn}(p_5(L, r, a, b))\,] = \sigma.$$

In this section, we show that there are more than 100 infeasible cases in Table 1 so that the necessary and sufficient condition for the piano movers' problem can be significantly simplified. The key is the following Lemma 3.

Lemma 3. *Let $p_1, p_2, \ldots, p_5$ be the polynomials in PS. Then the following relations hold:*

$$\begin{aligned}
&(1) \quad p_1 \geq 0 \Longrightarrow p_2 \leq 0,\\
&(2) \quad p_2 \geq 0 \Longrightarrow p_1 \leq 0,
\end{aligned}$$

$$
\begin{aligned}
&(3) \quad p_2 \geq 0 \Longrightarrow p_3 \geq 0,\\
&(4) \quad p_3 \leq 0 \Longrightarrow p_2 \leq 0,\\
&(5) \quad p_3 \geq 0 \Longrightarrow p_5 \leq 0,\\
&(6) \quad p_5 \geq 0 \Longrightarrow p_3 \leq 0,\\
&(7) \quad p_5 \leq 0 \Longrightarrow p_4 \leq 0,\\
&(8) \quad p_4 \geq 0 \Longrightarrow p_5 \geq 0.
\end{aligned}
$$

This lemma can be proved by a simplified cylindrical algebraic decomposition described in [13]. Actually, we used a Maple program `BOTTEMA` (see [13]) for establishing the results in Lemma 3. `BOTTEMA` can prove automatically an extensive class of inequalities whereof the conclusions are of type $\geq$ or $\leq$.

Although we can establish more implication relations by further investigation, it is enough now to give a proof to Theorem 1.

Proof (of Theorem 1). In view of Lemma 3, any sign list $[s_1, \ldots, s_5]$ which falls into one of the following 12 types is not feasible:

(T_1) : `[1,1,u,v,w]`, (T_2) : `[0,1,u,v,w]`, (T_3) : `[1,0,u,v,w]`,
(T_4) : `[u,1,-1,v,w]`, (T_5) : `[u,0,-1,v,w]`, (T_6) : `[u,1,0,v,w]`,
(T_7) : `[u,v,1,w,1]`, (T_8) : `[u,v,0,w,1]`, (T_9) : `[u,v,1,w,0]`,
(T_{10}) : `[u,v,w,1,0]`, (T_{11}) : `[u,v,w,1,-1]`, (T_{12}) : `[u,v,w,0,-1]`,

where u,v,w $\in$ {-1, 0, 1}. We have marked these types with subscript numbers in Table 1. It is easy to see that all possibly feasible sign lists in Table 1 are the followings ones (grouped into three categories):

(G_1) : `[-1,1,1,-1,-1];`
(G_2) : `[-1,0,0,-1,-1], [0,0,0,-1,-1],`
`[-1,0,1,-1,-1], [0,0,1,-1,-1];`
(G_3) : `[-1,-1,-1,-1,-1], [-1,-1,0,-1,-1], [-1,-1,1,-1,-1],`
`[0,-1,-1,-1,-1], [0,-1,0,-1,-1], [0,-1,1,-1,-1],`
`[1,-1,-1,-1,-1], [1,-1,0,-1,-1], [1,-1,1,-1,-1].`

According to the map from sign list to inequality system, the sign list in (G_1) represents the following inequality system:

$$p_1 < 0,\ p_2 > 0,\ p_3 > 0,\ p_4 < 0,\ p_5 < 0.$$

The four inequality systems decided by sign lists in (G_2) can be simplified to

$$p_1 \leq 0,\ p_2 = 0,\ p_3 \geq 0,\ p_4 < 0,\ p_5 < 0$$

in the following scheme:

$$
\left.
\begin{array}{l}
\left.\begin{array}{l} p_1 < 0,\ p_3 > 0,\ I_1 \\ p_1 = 0,\ p_3 > 0,\ I_1 \end{array}\right\} \Longrightarrow p_1 \leq 0,\ p_3 > 0,\ I_1 \\[2ex]
\left.\begin{array}{l} p_1 < 0,\ p_3 = 0,\ I_1 \\ p_1 = 0,\ p_3 = 0,\ I_1 \end{array}\right\} \Longrightarrow p_1 \leq 0,\ p_3 = 0,\ I_1
\end{array}
\right\} \Longrightarrow p_1 \leq 0,\ p_3 \geq 0,\ I_1,
$$

where I_1 stands for $p_2 = 0, p_4 < 0, p_5 < 0$ for short. In a similar way, the inequality systems related to the 9 sign lists in (G_3) can be simplified to

$$p_2 < 0,\ p_4 < 0,\ p_5 < 0.$$

The following is the simplification procedure:

$$\left.\begin{array}{l} \left.\begin{array}{l} p_1 < 0,\ p_3 < 0,\ I_2 \\ p_1 < 0,\ p_3 = 0,\ I_2 \\ p_1 < 0,\ p_3 > 0,\ I_2 \end{array}\right\} \Longrightarrow p_1 < 0,\ I_2 \\ \left.\begin{array}{l} p_1 = 0,\ p_3 < 0,\ I_2 \\ p_1 = 0,\ p_3 = 0,\ I_2 \\ p_1 = 0,\ p_3 > 0,\ I_2 \end{array}\right\} \Longrightarrow p_1 = 0,\ I_2 \\ \left.\begin{array}{l} p_1 > 0,\ p_3 < 0,\ I_2 \\ p_1 > 0,\ p_3 = 0,\ I_2 \\ p_1 > 0,\ p_3 > 0,\ I_2 \end{array}\right\} \Longrightarrow p_1 > 0,\ I_2 \end{array}\right\} \Longrightarrow I_2,$$

where I_2 stands for $p_2 < 0,\ p_4 < 0,\ p_5 < 0$.

We have reduced the inequalities corresponding to the possibly feasible sign lists in Table 1 to the following three:

$$\begin{aligned} &(1): \quad p_1 < 0,\ p_2 > 0,\ p_3 > 0,\ p_4 < 0,\ p_5 < 0; \\ &(2): \quad p_1 \le 0,\ p_2 = 0,\ p_3 \ge 0,\ p_4 < 0,\ p_5 < 0; \\ &(3): \quad p_2 < 0,\ p_4 < 0,\ p_5 < 0. \end{aligned}$$

This gives a necessary and sufficient condition for $G(x) = g(L, r, a, b, x)$ to be positive definite under the assumption $0 < r < b \le a < L$. Thus, if $0 < r < b \le a < L$ and $G(x)$ is positive definite, then $p_5 < 0$ is always true.

The following deduction diagram shows that if $L > a \ge b > r > 0$ and $p_5 < 0$, then one of the three systems of inequalities above must hold, and therefore $G(x)$ is positive definite.

$$\begin{array}{l} p_5 < 0 \\ p_5 < 0 \overset{-8}{\Longrightarrow} p_4 < 0 \\ - - - - - - - - - \\ p_4 < 0, p_5 < 0 \\ p_2 < 0 \vee p_2 < 0 \vee p_2 = 0 \\ - - - - - - - - - - - - \\ (p_2 > 0, p_4 < 0, p_5 < 0) \vee (p_2 < 0, p_4 < 0, p_5 < 0) \vee (p_2 = 0, p_4 < 0, p_5 < 0) \\ p_2 > 0 \overset{-1,-4}{\Longrightarrow} p_1 < 0, p_3 > 0 \qquad\qquad p_2 = 0 \overset{2,3}{\Longrightarrow} p_1 \le 0, p_3 \ge 0 \\ - - - - - - - - - - - \quad - - - - - - - - - - - \quad - - - - - - - - - - - \\ (p_1 < 0, p_2 \ge 0, p_3 > 0, p_4 < 0, p_5 < 0) \vee (p_2 < 0, p_4 < 0, p_5 < 0) \\ \qquad\qquad \vee (p_1 \le 0, p_2 = 0, p_3 \ge 0, p_4 < 0, p_5 < 0), \end{array}$$

where $\overset{-8}{\Longrightarrow}$ means the converse-negative proposition of statement (8) in Lemma 3, and $\overset{2,3}{\Longrightarrow}$ the statements (2), (3) in Lemma 3. Theorem 1 is proved.

Acknowledgements. The authors would like to thank the referees for many valuable comments and suggestions. We also wish to thank Dongming Wang for helpful discussions.

References

1. B. F. Caviness, J. R. Johnson (eds.), *Quantifier Elimination and Cylindrical Algebraic Decomposition* (Proceedings of the symposium held at the Research Institute for Symbolic Computation, Johannes Kepler University, Linz), Texts and Monographs in Symbolic Computation. Springer-Verlag, Vienna, 1998.
2. J. H. Davenport, Y. Siret, E. Tournier, *Computer Algebra: Systems and Algorithms for Algebraic Computation* (2nd edn.). Academic Press, London, pp. 138–139, 1993.
3. E. B. Feinberg, C. H. Papadimitriou, Finding Feasible Points for a Two-point Body. *J. Algorithms* **10**, 109–119, 1989.
4. S. R. Finch, Moving Sofa Constant. In: *Mathematical Constants*, Cambridge University Press, Cambridge, pp. 519–523, 2003.
5. H. Iba, T. Tohge, Y. Inoue, Cooperative Transportation by Humanoid Robots — Solving Piano Movers' Problem. *Int. J. Hybrid Intell. Syst.* **1**(4), 189–201, 2004.
6. H. Hong, Quantifier Elimination in Elementary Algebra and Geometry by Partial Cylindrical Algebra Decomposition (QEPCAD) version B 1.44, http://www.cs.usna.edu/~qepcad/.
7. J. Lengyil, M. Reichert, B. R. Dolnald, D. P. Greenberg, Real Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware. *Computer Graphics* (SIGGRAPH '90 Proceedings) **24**(4), 327–335, 1990.
8. D. Leven, M. Sharir, An Efficient and Simple Motion Planning Algorithm for a Ladder Moving in Two-Dimensional Space Amidst Polygonal Barriers. *J. Algorithms* **8**, 192–215, 1987.
9. J. T. Schwartz, M. Sharir, On the Piano Movers' Problem I: The Case of a Rigid Polygonal Body Moving Amidst Polygonal Barriers. *Comm. Pure Appl. Math.* **36**, 345–398, 1983.
10. J. T. Schwartz, M. Sharir, On the Piano Movers' Problem II: General Techniques for Computing Topological Properties of Real Algebraic Manifolds. *Adv. Appl. Math.* **4**(3), 298–351, 1983.
11. D. Wang, Geometry Machines: From AI to SMC. In: *Proceedings of AISMC-3* (Steyr, Austria, September 23–25, 1996) (LNCS **1138**), Springer-Verlag, Berlin Heidelberg, pp. 213–239, 1996.
12. L. Yang, X. Hou, Z. Zeng, A Complete Discrimination System for Polynomials. *Sci. China* Ser. **E 39**(6), 628–646, 1996.
13. L. Yang, J. Zhang, A Practical Program of Automated Proving for a Class of Geometric Inequalities. In: *Automated Deduction in Geometry* (LNAI **2061**), Springer-Verlag, Berlin Heidelberg, pp. 41–57, 2001.

Computing Curves Bounding Trigonometric Planar Maps: Symbolic and Hybrid Methods

Daniel Lichtblau

Wolfram Research, Inc., 100 Trade Centre Dr., Champaign, IL 61820
danl@wolfram.com

Abstract. A few years ago S-H Kim investigated some problems at the boundary of number theory, optimization, and geometry. One question regarded an optimal packing of certain "triangular oval" planar curves and another looked at some related transformations of $\mathbb{R}^2$ to $\mathbb{R}^2$. These were investigated primarily using tools from calculus but it turns out that computational algebra methods may instead be employed to particular advantage. Moreover, generalizations that are beyond the reach of such methods are still amenable to hybrid approaches using numeric and symbolic methods in tandem. We introduce some of the specific problems and generalizations, and show by detailed example how such techniques may be implemented and deployed.

1 Introduction and Overview

A common need in two and three dimensional computational geometry is to bound regions given by algebraic maps. For example, while it is typically easy to find the implicit form of a curve given parametrically by rational functions of a variable (or trigonometric functions, as discussed at length in [10]), it is not so well understood how to find boundary curves of maps between planar regions. Perhaps more to the point, known symbolic methods already become overwhelmed by complexity in examples of modest size. Hence there is a desire to find methods less prone to the vagaries of intermediate swell and related pitfalls of symbolic computation.

What we develop will fall into the category of computational methods. We cite theory when relevant but do not purport to develop any herein. That said, the hybrid methods to be presented draw from several areas of numeric and symbolic computation (optimization, differential equation solvers, linear algebra, lattice methods, and more), and illustrate nicely the advantages of such synergy. Moreover, as will be seen from the family of examples we study, these methods can then be applied to the study of mathematical problems.

In particular, we will investigate certain properties of trigonometric planar mappings of the form

$$x = \cos(ma) + \cos(mb) + \cos(m(a-b))$$
$$y = \cos(na) + \cos(nb) + \cos(n(a-b))$$

for fixed integers m and n. We begin with a history and analysis of a particular simple case, illustrating symbolic methods that have been presented in previous

H. Hong and D. Wang (Eds.): ADG 2004, LNAI 3763, pp. 70–91, 2006.

venues. We proceed to develop a hybrid approach that applies when the symbolic methods are no longer up to the task.

We remark that similar maps e.g. replacing cosine with sine are also of interest and can be addressed using the same methods we will develop.

2 Background of the Problems Under Consideration

We begin with a quick review of material presented in [18], based in part on [12, 14]. Given

$$f(n) = \underset{a_j \epsilon \mathbb{R}}{\text{MinMax}} \left[\left| \sum_{j=1}^{n} e^{ia_j} \right|, \left| \sum_{j=1}^{n} e^{ina_j} \right| \right]$$

We want to say something useful about this function $f(n)$.

Theorem (paraphrased).
(i) $f(1) = 1$.
(ii) $f(2) = 1$.
(iii) For $n > 3$ we have $f(n) = 0$.

Proofs are in [13] and [14]. One might be tempted to state, as a corollary, that the only case of interest is when $n = 3$. We will see presently why this is not so. First we discuss the case $n = 3$, which was of interest in Kim's work and also lays the groundwork for the more difficult problems we will attack.

Proposition. For $f(3) = \underset{a,b \in [0,2\pi]}{\text{Min}} \text{Max}[|1 + e^{ia} + e^{ib}|, |1 + e^{3ia} + e^{3ib}|]$ the two moduli are equal.

Theorem. $f(3)$ occurs at a tangential intersection of the curves

$$\cos(a) + \cos(b) + \cos(a - b) = k$$
$$\cos(3a) + \cos(3b) + \cos(3(a - b)) = k$$

for values $\{a, b, k\}$. [These new functions are half of (the squares of the moduli -3). Once we find k, we recover $f(3)$ as $\sqrt{2k + 3}$.] Moreover at extremal values of $f(3)$ there will be only finitely many such intersection points, up to periodicity.

Proofs again are in [13, 14]. The utility of this theorem is that it points the way to an algorithm because there are only finitely many values of k for which the intersections will be tangential, and we will see shortly how to find them via elimination theory.

In the theorem above we have parametrized pairs of trigonometric polynomials associated to pairs of moduli of exponential sums. As we seek a minmax it is not surprising that this would occur at parameter values that give tangential intersections of the pairs. We show some trigonometric curve pairs for $k = -1.3$ and $k = -1.1$. Computations below are done with the version 5 of *Mathematica* [21] (*Mathematica* (TM) is a registered trademark of Wolfram Research, Inc.).

```
Needs["Graphics`"]
trigpoly[n_, a_, b_] = Cos[na] + Cos[nb] + Cos[n(a - b)];
grad[expr_, a_, b_]:={∂a expr, ∂b expr}
trigsubs = {Cos[a] → c_a, Sin[a] → s_a, Cos[b] → c_b, Sin[b] → s_b};

pl1 = ImplicitPlot[{trigpoly[1, a, b] == -1.1, trigpoly[3, a, b] == -1.1},
  {a, 0, π}, {b, π, 2π}, PlotPoints → 200, AspectRatio → 1,
  PlotStyle → {{Thickness[0.012], GrayLevel[0.5], Dashing[{0.25, 0.1}]},
    {Automatic}}, DisplayFunction → Identity];
pl2 = ImplicitPlot[{trigpoly[1, a, b] == -1.3, trigpoly[3, a, b] == -1.3},
  {a, 0, π}, {b, π, 2π}, PlotPoints → 200, AspectRatio → 1,
  PlotStyle → {{Thickness[0.012], GrayLevel[0.5], Dashing[{0.25, 0.1}]},
    {Automatic}}, DisplayFunction → Identity];
Show[GraphicsArray[{pl1, pl2}], DisplayFunction → $DisplayFunction];
```

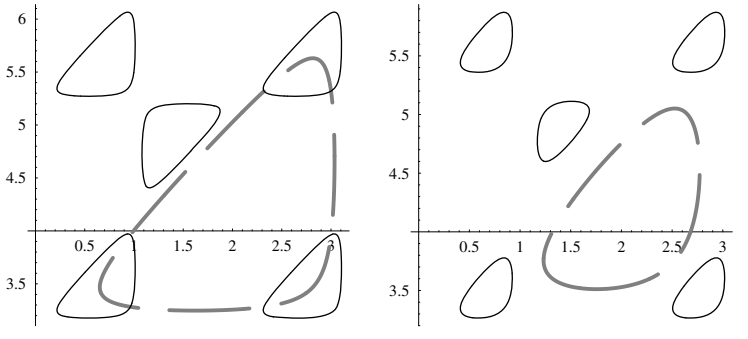


Fig. 1.

The level sets are apparently convex closed curves and for some value of $-1.3 < k < -1.1$ we will have tangential intersections which, it will turn out, gives the optimal value we seek. As the pictures provide a hint that the level curves are smooth and convex, that of course is already useful in pointing the way to the theory. But this is just the tip of the computational iceberg in regards to the problems that will come under scrutiny. To introduce them we will next find tangential intersections of these level sets by means of Lagrange multipliers.

The gist of the computation is as follows. We begin with the two curve equations from the theorem above. As the theorem tells us the intersection of interest is tangential, we augment with the standard Lagrange multiplier relation between their gradients. We transform the equations to explicitly algebraic ones by rewriting e.g. $\cos(a)$ as c_a. We must also augment by polynomials corresponding to algebraic relations such as $c_a^2 + s_a^2 = 1$. From these we construct a Gröbner basis to eliminate all variables but the one of interest, using a term order efficient for this task. Specifically, the variables to be eliminated are all weighted equally, and lexicographically larger than the ones that remain. Ties are broken with

graded reverse lexicographic ordering on the join of the set of elimination variables with the set of remaining variables. (This is the elimination order of Bayer and Stillman that is utilized to eliminate all but one variable. For background on construction and use of Gröbner bases, including elimination of variables from algebraic systems, see e.g. [1, 6]. The latter reference discusses this ordering in the exercises in chapter 3, section 1.)

{e1, e2} = Map[TrigExpand[trigpoly[#, a, b]]&, {1, 3}];
polys =
Flatten[{e1 − k, e2 − k, grad[e1, a, b] − λgrad[e2, a, b], Cos[a]² + Sin[a]² − 1,
Cos[b]² + Sin[b]² − 1}]/.trigsubs

$$\{-k + c_a + c_b + c_a c_b + s_a s_b, -k + c_a^3 + c_b^3 + c_a^3 c_b^3 - 3c_a s_a^2 - 3c_a c_b^3 s_a^2 + 9c_a^2 c_b^2 s_a s_b - 3c_b^2 s_a^3 s_b - 3c_b s_b^2 - 3c_a^3 c_b s_b^2 + 9c_a c_b s_a^2 s_b^2 - 3c_a^2 s_a s_b^3 + s_a^3 s_b^3,$$
$$-s_a - c_b s_a + c_a s_b - \lambda(-9c_a^2 s_a - 9c_a^2 c_b^3 s_a + 3s_a^3 + 3c_b^3 s_a^3 + 9c_a^3 c_b^2 s_b - 27c_a c_b^2 s_a^2 s_b + 27c_a^2 c_b s_a s_b^2 - 9c_b s_a^3 s_b^2 - 3c_a^3 s_b^3 + 9c_a s_a^2 s_b^3),$$
$$c_b s_a - s_b - c_a s_b - \lambda(9c_a^2 c_b^3 s_a - 3c_b^3 s_a^3 - 9c_b^2 s_b - 9c_a^3 c_b^2 s_b + 27c_a c_b^2 s_a^2 s_b - 27c_a^2 c_b s_a s_b^2 + 9c_b s_a^3 s_b^2 + 3s_b^3 + 3c_a^3 s_b^3 - 9c_a s_a^2 s_b^3), -1 + c_a^2 + s_a^2, -1 + c_b^2 + s_b^2\}$$

Timing[kpoly = First[GroebnerBasis[polys, k, {λ, c_a, s_a, c_b, s_b},
MonomialOrder → EliminationOrder]]]

$$\{0.38 \text{ Second}, 9 + 3k - 5k^2 + k^3 - 6k^4 - 4k^5 + 2k^6\}$$

We will factor it to see what values of k might be of interest.

Factor[kpoly]

$$(-3 + k)(-1 + k)(1 + k)(3 + 2k + 2k^2 + 2k^3)$$

The values -1, 1, and 3 do not in fact lead to the optimum for the original problem (at $k = -1$ the curves are particularly misbehaved insofar as the intersections are not even discrete). The root of interest turns out to be the real value for the cubic factor; as it is the most negative root it will give the minimal value for $f(3)$.

solns = NSolve[kpoly == 0, k]

$$\{\{k \to -1.20409\}, \{k \to -1.\}, \{k \to 0.102047 - 1.11146i\}, \{k \to 0.102047 + 1.11146i\}, \{k \to 1.\}, \{k \to 3.\}\}$$

A picture of the curve intersections for $k \approx -1.2$ will show they intersect tangentially.

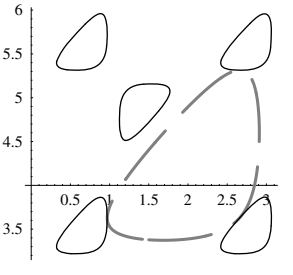


Fig. 2.

Note that we have what appear to be both horizontal and vertical tangencies (they really are). Hence we would have had trouble had we tried to solve for either variable as an implicit function of the other. One conclusion is that Lagrange multipliers constitute a good approach to this problem. Also notice that we have a "maximal" sort of lattice packing insofar as we cannot further expand the curve components without having their interiors intersect. This link between a problem in number theory and a lattice packing was an emphasis of [14]. We indicate pictorially the situation for the other roots in k. The case $k = -1$ gives intersections along line segments. For $k = 1$ we appear to have solved a different tangential intersection problem. For $k = 3$ we actually have degeneracies to points, which we show by plotting at $k = 2.9$ and observing small sets of ovals.

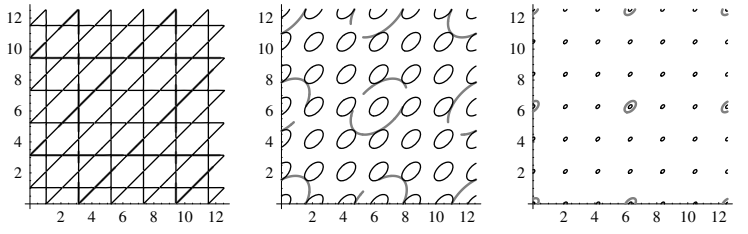


Fig. 3.

This sets the stage for the geometry problems we will consider. The focus will be on systems of equations of the type shown above that are based on Lagrange multiplier methods. This will allow us to find curves that satisfy certain extremal value conditions.

3 The Associated Planar Map

We now investigate the function from $\mathbb{R}^2$ to $\mathbb{R}^2$ that is motivated by the last example.

```
x[a_, b_] = trigpoly[1, a, b];
y[a_, b_] = trigpoly[3, a, b];
```

We plot the images of around 4000 random points from the square $[0, 2\pi] \times [0, 2\pi]$.

```
pointlist[m_]:=Table[2 * Pi * {Random[], Random[]}, {2^m}]
points2k = pointlist[12];
data[{point_}]:={x[point], y[point]};
datalist2k = Map[data, points2k];
ListPlot[datalist2k];
```

We have something a bit like a Lissajous figure (insofar as those also arise from trigonometric parametrizations). At $\{x, y\} = \{-1, -1\}$ we see the image pinched to a point as boundary curves cross. This may help to explain that

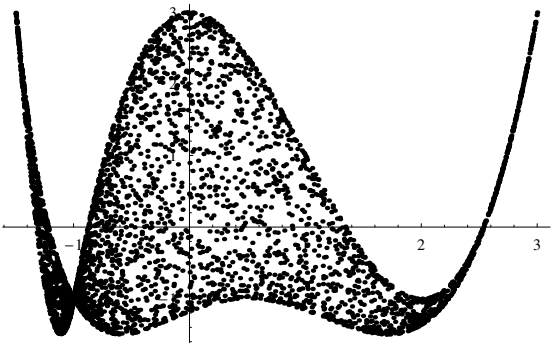


Fig. 4.

dimensional component of solutions to tangential intersection problem of the previous section.

The problem we now consider is to compute the boundary curve(s). Observe that our map may be regarded as rational functions of $z_a = e^{ia}$ and $z_b = e^{ib}$. Hence it is a rational map. As the boundary curves will be obtained from algebraic relations involving this map, they too are algebraic (so in a sense we are working with a bivariate flavor of [10]). To find these boundaries we again turn to Gröbner bases. We have two viable approaches. One is to see where the Jacobian of our map vanishes. The other possibility, similar to that used in the last section, is to find extremal values. We pursue this latter approach.

The idea is to write x and y as functions of trigonometric polynomials, again make substitutions so these become explicitly algebraic, and realize that at the extrema (that is, on the boundary curve(s)), the gradients of the two functions must be parallel. (Reason: on the boundary, for fixed $x = x(a, b)$ we extremize $y = y(a, b)$. This can be set up as a standard Lagrange multiplier problem with $\nabla y = \lambda \nabla x$.) Note that one might attempt to influence the computational efficiency by converting from trigonometric to exponential polynomials, thus reducing the number of polynomials and variables in the system. This is a method advocated in [10]. Our experience with this particular example is that the trigonometric formulation has the advantage. But one should bear in mind that this might be dependent on both Gröbner basis implementation and problem specific details.

Here is the trigonometric formulation.

$$\mathbf{parameters} = \{a, b\}; \mathbf{mainvars} = \{x, y\};$$
$$\mathbf{xpoly} = x - \mathbf{trigpoly}[1, a, b]; \mathbf{ypoly} = y - \mathbf{trigpoly}[3, a, b];$$
$$\mathbf{xypolys} = \{\mathbf{xpoly}, \mathbf{ypoly}\};$$
$$\mathbf{gradientpolys} = \mathbf{grad}[\mathbf{xpoly}, a, b] - \lambda \mathbf{grad}[\mathbf{ypoly}, a, b];$$
$$\mathbf{trigidentities} = \{c_a^2 + s_a^2 - 1, c_b^2 + s_b^2 - 1\};$$
$$\mathbf{elimvars} = \{\lambda, c_a, c_b, s_a, s_b\};$$
$$\mathbf{polys} = \mathbf{TrigExpand}[\mathbf{Join}[\mathbf{trigidentities}, \mathbf{xypolys}, \mathbf{gradientpolys}]]/.\mathbf{trigsubs}$$

$$\{-1 + c_a^2 + s_a^2, -1 + c_b^2 + s_b^2, x - c_a - c_b - c_a c_b - s_a s_b,$$
$$y - c_a^3 - c_b^3 - c_a^3 c_b^3 + 3c_a s_a^2 + 3c_a c_b^3 s_a^2 - 9c_a^2 c_b^2 s_a s_b + 3c_b^2 s_a^3 s_b + 3c_b s_b^2 + 3c_a^3 c_b s_b^2 -$$

$$9c_a c_b s_a^2 s_b^2 + 3c_a^2 s_a s_b^3 - s_a^3 s_b^3, s_a - 9\lambda c_a^2 s_a + c_b s_a - 9\lambda c_a^2 c_b^3 s_a + 3\lambda s_a^3 + 3\lambda c_b^3 s_a^3 - c_a s_b + 9\lambda c_a^3 c_b^2 s_b - 27\lambda c_a c_b^2 s_a^2 s_b + 27\lambda c_a^2 c_b s_a s_b^2 - 9\lambda c_b s_a^3 s_b^2 - 3\lambda c_a^3 s_b^3 + 9\lambda c_a s_a^2 s_b^3, - c_b s_a + 9\lambda c_a^2 c_b^3 s_a - 3\lambda c_b^3 s_a^3 + s_b + c_a s_b - 9\lambda c_b^2 s_b - 9\lambda c_a^3 c_b^2 s_b + 27\lambda c_a c_b^2 s_a^2 s_b - 27\lambda c_a^2 c_b s_a s_b^2 + 9\lambda c_b s_a^3 s_b^2 + 3\lambda s_b^3 + 3\lambda c_a^3 s_b^3 - 9\lambda c_a s_a^2 s_b^3\}$$

We now want to compute the polynomial in $\{x, y\}$ in the elimination ideal of the system as per [6]. That is, we eliminate from the system above all other variables. As before, this can be attempted via a Gröbner basis computation. Note that this is in effect a generalization of implicitization a la [11] insofar as we have a parametrization of the region rather than just its boundary.

```
Timing[gb = GroebnerBasis[polys, mainvars, elimvars, Sort → True,
  MonomialOrder → EliminationOrder]]
```

$$\{8.77 \text{ Second}, \{-216 - 324x + 135x^2 + 108x^3 - 243x^4 + 117x^5 + 216x^6 - 72x^7 - 48x^8 + 16x^9 - 108y - 270xy - 9x^2y + 342x^3y + 162x^4y - 48x^5y - 24x^6y + 63y^2 + 126xy^2 + 69x^2y^2 + 9x^3y^2 - y^3\}\}$$

Notice that we are left with one polynomial in the two variables, which means we indeed found a curve or curves. We will factor it to better understand what we have found.

```
envelope = Map[#[[1]]^#[[2]]&, Drop[FactorList[First[gb]], 1]]
```

$$\{3 - 3x^2 + x^3 - y, -72 - 108x - 27x^2 - 48x^3 - 72x^4 + 16x^6 - 60y - 126xy - 72x^2y - 8x^3y + y^2\}$$

We have a cubic function of x and an implicit relation that is quadratic in y. One may observe that the latter is not exactly trivial and is unlikely to be found without recourse to computer algebra. We next plot these curves to see specifically how they bound our map.

```
ImplicitPlot[Evaluate[Thread[envelope == 0]], {x, -1.5, 3.2}, {y, -2, 3.2},
  PlotPoints → 2000, AspectRatio → 1,
  PlotStyle → {{Thickness[.01], GrayLevel[.5], Dashing[{.15, .04}]}, {Automatic}}];
```

Before proceeding to the general case we use the picture to explain those nonminimal critical values for k derived in the previous section. The idea is to

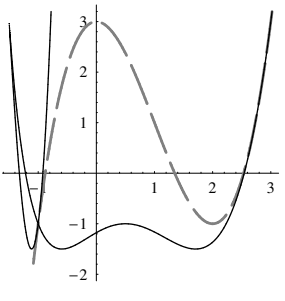


Fig. 5.

intersect with the line $y = x$ to see what happens at level curve intersections for particular values of k. Clearly $\{1, 1\}$ is a local maximum. For larger values the level curves do not intersect, but they gradually shrink to (colliding) points at the global maximum of $\{3, 3\}$. The remaining critical value is at $\{-1, -1\}$. From the graph we realize that when $x = -1$ we must have $y = -1$, but not vice versa. Hence the set of level curves for $y = -1$ contains those for $x = -1$ (one may wish to review the picture of level curves corresponding to $k = -1$; the larger dashed triangle boundaries are completely covered by their smaller solid counterparts, but not vice versa). As one set of level curves contains the other, we have a dimensional component in the intersection for this value.

4 Generalizing the Planar Maps

We have set the stage for the more general problem. While the work in [14] shows that, for the purposes of the original minmax problem, the cases of $n > 3$ are not relevant, in fact those give interesting planar maps from which one might wish to extract boundary curves. As an example we illustrate the $n = 5$ case below using around 4000 points.

```
x[a_, b_] = trigpoly[1, a, b]; y[a_, b_] = trigpoly[5, a, b];
pointlist[m_]:=Table[2 * Pi * {Random[], Random[]}, {2^m}]
points2k = pointlist[12];
data[{point_}]:={x[point], y[point]};
datalist2k = Map[data, points2k];
lplot = ListPlot[datalist2k];
```

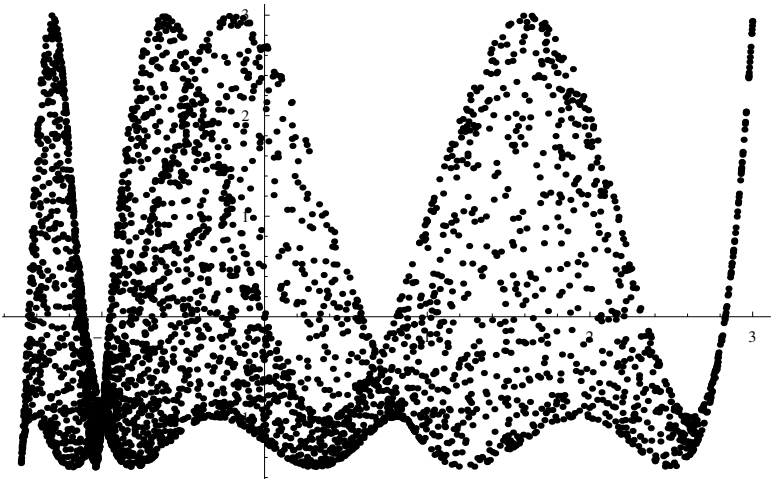


Fig. 6.

One can discern more complexity in this plot, insofar as there appear to be "ghost" curves inside the region that perhaps correspond to curves in the

domain where the Jacobian fails to have full rank. It turns out that for $n = 4$ we are already unable to find the boundary curves in reasonable time using the method of the previous section. After conversion to exponentials a la [10], the program Fermat was able to do the variable elimination via a Dixon resultant computation in about 15 minutes [Lewis, private communication]. For $n = 5$ it ran out of memory. While it is plausible that other programs might progress further, it seems unlikely that any will go far beyond the range for which results have been obtained by symbolic methods. Hence the motivation for the numeric-symbolic approach we describe below. We take for our working example the case where $n = 5$.

```
xpoly = x - trigpoly[1, a, b]; ypoly = y - trigpoly[5, a, b];
xypolys = {xpoly, ypoly};
gradientpolys = grad[xpoly, a, b] - λgrad[ypoly, a, b];
polys = TrigExpand[Join[trigidentities, xypolys, gradientpolys]]/.trigsubs;
```

Now we may set values for y and solve for the rest, retaining solutions in x. The method we first show was used to advantage in a computational geometry setting in [19], in a case where standard root finding approaches were problematic.

```
allbutyvars = {x, λ, c_a, c_b, s_a, s_b};
polyy = polys/.y → 11/10;
Timing[soln = NSolve[polyy, allbutyvars, Sort → True];]
```

```
{59.67 Second, Null}
```

```
Select[Union[x/.soln, SameTest → (Abs[#1 - #2]/(Abs[#1] + Abs[#2]) < 10^(-5)&)],
  Im[#] == 0&]
```

```
{-1.4246, -1.18145, -1.17679, -0.952203, -0.895835,
  - 0.684639, -0.191699, 0.382082, 1.02001, 2.15932, 2.9082, 2.9086}
```

While this does give results in modest time we perceive two difficulties. One is that it will take a very long time to get sufficiently many points to use to recover the curves. Another is that this method will not extend much further; it is quite sensitive to the degrees of the input polynomials and thus for larger n it is not likely to give us solution points in any reasonable amount of time. Hence we drop the idea of solving for several values of x for a given y, and focus instead on obtaining one such solution at a time. The most naive approach would be to use a simple root finder. These tend to be local methods and require reasonable initial guesses for solution values in order to converge to a root. Once we select a value, say for y, it is not hard to see what might be a reasonable value for x. But the best we can say for the "trigonometric" parameters is that they lie in the range between -1 and 1. Even worse, we have no information for the Lagrange multiplier. In such a situation it will turn out to be preferable to use a global method that allows for constraints on values. While one might use e.g. interval methods for constrained global root finding, in *Mathematica* it is more convenient to use the optimization function `NMinimize`. We have some latitude in how we do this. For example, we may use explicit constraints to enforce some of the polynomial equations, and take as objective function the sum of squares of

the remaining polynomials (we thus hope to find solution values for which this is close to zero), perhaps plus a constant. Some trial and error experimentation indicates that it is convenient to enforce all of the equations as constraints, working with just a constant objective function. Admittedly this is not the most obvious sort of thing to set up; further information regarding use of `NMinimize` may be found in [2]. Note that here it makes good sense to work with the trigonometric parametrization as we have a very good set of ranges for those variables. With a parametrization in terms of complex exponentials we would need to work separately with real and imaginary parts (in effect returning to the trigonometric parametrization), whereas with the "usual" rational parametrization of a torus our parameters would take on values in an infinite range.

```
Timing[{val, root} = NMinimize[{1, Flatten[{Thread[polyy == 0], .9 ≤ x ≤ 1.2}]},
  {{x, .9, 1.2}, {λ, −10, 10}, {c_a, −1, 1}, {c_b, −1, 1}, {s_a, −1, 1}, {s_b, −1, 1}},
   MaxIterations → 200, Method → "DifferentialEvolution"]]
```

$\{6.42\ \text{Second}, \{1., \{x \to 1.02001, \lambda \to 0.20024, c_a \to 0.010004,$
$c_b \to 1., s_a \to -0.99995, s_b \to 1.44181 \times 10^{-15}\}\}\}$

At this point we will note that the choice of $y = 11/10$ is very likely "generic", and hence the fact that the solution values clearly indicate $b = 0$ means that we are very likely on a boundary segment in the $\{a, b\}$ domain. Thus it should not be difficult to find this part of the boundary simply by implicitizing the domain boundary curve $b = 0$, discarding from our polynomial system the Lagrange multiplier part. We show this now.

```
envelopepiece1 =
  First[GroebnerBasis[Take[polys, 4]/.{c_b → 1, s_b → 0}, {x, y}, {c_a, s_a}]]
```

$-5x + 5x^2 + 5x^3 - 5x^4 + x^5 - y$

A glance at the plot will convince us that a part of it comprises a piece of the map boundary.

```
envplot1 = ImplicitPlot[Evaluate[envelopepiece1 == 0], {x, −2, 3.2}, {y, −2, 3.2},
  PlotStyle → {Thickness[.01], GrayLevel[.5], Dashing[{.15, .07}]}, PlotPoints → 1000];
```

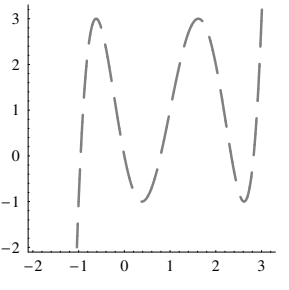


Fig. 7.

That piece was relatively easy due to a stroke of luck: we deduced that it was the image of a boundary segment. We now look at that part of the boundary in

the bottom right portion of the picture. We note that for $y = -11/10$ we will have $1.5 < x < 2$.

```
polyy = polys/.y → -11/10;
Timing[
 {val, root} = NMinimize[{1, Flatten[{Thread[polyy == 0], 1.2 ≤ x ≤ 2}]},
  {{x, 1.2, 2}, {λ, -10, 10}, {c_a, -1, 1}, {c_b, -1, 1}, {s_a, -1, 1}, {s_b, -1, 1}},
  MaxIterations → 200, Method → "DifferentialEvolution"]]
```

```
{15.65 Second, {1., {x → 1.72049, λ → 1.13162, c_a → 0.768955,
 c_b → 0.768955, s_a → 0.639303, s_b → -0.639303}}}
```

This time the solutions for the trigonometric parameters suggest the segment $b = -a$ in the domain. We obtain the boundary for this piece using the same method as above.

```
envelopepiece2 =
 First[GroebnerBasis[Take[polys, 4]/.{c_b → c_a, s_b → -s_a}, {x, y}, {c_a, s_a}]]
```

$$5400 + 11700x - 8675x^2 - 37800x^3 - 20600x^4 + 14880x^5 + 13680x^6 - 1920x^7 - 3200x^8 + 256x^{10} - 1980y - 6230xy - 7280x^2y - 3800x^3y - 800x^4y - 32x^5y + y^2$$

```
envplot2 = ImplicitPlot[Evaluate[envelopepiece2 == 0], {x, -2, 3.2}, {y, -2, 3.2},
 PlotStyle→{Thickness[.012], GrayLevel[.35], Dashing[{.12, .08}]}, PlotPoints→1000];
```

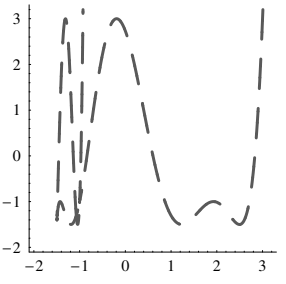


Fig. 8.

It appears that these together give most of the boundary. We now hunt for the part in the lower middle. For this we try to find a point near the lower crossing of the boundary curve with the y axis. It turns out to be expedient to make two changes to the minimization. One is that we now fix the value of x rather than y, and the other is that we will square the polynomial constraints (this seems to have the effect of better balancing them for the problem at hand).

```
polyx1 = polys/.x → 0;
Timing[
 {val, root1} = NMinimize[{1, Flatten[{Thread[polyx1^2 == 0], -1.6 ≤ y ≤ -1.2}]},
  {{y, -1.6, -1.2}, {λ, -10, 10}, {c_a, -1, 1}, {c_b, -1, 1}, {s_a, -1, 1}, {s_b, -1, 1}},
  MaxIterations → 200, Method → "DifferentialEvolution"]]
```

```
{10.88 Second, {1., {y → -1.25, λ → -0.8, c_a → -0.21616,
 c_b → -0.660006, s_a → 0.976358, s_b → 0.75126}}}
```

This time it is not obvious from the trigonometric parameter values how a domain segment or curve might give us this part of the boundary, so now we'll have to do some real work. We proceed to find more boundary points in order to reconstruct the curve therefrom. We begin with the solution point we just found. We will want to make sure we have a good approximation. As we used a global optimizer we may suspect that the root values are accurate to but a few decimal places (though the exact values for y and the Lagrange multiplier are fairly obvious). But we now have good initial values from which to proceed further, we can refine the result to arbitrarily high precision using a local root finder.

```
allbutxvars = {y, λ, c_a, c_b, s_a, s_b};
rootbetter1 =
 FindRoot[polyx1 == 0,
  Evaluate[Apply[Sequence, Transpose[{allbutxvars, allbutxvars/.root1}]]],
  PrecisionGoal → 30, AccuracyGoal → 30, WorkingPrecision → 40]
```

$\{y \to -1.250000000000000000000000000000000000000,$
$\lambda \to -0.800000000000000000000000000000000000000,$
$c_a \to -0.216160221673850756457782105666535422 0011,$
$c_b \to -0.660005866723161169829745025809548792 4794,$
$s_a \to 0.976357904953870746132758834833973953 6197,$
$s_b \to 0.751260444779976962667400856474058841 5410\}$

It turns out that by comparison with the values of `root1` we actually had a good result even then, correct to about 8 decimal places. This is most likely due to use of a local polishing step in `NMinimize`.

To deduce the actual curve we will require a number of $\{x, y\}$ pairs of points lying on it. We can obtain these in various ways. One is simply to invoke `NMinimize` as above but with different values for y. This can be tedious, especially as the minimization can present problems in some regions (possibly due to proximity of multiple solutions, or difficulty balancing the various constraint equations). Instead we will treat this as a homotopy continuation problem wherein the roots found above will be initial values for a differential system. As it is easier to see y as a function of x in this part of the curve, we will use the latter as independent variable. We create the system below but display it in abbreviated form due to excessive size.

```
varsinx = Through[allbutxvars[x]];
parampolys = polys/.Thread[allbutxvars → varsinx];
diffpolys = D[parampolys, x];
Short[
 odesystem1 = Join[Thread[diffpolys == 0],
  Thread[(varsinx/.x → 0) == (allbutxvars/.rootbetter1)]], 12]
```

$\{2c_a[x]c_a'[x] + 2s_a[x]s_a'[x] == 0, 2c_b[x]c_b'[x] + 2s_b[x]s_b'[x] == 0,$
$1 - c_a'[x] - c_b[x]c_a'[x] - c_b'[x] - c_a[x]c_b'[x] - s_b[x]s_a'[x] - s_a[x]s_b'[x] == 0,$
$y'[x] - 5c_a[x]^4c_a'[x] - 5c_a[x]^4c_b[x]^5c_a'[x] + 30c_a[x]^2s_a[x]^2c_a'[x] +$
$30c_a[x]^2c_b[x]^5s_a[x]^2c_a'[x] - 5s_a[x]^4c_a'[x] - 5c_b[x]^5s_a[x]^4c_a'[x] -$
$100c_a[x]^3c_b[x]^4s_a[x]s_b[x]c_a'[x] + 100c_a[x]c_b[x]^4s_a[x]^3s_b[x]c_a'[x] + \langle\langle 82\rangle\rangle +$

$30c_b[x]^2 s_a[x]^5 s_b[x]^2 s_b'[x] - 20c_b[x]s_b[x]^3 s_b'[x] - 20c_a[x]^5 c_b[x]s_b[x]^3 s_b'[x] +$
$200c_a[x]^3 c_b[x]s_a[x]^2 s_b[x]^3 s_b'[x] - 100c_a[x]c_b[x]s_a[x]^4 s_b[x]^3 s_b'[x] -$
$25c_a[x]^4 s_a[x]s_b[x]^4 s_b'[x] + 50c_a[x]^2 s_a[x]^3 s_b[x]^4 s_b'[x] - 5s_a[x]^5 s_b[x]^4 s_b'[x] == 0,$
$\langle\langle 135\rangle\rangle + \langle\langle 1\rangle\rangle == 0, \langle\langle 1\rangle\rangle, \langle\langle 1\rangle\rangle, \langle\langle\ 1\rangle\rangle, c_a[0] == -\langle\langle 63\rangle\rangle,$
$c_b[0] == -0.660005866723161169829745025809548792 4794,$
$s_a[0] == 0.976357904953870746132758834833973953 6197,$
$s_b[0] == 0.751260444779976962667400856474058841 5410\}$

```
desoln1 = NDSolve[odesystem1, varsinx, {x, 0, 1.2}];
```

NDSolve::ndsz :
At x == 0.3090169939494037‘, step size is effectively zero;
singularity or stiff system suspected.

The message tells us we were not able to get very far. A reasonable guess as to the cause is that we may be near a tangential intersection of solution curves (which would amount to a bifurcation in the solution set to what is really a differential algebraic system). In any case, what we have already suffices to give us points that lie approximately on the boundary curve. We will use several of them as initial values to find boundary points to high precision.

```
solnfunctions1 = First[varsinx/.desoln1];
valuelists1 =
 Table[Join[{t → x}, Thread[List[allbutxvars, solnfunctions1]]],
  {x, 0, 3/10, 5/1000}]/.t → x;
highprecsolns1 =
 Table[
  Flatten[{First[valuelists1[[j]]],
  FindRoot[Evaluate[(polys/.First[valuelists1[[j]]]) == 0],
   Evaluate[Apply[Sequence, Rest[valuelists1[[j]]]]], PrecisionGoal → 100,
   AccuracyGoal → 100, WorkingPrecision → 150]}], {j, Length[valuelists1]}];
```

We now find more solutions by moving to another part of (what we suspect will be) the same curve. We repeat the process of finding one solution, refining it, approximating a part of the curve through it, and refining individual points thereon.

```
polyx2 = polys/.x → -2/5;
Timing[
 {val, root2} = NMinimize[{1, Flatten[{Thread[polyx2^2 == 0], -1.2 ≤ y ≤ -.9}]},
  {{y, -1.2, -.9}, {λ, -10, 10}, {c_a, -1, 1}, {c_b, -1, 1}, {s_a, -1, 1}, {s_b, -1, 1}},
  MaxIterations → 200, Method → "DifferentialEvolution"]]
```

{10.79 Second, {1., {$y \to -1.02904$, $\lambda \to 1.5674$,
$c_a \to -0.847762$, $c_b \to -0.337837$, $s_a \to 0.530377$, $s_b \to 0.941204$}}}

```
rootbetter2 =
 FindRoot[polyx2 == 0,
 Evaluate[Apply[Sequence, Transpose[{allbutxvars, allbutxvars/.root2}]]],
 PrecisionGoal → 30, AccuracyGoal → 30, WorkingPrecision → 40];
odesystem2 = Join[Thread[diffpolys == 0],
 Thread[(varsinx/.x → -2/5) == (allbutxvars/.rootbetter2)]];
desoln2 = NDSolve[odesystem2, varsinx, {x, -2/5, -1}];
```

```
NDSolve::ndsz :
  At x == -0.809017, step size is effectively zero;
  singularity or stiff system suspected.
```

Again we did not get as far as we might like, but no matter. We will take several estimated solutions and refine them as before.

```
solnfunctions2 = First[varsinx/.desoln2];
valuelists2 =
  Table[Join[{t → x}, Thread[List[allbutxvars, solnfunctions2]]],
    {x, -2/5, -4/5, -5/1000}]/.t → x;
highprecsolns2 =
  Table[
    Flatten[{First[valuelists2[[j]]],
    FindRoot[Evaluate[(polys/.First[valuelists2[[j]]]) == 0],
    Evaluate[Apply[Sequence, Rest[valuelists2[[j]]]]], PrecisionGoal → 100,
    AccuracyGoal → 100, WorkingPrecision → 150]}], {j, Length[valuelists2]}];
```

With these values we can now attempt to reconstruct the curve. As reasonable inferences from the picture we guess that it is a polynomial of degree at most 2 in y, and 12 in x. We form the possible monomials as our "basis" set, evaluate these at the $\{x, y\}$ values corresponding to our high precision approximations of points on the boundary, and find the appropriate null space. Given that we have vectors of approximate numbers, the reliability of such an undertaking depends heavily on the method. Specifically it is based on singular value decomposition rather than, say, Gaussian elimination (see e.g. [7]).

```
xyvals = Join[highprecsolns1, highprecsolns2]/.
  ((a_/;!MatchQ[a, x|y]) → b_) :→ Sequence[];
xypowers = Flatten[Table[x^j * y^k, {j, 0, 12}, {k, 0, 2}]];
valuevectors = xypowers/.xyvals;
ns = NullSpace[valuevectors];
Length[ns]
```

```
16
```

We find that we have 16 null vectors, so we cannot formulate a unique (to scalar multiples) polynomial. But this merely indicates that we have more monomials in our basis than we require; that is, the defining polynomial is (not surprisingly) using far fewer than the full set we provided. We "squeeze" out a null vector in terms of lower degree monomial basis members by extracting the last element of the null set in row echelon form. Before we proceed we will do two things. One is to replace the null vector values that are very clearly zeros to high accuracy with exact zeros (so as not to mistakenly use them as pivots). The other is to reverse the vectors so that entries that correspond to higher degree basis members are at the beginning. Strictly speaking it would probably be better to order by total degree, but this will suffice for our purposes. After we row reduce we may again have artificial values which we replace by zero.

```
ns2 = Chop[ns, 10^(-145)];
ns3 = Map[Reverse, ns2];
```

```
rred = RowReduce[ns3];
rred2 = Chop[rred, 10^(-145)];
N[Last[rred2]]
```

$$\{0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., -1.25, 0., 0., 0., 0., 0., 0.3125, 0., 0.25, 0.3125\}$$

This looks quite promising. We rationalize, find the common denominator, remove it by multiplication, and finally form the curve by associating the resulting vector of coefficients with the monomial basis.

```
coeffs = Rationalize[Last[rred2]];
mult = Apply[LCM, Denominator[coeffs]];
coeffs2 = coeffs * mult;
envelopepiece3 = coeffs2.Reverse[xypowers]
```

$$5 + 5x - 20x^3 + 16x^5 + 4y$$

```
envplot3 = ImplicitPlot[Evaluate[envelopepiece3 == 0], {x, -2, 3.2}, {y, -2, 3.2},
  PlotStyle→{Thickness[.008], GrayLevel[.2], Dashing[{.12, .08}]}, PlotPoints→1000];
```

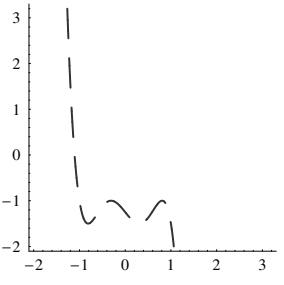


Fig. 9.

This certainly looks like that missing lower middle portion of the boundary. We'll put them all in one picture to see that this is so.

```
ll = Show[{lplot, envplot1, envplot2, envplot3}];
```

We now want to verify that our curve is indeed part of the boundary of the planar map. To this end we pick specific exact values on the curve and see if they are consistent with the original polynomial system. This will happen if and only if they lie on a curve that satisfies the original system. For example, we solve for y when $x = -1/2$.

```
xycoord = Prepend[First[Solve[(envelopepiece3/.x → -1/2) == 0, y]], x → -1/2]
```

$$\{x \to -\tfrac{1}{2}, y \to -\tfrac{9}{8}\}$$

We plug these values into the polynomial system and check whether we get a nontrivial Gröbner basis.

```
GroebnerBasis[polys/.xycoord, {λ, c_a, c_b, s_a, s_b},
  MonomialOrder → DegreeReverseLexicographic]
```

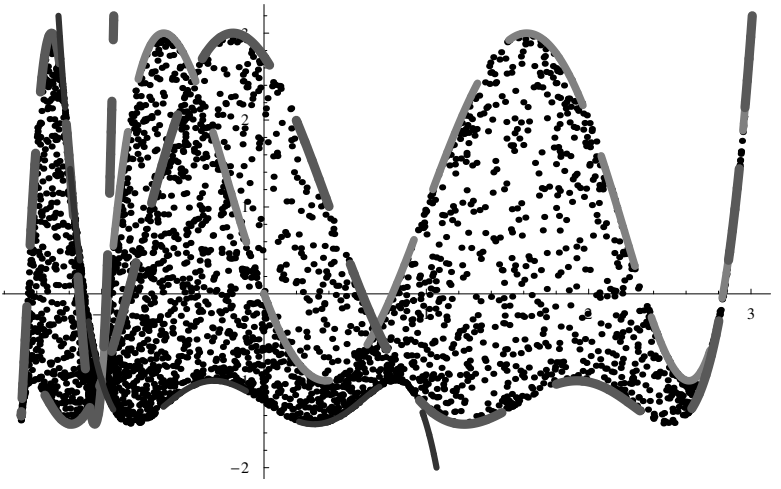


Fig. 10.

$$\{4-5\lambda, 7-4c_a-4c_b-8s_a^2-8s_as_b-8s_b^2, -1+c_b^2+s_b^2, -1-2c_a-2c_b-2c_ac_b-2s_as_b,$$
$$1+4c_a-8c_a^2+4c_b+8s_as_b+8s_b^2, 2s_a-2c_bs_a-5s_b+2c_as_b+4c_bs_b+8s_b^3,$$
$$5s_a+4c_as_a+3c_bs_a-s_b-7c_as_b-4c_bs_b-16s_as_b^2, -2-3c_b+4s_b^2+8c_bs_b^2,$$
$$-1+c_b+2s_as_b-2c_bs_as_b+2s_b^2+2c_as_b^2, -3-4c_a+4s_as_b+8c_as_as_b+8s_b^2+8c_as_b^2\}$$

This indicates that the point is in fact on a boundary curve. We try another such point. For brevity we will simply ascertain that the basis is not the trivial one (containing just 1) that would arise from an inconsistent system.

```
xycoord2 = Prepend[First[Solve[(envelopepiece3/.x → -1/4) == 0, y]], x → -1/4]
GroebnerBasis[polys/.xycoord2, {λ, c_a, c_b, s_a, s_b},
  MonomialOrder → DegreeReverseLexicographic]=!={1}
```

$\{x \to -\frac{1}{4}, y \to -\frac{259}{256}\}$

True

By way of contrast, we now demonstrate that a point not on the boundary will, as claimed above, give a basis containing just 1. For this we perturb the y coordinate from the last example. (That an inconsistent polynomial system has a reduced Gröbner basis of 1 is well known from general theory, but all the same should be demonstrated from time to time in actual computation.)

```
GroebnerBasis[polys/.{x → -1/4, y → -261/265}, {λ, c_a, c_b, s_a, s_b},
  MonomialOrder → DegreeReverseLexicographic]
```

{1}

By verifying sufficiently many points—the number being dependent on the degree of the boundary curve—one can prove that the entire curve satisfies the boundary curve polynomial system. This might be done using interpolation methods as in [9], chapter 5.

For general interest we indicate another way to recover the boundary curve, this time from one point. We will start with the exact value but it will be seen that all we actually require is a high precision approximation. For this to work it is advisable that the coordinates not be algebraic numbers (so as to avoid any algebraic dependencies not forced by the curve polynomial itself). We will take the point with $x = \frac{\pi}{4}$.

```
xycoord = Prepend[First[Solve[(envelopepiece3/.x → -Pi/4) == 0, y]], x → -Pi/4]
valuevector = xypowers/.xycoord;
```

$\{x \to -\frac{\pi}{4}, y \to \frac{1}{256}(-320 + 80\pi - 20\pi^3 + \pi^5)\}$

This time we form a lattice with these evaluated basis monomials multiplied by a suitably large constant in the first column and an identity matrix adjoined to the right.

```
lat = Transpose[Prepend[IdentityMatrix[Length[valuevector]],
  Round[10^200 * N[valuevector, 210]]]];
```

We now reduce the lattice and take as approximate null vectors all those for which the first column is fairly small. Again we reverse the ordering so as to have later components correspond to lexicographically smaller monomials, row reduce the matrix of null vectors, and use the last vector as coefficients for our implicit polynomial.

```
nullvecs = Map[Drop[#, 1]&, Select[redlat, Abs[#[[1]]] < 20&]];
Last[RowReduce[Map[Reverse, nullvecs]]]
```

$\{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, -5/4,$
$0, 0, 0, 0, 0, 5/16, 0, 1/4, 5/16\}$

We see that we did in fact recover the same polynomial coefficients as with the previous approach. We remark that this method is quite similar to that of finding a minimal polynomial for a given approximate number (which, for example, is the method behind the *Mathematica* function `NumberTheory‘Recognize`). This in turn may be utilized to factor a univariate polynomial as in [15]. One might also use for these purposes the efficient integer relation finding method known as PSLQ [8].

5 Discussion and Directions for Future Work

We began with a brief review of a problem that arose in a number theory setting and in some way migrated to computational geometry as an example of a sort of extremal planar packing. We showed how symbolic computation methods may be applied to determine important values in that problem. We next discussed a planar trigonometric polynomial parametrization associated in a natural way with the original problem. It gave rise to a region for which the boundary in a sense generalizes a Lissajous figure. The next step in our enquiry was to utilize symbolic algebra to determine the equation for each part of that boundary. As

we did not have a direct parametrization of that boundary this task could not be done by the usual implicitization methods (which tend to be tractable to at least moderate degree).

Thus far this was a warm up to the real problem at hand. When we look at examples of such maps in higher degrees the symbolic methods become mired by computational complexity. While certainly it might be possible that a clever reformulation, or a more powerful symbolic elimination implementation, might suffice, the methods we next developed are in a sense less troubled by degree or by specifics of the problem. Hence they might be utilized when one reaches the point at which clever symbolic approaches simply fail to yield a result.

For the harder problem we employed a mix of heuristics and numeric methods. We found approximations of isolated boundary curve points using both a numeric algebraic solver and optimization software (the latter being more readily applied in harder cases). In certain cases luck was on our side and we were able to "guess" the domain segment that gave rise to the part of the boundary on which our point was found. When this was not possible we showed how to refine the approximations and generate more approximate boundary points by setting up and solving what amounts to a DAE system; we worked with it instead as an ODE system in order to maintain simplicity in the exposition and code. We then used simple numeric linear algebra to fit to a guessed curve. For this we used far more points than we had basis vectors, so as to be confident of generating the correct null space to good approximation (this seems to be all the more important given how close our sample points were clustered). We next employed row reduction to find a "minimal" null vector corresponding to the "smallest" basis monomials. This gave our solution curve polynomial.

We remark that the steps of solving an ODE (or DAE) system to obtain points on the graph, and computing a null space for a monomial basis evaluated at those points, are also utilized in [4] for the purpose of approximate factorization of bivariate polynomials. This in turn is related to work in [5] for approximate implicitization of hypersurfaces. Our usage is actually more closely related to the former reference, in that the latter uses a calculus of variations formulation involving an integral functional, which first requires a parametrization of the curve. A big difference between our problem and that of bivariate factorization is that our hypersurface (curve, in this case) arises from an extremality condition on a surface map and hence there are added complications both to finding points on and following paths along it. One also recognizes at this point that our method might find a factor over the (approximate) reals that is a proper factor of a factor over the rationals. We will discuss this possibility in the last section.

An issue that did not arise in our particular example is as follows. The method of finding an approximate factor of the boundary curve might yield a factor over an extension field of the rationals. In this case we are faced with a few choices, and what we do will depend on what we require. If our interest is in the computational geometry aspects of the problem we might be quite happy to work with that approximate factorization over the reals. If we are working

on a problem from mathematics that requires the exact factor (as was the case in [14]) then we can attempt to find the exact form by methods that recognize algebraic numbers from numeric approximations of sufficiently high precision as discussed in [15, 8]. Alternatively we might try to find approximations of the algebraic conjugate factors, e.g. using methods from [3], and combine the factors to deduce the correct factor over the rationals. Finally we should note that if only a crude approximation to the curve is needed, then fast methods of edge detection from the field of image processing may suffice.

One thing to notice is that in the type of problem we discussed this issue cannot arise unless a similar factorization takes place in the domain. To be specific, recall that a necessary condition that a point be on the boundary curve is that it be the image of a point in the domain at which the Jacobian vanishes. Thus distinct factors in the boundary must come from distinct factors of the Jacobian. Since we have the Jacobian in explicit form, methods of absolute factorization might be employed to see whether there are factors over a nontrivial algebraic extension of the rationals.

It is important to realize that hybrid methods appear in several ways in this work, some lurking behind the scenes. For example, the numeric solver `NSolve` is based on a mix of Gröbner basis and numeric methods [16]. The function `Rationalize` is based on continued fraction methods (see e.g. chapter 4 of [9]), which themselves can use mixed exact and approximate numeric methods when operating at high precision (though the example we showed was quite tame). Various implementations of lattice reduction make use of approximate arithmetic. Though we did not do so, a lattice reduction method may be employed to rationalize a set of approximate values collectively so as to obtain a common denominator. This is generally referred to as simultaneous diophantine approximation, and the underlying method is quite similar to that of recognizing a minimal polynomial from a root approximation; the principle difference in implementation involves whether or not a lattice is transposed. See [9], chapter 17.

One obvious question to pose is just how generally useful are the methods we have discussed. To a large extent this depends on the sort of computational problem one wishes to solve. For purposes of obtaining a numeric approximation e.g. for graphing, what we have shown clearly requires too much work. It is only when one requires either an exact implicit curve or a high precision approximation that something of this sort must be done. This can arise, for example, in finding algebraic surface level curves (in effect we did that, taking as our third surface coordinate the Jacobian, and looking at the set where it vanishes). Another such computation might be finding offset curves from an algebraic planar object in cases where the complexity of a purely symbolic approach is too great. For this one might start with approximations based on fast marching methods as described in [20] (a simple *Mathematica* implementation of which may be found in [17]).

We conclude with several questions and remarks that indicate directions for further investigation.

• Could methods from real algebraic geometry e.g. cylindrical algebraic decomposition be employed to find boundary curves efficiently?

• Can the process we described be automated? In our work it required a lot of eyeballing of the list plot to see what might be viable places to look for approximate curve points. Indeed, there were several failed attempts (not shown).

• Are there more robust ways to find individual points on the curves? While we used some global optimization software, other methods e.g. with intervals might work better. Image processing methods can be used to find approximate coordinate values for $\{x, y\}$ but one also requires corresponding values from the domain space in order to obtain high precision refinements. A possible approach would be to regard $\{x, y\}$ as a Taylor series in $\{a, b\}$ and then "invert". Of course on the curve of interest the inverse function theorem does not apply, because we are precisely where the Jacobian vanishes! But this is not really a concern because a "multivalued" inverse function approximation e.g. from a Puiseux series will suffice for our purposes. Alternatively we might lift the problem to three dimensions, say by adding a coordinate function $z = \text{Jacobian}(x(a, b), y(a, b))$, and change coordinates to invert locally the map from plane to surface.

•In the symbolic computations of boundary curves we found the extremal formulation in terms of Lagrange multipliers to be preferable to the vanishing Jacobian approach (possibly because the latter gives factors that are extraneous for our purposes). A drawback to the Lagrange multiplier method in our later numeric approach is that we really do not have a good idea of the range of values the multiplier variable might take. So it might be advantageous to revisit the vanishing Jacobian formulation in conjunction with the numeric techniques.

• Are there better ways to "track" a piece of curve to obtain a set of approximate points thereon?

• We first found candidate boundaries. We outlined a method of validating them algorithmically, by showing that individual points satisfy the boundary curve system. We chose this approach because the more direct route of computing an elimination ideal appears to be infeasible using the software available. It would be interesting to know whether there might be a feasible direct approach.

• Our method of deducing the implicit polynomial from rationalizing relied on those curves being defined by polynomials over the rationals. Is there a way to extend to curves with minimal polynomials having algebraic numbers for coefficients?

• Might this method generalize in a reasonable way to implicitization of bounding surfaces of parametrized volumetric maps?

Acknowledgements

I thank Ken Stolarsky (thesis advisor to Seon-Hong Kim) for prompting me to consider the many computational aspects of this family of problems. I thank Josef

Schicho for reminding me of the sometimes more efficient approach to similar problems that was used in [10]. I thank the organizers of the ACA 2003 session on Symbolic-Numeric methods for Curves and Surfaces, John Wetzel (emeritus of the UIUC math department and organizer of the Geometric Potpourri seminar series), and the organizers of the ADG 2004 conference, all of whom provided opportunities for me to speak on various stages of this work.

References

1. B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In *Multidimensional Systems Theory*, chapter 6. N. K. Bose, ed. D. Reidel Publishing Company, 1985.
2. B. Champion. Numerical Optimization in Mathematica: An Insider' s View of NMinimize. In SCI2002, *Proceedings of the 6th World Multiconference on Systemics, Cybernetics, and Informatics*. Volume 16, N. Callaos, T. Ebisuzaki, B. Starr, J. M. Abe, D. Lichtblau, eds., pages 136-140. International Institute of Informatics and Systemics, 2002. A *Mathematica* notebook version may be found at: http://library.wolfram.com/infocenter/Conferences/4311/ Up to date documentation regarding NMinimize may be found at: http://documents.wolfram.com/v5/Built-inFunctions/AdvancedDocumentation/ Optimization/NMinimize/
3. R. M. Corless, A. Galligo, I. S. Kotsireas, S. M. Watt. A Geometric-numeric algorithm for absolute factorization of multivariate polynomials. In: *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation* (ISSAC 2002), T. Mora, ed., pages 37-45. ACM Press, 2002.
4. R. M. Corless, M. W. Giesbrect, M. van Hoeij, I. S. Kotsireas, S. M. Watt. Towards factoring bivariate approximate polynomials. In: *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation* (ISSAC 2001), B. Mourrain, ed., pages 1-8. ACM Press, 2001.
5. R. M. Corless, M. W. Giesbrect, I. S. Kotsireas, S. M. Watt. Numerical implicitization of parametric hypersurfaces with linear algebra. Artificial Intelligence and Symbolic Computation (AISC 2000), J.A. Campbell and E. Roanes-Lozano, eds. Lecture Notes in Artificial Intelligence **1930**, pages 174-183. Springer-Verlag, 2001.
6. D. Cox, J. Little, D. O' Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Computer Algebra*. Undergraduate Texts in Mathematics. Springer-Verlag, 1992.
7. J. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1992.
8. H. Ferguson, D. Bailey, S. Arno. Analysis of PSLQ, an integer relation finding algorithm. *Mathematics of Computation* **68**(225):351-369, 1999.
9. J. von zur Gathen, J. Gerhard. Modern Computer Algebra. Cambridge University Press, 1999.
10. H. Hong, J. Schicho. Algorithms for trigonometric curves (simplification, implicitization, and parameterization). *Journal of Symbolic Computation***26**:279-300, 1998.
11. M. Kalkbrenner. Implicitization of rational parametric curves and surfaces. In AAECC8, *Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, S. Sakata, ed. Springer Lecture Notes in Computer Science **508**, pages 249-259. Springer-Verlag, 1990.

12. S.-H. Kim. Packing of ovals and an extremal problem for exponential sums. Talk presented in the Geometric Potpourri Seminar series, Department of Mathematics, University of Illinois at Urbana-Champaign, September 15, 1998. Abstract may be found at: http://www.math.uiuc.edu/Bulletin/Abstracts/September/sep11-98geompot.html
13. S.-H. Kim. Sums of Polynomials, Minmax Problems and Number Theory. Ph.D thesis, Department of Mathematics, University of Illinois at Urbana-Champaign, 2000.
14. S.-H. Kim. Planar packings and mappings related to certain minmax problems. *Mathematical Inequalities and Applications* **6**(2):351-374, 2003.
15. A. K. Lenstra. Polynomial factorization by root approximation. In EUROSAM 84, *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, J. Fitch ed. Lecture Notes in Computer Science **174**, pages 272-276, Springer-Verlag, 1984.
16. D. Lichtblau. Solving finite algebraic systems using numeric Gröbner bases and eigenvalues. In SCI2000, *Proceedings of the World Conference on Systemics, Cybernetics, and Informatics*. Volume 10, (Concepts and Applications of Systemics, Cybernetics, and Informatics). M. Torres, J. Molero, Y. Kurihara, and A. David, eds., pages 555-560. International Institute of Informatics and Systemics, 2000.
17. D. Lichtblau. Ordered heaps and fast marching method. 2003. Mathematica notebook available at: http://library.wolfram.com/infocenter/Demos/4928/
18. D. Lichtblau. Talk presented in the session on Symbolic-Numeric methods for Curves and Surfaces at the International Conference on Applications of Computer Algebra (ACA 2003), Raleigh, North Carolina, 2003. Abstract may be found at: http://www.risc.uni-linz.ac.at/people/shalaby/ACA03/2003.
19. D. Lichtblau. Cylinders through five points in $\mathbb{R}^3$. Submitted, 2004.
20. J. Sethian. *Level Set Methods and Fast Marching Methods: Evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science* (2nd edition). Cambridge University Press, 1999.
21. S. Wolfram. *The Mathematica Book* (5th edition). Wolfram Media, 2003.

Towards Solving the Dynamic Geometry Bottleneck Via a Symbolic Approach

Francisco Botana* and Tomás Recio**

Departamento de Matemática Aplicada I, Universidad de Vigo,
Campus A Xunqueira, 36005 Pontevedra, Spain
http://rosalia.uvigo.es/fbotana
Departamento de Matemáticas, Estadística y Computación,
Universidad de Cantabria, Santander, Spain
http://www.recio.tk

Abstract. The goal of this paper is to report on a prototype of a new dynamic geometry software, GDI (Geometría Dinámica Inteligente). We will describe how, apart from being a standard dynamic environment for elementary geometry, GDI addresses some key problems of the dynamic geometry paradigm, by including enhanced tools for loci generation and automatic proving, plus another distinguished feature, namely, a *discovery* option, allowing the user to find complementary hypotheses for arbitrary statements to become true. The key technique for all these improvements is the development of an automatic "bridge" between the graphic and the algebraic counterparts of the program (calling on an external computer algebra system).

1 Introduction

In the late eighties, two computer programs allowing dynamic changes in plane geometric constructions were simultaneously introduced: Cabri [20] and The Geometer's Sketchpad, GSP [16]. The key feature of this software is that unconstrained parts of the construction can be dragged on by the user and, as they move, all other elements in the construction automatically self–adjust, preserving mutual dependency relations and constraints [17]. Because of its evident impact in computer aided mathematics instruction, a specific name, i.e. dynamic geometry, was soon coined for programs presenting this kind of feature. Dynamic geometry software offers a virtual environment where accurate construction of geometric configurations can be carried out. Besides Cabri and GSP, Cinderella [28, 18] and Geometry Expert, GEX [11] can be also quoted as well known and performing dynamic geometry environments (see http://mathforum.org/dynamic/classroom.html for further information on different programs of this kind).

On the other hand, even in this summary introduction to the topic, we would like to highlight three problems (that are creating, perhaps, a real bottleneck

* Partly supported by grant MTM2004-03175 from the Spanish MEC and by a travel grant from Xunta de Galicia.
** Supported by grant BFM2002-04402-C02-02 from the Spanish MCyT.

H. Hong and D. Wang (Eds.): ADG 2004, LNAI 3763, pp. 92–110, 2006.

for future developments) that dynamic geometry software has to face in general: the so called continuity problem, the loci generation and the proof capability.

A frequent case in standard dynamic geometry environments occurs when a small move of a free object causes a sudden jump of a dependent element. Dynamic geometry developers and other researchers have paid a lot of attention to this aspect, concluding that continuity issues are not just a programming matter, but a central point, closely related to other topics such as the conservative or deterministic behavior of the dynamic setup (a dynamic geometry program is called conservative when it always happens that, departing from any configuration and after moving some free elements, returning them to the initial position yields always the starting configuration). See Section 2 below.

Regarding loci generation, most dynamic geometry environments heavily rely on a "sampling" approach [2] . Thus, in order to build up some geometric locus, they simulate dragging an element of a construction, usually a point, and then –by considering many instances of the construction as the point is dragged on– the program automatically generates the path of some other element which is supposed to be activated by the displacement of the first object. This approach, quite successful in many cases, requires, first, to construct an instance of the geometric locus and, thus, it makes difficult or impossible the computation of loci defined through conditions that, albeit reasonable, the user does not know how to exemplify. For example, given a circle with center O and going through another fixed point U, suppose we want to compute the locus of points B such that they are in the bisector of the points O, A (where A belongs to the given circle) and such that the segments OB and AB are perpendicular (see Fig. 6). Then, else we know how to construct such a point B (recalling a property of euclidean geometry that states that the perpendicularity of OB and AB implies point B is on the circle with diameter OA), or we will be unable to use the locus generation tool in most Dynamic Geometry programs. Even worse, in the "sampling" approach, the loci equations, in general, are not provided by the machine for further computation purposes. See Section 3 for details.

Finally, a common use of dynamic geometry environments such as Cabri and GSP, in elementary geometry teaching and learning, involves an activity that has been termed as 'visual proving' (of properties and theorems). The numerical accuracy of the constructions and the possibility to experiment with different instances (by dragging the basic geometric objects in a given statement) get the user convinced about the truth or falsity of some conjecture. Partly reacting to this approach, last generation programs such as Cinderella, GEX and GEOTHER [29] start including formal tools for automatic geometric theorem proving. But they are of little help to proceed any further once the user is informed about the falsity of some conjectural statement. See Sections 4 and 5 for new findings in this respect.

In this context, the goal of this note is to provide some news on a prototype of a recent program, GDI [3, 4], a Spanish acronym of Intelligent Dynamic Geometry, freely distributed at request, from the first author of this paper. We will describe how, apart from being a standard dynamic geometry environment,

GDI includes enhanced tools for loci generation and automatic proving, plus another distinguished feature, namely, a *discovery* option, allowing the user to find complementary hypotheses for arbitrary statements to become true, or, in other words, to find the missing hypotheses so that a given conclusion follows from a given (perhaps drastically) incomplete set of hypotheses.

The key technique for all these improvements is the development of an automatic "bridge" between the graphic and the algebraic counterparts of the program (CoCoA [5] and Mathematica), as proposed in [26, 23].

2 Continuity

Roughly speaking, the problem of continuity appears when performing –by dragging– small changes in some element of a construction involve sudden changes in some other parts of the construction, as it happens in the paradigmatic construction shown in Fig. 1 (taken from [18], p. 88): A circle (several positions drawn with thin lines) moves from left to right through another circle (one position drawn with thick line) of the same radius. On the left figure, the expected continuity behavior, the one would like to see: the selected intersection point of the two circles stays above the horizontal line while dragging the moving circle. On the right figure, the chosen intersection point suddenly jumps. It should be remarked that the mathematical theory behind Cinderella has solved this problem, as explained in [18], [19] but it is not the case of Cabri or GSP (see [13], p. 139, for a detailed analysis of an example with Cabri).

The continuity problem is not simply an issue about finding the right implementation strategies for visualization: it is mathematically involved and it is related to the so called "conservative" or "deterministic" behavior (namely, that for any concrete position of the base points in a construction, the position of all the constructed elements should be uniquely determined) of the dynamic geometry software, see [21], [8]). An example of a non-deterministic situation is shown in Fig. 2 (taken from [8]): the incenter I of a triangle is constructed, with Cinderella, for a triangle ABC. Now in Fig. 2, left and right, two different positions of the output point (the incenter) are shown, both corresponding to the *same* positions of the vertices of the triangle: each one can be obtained from the other by merely dragging around (in some way) one of the vertices, and then returning it to the original position. To make things more tangled, it is known that continuity and deterministic principles can not simultaneously hold, for mathematical reasons, on a dynamic geometry software ([8], [13]).

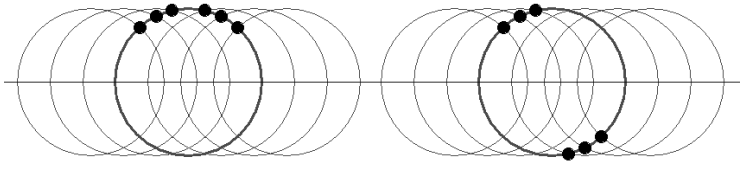


Fig. 1. A case of discontinuity

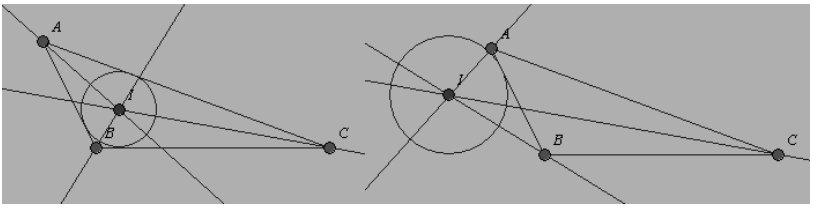


Fig. 2. An example of non deterministic behavior

The symbolic approach taken by GDI to represent internally geometric constructions gives a way of circumventing the continuity/deterministic issue. Besides including some heuristics that avoid discontinuity in the visualization of some constructions (but that fail in some other cases), GDI favors (when the user aims to perform proving or discovering tasks) to keep construction intermediate steps (such as the intersection of two circles) in a raw form, as a system of equations, without aiming to solve this system until it is unavoidable for obtaining the output (i.e. performing a kind of lazy evaluation).

For instance, in order to display the geometric locus of points such that the product of their distances to two given fixed points is constant, one may proceed by constructing, first, two non-oriented segments BD, AC such that the product of their lengths is equal to a fixed quantity. One of the segments has a free endpoint, say D, on the line BD. Then two base points F_1, F_2 are constructed and two circles around the fixed points, with radius equal, respectively, to the length of AC or BD. Finally, one searches for the locus of points that are simultaneously in both circles, when D moves on the line BD.

Now Cabri and Cinderella output (according to the different relative positions of F_1, F_2) the graphics shown in the figures (see Fig. 3 for an output from Cinderella or Fig. 4 for Cabri, in a different position). Their behavior is similar: Cabri requires the user to select both points on the intersection of the two circles to achieve just one oval of Cassini, while Cinderella outputs the same single oval selecting just one of the intersection points (but it does not find the two ovals

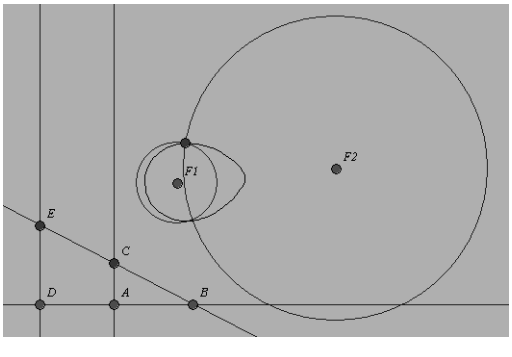


Fig. 3.

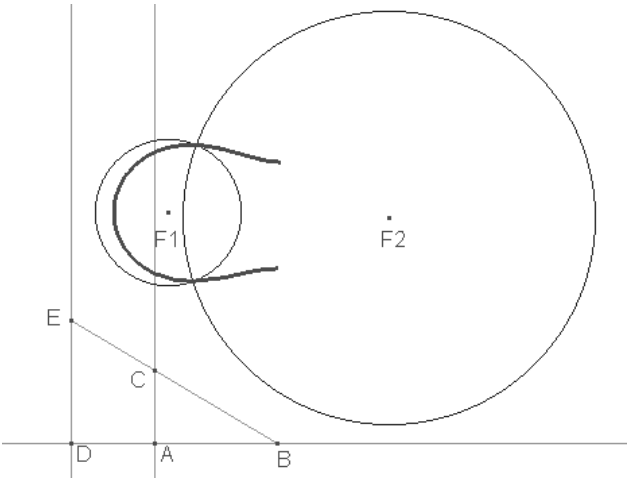


Fig. 4.

even if the two intersection points are selected). This is due to the limitations of their approaches concerning the compromise between continuity/determinism, which in the case of Cinderella implies considering complex coordinate solutions for the locus point, constrained, by construction, by several algebraic conditions. Thus, in Cinderella, its strategy of returning loci as the positions only accessible by real continuous moves where the dependent point has real valued coordinates, sometimes avoids their correct generation. See ([18], p. 137) for details.

On the other hand GDI obtains the actual full locus, the two Cassini ovals, since it works at the symbolic level, transforming the input data into a set of algebraic equations, then performing some elimination of the dependent variables and, finally, displaying them via an implemented connection to a standard computer algebra package, see Fig. 5. It is rather simple, but standing on the shoulders of giants.

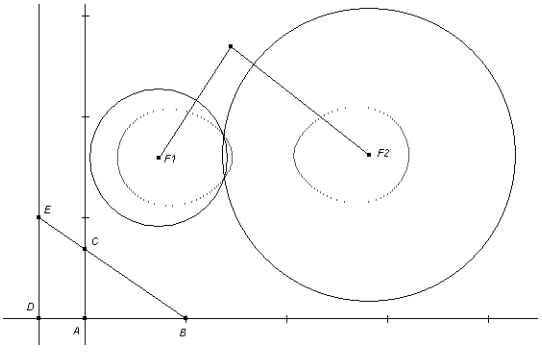


Fig. 5.

3 Loci Discovery

In the sequel, to describe a geometry problem by a finite set of polynomials, we will only consider problems involving incidence, congruence and parallelism relations. Given a geometric construction and a distinguished point, in order to look for its locus with respect to some conditions, GDI begins by translating the geometric properties into algebraic expressions, after selecting a coordinate system. We use the field of rational numbers Q and C, the field of complex numbers, as an algebraically closed field containing the former. The collection of construction properties is then expressed as a set of polynomial equations

$$p_1(x_1, \ldots, x_n) = 0, \ldots, p_r(x_1, \ldots, x_n) = 0,$$

where $p_1, \ldots, p_r \in Q[x_1, \ldots, x_n]$. Thus, the affine variety defined by $V = \{p_1 = 0, \ldots, p_r = 0\} \subset C^n$ contains all points $(x_1, \ldots, x_n) \in C^n$ which satisfy the construction requirements, that is, the set of all common zeros of $p_1, \ldots, p_r$ in the n-dimensional affine space of C describe all the possible positions of the construction points. In particular, the positions of the locus point define the locus we are searching for. Thus, supposing that the locus point coordinates are x_{n-1}, x_n, the projection

$$\pi_{n-2} : V \subset C^n \to C^2$$

gives an extensional definition of the locus in the affine space C^2. This projection can be computed via I_{n-2}, the $(n-2)$th elimination ideal of $\langle p_1, \ldots, p_r \rangle$. The Closure theorem states that $V(I_{n-2})$ is the smallest affine variety containing $\pi_{n-2}(V)$, or, more technically, that $V(I_{n-2})$ is the Zariski closure of $\pi_{n-2}(V)$. So, except some missing points that lie in a variety strictly smaller than $V(I_{n-2})$, we can describe the locus computing a basis of I_{n-2}. This basis is computed as follows: given the ideal $\langle p_1, \ldots, p_r \rangle \subset Q[x_1, \ldots, x_n]$, let G be a Gröbner basis of it with respect to lex order where $x_1 > x_2 > \ldots > x_n$. The Elimination theorem states that $G_{n-2} = G \cap Q[x_{n-1}, x_n]$ is a Gröbner basis of I_{n-2}.

This approach can be illustrated with two different examples, each one addressing the two issues we have mentioned in the introduction: loci generation for points we do not know how to construct and reutilization of loci for further computations within GDI. First we will consider finding the locus of points B, given a circle with center O and going through another fixed point U, such that B is in the bisector of the points O, A (where A belongs to the given circle) and such that the segments OB and AB are perpendicular (see Fig. 6).

In GDI, once points O, U are defined (and, in order to simplify the exposition, taken as $(0, 0)$ and $(1, 0)$, respectively), a semi-free point A is constructed belonging to the given circle. Then a new free point B is introduced, and, through the Discovery menu, we impose two conditions on B, namely, the equality and perpendicularity of OB and AB. A set of equations is created (GDI automatically assigns indeterminate coordinates to the given points, enumerate the different components of the construction, and translates the constraints into algebraic

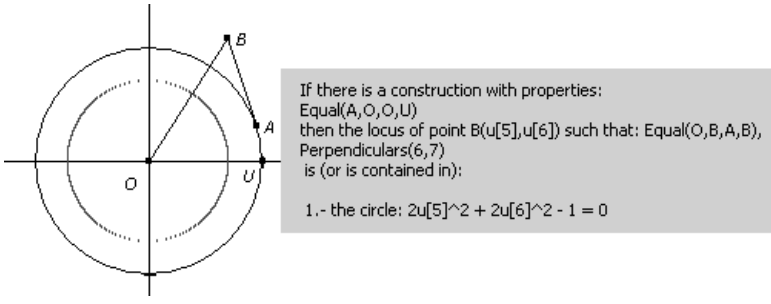


Fig. 6. Finding the locus of B

equations) and then GDI interrelates with CoCoA or Mathematica (at user's choice) or via web (see below Section 6), and proceeds to eliminate all variables but those of B. The result is, correctly, interpreted by GDI as a circle, centered at the origin and of radius $\sqrt{(2)}/2$.

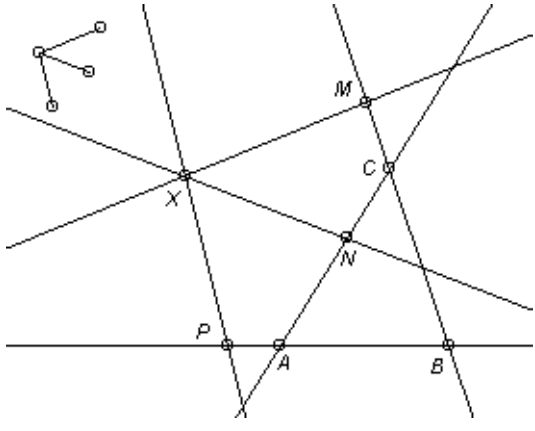


Fig. 7. The construction for Giering-de Guzmán's theorem

A less trivial example of the above procedure could be using GDI to discover a recent generalization of Wallace–Steiner theorem [14, 12]: Given a triangle ABC and three directions, not all three equal, nor parallel to the triangle sides, find the locus of points X such that its projections M, N, P along the three directions determine a triangle of oriented area k (Fig. 7). Once the geometric construction is done, the user imposes the condition that the oriented area of MNP is, say, 1, area$(M, P, N) = 1$. The geometric predicates are automatically translated by GDI into polynomials, and CoCoA is used (in the background) to perform the elimination task. Finally, the locus equation is returned to the dynamic environment, where the curve is plotted (Fig. 8), yielding a conic, as stated by Giering-de Guzmán's theorem.

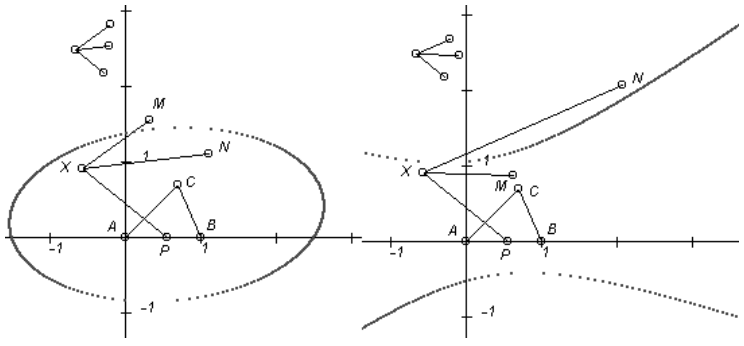


Fig. 8. An ellipse $(43890x2 - 16139xy + 143719y2 - 43890x - 76139y - 165360 = 0)$ and a hyperbola $(64470x2 + 87313xy - 521163y2 - 64470x + 323036y + 532680 = 0)$ in Giering–de Guzmán's theorem

If required by the user, the same method can be applied to obtain the equation of the corresponding conic for generic input data. The output in this case is not displayed here because of its length. Note that since the locus equation is now known by the system, this new object can be used to construct new objects. For instance, if the imposed condition was the alignment of M, N, P, GDI could compute the envelope of lines MNP, a tricuspidal hypocycloid (Fig. 9).

Finally notice that, since we are working on the plane and with complex geometry, the locus set will be, in general, either a curve or a finite set of points. In the former case, the elimination ideal will be principal and, then its Gröbner basis will have just one element. Otherwise, all equations appearing in the G–basis will be required to describe the locus: for instance, one can construct the

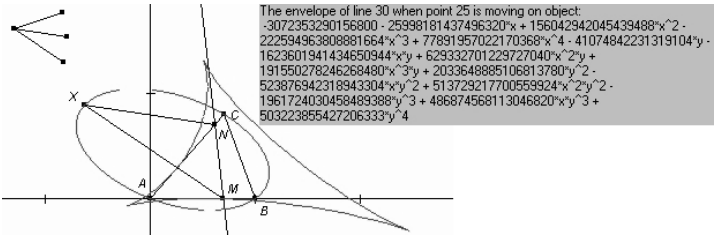


Fig. 9. An envelope of lines MNP

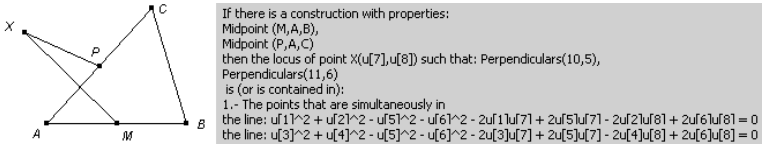


Fig. 10. The equations of the circumcenter

circumcenter of a triangle as the locus of points equidistant to the three vertices. Obviously, the answer by GDI is not a single equation, but a pair of equations on the coordinates of the circumcenter, with coefficients that are polynomials in the indeterminate coordinates of the vertices. The intersection of the two curves defined by these equations yields, of course, the circumcenter (Fig. 10).

4 The Algebraic Approach to Automatic Theorem Proving

For the last 20 years, symbolic algebraic techniques have been successfully used for automatically proving theorems in elementary geometry (see [31] for an exhaustive repository of related papers). The practical interest of this goal (i.e to automate, through the algebraic translation of hypotheses and theses, theorem proving) could be related, for instance, to its potential applications in geometric constraint solving and parameterized CAD [15, 9, 10].

The algebraic approach roughly proceeds as follows. A geometric statement (a finite set of hypotheses and a thesis) is translated into two multivariate polynomial systems, H, T. The statement is declared to be true if the hypotheses variety is contained in the thesis variety, $Var(H) \subseteq Var(T)$. Different approaches exist to test this inclusion in commutative/algebraic geometry, mainly Wu–Ritt characteristic sets and Gröbner bases. Moreover, it usually happens that the inclusion does not happen because of some small set of points; in this case the algebraic approach takes care of detecting these degenerate cases that should be removed from the hypothesis. Both methods work in an algebraically closed field, so the decision about the truth of a geometric statement involves not only real solutions, but the complex ones also.

In GDI, following this algebraic approach, and through the cooperation of the graphic environment with a symbolic computation program (such as Mathematica or CoCoA, at user's choice), the user can state and prove (opening a suitable Menu) or disprove different conjectures on a given construction. As in the sections above, in GDI the construction steps (for the hypothesis) and the thesis condition (chosen from a Menu) generate a file with algebraic equations, which is then exported to the symbolic computation package, where it is subject to some standard algebraic manipulation for theorem proving (see [26], [25] or [1]). The output of the computation is visualized in a dedicated window, with some indications about the validity of the theorem and of the degeneracy conditions.

Below we provide one example of this procedure. It concerns Pappus theorem (see Fig. 11), where the thesis (of alignment of the three selected points P, Q, R) is imposed clicking on a Menu command.

The output window shows the theorem is true under the disjunction of three non-degeneracy conditions; it must be read as $u_2u_3 \neq 0 \vee u_1u_4 - u_4 \neq 0 \vee u_2u_4 \neq 0$, where u_i are coordinates of some of the basic points of the construction. They are labeled by GDI according to the order in which the points are introduced and following some other simple rules (such as the u_i's represent free variables, the x_j's are dependent ones, etc.) that can be learned from GDI on–line manual.

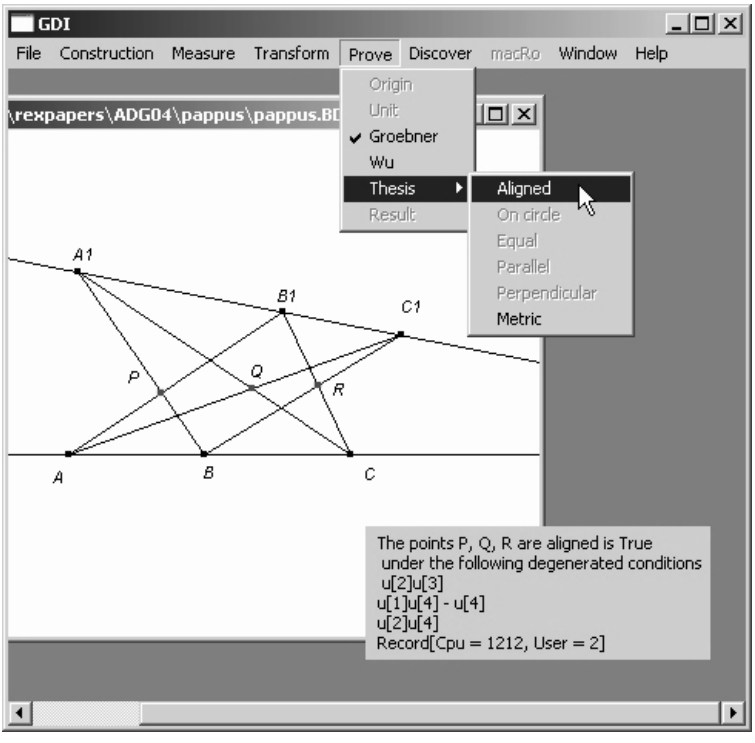


Fig. 11. Proving Pappus theorem

It is not important, therefore, to detail here the geometric meaning of these conditions, since they are dependent on the order the construction has been made in this particular case.

5 Discovering Through GDI

It is relevant to remark that GDI distinguished feature is that of *discovering* complementary hypothesis, but not only for degenerate cases, when we have a nearly true statement, but in a *generally false statement*, following the approach of [26], where we send the reader for the algebraic theory behind it. Again, as in the case of Proving, the user must perform the corresponding construction in GDI and, then, manipulating the Discovery Menu, the user establishes a conjecture on some elements of the construction. A file with the corresponding algebraic data is sent to the symbolic computation engine associated to GDI and, finally, a window with the result (additional hypothesis plus non-degeneracy conditions) opens up.

In the first example (see Fig. 12), taken from [26], it is stated that the symmetrical images of a point, with respect to the three sides of a triangle, are on a line. It is obviously false, but GDI finds, through algebraic manipulation, that

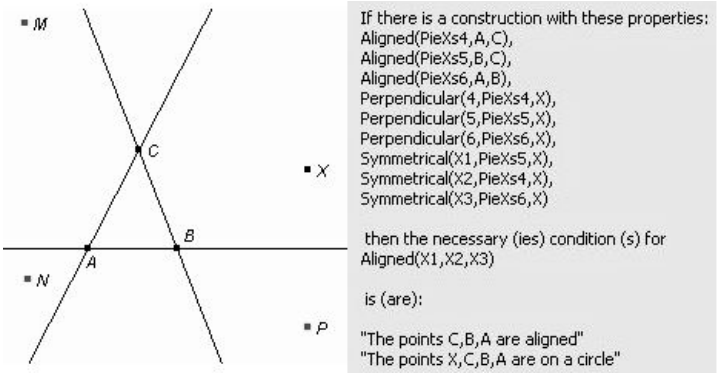


Fig. 12. Discovering conditions for the alignment of the symmetrical points

it is true if the given point lies on the circle passing through the three vertices of the triangle. It is not easy to guess this result without GDI's help!

The following two examples come from mathematics education, more precisely, from the recent book by Richard ([27], on proof strategies in mathematics courses, for students 14-16 years old. Richard's book reports the results of a five questions test (concerning proving strategies) which has been passed to a number of students. We have considered questions 1 and 3 as the more suitable both for GDI characteristics (since questions 2 and 4 had some numerical coordinates which where quite essential) and for exploring its performance on tasks that have not been directly conceived for computer assisted instruction (since question 5 was already designed to be worked out by means of a Dynamic Geometry software). Question 1 presents the following situation (we are summarizing, for our purposes, very much the content of the test): A circle of diameter AB, its bisector d, a point X in the intersection of d and the circle. It asks the student to consider which of the following statements (can be more than one) concerning triangle ABC are correct and why.

- The construction is impossible
- The triangle is isosceles
- The triangle is equilateral
- It is a right triangle
- None of the above

GDI concludes, via Discovery, that the construction is possible, that the triangle is isosceles and that it has at vertex X a right angle (Fig. 13). If we conjecture the triangle is equilateral, GDI answers, as well, that this is not correct, see Fig. 14.

Of course, as it is in general the case with the algebraic approach to theorem proving, GDI does not help much with providing a student-readable proof of the result. But it helps, at least, with solving the different conjectures that one might be considering during the heuristic phase of proving. Question 3 presents

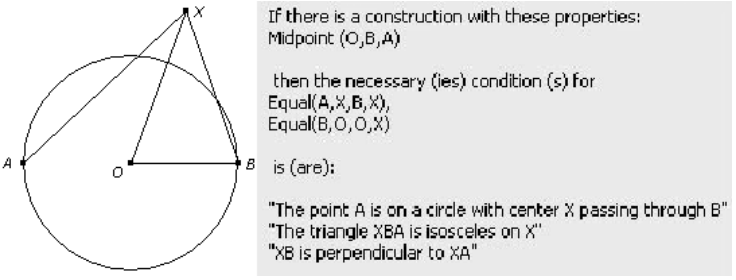


Fig. 13.

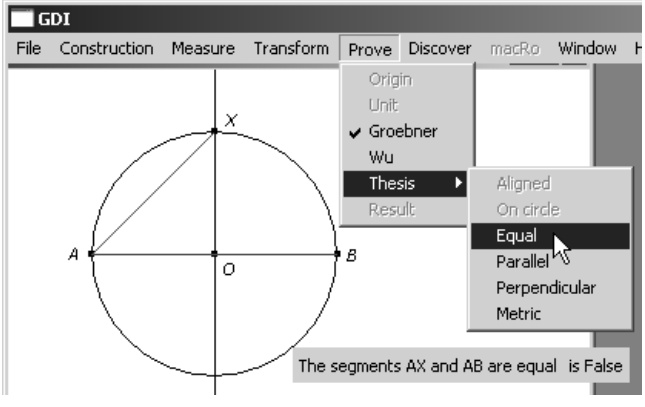


Fig. 14. The triangle ABX is not equilateral

a classical pseudo-construction (see Fig. 15): an isosceles triangle ABC, where AB, AC are the two equal sides, and assumes that the two bisectors of angles ABC and ACB intersect perpendicularly at an interior point X. Again, it is required that the student considers the same set of possible answers as above.

Here, GDI immediately shows that the construction is not possible under these hypotheses, stating that there are no point in the locus ("No hay puntos en el lugar").

All these are quite simple examples, performed on a basic lap-top within a few seconds. Their interest is, precisely, to show that our approach is successful in many cases that appear naturally at school level, where GDI could be a helpful tool for guessing and conjecturing (although not for providing conviction or human acceptable explanations).

A final example of GDI discovery possibilities concerns a formula of Euler on the relation between the radii of the incircle and the excircle of a triangle, and the distance between their centers. It is a case where it is very difficult (except for Euler!) to guess such a formula. This problem has been approached by Wang and Zhi [30], but for automatic proving, giving explicitly the formula of Euler (arguably named Poncelet's

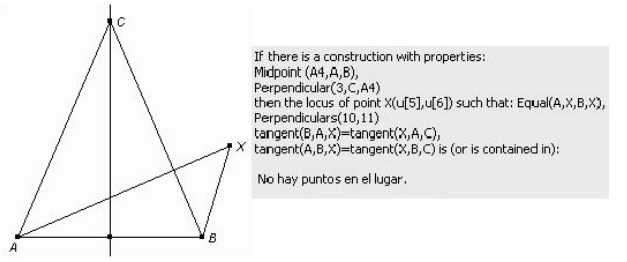


Fig. 15. The construction for Question 3

theorembythem,seepagesmathworld.wolfram.com/EulerTriangleFormula.htmlor enriques.mathematik.uni–mainz.de/intgeo/poncelet.html) and requiring to prove its validity.

Here we proceed with GDI by constructing the center of the circumcircle as the intersection of two perpendicular bisectors of two sides of the triangle, then constructing the center of the incircle as a point equidistant to the feet of its orthogonal projections over the three sides of the triangle. Finally, we ask GDI to discover whatever relation it might exist among the variables representing the coordinates of the circumcenter, the incenter and the vertices of the triangle.

In order to simplify the computation we have chosen (this is an option of GDI) two vertices of the triangle to be the origin and the unit point in the x-axis, and GDI assigns then (u_5, u_6) as coordinates for the third vertex, (u_7, u_8) as coordinates for the incenter, and $(u_9, u_1 0)$ for the circumcenter. The result (via webDiscovery, see next Section) is displayed in the figure below.

Fig. 16. Trying to discover Euler formula

Thus, if the triangle is not degenerate, we have obtained a collection of eight polynomial conditions among $u_5 \dots u_{10}$, plus the condition $u_6 \neq 0$. Now it is just a matter of rewriting these nine conditions in terms of the distance between the two centers and the two radii. Currently it is not possible to automatically perform such human-driven interpretation of the output, but GDI provides, as well, a file with the algebraic output, so it is quite easy to perform (by the user) the following elimination, yielding Euler formula (generalized in the sense that it applies to all different incenters, and not just to the one interior to the triangle).

```
Use R::=Q[hu[5..10]drc];

Elim(h..u[10],Ideal(h u[6]-1,
d^2-((u[9]-u[7])^2+(u[10]-u[8])^2),
r-u[8],
c^2-(u[9]^2+u[10]^2),
2u[9] - 1,
-u[5]^2 - u[6]^2 + 2u[5]u[9] + 2u[6]u[10],
-u[5]u[6]u[7] + 1/2u[6]u[7]^2 - u[6]^2u[8] - 1/2u[6]u[8]^2 +
2u[6]u[8]u[10] + 1/2u[5]u[6],
u[6]^2u[7] - u[5]u[6]u[8] - u[6]u[7]u[8] - 1/2u[6]^2 + u[6]u[8],
1/4u[6]u[7]^2u[8] + 1/4u[6]u[8]^3 - u[6]u[8]^2u[10] -
1/4u[6]u[7]u[8] + 1/8u[6]^2 - 1/4u[6]u[8],
1/2u[6]u[7]^3 + 1/2u[6]u[7]u[8]^2 - 2u[6]u[7]u[8]u[10] -
3/4u[6]u[7]^2 - 1/4u[6]u[8]^2 + u[6]u[8]u[10] + 1/4u[5]u[6],
1/2u[6]u[8]^4 + u[6]^2u[8]^2u[10] - 2u[6]u[8]^3u[10] -
1/8u[6]^3 + 1/2u[6]^2u[8] - 1/2u[6]u[8]^2,
1/2u[6]u[7]u[8]^3 + u[5]u[6]u[8]^2u[10] - u[6]u[7]u[8]^2u[10]
- 1/4u[6]u[8]^3 - 1/8u[5]u[6]^2 +3/8u[5]u[6]u[8] -1/8u[6]u[7]u[8]
+ 1/16u[6]^2 - 1/8u[6]u[8]));

Ideal(1/2d^4 - d^2c^2 - 2r^2c^2 + 1/2c^4)
-------------------------------
```

6 webDiscovery

The GDI symbolic algorithms have been used to develop an Internet application. *webDiscovery* (http://rosalia.uvigo.es/sdge/web/2D) is an open web–based tool for automatic discovery in elementary Euclidean geometry. It accepts user–defined geometric constructions, which are uploaded to a Java Servlet server, where two computer algebra systems, CoCoA and Mathematica, return the discovered facts about the construction. As a simple illustration, we consider a triangle ABC and its circumcircle O. In order to discover the necessary conditions for the (obviously false in general) collinearity of B, O and C, the user makes a construction with GDI, outputs a text file with the algebraic information, such as

```
Points
C(u[1],u[2])
B(1,0)
A(0,0)
D(x[1],x[2])
E(x[3],x[4])
O(x[5],x[6])

LingProperties
Midpoint(D,B,A)
Midpoint(E,C,A)
Perpendicular(B,A,O,D)
Perpendicular(C,A,O,E)

LingConditions
Aligned(B,O,C)

DiscProperties
```

and then uploads it through the web. Then webDiscovery returns, via web, a page containing the conditions for the collinearity, that is, the rightness of the angle BAC and a degenerated condition (Fig. 17).

A final example deals with the celebrated MacLane 8_3 theorem [22], that can be stated as follows: *Consider eight points A, B,..., H such that the following (also*

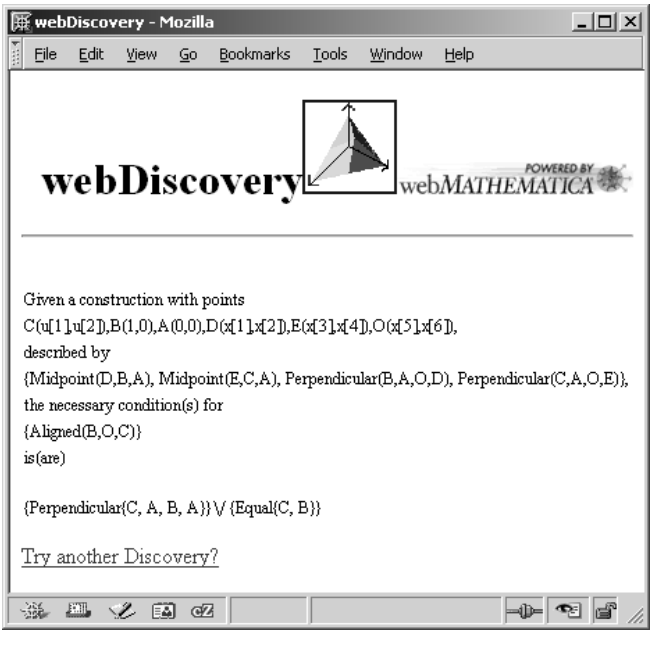


Fig. 17. The necessary conditions for the alignment of the circumcenter

eight) triples are collinear ABD, BCE, CDF, DEG, EFH, FGA, GHB, HAC. *Then all eight points lie on a line.* This is a controversial result: Dolzmann, Sturm and Weispfenning [7] point out, without further discussion, that this theorem holds in the real plane, but fails in the complex one, while Conti and Traverso [6] consider it as "an example of an obviously false theorem that is true". Wang and Zhi [30] also consider it as a false theorem both over the complex and the real numbers. Leaving aside reality issues, i.e. considering just the complex case (as it is usual up to now in most theorem provers), we have attempted to verify with GDI the truth/falsity of this result. But it is rather difficult to approach it via the Prover, because it is precisely a theorem of non-constructible type (one can not make a construction without actually placing all eight points on a line). See [24] for further details on the solution of this theorem with GDI, showing that it is much more feasible to study this theorem using the automatic Discovery feature of GDI.

Since this approach consists of imposing some extra, hypothetical, conditions on a construction, we can sketch the construction as above, namely, we construct two free points $A(u_1, u_2)$ and $B(u_3, u_4)$ which define a line containing the point $D(u_5, x_1)$. A new free point $C(u_6, u_7)$ is used to define line BC containing the point $E(u_8, x_2)$. Analogously, the line CD contains the point $F(u_9, x_3)$, the line DE contains the point $G(u_{10}, x_4)$, and the line EF contains the point $H(u_{11}, x_5)$.

There are eleven free variables in the above construction: A, B and C each contribute with two new variables, while the other five points give only one free coordinate each. Then the last three alignment conditions are imposed on

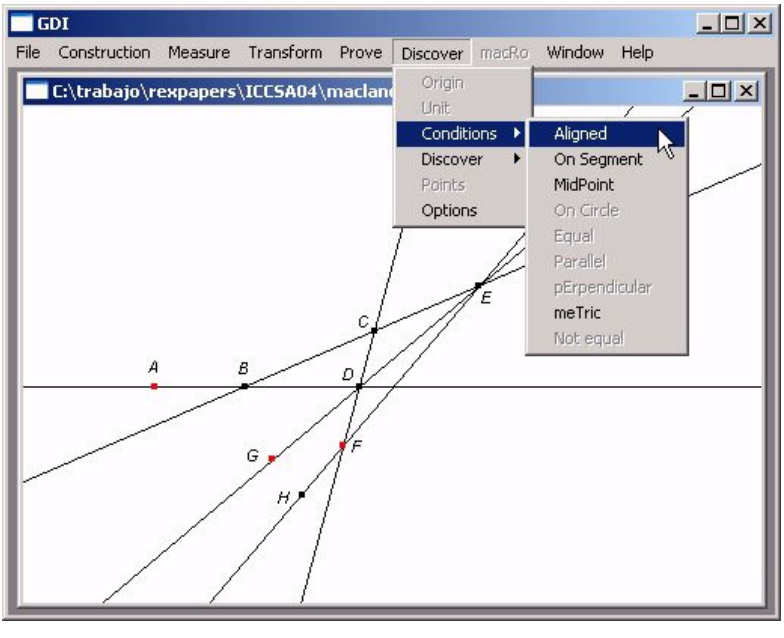


Fig. 18. Imposing the alignment of F, G and A in GDI

the sketch by selecting the corresponding triple of points and declaring them as collinear (Fig. 18).

For symbolic purposes, the sketch is internally (in GDI) translated as follows, where A and B have been declared in the construction as the origin and the x-axis unit point, without loss of generality and in order to reduce the number of involved variables:

```
Points
A(0,0)
B(1,0)
C(u[6],u[7])
D(u[5],x[1])
E(u[8],x[2])
F(u[9],x[3])
G(u[10],x[4])
H(u[11],x[5])

Properties
Aligned(D,A,B) (0-x[1])*(1-0)-(0-0)*(0-u[5])
Aligned(E,B,C) (0-x[2])*(u[6]-1)-(u[7]-0)*(1-u[8])
Aligned(F,D,C) (x[1]-x[3])*(u[6]-u[5])-(u[7]-x[1])*(u[5]-u[9])
Aligned(G,D,E) (x[1]-x[4])*(u[8]-u[5])-(x[2]-x[1])*(u[5]-u[10])
Aligned(H,E,F) (x[2]-x[5])*(u[9]-u[8])-(x[3]-x[2])*(u[8]-u[11])

Conditions
Aligned(F,G,A) (x[4]-x[3])*(0-u[10])-(0-x[4])*(u[10]-u[9])
Aligned(G,H,B) (x[5]-x[4])*(1-u[11])-(0-x[5])*(u[11]-u[10])
Aligned(H,A,C) (0-x[5])*(u[6]-0)-(u[7]-0)*(0-u[11])
```

Let I be the ideal automatically generated by the eight polynomials (properties + conditions) in the ring $R = Q[x_1, \ldots, x_5, u_5, \ldots, u_{11}]$. GDI proceeds in a mechanic way (at user's request) towards the discovery of properties induced on the eight points configuration after imposing the last three conditions. This is (roughly) done by eliminating the (construction declared) bounded variables x_i in order to find necessary conditions expressed by means of the supposedly free variables. This elimination ideal, computed with CoCoA in a few seconds, is generated by 28 polynomials, say $p_i, i = 1, \ldots, 28$.

The vanishing of any of these polynomials, involving just the variables $u_5, \ldots, u_{11}$, is a mere consequence of the proposed hypotheses and gives some necessary conditions for the configuration to verify the construction and the extra conditions. Now the surprise comes when we realize that the *gcd* of these polynomials is u_7. But $u_7 = 0$ gives the collinearity of the eight given points! In fact, $u_7 = 0$ expresses that point C is aligned with points A, B. Let us call $q_i, i = 1, \ldots, 28$ the polynomials such that $u_7 * q_i = p_i$. Therefore, if all the polynomials in I vanish (i.e. if we are on the hypothesis variety) and some $q_i \neq 0$, then $u_7 = 0$, that is, the theorem is true over the open set of the hypothesis variety described by the union of $q_i \neq 0, i = 1, \ldots, 28$.

Finally, as a personal and non–automatic remark, we might observe that $u_5, u_6, u_8, \ldots, u_{11}$ is a maximal set of 6 independent variables modulo I, and that performing some algebraic operation (saturation), we get that $Sat(I, Ideal(u_7)) \cap Q[u_5, u_6, u_8, \ldots, u_{11}]$ is not zero (for instance, it contains $q_1 = -u_5u_6u_8u_{11} + u_5u_6u_{10}u_{11} + u_5u_6u_8 - u_6u_8u_{10} + u_6u_8u_{11} - u_5u_{10}u_{11} - u_6u_{10}u_{11} + u_8u_{10}u_{11} - u_5u_6 + u_6u_{10} + u_5u_{11} - u_8u_{11}$) which implies [25] that the theorem holds over all components of the hypothesis variety where these variables (of number equal to the dimension of the variety) remain independent; this fact (for some of us: it depends on the different concepts of truth in theorem proving [6]) supports calling this a *generally true theorem*.

7 Conclusions

Perhaps the most astonishing fact is that GDI performs quite well avoiding some problems derived from continuity issues, opening new possibilities for loci generation and for proving and discovering new results, from trivial facts to rather complicated explorations in fairly open contexts –as we hope to have exhibited through the many examples roughly described in this paper. But we must humbly admit that this is mainly due to the high performance of the current algebraic elimination engines and to the simple, but careful connection of the graphic and symbolic engines. Currently we work towards enhancing GDI and webDiscovery with new features, such as the capability to export/accept geometric constructions coded in OpenMath.

References

1. Bazzotti, L., Dalzotto, G., Robbiano, L.: Remarks on geometric theorem proving. *Proc. 3rd Int. Workshop on Automated Deduction in Geometry* (ADG 2000), LNAI, 2061, 104–128 (2001)
2. Botana, F.: Interactive versus symbolic approaches to plane loci generation in dynamic geometry environments. *Proc. I Int. Workshop on Computer Graphics and Geometric Modelling* (ICCS 2002), LNCS, 2330, 211–218 (2002)
3. Botana, F., Valcarce, J.L.: A software tool for the investigation of plane loci. *Mathematics and Computers in Simulation*, 61(2), 141–154 (2003)
4. Botana, F., Valcarce, J.L.: Automatic determination of envelopes and other derived curves within a graphic environment. *Mathematics and Computers in Simulation*, 67(1–2), 3–13 (2004)
5. Capani, A., Niesi, G., Robbiano, L.: CoCoA, a system for doing Computations in Commutative Algebra. Available via anonymous ftp from: cocoa.dima.unige.it
6. Conti, P., Traverso, C.: Algebraic and semialgebraic proofs: Methods and paradoxes. *Proc. 3rd International Workshop on Automated Deduction in Geometry* (ADG 2000), LNAI, 2061, 83–103 (2001)
7. Dolzmann, A., Sturm, A., Weispfenning, V.: A new approach for automatic theorem proving in real geometry. *Journal of Automated Reasoning*, 21, 357–380 (1998)
8. Gawlick, T.: Towards a theory of visualization by dynamic geometry software. Paradigms, phenomena, principles. *Int. Conf. on Mathematical Education 10* (ICME 10). Topic Study Group 16. http://www.icme-organisers.dk/tsg16/papers/Gawlick.theoryofvisDGS.pdf

9. Gao, X.S., Chou, S.C.: Solving geometric constraint systems. I. A global propagation approach. *Computer–Aided Design*, 30, 47–54 (1998)
10. Gao, X.S., Chou, S.C.: Solving geometric constraint systems. II. A symbolic approach and decision of Rc–constructibility. *Computer–Aided Design*, 30, 115-122 (1998)
11. Gao, X.S., Zang, J.Z., Chou, S.C: Geometry expert (Nine Chapters Publ., Taiwan, 1998)
12. Giering, O.: Affine and projective generalization of Wallace lines. *Journal of Geometry and Graphics*, 1(2), 119–133 (1997)
13. González López, M.J.: Using dynamic geometry software to simulate physical motion. *International Journal of Computers for Mathematical Learning*, 6(2), 127–142 (2001)
14. de Guzmán, M.: An extension of the Wallace–Simson theorem: projecting in arbitrary directions. *American Mathematical Monthly*, 106(6), 574–580 (1999)
15. Hoffmann, C., Bouma, W., Fudos, I., Cai, J., Paige, R.: A geometric constraint solver. *Computer–Aided Design*, 27, 487–501 (1995)
16. N. Jackiw, The Geometer's Sketchpad (Key Curriculum Press, Berkeley, 1997)
17. King, J., Schattschneider, D.: Geometry turned on (Mathematical Association of America, Washington DC, 1997)
18. Kortenkamp, U.: Foundations of dynamic geometry, Ph.D. Thesis, ETH, Zurich, 1999.
19. Kortenkamp, U., Richter-Gebert, J.: A dynamic setup for elementary geometry. *Proc. Multimedia Tools for Comunicating Mathematics* (MTCM 2000), Springer-Verlag (2001)
20. Laborde, J.M., Bellemain, F.: Cabri Geometry II (Texas Instruments, Dallas, 1998)
21. Laborde, J.M.: Some issues raised by the development of implemented dynamic geometry as with Cabri-Geometre. *Proc. 15th European Workshop on Computational Geometry*, 7–19 (1999)
22. MacLane, S.: Some interpretation of abstract linear dependence in terms of projective geometry. *American Journal of Mathematics*, 58, 236–240 (1936)
23. Recio, T.: Cálculo simbólico y geométrico (Síntesis, Madrid, 1998)
24. Recio, T., Botana, F.: where the truth lies (in automatic geometry theorem proving). *Proc. International Conference on Computational Science and Its Applications* (ICCSA 2004), LNCS, 3044, 761–770 (2004)
25. Recio, T., Sterk, H., Vélez, P.: Project: Automated Geometry theorem proving. In A.M. Cohen, H. Cuypers and H. Sterk, editors, *Some Tapas of Computer Algebra* (Algorithms and Computations in Mathematics, Vol. 4), Springer, Berlin, 276–296 (1999)
26. Recio, T., Vélez, M. P.: Automatic discovery of theorems in elementary geometry. *Journal of Automated Reasoning*, 23, 63–82 (1999)
27. Richard, P.: Raisonnement et stratégies de preuve dans l'enseignement des mathématiques (Peter Lang, Berne, 2004)
28. Richter–Gebert, J., Kortenkamp, U.: The interactive geometry software Cinderella (Springer, Berlin, 1999)
29. Wang, D.: GEOTHER: A geometry theorem prover. *Proc. 13th International Conference on Automated Deduction* (CADE 1996), LNCS, 1104, 166–170 (1996)
30. Wang, D., Zhi, L.: Algebraic Factorization Applied to Geometric Problems. *Proc. Third Asian Symposium on Computer Mathematics* (ASCM '98), Lanzhou University Press, Lanzhou (China), 23–36 (1998)
31. http://www-calfor.lip6.fr/ wang/GRBib/Welcome.html

On the Decidability of Tracing Problems in Dynamic Geometry

Britta Denner-Broser

Institut für Informatik der FU-Berlin, Takustr. 9, 14195 Berlin, Germany
broser@inf.fu-berlin.de

Abstract. We investigate two important problems arising in Dynamic Geometry, the *Tracing Problem* and the *Reachability Problem*. After presenting purely algebraic algorithms for these problems, we give an alternative approach.

1 Introduction

In Dynamic Geometry geometric constructions are represented by Geometric Straight-Line Programs (GSP). They consist of free points and dependent elements (like a line connecting two points, the intersection point of two lines, one of the at most two intersection points of a line and a circle). An instance of a GSP is an assignment of fixed values to all free parameters and choices (see e.g. [10, 8]). Initially these values are elements of $\mathbb{R}$. Let k be the number of free points in the GSP and $m - k$ the number of dependent elements.

Since we work on dynamic geometry, we have to formalize movements of constructions. This is done via *continuous evaluations* ([10]): Given are continuous paths $p_i(t)$, $t \in [0, 1]$, of the free points $(i = 1, \ldots, k)$.

A continuous evaluation under the movement $\{p_i\}$ is an assignment of continuous paths o_j, $j = k + 1, \ldots, m$, to all the dependent elements, such that for all $t \in [0, 1]$ the objects $(p_1(t), \ldots, p_k(t), o_{k+1}(t), \ldots, o_m(t))$ form an admissible instance of the GSP.

There are two problems arising naturally from this setup:

Problem 1. (Reachability Problem)
Given are two instances A and B of a GSP, where A is called starting instance and B a final instance.

Decide whether there are paths $\{p_i\}$ of the free points for which a continuous evaluation from A to B exists. In [10] it is shown by a reduction of 3-SAT that this problem is NP-hard (in $\mathbb{R}$).

Problem 2. (Tracing Problem)
Like in the reachability problem there are given a starting instance A and a final instance B. Let p_A be the position of the free points at instance A, and p_B their position at B. Furthermore, a movement $\{p_i\}$ of the free points from p_A to p_B is given for which there is a continuous evaluation.

H. Hong and D. Wang (Eds.): ADG 2004, LNAI 3763, pp. 111–129, 2006.

Decide whether a continuous evaluation given by the paths p_i and the starting instance A ends at B. In [10] it is shown, again by a reduction of 3-SAT, that this problem is NP-hard.

The Tracing Problem occurs in the following situation: Assume that a user of a Dynamic Geometry software has done a geometric construction. Then what is seen on the screen is an instance A of the GSP describing the underlying construction. Now he drags a free point with his mouse. This describes a continuous path of the free points: The path of the dragged point is piecewise linear whereas the paths of the other free points are constant. The Dynamic Geometry software has to decide which instance B should be drawn after the motion (and at all intermediate positions).

A solution to the Reachability Problem could help in automated theorem proving. Here a geometric problem is given as an instance A of a certain GSP. If a probabilistic approach is used as in [7] one has to create instances B of the geometric problem at random. For generating an instance, one has to assign fixed values to all free parameters *and* also to the choices (e.g. one has to specify which intersection point of a line and a circle is chosen). If one can reach the instance B with a continuous evaluation starting at A we either have found a counterexample and know that the conjecture is wrong or a "positive example", which increases the probability that the conjecture holds (see [7]).

There are some situations, where it is useful to consider complex coordinates of the objects, too. For example, the complex Tracing Problem occurs when we are dealing with singularities (e.g. the intersections of identic circles or lines): Depending on the path p of the free variables, tracing along p might enforce us to consider such degenerate situations (see Figs. 1 and 2): Let the free point P move on the linear path $p(t) = \binom{1.5}{1} + t \cdot \binom{-3}{0}$. Then at time $t_0 = \frac{1}{2}$ the point P lies on the y-axis l_y. Hence Q reaches the origin and the two circles are identic. Thus we have a degenerate situation and the intersection point M is not defined. One approach is to avoid a singularity $S = p(t_0)$ by bypassing it with a little detour (through $\mathbb{R}^{2k}$), e.g. by modifying p in a neighbourhood of t_0. Unfortunately, this might be impossible as our example shows (see Figs. 1 and 2): By construction, the dependent point Q is always incident to the line l_x, and the singularity occurs when Q is moved to the origin. Since Q has to stay on l_x by construction, the singularity S cannot be avoided by modifying the path p of the free point P. A way out of this problem is to consider detours for which the coordinates of the free points may have nonreal coordinates. Now the line l_x of our example becomes a two-dimensional object, and the point Q might be able to bypass S without leaving l_x. Additionally, if the singularity is *removable*, the instance being reached after finishing the detour does not depend on the detour itself (of course this is only true, if the detour does not "catch" other singularities). This is an application of the complex Tracing Problem since we have to trace the complex detour instead of the original real path p of the free variables.

The complex Reachability Problem seems to be useful for automated theorem proving. This is due to the fact that if a theorem holds over $\mathbb{C}$ then it also holds over $\mathbb{R}$.

$P \leftarrow \text{FREE}$
$l_x \leftarrow \text{JOIN}((0,0),(1,0))$
$l_y \leftarrow \text{JOIN}((0,0),(0,1))$
$l \leftarrow \text{JOIN}((0,1),P)$
$Q \leftarrow \text{MEET}(l, l_x)$
$C_1 \leftarrow \text{CIRCLE}((0,0),\frac{1}{2})$
$C_2 \leftarrow \text{CIRCLE}(Q,\frac{1}{2})$
$M \leftarrow \text{MEET}(C_1, C_2)$

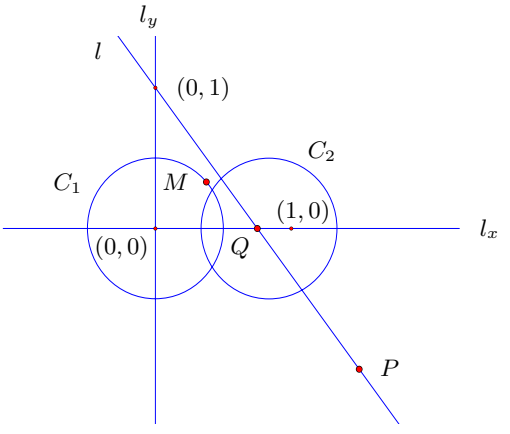


Fig. 1. On the left there is a GSP, on the right there is drawn an instance of the GSP. By construction Q has to stay on the line l_x.

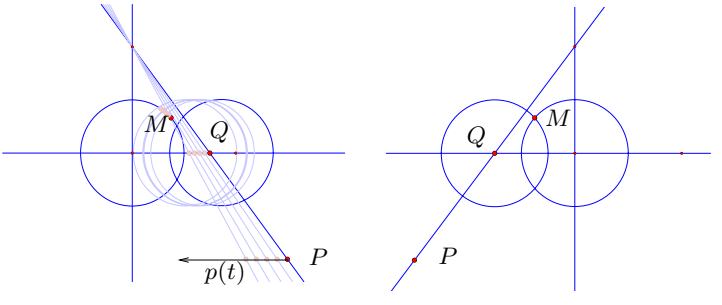


Fig. 2. A singularity occurs if the free point P is moved towards the y-axis

Concerning the Tracing Problem, switching to complex coordinates does not cause many changes since we are tracing given paths. In contrast to this the Reachability Problem over $\mathbb{R}$ and the complex Reachability Problem are somehow different problems (see Fig. 2): In the real situation the right instance in Fig. 2 cannot be reached from the left one, whereas it can be reached via a complex path as described previously.

The geometric situation translates easily into an algebraic model where the objects are numbers (real or complex) and the operations are $+$, $-$, $\cdot$, $/$ and $\sqrt{\square}$. We discuss decision algorithms for the Tracing Problem and the Reachability Problem in this algebraic context. For the Tracing Problem we only allow (piecewise) polynomial paths of the free variables.

In Sect. 2 we refine the model for Dynamic Geometry described on page 111 and explain the Tracing Problem and the Reachability Problem. In Sect. 3 we describe an algebraic approach for both problems over $\mathbb{R}$ and over $\mathbb{C}$. Unfortunately the presented algoithms are just of a theoretical interest, they do not

lead to practical methods. In Sect. 4 we consider the complex situation. We describe algorithms for the complex Tracing Problem (for polynomial paths of the free variables leading only to finitely many degenerate situations) and the complex Reachability Problem for GSPs with one free variable having finitely many critical points. Here some basic concepts from complex analysis are used, and Voronoi diagrams (see [9]) play an important role. We hope that we can extend these ideas to algorithms which can be used in some Dynamic Geometry software in the future. In the appendix the relation between continuous evaluations and Riemann Surfaces is pointed out.

2 Some Basic Concepts from Dynamic Geometry

Geometric Straight-Line Programs. As mentioned in the introduction, geometric constructions can be described by Geometric Straight-Line Programs (GSP, [10, 8]). The objects are points, lines and circles (or more general conics) with the standard geometric operations like computing the line connecting two points, the intersection point of two lines or one of the at most two intersection points of a circle and a line.

Here we regard an algebraic situation: Our objects are real or complex numbers with the operations $+$, $-$, $\cdot$, $/$ and $\sqrt{\ }$. Geometric Straight-Line Programs in this context describe algebraic expressions like $\sqrt{z^2-1}$ instead of geometric constructions. Important is that the instructions of a GSP are defined by relations which is important for the $\sqrt{\ }$-operation. If we regard complex numbers ($\mathbb{K}=\mathbb{C}$), we usually have two possible solutions (as long as the radicant, i.e. the input variable, is not zero). Zero is defined not to be a valid input for the $\sqrt{\ }$-operation, since here the square-root-function is not analytic. In the real situation ($\mathbb{K}=\mathbb{R}$) only positive numbers are valid inputs for the $\sqrt{\ }$-operation.

An *instance* of a GSP Γ is an assignment of objects (in our casereal or complex numbers) such that all relations given by the GSP Γ are fulfilled.

Example 1. The expression $\sqrt{z^2-1}$ can be described by the following GSP Γ_0:

$$\begin{aligned} z &\leftarrow \text{FREE} \\ v_1 &\leftarrow z \cdot z \\ v_2 &\leftarrow v_1 - 1 \\ v_3 &\leftarrow \sqrt{v_2} \qquad \text{(described by } v_3^2 = v_2 \text{ and } v_2 \gneq 0.\text{)} \end{aligned}$$

If $\mathbb{K}=\mathbb{C}$ then $(0,0,-1,i)$ and $(0,0,-1,-i)$ are instances of this GSP, whereas $(1,1,0,0)$ is not an instance. For $\mathbb{K}=\mathbb{R}$ non of the three tuples is an instance.

Note 1.
1. For the operations $+$, $-$ and $\cdot$ all inputs are "admissible", and these operations are deterministic: For each admissible input there is exactly one "admissible" output.
2. The operation $/$ only allows inputs (a,b), where $b \neq 0$. As $+$, $-$, $\cdot$ it is deterministic.

3. The only non-deterministic operation is $\sqrt{\ }$. For a complex number $z \neq 0$ or a real number $z > 0$ the instruction

$$w \leftarrow \sqrt{z}$$

has two possible outputs "$+\sqrt{z}$" and "$-\sqrt{z}$" (e.g. $z = 4$, then ± 2 are the two possible outputs).
4. Variables, which are created by the instruction FREE are called free variables. Elements created by $+$, $-$, $\cdot$, / or $\sqrt{\ }$ are called dependent elements since they depend on their inputs.

Now we introduce the notion of *critical points of a GSP*. They play an important role for the structure of the Tracing Problem and the Reachability Problem.

Definition 1. *A point $(p_1, \ldots, p_k) \in \mathbb{K}^k$ is a* critical point *of a GSP Γ with k free variables $z_1, \ldots, z_k$, if there is at least one "invalid" instance lying over $(p_1, \ldots, p_k) \in \mathbb{K}^k$ (i.e. the relations of Γ are fulfilled but at one dependent variable $\sqrt{0}$ or a division by 0 occurs).*

Example 2. The critical points of the expression $\sqrt{z^2 - 1}$ are 1 and -1.

This definition also makes sense in the case $\mathbb{K} = \mathbb{R}$ since if the first radicant of a $\sqrt{\ }$-operation becomes negativ during a continuous evaluation, it must have been zero at some previous time.

If the GSP has just **one** free variable z and it is $\mathbb{K} = \mathbb{C}$, we can apply theorems from complex analysis. By the identity theorem we get the following

Lemma 1. *If $\mathbb{K} = \mathbb{C}$ then the set of critical points of a GSP with one free variable is bounded if and only if the number of critical points is finite.*

Proof. The identity theorem from complex analysis tells, that if the set of critical points has an accumulation point, then all points are critical points. So all points are critical points (as e.g. in the expression $\sqrt{\sqrt{z^2} - z}$) or they build a discrete set. Since we are regarding expressions defined by a GSP we can exclude the case where there are infinitly many critical points forming a discrete set. □

Continuity. In Dynamic Geometry we are dealing with dynamic constructions: If a free point is moved in a continuous way, the whole construction should follow continuously. It means that whenever the free points move on continuous paths, the dependent elements have to move on continuous paths as well (as long as no critical point lies on the paths). This concept is formalized in the following definition which is taken from [10].

Definition 2. *Let Γ be a GSP having k free variables and $m - k$ dependent elements (w.l.o.g. let the first k variables $p_1, \ldots, p_k$ be the free variables and $o_{k+1}, \ldots, o_m$ the dependent ones). Furthermore for each free variable p_i we are given a continuous path $p_i(t) : [0, 1] \to \mathbb{K}$.*

A continuous evaluation *of the GSP Γ under the movement $\{p_i(t)\}$ is an assignment of continuous functions*

$$o_i(t) : [0,1] \to \mathbb{K}$$

for each $i \in \{k+1, \ldots, m\}$ (i.e. for each dependent element there is one function) such that for all $t \in [0,1]$

$$\big(p_1(t), \ldots, p_k(t), o_{k+1}(t), \ldots, o_m(t)\big)$$

is an instance of the GSP Γ.

Remark 1. 1. If the GSP does not contain a $\sqrt{\ }$-operation, then the values of the dependent variables are determined by the values of the free variables. Since $+$, $-$, $\cdot$ and $/$ are continuous functions (as long as there is no division by zero), there is exactly one continuous evaluation for a given continuous motion of the free variables (avoiding a division by zero).
2. If the GSP Γ contains $\sqrt{\ }$-operations, the values of the dependent variables are determined by the values of the free variables *up to a number of binary choices*, which correspond to the two branches of the $\sqrt{\ }$-function. The definition of continuous evaluation ensures that we always choose the "right" branch and do not jump from one branch to the other one.
3. In the appendix is shown that whenever a starting instance is fixed for the continuous evaluation (i.e. values for $o_{k+1}(0), \ldots, o_m(0)$) and we do not hit a critical point, then there is exactly one continuous evaluation for the given motion starting at this starting instance.

The Tracing Problem and the Reachability Problem. Here we give short definitions of the complex tracing problem and the complex reachability problem. Remember that we are regarding GSPs using the operations $+$, $-$, $\cdot$, $/$ and $\sqrt{\ }$, the object set is $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$.

Definition 3. *Given is a GSP Γ with k free variables and $m-k$ dependent variables.*

1. The Tracing Problem:
   Given *is a starting instance $A = (p_A, o_A)$ and a final instance $B = (p_B, o_B)$ of Γ. Furthermore we are given continuous paths $p_1, \ldots p_k : [0,1] \to \mathbb{K}$ of the free variables of Γ with $(p_1(0), \ldots, p_k(0)) = p_A$ and $(p_1(1), \ldots, p_k(1)) = p_B$. Here we assume, that $p_1, \ldots, p_k$ are polynomials in t which are described by their coefficients.*
   Decide: *Is there a corresponding continuous evaluation from A to B*[1].
2. The Reachability Problem:
   Given *is a starting instance $A = (p_A, o_A)$ and a final instance $B = (p_B, o_B)$ of Γ.*

[1] As long as there is no critical point on the path $(p_1(t), \ldots, p_k(t))$ one "just" has to decide whether the unique continuous evaluation under the movement $\{p_i(t)\}$ starting at A ends at B (see Remark 1).

Decide: *Are there continuous paths $p_i : [0,1] \to \mathbb{K}$ of the free variables for which there is a corresponding continuous evaluation from A to B.*

The difference between both problems is, that for the Tracing Problem, paths of the free variables are given, whereas for the Reachability Problem they are not given (see Fig. 3).

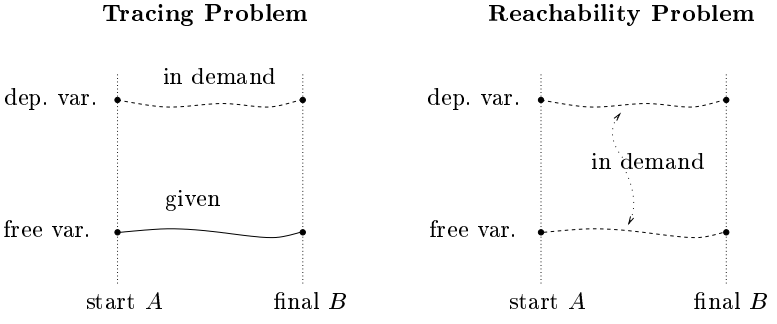


Fig. 3. Exemplification of the difference between the tracing and the reachability problem

3 Algebraic Solution for the Reachability and the Tracing Problem

The Reachability Problem and the Tracing Problem can be decided in the following way: In the first step the connected components of a suitable semi-algebraic set are computed (see [2]). In the second step, it is checked whether the starting instance A and the final instance B lie in the same component. A semi-algebraic set is the set of points in an $\mathbb{R}^n$ satisfying a boolean combination of polynomial equalities and inequalities.

The main problem of this approach is that it seems that these algorithms are too slow to be adapted to a use in praxis in our context.

The Reachability Problem. First we focus on the real situation (i.e. $\mathbb{K} = \mathbb{R}$). Afterwards we describe how the algorithm can be extended to the case $\mathbb{K} = \mathbb{C}$. A GSP Γ defines in a natural way a semi-algebraic set $\mathcal{R}_{\mathbb{R}}(\Gamma)$, as we first explain with Example 1 from page 114. The following GSP Γ_0 describes the expression $\sqrt{z^2-1}$:

$$
\begin{aligned}
z &\leftarrow \text{FREE} \\
v_1 &\leftarrow z \cdot z \\
v_2 &\leftarrow v_1 - 1 \\
v_3 &\leftarrow \sqrt{v_2} \qquad \text{(described by } v_3^2 = v_2 \text{ and } v_2 > 0.\text{)}
\end{aligned}
$$

The corresponding semi-algebraic set is

$$\mathcal{R}_{\mathbb{R}}(\Gamma_0) := \{(z, v_1, v_2, v_3) \in \mathbb{C}^4 \mid v_1 = z^2 \wedge v_2 = v_1 - 1 \wedge v_3^2 = v_2 \wedge v_2 > 0\}.$$

Now we regard the general case, where we are given an arbitrary GSP Γ. For each instruction γ of Γ describing a dependent variable we define a term $\tau(\gamma)$ as follows:

instruction γ	term $\tau(\gamma)$
$a \leftarrow b + c$	$a = b + c$
$a \leftarrow b - c$	$a = b - c$
$a \leftarrow b \cdot c$	$a = b \cdot c$
$a \leftarrow b/c$	$b = a \cdot c$
$a \leftarrow \sqrt{b}$	$a^2 = b \wedge b > 0$

Furthermore let Γ have k free variables and $m - k$ dependent ones defined by the instructions $\gamma_{k+1}, \ldots, \gamma_m$. Then it is

$$\mathcal{R}_{\mathbb{R}}(\Gamma) := \{(p_1, \ldots, p_k, o_{k+1}, \ldots, o_m) \in \mathbb{R}^m \mid \tau(\gamma_{k+1}) \wedge \tau(\gamma_{k+2}) \wedge \cdots \wedge \tau(\gamma_m)\}.$$

To decide the Reachability Problem we have to check whether the starting instance A and the final instance B are in the same connected component of $\mathcal{R}_{\mathbb{R}}(\Gamma)$. This fact is stated in the following lemma.

Lemma 2. *Let $\mathbb{K} = \mathbb{R}$ and A and B two instances of the GSP Γ. Then there is a continuous evaluation connecting A and B if and only if A and B lie in the same connected component of $\mathcal{R}_{\mathbb{R}}(\Gamma)$.*

Proof. Let A and B be instances of Γ. Then by definition the coordinates of A and of B fulfill the relations of Γ, hence $A, B \in \mathcal{R}_{\mathbb{R}}(\Gamma)$. The same argument shows, that a continuous evaluation is a (continuous) path in $\mathcal{R}_{\mathbb{R}}(\Gamma)$ and each path in $\mathcal{R}_{\mathbb{R}}(\Gamma)$ is a continuous evaluation. This implies that there is a continuous evaluation connecting A and B if and only if A and B lie in the same path-connected component of $\mathcal{R}_{\mathbb{R}}(\Gamma)$.

Since $\mathcal{R}_{\mathbb{R}}(\Gamma)$ is a semi-algebraic set the path-connected components of $\mathcal{R}_{\mathbb{R}}(\Gamma)$ coincide with the connected components of $\mathcal{R}_{\mathbb{R}}(\Gamma)$ (see [2]).

The algorithm from [2] outputs for each connected component a boolean combination of polynomial equalities and inequalities, which describes this component. A component contains the instances A and B if and only if the coordinates of A and of B fulfill this boolean combination of polynomial equalities and inequalities.

If we are dealing with the complex Reachability Problem we can use the same approach. Then the term $\tau(\gamma)$ of a $\sqrt{\ }$-instruction $\gamma = (a \leftarrow \sqrt{b})$ is defined as $\tau(\gamma) := (a^2 = b \wedge b \neq 0)$. Additionally each complex variable v ofthe GSP Γ is split into two variables v_r and v_i representing the real and imaginary part of v. We call the corresponding semi-algebraic set $\mathcal{R}_{\mathbb{C}}(\Gamma)$.

The Tracing Problem. As above we start with the real situation, and we use the same GSP Γ_0 as example to explain the algorithm. Additionally we are given a polynomial path $p(t) : [0, 1] \to \mathbb{R}$ of the free variable z. This defines the following semi-algebraic set:

$$\mathcal{T}_{\mathbb{R}}(\Gamma_0) := \{\ (t, z, v_1, v_2, v_2) \in \mathbb{R} \times \mathbb{R}^4 \mid 0 \leq t \leq 1\ \wedge\ z = p(t)\ \wedge\ v_1 = z^2 \wedge v_2 = v_1 - 1\ \wedge\ v_3^2 = v_2\ \wedge\ v_2 > 0\}$$

More generally let Γ an arbitrary GSP having the free variables $p_1, \ldots, p_k$ and the dependent variables $o_{k+1}, \ldots, o_m$. Let $p(t) = (p_1(t), \ldots, p_k(t)) : [0,1] \to \mathbb{R}^k$ be a polynomial path of the free variables. Using the same notation as in the definition of $\mathcal{R}_{\mathbb{R}}(\Gamma)$ we define

$$\mathcal{T}_{\mathbb{R}}(\Gamma) := \{\ (t, p_1, \ldots, p_k, o_{k+1}, \ldots, o_m) \in \mathbb{R} \times \mathbb{R}^m \mid 0 \le t \le 1 \\ \wedge\ p_1 = p_1(t) \wedge \cdots \wedge p_k = p_k(t) \wedge \tau(\gamma_{k+1}) \wedge \tau(\gamma_{k+2}) \wedge \cdots \wedge \tau(\gamma_m)\}.$$

To decide the Tracing Problem we have to determine whether A and B lie in the same connected component of $\mathcal{T}_{\mathbb{R}}(\Gamma)$ and at A it is $t = 0$ and at B it is $t = 1$:

Lemma 3. *Let $A = (p_A, o_A)$ and $B = (p_B, o_B)$ be instances of the GSP Γ and $p(t)$ be a polynomial path with $p(0) = p_A$ and $p(1) = p_B$.*

There exists a continuous evaluation along $p(t)$ starting at the instance A, and this continuous evaluation ends at B, if and only if A and B lie in the same connected component of $\mathcal{T}_{\mathbb{R}}(\Gamma)$, and at A it is $t = 0$ and at B it is $t = 1$.

Proof. The proof is similar to the proof of Lemma 2. Two continuous evaluations could only meet at an instance where the radicant of a $\sqrt{\ }$-instruction is zero. Since these are excluded in $\mathcal{T}_{\mathbb{R}}(\Gamma)$, a continuous evaluation along the path $p(t)$ is a connected component of $\mathcal{T}_{\mathbb{R}}(\Gamma)$.

For the complex Tracing Problem we choose the same approach as for the complex Reachability Problem and define the semi-algebraic set $\mathcal{T}_{\mathbb{C}}(\Gamma)$.

4 Alternative Approaches

Here we give an alternative approach for the complex Tracing Problem and the complex Reachability Problem. The presented methods might be a starting point for finding algorithms which are fast enough for a practical usage. To the authors opinion they take the structure of both problems more directly into account than the algebraic approaches of Sect. 3.

In the beginning we discuss an algorithm for the complex Tracing Problem for polynomial paths which do not contain critical points. Later we show how it can be extended to polynomial paths containing a finite number of critical points.

In Sect. 4.2 we present an algorithm for the complex Reachability Problem for GSPs with just **one** free variable having finitely many critical points. It reduces the Reachability Problem to the Tracing Problem.

4.1 A Decision Algorithm for the Tracing Problem

An alternative way to decide the Tracing Problem is to follow the corresponding continuous evaluation in discrete steps. This gives a discrete approximation of the continuous evaluation, which is an advantage compared to the method presented in Sec. 3. Here the main problem is to compute a suitable steplength.

Again we assume that the free variables of the given GSP move on a polynomial path. First we regard the case where there is no critical point on this path, afterwards we extend the algorithm to paths containing finitely many critical points of the GSP.

Tracing Without Critical Points. The Algorithm proceeds in three steps.

1. Check whether there is no critical point on the path $p(t)$ of the free variables of the GSP Γ.
2. Compute a suitable steplength for following the continuous evaluation.
3. Follow the continuous evaluation using the steplength from 2.

The three steps are discussed separately:

ad 1. To decide whether there is a critical point on a polynomial path is an interesting problem itself. It can be formulated as follows:

At each instance (valid or invalid) lying over the path p the radicant of each $\sqrt{}$-instruction and the denominator of each /-instruction must not be zero. This can be formulated as a Tarski-formula with just one sort of quantifiers.

Example 3. Again we regard the GSP Γ_0 describing the expression $\sqrt{z^2-1}$ from page 114. The following formula is true if and only if there is no critical point on the (polynomial) path p:

$\forall t \in \mathbb{R}\ \forall z, v_1, v_2, v_3 \in \mathbb{C} \hat{=} \mathbb{R}^2$:
$\big(0 \le t \le 1 \wedge z = p(t) \wedge v_1 = z \cdot z \wedge v_2 = v_1 - 1 \wedge v_3^2 = v_2\big) \implies v_2 \neq 0$

ad 2. If there are no critical points on the path p of the free variables we know by Corollaries 1 and 2 of the appendix that there is a unique continuous evaluation of this path starting at a given instance $A = (p_A, o_A)$.

Since the graph of this continuous evaluation is a path-connected component of the compact set $\mathcal{M}$ of all instances lying over the path p, we know that there must be a suitable steplength for the discrete approximation:

There are ν, $\eta > 0$ with the following property: If we fix an instance $R = (p_{01}, \ldots, p_{0k}, o_{0k+1}, \ldots, o_{om})$ of the GSP Γ at time $t_0 \in [0,1]$, then for each $t \in [0,1]$ with $|t - t_0| < \nu$ there is exactly one instance of Γ contained in the circle with center R and radius η.

This phrase can easily be translated to a first-order-formula and the steplength ν can be computed by quantifier elimination over the reals using the intermediate value theorem.

Example 4. Again we regard the GSP Γ_0 from example 1. The free variable moves on the path $p(t) = i \cdot t + 2$. Let η_0 be the minimum distance of all instances over $p(0) = 2$, which is $\|(2,4,3,\sqrt{3}) - (2,4,3,-\sqrt{3})\| = \sqrt{6}$. The steplength ν can be described as follows:

$\exists \nu, \eta\ \forall t_0, t,\ z_0, v_{01}, v_{02}, v_{03},\ z, v_1, v_2, v_3$:
$\big(0 < \eta < \frac{1}{3}\eta_0 \wedge \nu > 0 \wedge |t - t_0| < \nu \wedge z_0 = it_0 + 2 \wedge v_{01} = z_0^2 \wedge v_{02} = v_{01} - 1 \wedge v_{03}^2 = v_{02} \wedge z = it + 2 \wedge v_1 = z^2 \wedge v_2 = v_1 - 1 \wedge v_3^2 = v_2 \wedge \|(z_0, v_{01}, v_{02}, v_{03}) - (z, v_1, v_2, v_3)\| \ge \eta\big) \implies \|(z_0, v_{01}, v_{02}, v_{03}) - (z, v_1, v_2, v_3)\| \ge 2\eta$

ad 3. We get the following algorithm (see Fig. 4). To simplify matters we assume that $\frac{1}{2\nu} \in \mathbb{Z}$.

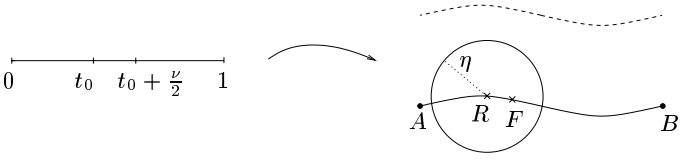


Fig. 4. Notation of the approximation algorithm

Reference-time $t_0 := 0$;
Reference-instance $R :=$ starting instance A;
while $t_0 < 1$ do
$t_0 := t_0 + \frac{\nu}{2}$;
compute the unique instance F at t_0 with smallest distance to R;
this will be the new reference-instance R.
Return $(R = B)$

Table 1. Complexity of the general algorithms for the complex Tracing Problem (see [2]), l denotes the total number of $\sqrt{\text{-}}$ and /-operations; if the path of the free variables is linear the max. degree d of the polynomials is 2

Algorithm	No. of Vars. s	No. of Pols. k	Complexity
Step 1 of Sec. 4.1	$2m+1$	$2+2(m+l)$	$s^{k+1}d^{\mathrm{O}(k)} = m^{\mathrm{O}(m)}$
Step 2 of Sec. 4.1	$\underbrace{2}_{k_1} + \underbrace{2+4m}_{k_2}$	$6+4m$	$s^{(k_1+1)(k_2+1)}d^{\mathrm{O}(k_1)\mathrm{O}(k_2)} = m^{\mathrm{O}(m)}$
ALg. from Sec. 3	$2m+1$	$2+2(m+l)$	$s^{k+1}d^{\mathrm{O}(k^4)} = m^{\mathrm{O}(m)}2^{\mathrm{O}(m^4)}$

Remark 2. 1. The runtime of step 1 and of deciding whether a given ν is a proper steplength in setp 2 is slightly better than the runtime of the algorithm from Sec. 3 (see Tab. 1). The runtime of step 3 depends on the steplength. Since non of the methods is elaborated for a practical usage we omit a detailed runtime discussion.
(Remember, that the Tracing Problem is NP-hard).

2. There might be exponentially many critical points on a path p of the free variables as the following example shows:
The expression $\sqrt[\frac{n}{2}]{z^{\frac{n}{2}} - 1}$ can be described by a GSP of length $n+1$. Then all $2^{\frac{n}{2}}$ critical points lie on the path $p : [0, \sqrt{\frac{1}{2}}] \to \mathbb{C}$, $t \mapsto (+\sqrt{1-t^2} + it)^8$ whose graph is the unit circle[2].
Over each regular point (i.e. noncritical point) there are $2^{\frac{n}{2}}$ instances.

3. There are several numerical continuation methods which could also be applied to the Tracing Problem (see [1, 4]). They can also deal with bifurcation points and critical points, but usually there is no guarantee for the correctness of the solution.

[2] One could also enlarge the GSP by a constant number of instructions and regard the path $q(t) = t$.

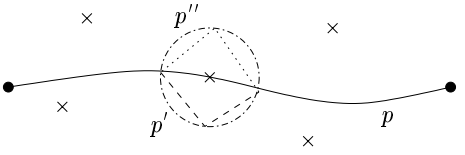


Fig. 5. The little crosses mark critical points; the path p contains one of them, which can be omitted by following the dashed or dotted detour. The radius of the circle is $\frac{1}{2}\min\{|x-y| \mid x \neq y \text{ critical points}\}$.

Tracing with Finitely Many Critical Points. Now we consider the case in which there are critical points on the path p of the free variables. Here there are two quite different situations, which can be distinguished using Lemma 1 from page 115: In the first case there is a finite number of critical points on p, in the second case all points are critical points. We only consider the first one.

We begin with the case in which there is one singularity over the path p of the free variables (it is clear that this can be generalized for finitely many singularities).

First we restrict on GSPs with just one free variable. Let t_0 be the time at which the singularity occurs and as above let p be the path of the free variable. Now we disturb p in a neighbourhood of t_0 twice, so that we pass the critical point once at both sides (see Fig. 5). This can be done by adding "small" piecewise linear functions to p. Let the resulting paths be p' and p'', they can be chosen such that they do not hit a singularity and also do not catch other singularities. Now we "trace" p' and p'' starting at our given starting instance A. Let B' and B'' be the instances at which we end. The residue theorem from complex analysis (resp. Riemann Surfaces) implies:

If $B' = B''$**,** the continuous evaluation of our original path p has not hit an invalid instance and we just have to check whether B' is the given final instance B.

If $B' \neq B''$**,** we must have hit an invalid instance, and we cannot solve the tracing problem unambiguously.

If we are dealing with a GSP having more than one free variable, we can argue as follows: Since we assume that the paths of the free variables are polynomials in t, we can encode all these paths as a GSP (so that our original GSP just becomes a bit longer). Hence we have just one free variable t left (the path we regard is the embedding $[0,1] \hookrightarrow \mathbb{C}$). Thus we have reduced the problem to the situation with just one free variable.

Remark 3. An interesting problem is how to deal with infinitely many singularities.

4.2 A Decision Algorithm for the Reachability Problem

Now we are going to describe an algorithm for deciding the reachability problem for GSPs with just **one** free variable having finitely many critical points. It uses the decision algorithm for the tracing problem from the previous subsection.

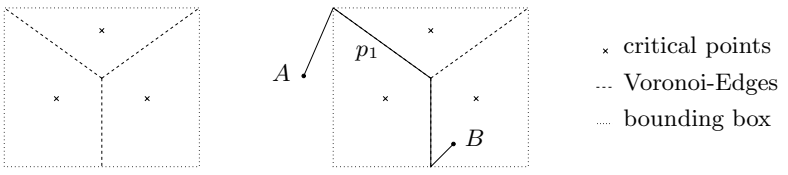


Fig. 6. The left figure shows the Voronoi diagram of three points in a bounding box, in the right figure there is a path p_1 from A to B which is essentially composed of edges of the Voronoi diagram

1. First we check whether the number of critical points is finite (see Lemma 1). If it is true, we compute the critical points and a bounding box $\mathcal{D} = [-C, C] \times [-iC, iC]$ containing all of them in its interior.
2. We compute a finite set of paths which suffices to decide the Reachability Problem. For this we use the Voronoi diagram (see [3, 9]) of the critical points in the bounding box $\mathcal{D}$. It is a planar graph having the following properties:
   - The diagram has finitely many edges, and the edges are line segments,
   - there lies no critical point on an edge of the diagram, and
   - each facet (Voronoi region) contains exactly one critical point.

   The paths basically consist of edges of the Voronoi diagram (see Fig. 6). The details to this step are described in Sect. 4.3.
3. To decide the Reachability problem we have to trace all these paths.

In the first step it suffices to approximate the critical points up to the precision $\frac{1}{4} \cdot \epsilon$, where ϵ is the minimum distance between two critical points. Then the Voronoi diagram of the approximated points still has the properties required in step 2.

We briefly recall the notion of the Voronoi diagram of a finite set $S = \{c_1, \ldots, c_f\}$ of points in the plane (see Fig. 6). In our case these are the critical points of the GSP Γ. The Voronoi region of a point c_i contains all points of $\mathbb{R}^2 \hat{=} \mathbb{C}$ which are closer to c_i than to all other points of S. The line segements (or rays) separating two Voronoi regions are called Voronoi edges, their endpoints are the Voronoi vertices. A formal defintion can e.g. be found in [9].

4.3 Details and Correctness of the Algorithm

We discuss the second step of the algorithm from 4.2 and the correctness of the algorithm.

A first observation is that "similar" paths (i.e. homotopic paths) of the free variable z lead to "similar" continuous evaluations (i.e. if they both have the same starting instance, then they also have the same final instance). Roughly speeking two paths having the same endpoints are called homotopic, if one of them can be continuously transformed into the other one, and the transformation leaves the endpoints of the curves fixed (see Definition 6 in the appendix). This defines an equivalence relation on the set of continuous paths, the equivalence classes are called homotopy classes.

Let $c_1, \ldots, c_f$ be the critical points of the GSP Γ. Lemma 4 is a consequence of Corollary 3 from the appendix.

Lemma 4. *Let $S = \{c_1, \ldots, c_f\}$ be the set of critical points of a GSP Γ of length m having one free variable z and $m-1$ dependent variables $w_2, \ldots, w_m$. Furthermore let $p : [0,1] \to \mathbb{C} \setminus S$ and $p' : [0,1] \to \mathbb{C} \setminus S$ be homotopic continuous paths of the free variable z of the GSP Γ and $A = (a_1, \ldots, a_m) \in \mathbb{C}^m$ an instance of Γ with $a_1 = p(0) = p'(0)$.*

Then the continuous evaluations of p and p' starting at A end at the same instance of Γ.

Hence for deciding the Reachability Problem it is enough to take one path $p : [0,1] \to \mathbb{C} \setminus \{c_1, \ldots, c_f\}$ per homotopy class of paths $[0,1] \to \mathbb{C} \setminus \{c_1, \ldots, c_f\}$ starting and ending at the corresponding instances. The two paths p_1 and p_2 in Fig. 7 are not homotopic and represent different homotopy classes.

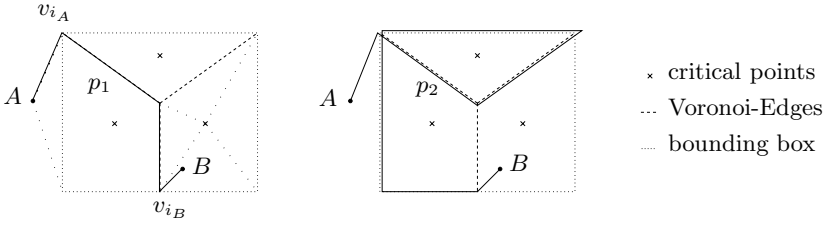


Fig. 7. Two paths p_1 and p_2 from A to B are drawn, which we have to consider in our algorithm. In the left figure the construction of the Voronoi vertices v_{i_A} and v_{i_B} is indicated.

Since each Voronoi cell contains exactly one critical point, each homotopy class has a representative which is essentially composed by a finite number of Voronoi edges of the Voronoi diagram of the critical points $c_1, \ldots, c_f$ (see Fig. 7 for an example).

If the starting instance $A = (p_A, o_A)$ (or the final instance $B = (p_B, o_B)$) does not lie on a Voronoi vertex, we choose a vertex v_{i_A} (resp. v_{i_B}) of the diagram and regard the linear path γ_A (resp. γ_B) connecting p_A and v_{i_A} (or p_B and v_{i_B}). The vertices v_{i_A} and v_{i_B} can be chosen such that the linear paths γ_A and γ_B do not hit critical points (see Remark 4 and Fig. 7).

There is still an infinite number of paths left which we have to consider. We give a construction where it is sufficient to trace a finite number of edges of the Voronoi diagram to decide the Reachability Problem:

Let l be the number of instances lying over a regular point[3], and let VD be the Voronoi diagram in the bounding box of the critical points of the GSP having the Voronoi vertices $v_1, \ldots, v_g \in \mathbb{C} = \mathbb{R}^2$.

We define a graph $G = (V, E)$ combining the idea of Voronoi diagrams and coverings (resp. Riemann Surfaces of algebraic functions). The vertex set V

[3] For each $\sqrt{\ }$-operation, the number of instances is doubled unless the radicant is zero.

consists of l copies $v_{i,1}, \ldots, v_{i,l}$ of each Voronoi vertex v_i. Two vertices $v_{i,j}$ and $v_{i',j'}$ are connected by an edge if the following holds: There is a Voronoi edge in the Voronoi diagram between the Voronoi vertices v_i and $v_{i'}$ and the continuous evaluation of this Voronoi edge starting at the instance $v_{i,j}$ ends at $v_{i',j'}$.

The Reachability Problem can be decided as follows:

1. We regard a linear path $\gamma_A : [0,1] \to \mathbb{C}$ from the starting instance A (or more precisely p_A) to a Voronoi vertex v_{i_A} and another path $\gamma_B : [0,1] \to \mathbb{C}$ from the final instance B to a Voronoi vertex v_{i_B}. The corresponding continuous evaluations starting at A resp. B end at the instances of the GSP which correspond to the vertices $v_{i_A,j}$ resp. $v_{i_B,j'}$ of our graph G.

   As long as there is no critical point on the linear paths γ_A and γ_B, it does not matter which Voronoi vertices are chosen for v_{i_A} and v_{i_B}: For another choice of $v_{i'_A}$ ($v_{i'_B}$) the linear path connecting p_A (p_B) and $v_{i'_A}$ ($v_{i'_B}$) is homotopic to a concatenation of γ_A (γ_B) and some paths which follow edges of the Voronoi diagram.
2. Check whether the vertices $v_{i_A,j}$ and $v_{i_B,j'}$ are in the same connected component of the graph G.

Remark 4. 1. The Voronoi vertices v_{i_A} (resp. v_{i_B}) can be chosen as follows: If p_A lies outside the bouding box, then v_{i_A} is the nearest vertex on the bounding box such that the line segment $|p_A v_{i_A}|$ does not intersect the interior of the bounding box. Otherwise p_A is contained in a Voronoi cell. We triangulate this cell by adding all edges connecting a vertex of this cell with the critical point defining the cell. Then v_{i_A} is the nearest Voronoi vertex of the triangle containing p_A. If in the first step of the algorithm in 4.2 the critical points have been approximated, we have to choose v_{i_A} and v_{i_B} more carefully.

2. The graph G describes somehow a discretisation of the Riemann Surface of the algebraic function of the GSP.
3. In the following example the configuration space (i.e. the space containing all instances of the GSP) of the GSP is not path-connected:

$$\begin{aligned} z &\leftarrow \text{FREE} \\ v_1 &\leftarrow z \cdot z \\ v_2 &\leftarrow \sqrt{v_1} \\ v_3 &\leftarrow v_2 / z \end{aligned}$$

   This GSP describes the expression $\frac{\sqrt{z^2}}{z}$. If $z \neq 0$ we have $v_3 = \pm 1$.

5 Outlook

There are several important problems remaining. In order to get an applicable algorithm for the Tracing Problem, we have to find a "good" steplength for the approximation of the continuous evaluation. Additionally it would be useful, if this method could detect critical points on the given path of the free variable.

For the complex Reachability Problem it would be nice to get some better complexity bounds, since here the NP-completeness proof from [10] does not work. Another interesting question concerning both problems is how to deal with infinitely many critical points in practical applications.

Appendix: Continuous Evaluations and Coverings

Here we briefly explain the notion of coverings as they are used in the theory of Riemann surfaces ([6, 5]). They are a useful tool for proving the uniqueness and existence of continuous evaluations.

Lemma 4 from 4.2 is an easy consequence of Corollary 3. It is very crucial for the second step of the algorithm presented in 4.2.

Definition 4. *([6], def. 4.1) Let X be a topological space. A topological space Y together with a map $\pi : Y \to X$ is called* covering *of X, if π is continuous and open and if for each point $x \in X$ the inverse image $\pi^{-1}(x)$ is empty or discrete in Y. The set $\pi^{-1}(x)$ is called* fiber over x *(see [5], p. 18).*

A point $y \in Y$ is called branchpoint *of the covering π, if there is no neighbourhood V of y such that $\pi_{|V}$ is injective (see [6], def. 4.3).*

Example 5. Is $X = \mathbb{C}$, then $Y = \mathbb{C}$ and $\pi : Y = \mathbb{C} \to X = \mathbb{C}$, $z \mapsto z^2$, is a covering. Indeed π is continuos and open and $\pi^{-1}(z)$ consists of two elements if $z \neq 0$ and $\pi^{-1}(0) = \{0\}$ contains one element. $0 \in Y = \mathbb{C}$ is a branchpoint of the covering (Y, π), since for each neighbourhood $V \subset Y$ of 0 the restriction $\pi|_V$ of π to V is not injective (ϵ and $-\epsilon$ are both mapped to ϵ^2).

Fig. 8 shows the common visualisation of this covering, which is the Riemann surface of the function $\sqrt{\ }: \mathbb{C} \to \mathbb{C}$.

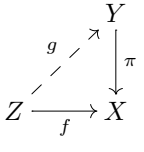
It is important to remark that this surface does not have selfintersections. The "dashed line" is due to the embedding into the $\mathbb{R}^3$. Observe that the map π is drawn as a projection.

The following lemma helps to understand the definition of coverings.

Lemma 5. *([6], thm. 4.2) Let X, Y be Riemann surfaces and $p : Y \to X$ a non-constant holomorphic map. Then p is a covering.*

We want to give some more standard definitions and properties of coverings (see [6]) and apply them to the notion of continuous evaluations.

Definition 5. *([6], def. 4.7) Let X, Y and Z be topological spaces, $\pi : Y \to X$ a covering and $f : Z \to X$ a continuous map. A* lifting *of f (corresponding to π) is a continuous map $g : Z \to Y$ with $f = \pi \circ g$, i.e. the following diagram commutes.*



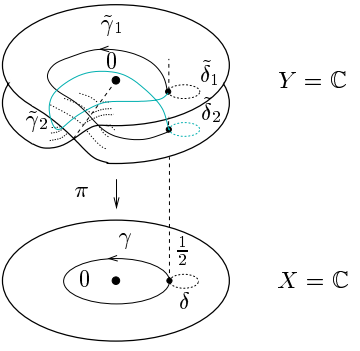


Fig. 8. Two-fold covering of $\mathbb{C}$ defined by $Y = \mathbb{C}$ and $\pi(z) = z^2$ with the liftings $\tilde{\gamma}_1$, $\tilde{\gamma}_2$, $\tilde{\delta}_1$ and $\tilde{\delta}_2$ of the closed curves γ and δ

Example 6. Again we regard the covering $\pi : \mathbb{C} \to \mathbb{C}$, $z \mapsto z^2$. In Fig. 8 all liftings of the curves $\gamma(t) = \frac{1}{2}e^{2\pi it}$ and $\delta(t) = \frac{1}{2}e^{2\pi i(t-\frac{1}{2})} + 1$, $t \in [0,1]$, are drawn. The liftings $\tilde{\gamma}_1$ and $\tilde{\gamma}_2$ of γ change the leaves of the covering π whereas the liftings $\tilde{\delta}_1$ and $\tilde{\delta}_2$ of δ don't.

Theorem 1. *([6], thm. 4.8) (Uniqueness of liftings) Let X, Y be Hausdorff-spaces (i.e. topological spaces with a separation property) and $\pi : Y \to X$ a covering* without *branchpoints. Furthermore let Z be a connected topological space (for example the interval $[0,1]$ in case we want to lift curves) and $f : Z \to X$ a continuous map.*

If g_1, $g_2 : Z \to Y$ are two liftings of f and is $g_1(z_0) = g_2(z_0)$ for one point $z_0 \in Z$ then $g_1 = g_2$.

Theorem 1 implies that liftings of curves (on which there lies no basepoint of a branchpoint of the covering) are unique if e.g. the starting point is specified.

Corollary 1 of Theorem 1 is the exact statement of an obvious fact.

Corollary 1. *(Uniqueness of continuous evaluations)*
Let $S \subset \mathbb{C}^k$ be the set of critical points of a GSP Γ of length m having k free variables $z_1, \ldots, z_k$ and $m-k$ dependent variables $w_{k+1}, \ldots, w_m$. Furthermore let $p = (p_1, \ldots, p_k) : [0,1] \to \mathbb{C}^k \setminus S$ be a continuous path of the free variables $z_1, \ldots, z_k$ of the GSP Γ and $A = (a_1, \ldots, a_m) \in \mathbb{C}^m$ an instance of Γ with $a_1 = p_1(0), \ldots, a_k = p_k(0)$.

If $\tilde{p} = (p_1, \ldots, p_k, o_{k+1}, \ldots, o_m) : [0,1] \to \mathbb{C}^m$ and $\hat{p} = (p_1, \ldots, p_k, o'_{k+1}, \ldots, o'_m) : [0,1] \to \mathbb{C}^m$ are two continuous evaluations of $p = (p_1, \ldots, p_k)$ starting at A then $\tilde{p} = \hat{p}$.

Proof. (Induction by the length m of Γ)
W.l.o.g. let the free variables $z_1, \ldots, z_k$ be the first k variables of Γ. If $m \leq k$ then Γ just consists of free variables and $\tilde{p} = \hat{p}$ is fulfilled.

Now let Γ be a GSP of length $m > k$. By induction we know, that

$$(p_1, \ldots, p_k, o_{k+1}, \ldots, o_{m-1}) = (p_1, \ldots, p_k, o'_{k+1}, \ldots, o'_{m-1}),$$

so that it remains to prove $o_m = o'_m$. If the last instruction of Γ is one of the operations $+, -, \cdot$ or $/$, $o_m = o'_m$ holds since these operations are determined by their input variables. If the last instruction of Γ is a $\sqrt{-}$-operation w.l.o.g. having the variable v_l of Γ as input (so it is $v_l = z_i$ for an $i \in \{1, \ldots, k\}$ or $v_l = w_j$ for a $j \in \{k+1, \ldots, m-1\}$). By the assumptions we know that v_l moves on the continuous path α, where $\alpha = p_i$ (if $v_l = z_i$) or $\alpha = o_j = o'_j$ (if $v_l = w_j$), and α does not take $0 \in \mathbb{C}$ as value. Since o_m and o'_m are two liftings of α (corresponding to the covering $\pi : \mathbb{C} \to \mathbb{C}$, $z \mapsto z^2$) with $o_m(0) = o'_m(0) = a_m$ it follows by Theorem 1 that $o_m = o'_m$. □

Now we discuss the existence of continuous evaluations. We need the following lemma, which is a consequence of Definition 4.11 and Theorem 4.14 from [6].

Lemma 6. *The covering*

$$\begin{array}{rcl} \pi : \mathbb{C} \setminus \{0\} & \to & \mathbb{C} \setminus \{0\} \\ z & \mapsto & z^2 \end{array}$$

has the curve-lifting-property, i.e.:

For each continuous function $u : [0,1] \to \mathbb{C} \setminus \{0\}$ and each point $y_0 \in \mathbb{C} \setminus \{0\}$ with $\pi(y_0) = u(0)$ there is a lifting $\tilde{u} : [0,1] \to \mathbb{C} \setminus \{0\}$ of u with $\tilde{u}(0) = y_0$.

Corollary 2. *Existence of continuous evaluations*
Let $S \subset \mathbb{C}^k$ be the set of critical points of a GSP Γ of length m having k free variables $z_1, \ldots, z_k$ and $m-k$ dependent variables $w_{k+1}, \ldots, w_m$. Furthermore let $p = (p_1, \ldots, p_k) : [0,1] \to \mathbb{C}^k \setminus S$ be a continuous path of the free variables $z_1, \ldots, z_k$ of the GSP Γ and $A = (a_1, \ldots, a_m) \in \mathbb{C}^m$ an instance of Γ with $a_1 = p_1(0), \ldots, a_k = p_k(0)$.

Then there is a continuous evaluation $\tilde{p} = (p_1, \ldots, p_k, o_{k+1}, \ldots, o_m) : [0,1] \to \mathbb{C}^m$ of $p = (p_1, \ldots, p_k)$ starting at A.

Proof. Corollary 2 can be proven in a similar way as Corollary 1 using Lemma 6. □

For investigating the Reachability Problem the notion of homotopic curves is helpful as seen in 4.2. Roughly speeking two paths with the same endpoints a, b are called homotopic, if one of them can be continuously transformed into the other one, and the transformation leaves the endpoints of the curves fixed.

Definition 6. *([6], def. 3.1) Let X be a topological space and $a, b \in X$. Let $c_1, c_2 : [0,1] \to X$ be (continuous) curves from a to b. The two curves c_1 and c_2 are called homotopic, if there is a map $H : [0,1] \times [0,1] \to X$, called homotopy, with the following properties:*

1. *$\forall t \in [0,1] :$ $H(t,0) = c_1(t)$, i.e. for $s = 0$ we get the path c_1.*
2. *$\forall t \in [0,1] :$ $H(t,1) = c_2(t)$, i.e. for $s = 1$ we get the path c_2.*
3. *$\forall s \in [0,1] :$ $H(0,s) = a$ and $H(1,s) = b$, i.e. the endpoints a and b of c_1 and c_2 are fixed.*

This defines an equivalence relation on the set of continuous paths $[0,1] \to X$, i.e. two paths $c_1, c_2 : [0,1] \to X$ are equivalent if and only if they are homotopic (see [6], Thm. 3.2).

The equivalence classes are called homotopy classes. Using the following theorem it can be shown that the continuous evaluations of homotopic paths starting at the same instance have the same final instance.

Theorem 2. *([6], thm. 4.10) (Lifting of homotopic curves) Again let X and Y be Hausdorff-spaces and $\pi : Y \to X$ a covering without branchpoints. Let a, $b \in X$ and $\hat{a} \in Y$ with $\pi(\hat{a}) = a$. Additionally a continuous map $H : [0,1] \times [0,1] \to X$ is given with $H(0,s) = a$ and $H(1,s) = b$ for all $s \in [0,1]$. We define*

$$u_s(t) := H(t,s)$$

and assume that each curve u_s can be lifted to a curve $\hat{u}_s$ with starting point $\hat{a}$.

Then $\hat{u}_0$ and $\hat{u}_1$ have the same endpoint and they are homotopic.

Corollary 3. *Let $S \subset \mathbb{C}^k$ be the set of critical points of a GSP Γ of length m having k free variables $z_1, \ldots, z_k$ and $m-k$ dependent variables $w_{k+1}, \ldots, w_m$. Furthermore let $p = (p_1, \ldots, p_k) : [0,1] \to \mathbb{C}^k \setminus S$ and $p' = (p'_1, \ldots, p'_k) : [0,1] \to \mathbb{C}^k \setminus S$ be homotopic continuous paths of the free variables $z_1, \ldots, z_k$ of the GSP Γ and $A = (a_1, \ldots, a_m) \in \mathbb{C}^m$ an instance of Γ with $a_1 = p_1(0) = p'_1(0), \ldots, a_k = p_k(0) = p'_k(0)$.*

Then the continuous evaluations of p and p' starting at A end at the same point.

Proof. Corollary 3 can be proven in a similar way as Corollary 1 using Thm. 2. □

References

1. E. L. Allgower, K. Georg, *Numerical Continuation Methods*, Springer-Verlag Berlin Heidelberg 1990
2. S. Basu, R. Pollack, M.-F. Roy, *Algorithms in Real Algebraic Geometry*, Springer-Verlag Berlin Heidelberg 2003
3. M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry*, Springer-Verlag Berlin Heidelberg 1997, 2000
4. L. Blum, F. Cucker, M. Shub, S. Smale, *Complexity and Real Computation*, Springer-Verlag New York 1998
5. W. Ebeling, *Funktionentheorie, Differentialtopologie und Singularitäten*, Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden 2001
6. O. Forster, *Riemannsche Flächen*, Springer-Verlag Berlin Heidelberg 1977
7. U. Kortenkamp, *Foundations of Dynamic Geometry*, PhD-thesis, ETH Zürich, 1999
8. U. Kortenkamp, J. Richter-Gebert, *Decision Complexity in Dynamic Geometry*, Springer Lecture Notes in Artificial Intelligence 2061 (Automated Deduction in Geometry, Workshop 2000), p. 193-199, 2001
9. F. P. Preparata, M. I. Shamos, *Computational Geometry*, Springer-Verlag New-York Inc. 1985
10. J. Richter-Gebert, U. Kortenkamp, *Complexity Issues in Dynamic Geometry*, Proceedings of the Smale Fest 2000, 2001

Towards a Geometric-Object-Oriented Language

Tielin Liang[1] and Dongming Wang[2,3]

[1] Department of Mathematics, University of Science and Technology of China, Hefei 230026, Anhui, China
[2] LMIB – School of Science, Beihang University, Beijing 100083, China
[3] Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie – CNRS, 8 rue du Capitaine Scott, F-75015 Paris, France

Abstract. This paper proposes a geometric-object-oriented language for symbolic geometric computation, reasoning, and visualization. In this language, geometric objects are constructed with indefinite parametric data. Modifications and basic operations on these objects are enabled. Degeneracy and uncertainty are handled effectively by means of imposing conditions and assumptions and geometric statements are formulated by declaring relations among different objects. A system implemented on the basis of this language will allow the user to perform geometric computation and reasoning rigorously and to prove geometric theorems and generate geometric diagrams and interactive documents automatically. We present the overall design of the language, explain the capabilities, features, main components of the proposed system, provide specifications for some of its functors, report our experiments with a preliminary implementation of the system, and discuss some encountered difficulties and research problems.

1 Introduction and Motivation

The axiom system of geometry developed since Euclid and finalized by David Hilbert [3] not only laid a solid foundation for elementary geometry, but has also become a model of axiomatization for modern mathematical reasoning. However, as pointed out first by Wen-tsün Wu [10], Hilbert's axiom system is far from being rigorous and in fact it is very difficult (if possible at all) to establish an axiom system under which geometric problems may be stated rigorously according to Hilbert's axiomatic approach. The reason is that geometric relations, axioms, and theorems are usually stated under certain not explicitly mentioned assumptions that the considered geometric configurations are in generic position. For example, while speaking about a triangle, one implicitly assumes that the triangle does not degenerate to a line. Otherwise, subsequent geometric constructions such as of the orthocenter and circumcircle of the triangle cannot proceed. Axioms and theorems about this triangle may be false or become meaningless in some degenerate cases.

Degenerate cases and conditions did not draw much attention of classical geometers. This is in accordance with the tradition of mathematics in which

H. Hong and D. Wang (Eds.): ADG 2004, LNAI 3763, pp. 130–155, 2006.

the treatment of special and trivial cases is often ignored. The situation is totally different when our concern is how to represent and manipulate geometric objects and how to perform computation and reasoning with them on modern computer. In this case, to ensure the involved manipulation and computation correct and rigorous in the logical sense, we must deal with all the degenerate cases. Moreover, geometric objects under consideration are usually indefinite and symbolic. For instance, while speaking about a circle, we may mean a general circle without specifying its radius and center numerically. How to handle such symbolic geometric objects effectively, how to perform computations with them symbolically, and how to reason about them correctly are some of the questions we are concerned with.

Although nondegeneracy conditions have been considered in most of the existing geometric theorem provers [2], there is little investigation on the foundation of computer-based geometry, for which a formal and comprehensive language must be developed so that symbolic geometric objects and relations may be specified, represented, visualized, and manipulated and computation and reasoning with them can be performed correctly and effectively even in the presence of uncertainty and degeneracy. Such a language should also be capable of representing and processing geometric knowledge. A software system developed on the basis of this language will have applications in diverse areas of modern geometry engineering such as CAGD, geometric modeling, computer vision, and computer graphics, in addition to geometry research and education.

The purpose of this paper is to propose such a language, that is geometric-object-oriented, referred to hereafter with its acronym Gœl. Our emphasis here is placed on the design methodology and implementation of the language is in progress. We will outline the overall design, capabilities, and features of Gœl in the next section, describe its main components in Sections 3 and 4, with specifications for some of the functors. In Section 5, we will discuss some implementation issues and report our experiments with a Gœl prototype. Some difficulties and challenges encountered in the development of Gœl will be addressed in Section 6. The paper will end with a brief discussion on a few technical problems for future research.

2 Overall Design of Gœl

We consider Gœl both as a language and as a system. As a language, its basic elements are *symbolic* geometric objects (i.e., geometric objects whose shape, size, and position are not numerically specified) and the language is oriented around these objects. Gœl is regarded as a special kind of object-oriented language (OOL for short) because it is geometric-object-oriented (see Section 6.1) and has almost all the attributes of generic OOL. Generic OOL has many standard and perfect properties such as strong and strict structure, hierarchical encapsulation and hiding (to protect information), miscellaneous method overloading (satisfying different requirements), and classified polymorphic inheritance (ensuring easy deriving of new objects) that Gœl will need. As a special instance of OOL,

Gool keeps most of these properties: every geometric object is strongly typed with information entries and miscellaneous methods, information can be purposely encapsulated and hidden in, methods can be overloaded with the same function names but different arguments, and many kinds of subobjects or new geometric objects can be derived or inherited from the known geometric objects.

In addition to the object-oriented paradigm, the design of Gool also follows the principles outlined below.

Gool is an interpreting language, like the batch process in DOS and the shell language in Unix/Linux. For such a language, programs need not be compiled before they are executed. The user may simply read a command or a segment of program statements into the shell and the shell will interpret and execute it and output the results. Interpreting languages have many advantages: for example, they are simple and intuitive and the user can easily know what a program does and can modify the running program quickly and dynamically. This kind of languages may work sometimes in low efficiency and thus have limited applicability for real-world problems, but they are adequate for our purpose of exploration on the foundational aspect of geometry software design.

Gool is class-based: it is based on various classes, which represent different classes of geometric objects and relations. However, the user of Gool does not need to use the formal syntax such as

```
class circle {
    data_type  data
    method_type  method_name( method_args )
      ⋮
}
```

to define geometric objects. Symbolic geometric objects and relations may be embedded in the core of Gool. When the user wants to define an object, he or she only needs to specify some parameters to a built-in function; then the geometric object will be defined automatically as wanted. Although Gool is object-oriented and has polymorphism and inheritance, its inheritance is not explicit. This means that there is no need for the user to derive new objects explicitly with the syntax

```
class sub_class extends parent_class { ····· }
```

because such inheritance may also be embedded in Gool. The user can get the inherited objects by giving some values to the parameters of a built-in function.

Gool is a special-purpose language/system: it is not for common programming and general problem-solving, but for doing geometry in two-, three- or higher-dimensional Euclidean or projective space. This language or system is designed specifically for constructing symbolic geometric objects (such as circles, triangles, polygons, and spheres) with uncertain data and indefinite parameters, manipulating the constructed objects (e.g., modifying parts of them by changing some of their parametric values), calculating basic geometric quantities (such as the perimeter of a triangle and the volume of a tetrahedron), visualizing two- or three-dimensional geometric objects, and performing various kinds of exact computation and formal

reasoning in elementary geometry. An ideal version of Gool may become a powerful software tool for geometry research, education, and application.

General-purpose computer algebra systems have been widely used because of their capabilities and power for symbolic, algebraic, and numeric computation, graphic representation, document processing, and high-level functional programming. It is desirable that some of these capabilities be integrated into Gool to support geometric computation and reasoning. However, at this early stage of our investigation we have no intention to implement any standard module for symbolic and algebraic computation because of the inherent difficulties and necessary effort required for the implementation. We suggest to make use of those modules that have already been available in many of the computer algebra systems, such as Maple and Mathematica, by creating an interface. The dependence of Gool on a specific computer algebra system may be dropped at a late stage by supplying the symbolic and algebraic computation modules.

In what follows, we list some of the expected capabilities and features of Gool.

- It can represent and construct symbolic geometric objects, such as points, lines, segments, circles, triangles, and polygons, with built-in information entries and computational methods. Such entries may cover information about properties, well-known facts, remarkable theorems, and history background about the objects, subobjects and objects derived from them, assumptions and conditions generated automatically or provided by the user, internal relations among subobjects and derived objects, and external relations with other objects. We may name some of the typical information entries as `representation`, `derivedObject`, `condition`, `assumption`, `property`, `knowledge`, `internalRelation` and `externalRelation` (see Section 3.2 for details). In particular, it is necessary to include the `condition` entry, that collects nondegeneracy conditions about an object during its construction for late use. This is the first step of taking nondegeneracy conditions into account. Every geometric object in Gool is strongly typed, ensuring that all manipulations are performed correctly. Algorithms are built-in for the involved computation and reasoning within the object and to handle the conditions and assumptions.
- Constructed objects may be easily modified, for example, by renaming symbols, changing parametric values, assigning new values to parameters, and adding or removing properties, conditions, and assumptions. Consistency checking after the modification may be done effectively and automatically.
- Basic geometric operations such as intersection, union, conversion, and calculation (of area, volume, offset, and convex hull, etc.) with constructed objects may be performed efficiently and correctly.
- Relations among different geometric objects may be declared and their consistency may be efficiently checked. With these relations, geometric statements, propositions, and theorems may be formulated using a simple syntax.
- Advanced methods and techniques are implemented to prove known geometric theorems, to discover new ones, to verify the consistency of geometric relations, and to derive new geometric properties.

- Symbolic geometric objects may be visualized and displayed on the screen. Automatically generated diagrams can be manipulated, modified, and animated with mouse clicks and dragging.
- Geometric knowledge (such as well-known facts, remarkable theorems, historical background, and literature information) can be represented and managed, allowing indexing and searching.
- Geometric uncertainty and degeneracy are handled systematically and efficiently. Degeneracy conditions may be generated, collected, and analyzed with different techniques and heuristics, and their geometric meanings may be interpreted in most cases.

We group Gœl into seven main components, a core and six modules of *functors*: constructors, modifiers, declarers, operators, reasoners, and visualizers, in accordance with the capabilities and features listed above. We will discuss our primitive ideas about each component: what it looks like and what it does. Some of these ideas have been explained in [9]. It is Kutzler [6] who first proposed careful translations of geometric statements into algebraic relations, taking degeneracy into account. We are not aware of any other similar work on the foundational aspect of geometry system design.

3 Gœl Components: Core and Construction

In this and the next section, we describe the basic elements of each Gœl component and explain what they can do. We will also address some of the problems and difficulties that may be encountered in the design and implementation of each component and provide specifications for some of the functors. The focus of this work is on the design issues of geometry software. Precise discussion on concrete algorithms and functions to be implemented, their applicability and limitation, and comparison of capability and performance between Gœl and other existing software systems cannot take place at this stage when the implementation of our system has just started.

3.1 Core

As in any language or system, there should be a *core* which has its own internal data and tree structures, syntax rules, and user interface. The core may handle input and output streams, interaction between different components, and interaction with external systems (e.g., interface with Maple for symbolic and algebraic computation). It may also manage directory structure, file opening and closing, memory and disk allocation, and foreground and background processing.

In the core of Gœl should also be implemented a number of basic categories such as `Point`, `Line`, `Segment`, `Triangle`, and `Circle` in two- and three-dimensional Euclidean space for geometric computation and reasoning. Basic data types such as `Boolean`, `List`, `Real`, `Array`, `Vector`, `String`, and `Table` should be defined and functions for handling these basic types should be implemented. In order that every geometric object is visualizable, an internal graphic representation of the object also need be maintained by the core.

A simple programming language grammar such as the condition syntax `if-else`, `while-do` repetition, `for` iteration, and `break` may be implemented and a parser is needed to analyze the input sequence of statements. As these programming elements are standard and available in almost all programming languages, they are not our main concern in the present study.

3.2 Constructors

Symbolic geometric objects such as points, lines, segments, triangles, circles, polygons, and algebraic curves and surfaces are the primary objects that Gool will manipulate. The construction of these objects is the first task that the system must carry out. We propose to group geometric objects into *classes* with data and (methods for) operations, and all objects being strongly typed. The data and operations for an object $\mathcal{O}$ may be partitioned into several entries detailed below.

- `representation`: different representations of $\mathcal{O}$ (for example, a triangle may be *represented* by its three vertices, or by two sides and their included angle, or by two angles and their included side, and a circle may be *represented* by its center and radius, or by its center and a point on the circle, or by three distinct points on the circle). Since different representations of $\mathcal{O}$ may correspond to different construction methods and operations, it is useful to include several representations in the constructor.
- `derivedObject`: geometric objects derived from $\mathcal{O}$ (such as the area, perimeter, centroid, incenter, orthocenter, circumcenter, circumcircle, and inscribed circle derived of a triangle).
- `condition`: conditions needed for the construction of $\mathcal{O}$, so that $\mathcal{O}$ is well defined (e.g., when a circle passing through three points is constructed, the nondegeneracy condition that the three points are not collinear is added, and when a circle is constructed with radius r, the condition $r > 0$ is added to the condition list automatically).
- `assumption`: assumptions imposed on $\mathcal{O}$ by the user (e.g., the radius r of a circle is greater than 2 and smaller than 5 and one side of a triangle is equal to another side of the triangle).
- `property`: basic properties about $\mathcal{O}$ (e.g., a triangle has the *properties* that the sum of its three internal angles is 180° and the sum of its two sides is greater than its third side).
- `knowledge`: well-known facts, remarkable theorems, literature information, and historical background about $\mathcal{O}$ (e.g., any side of a triangle is a segment, the three vertices of a triangle are usually assumed noncollinear, there are three escribed circles and one inscribed circles that are tangent to the three sides of a triangle, and an angle cannot be trisected with ruler and compass).
- `internalRelation`: geometric relations among subobjects and derived objects of $\mathcal{O}$ (e.g., the diameter of a circle is twice the radius of the circle and the area of an triangle is half of the product of the length of one side and the length of the altitude on that side).

- `externalRelation`: geometric relations between $\mathcal{O}$ and other objects (e.g., a circle is contact to another circle and tangent to a line and a triangle is located inside a circle).
- `method`: methods required for computation and reasoning with $\mathcal{O}$ (e.g., converting between different representations, computing derived objects, managing knowledge and properties, verifying consistency of conditions and assumptions, and handling internal and external relations).

As an example, we specify a constructor that constructs circles in plane Euclidean geometry. We use `Point`n`E`, `Line`n`E`, `Triangle`n`E`, `Circle`n`E`, etc. to denote the categories of points, lines, triangles, circles, etc. respectively in n-dimensional Euclidean geometry. The notation `o::T` means that `o` is an object of the category `T` or `o` is of type `T`. Moreover, | denotes the logical "or" and `Real` the type of real numbers.

Constructor `circle`

```
circle.var(args)::Circle2E {
    var = PPP | CR | CP
    args = A1, A2, ..., An
    Case PPP {
      n = 3
      A1, A2, A3::Point2E
      representation = [CR, CP, ...]
      derivedObject = [center::Point2E, radius::Real, diameter::Real,
                       area::Real, perimeter::Real, ...]
      condition = [not collinear(A1, A2, A3), ...]
      assumption = [ ]
      property = [ ]
      knowledge = [ ]
      internalRelation = [diameter = 2*radius, area = π*radius^2, ...]
      externalRelation = [ ]
      method = [Center, Radius, Area, ...]
        Center(args_c)::Point2E { ··· }
        Radius(args_r)::Real { ··· }
                ⋮
    }
    Case CR {
      n = 2
      A1::Point2E
      A2::Real
      representation = [PPP, CP, ...]
      condition = [A2 > 0]
                ⋮
    }
    Case CP {
      n = 2
      A1, A2::Point2E
      representation = [PPP, CR, ...]
```

```
        condition = [A2 ≠ A1]
                    ⋮
    }
}
```

When the class for a general circle is implemented, any concrete circle, symbolic or numeric, generic or specialized, is an instance of the general circle, obtained by giving values to some of the arguments of the class.

To implement a constructor, one has to deal with many technical issues such as data structure, constraint (condition, assumption, and relation) handling, and knowledge representation. We will discuss some of these issues in the last section.

3.3 Modifiers

After a geometric object is constructed, one may need to modify some part of the object: to change its parametric values, to assign new values to its parameters, and/or to add or delete assumptions. This will be done by *modifiers*. For a concrete geometric object constructed by a constructor, some entries such as `assumption`, `knowledge`, and `externalRelation` may be empty initially. New assumptions, knowledge, and external relations may be added to these entries and the existing ones may be removed or substituted by means of the modifiers.

Although there is no essential difficulty, it is not easy to modify a constructed geometric object. The change of a single parameter of the object may result in the change of other parameters and its subobjects and derived objects. So it has to be ensured that the modification does not cause any inconsistency. For example, if it has already been assumed that the radius of a circle is larger than 2 and smaller than 5, then it should not be allowed to add the assumption that the area of the circle is equal to 100. Moreover, a geometric object may have external relations with other geometric objects. This makes modification more complicated because modifying part of the geometric object may cause modifications for such relations or even some parts of the other objects. To deal with this kind of problems, one needs powerful mechanisms for consistency checking.

A geometric object $\mathcal{O}$ is said to be a *parent* of a symbol or another geometric object $\mathcal{P}$ if $\mathcal{P}$ is derived from $\mathcal{O}$ or $\mathcal{P}$ is a parameter in $\mathcal{O}$. The geometric object or parameter to be modified is called *target* and the modification is to assign a new *value* to the target. For example, let O be the center of a circle C. We may assign a value P to O; then C is a parent of O, O is the target, and P is the value to be assigned. A modifier may be defined as a class with the following operations.

- `collect`: collect basic information about the target, its parents, and the value. Such information includes the types, categories, value ranges, and definitions of these objects.
- `search`: search for the target to be modified, those geometric objects that are related to the target or its parents, and their relations and build up a temporary collection of geometric objects and relations.

- `ascertain`: from the temporary collection of objects and relations, ascertain which geometric objects and relations may be affected when the modification is made.
- `check`: check whether the type of the value matches that of the target and whether the modification may lead to a contradiction.
- `make`: make the modification (addition, deletion, renaming, substitution, etc., as appropriate) if the types match and no contradiction is derived, or report the types or the contradiction and the geometric objects and relations involved and ask the user to interact in order to continue the modification otherwise.

We do not give any example for the specification of modifiers. Since a target may have several parents that have relations with other geometric objects, searching for such objects and ascertaining which ones will be affected by the modification is a complicated process that needs effective algorithms. Consistency checking is also one of the tough problems for the implementation of modifiers.

4 Gœl Components: Computation and Reasoning

4.1 Operators

Operators may be defined as procedures that perform operations such as intersection, union, conversion, and calculation (of center, area, volume, offset, etc.) with geometric objects. As we have already emphasized, geometric objects under discussion (even after instantiation) are symbolic, the radius of a circle may still remain as a symbol without assigned numeric value and the vertex of a triangle may have symbolic coordinates. Such objects are indefinite in character and how to perform operations and computations with them efficiently and correctly is one of the main questions.

We view an operator as a function that maps several geometric objects to a single geometric object. It is nontrivial to determine the definition domain of the function and the type of its value because of the uncertainty of the symbolic objects. Consider for example the operator `intersection` of two geometric objects. Let `Object2E` denote the category of objects in plane Euclidean geometry. Since the intersection of a point and another geometric object in `Object2E` may be meaningless, the definition domain of `intersection` in plane Euclidean geometry is at most $(\texttt{Object2E} \setminus \texttt{Point2E}) \times (\texttt{Object2E} \setminus \texttt{Point2E})$. Two indefinite circles, without being given numeric values to the symbolic radii and coordinates of their centers, may coincide, may have no intersection, may intersect at two points or at a single point, and the two intersection points may be complex or real, depending on the size and location of the circles as well as the geometry-associated field. So the intersection of two indefinite circles may be an empty set, a single point, two points, or a circle. In general, it is impossible to give a yes or no answer to the question whether two indefinite circles intersect. The value returned by `intersection` may be a sequence of instances of `CondType`. This

kind of uncertainty makes type checking a delicate issue for symbolic geometric operations.

The following specification of `intersection` shows the complexity of operators and their implementation.

Operator `intersection`: `Object2E` $\times$ `Object2E` $\longrightarrow$ `Object2E`

```
intersection (A1::Object2E, A2::Object2E) :: Object2E {
     A1, A2 :: Circle2E | Line2E | Segment2E | ···
     Case A1, A2 :: Circle2E
       Intersection.CC(A1, A2)
     Case A1 :: Circle2E & A2 :: Line2E
       Intersection.CL(A1, A2)
     Case A1 :: Line2E & A2 :: Line2E
       Intersection.LL(A1, A2)
                  ⋮
     Intersection.CC(C1 :: Circle2E, C2 :: Circle2E) :: Object2E {
          if isIntersect(C1, C2) == true then {
            if C1 == C2 then
              return C1 :: Circle2E
            else if isContact(C1, C2) then {
                  ⋮
              return P1 :: Point2E
            }
            else {
                  ⋮
              return [P1::Point2E, P2::Point2E]
            }
          }
          else if isIntersect(C1, C2) == false then
            return Null
          else { // cannot determine whether C1 and C2 intersect
                  ⋮
            return new CondType[]{{P1::Point2E, isContact(C1, C2),
                    Circles C1 and C2 are contact},
                  {[P1::Point2E, P2::Point2E],
                    isIntersect(C1, C2) && not isContact(C1, C2),
                    Circles C1 and C2 intersect but are not contact},
                  {Null, not isIntersect(C1, C2),
                    Circles C1 and C2 do not intersect}}
          }
     }
                  ⋮
}
```

The above discussion is only for a simple case without any subsidiary conditions or assumptions. In reality, it is more complicated to compute the intersection of two geometric objects when conditions and assumptions are taken

into account. It may have already been assumed that the two objects do not intersect, so that their intersection set is empty. However, such an assumption may be made somewhere implicitly or is a consequence of other conditions and assumptions. Therefore, to draw the conclusion that the two objects do not intersect, formal reasoning is required. In fact, most of the operations on indefinite geometric objects require the support of practical reasoners (see Section 4.3).

4.2 Declarers

The main theme of geometry is to study properties of and relations among various geometric objects, so an ideal geometric software system should be capable of specifying and manipulating such *geometric relations*. In Gœl, geometric objects will interact each other through standard geometric relations specified by *declarers*. A declarer is a procedure that implements a general geometric relation with data and information entries and internal methods for operations, similar to the predicates defined in GEOTHER [9].

Besides some of the operations such as `search`, `ascertain`, and `check` in modifiers, declarers also need information entries about geometric meanings and algebraic expressions of the declared relations. Data, information entries, and (methods for) operations in declarers may be grouped as follows.

- `value`: value to be declared (e.g., `intersect`$(\mathcal{O}_1, \mathcal{O}_2)$ may be declared to have value `Null`, $\mathcal{P}$ :: `Point2E`, or another geometric object and `isIntersect` $(\mathcal{O}_1, \mathcal{O}_2)$ may be declared to be `true` or `false` :: `Boolean`).
- `geometricMeaning`: geometric meaning of the declared relation (e.g., the declaration of `intersect`$(\mathcal{O}_1, \mathcal{O}_2)$ to be $\mathcal{P}$ :: `Point2E` means that the two objects $\mathcal{O}_1$ and $\mathcal{O}_2$ intersect at point $\mathcal{P}$).
- `algebraicExpression`: algebraic representation of the declared relation (e.g., for two circles $\mathcal{C}_1$ and $\mathcal{C}_2$ centered at $[x_1, y_1]$ and $[x_2, y_2]$ with radii r_1 and r_2 respectively, the algebraic expression of the relation `isContact`$(\mathcal{O}_1, \mathcal{O}_2)$ = `true` is $(r_1 \pm r_2)^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$).
- `method`: methods for conversion, coordinatization, modification, and translation (e.g., converting geometric expressions to algebraic expressions and vice versa, coordinatizing the parameters in the declared relation, modifying the coordinates of the parameters, and translating the relation into statements in natural languages).
- `collect`, `search`, `ascertain`, `check`, `make`: similar to those in modifiers.

Here is an example specification of a declarer that specifies the collinearity of three points.

Declarer `collinear`

```
collinear (args) = val :: Boolean {
     args = A1 , A2 , A3 :: Point2E
     value = val :: Boolean
     Case val == true {
       geometricMeaning = [
```

```
                the three points A1, A2 and A3 are collinear]
    algebraicExpression = [
      A1_x A2_y - A1_x A3_y + A2_x A3_y  - A2_x A1_y + A3_x A1_y - A3_x A2_y = 0]
                ⋮
  }
  Case val == false {
    geometricMeaning = [
                the three points A1, A2 and A3 are not collinear]
    algebraicExpression = [
      A1_x A2_y - A1_x A3_y + A2_x A3_y  - A2_x A1_y + A3_x A1_y - A3_x A2_y  ≠ 0]
                ⋮
  }

  method = [Let, Coordinate, Convert, Translate, Modify, ...]
    Let(args_l) {  ···   }
    Coordinate(args_c) {  ···   }
                ⋮
  }
}
```

Modifiers are used to modify geometric objects locally, while declarers declare relations among different geometric objects globally. The relations produced by a declarer are associated to all the involved geometric objects. Such relations have to be well organized to facilitate the management and consistency checking.

4.3 Reasoners

As we have mentioned, several components of Gool need to handle geometric constraints and to check their consistency. Moreover, one of the main themes of geometry is to study relations among various geometric objects. Once fundamental geometric objects are constructed with conditions and assumptions, we may use a simple syntax to formulate geometric statements, propositions, theorems, etc. Advanced methods and techniques need be developed and used to handle them: checking the consistency of geometric constraints, verifying geometric properties, proving, generalizing, or discovering geometric theorems, and deriving new geometric relations and properties. We refer to functions implemented for these tasks as geometric *reasoners*.

Geometric reasoning may be performed locally within one geometric object (for example, verifying the consistency of assumptions) or globally about relations among different geometric objects. The language Gool will provide a basis for further investigations and studies on computer-aided geometric reasoning. As geometric conditions, assumptions, and relations involve both algebraic equations and inequalities, computation and reasoning have to be performed over the field $\Re$ of real numbers. This is an outstanding problem of research in the area of computational real algebra and geometry.

As a concrete example, we now consider the specification for `theoremProver`, a reasoner whose arguments are the hypothesis and conclusion of a geomet-

ric theorem. The hypothesis and conclusion may consist of geometric relations declared by declarers or algebraic relations (equations, inequations, and/or inequalities). Such relations will be transformed into a standard representation, for instance, like the predicate representation of geometric statements in GEOTHER [9], to facilitate different kinds of manipulations. The reasoner will be able to translate the standard representation into geometric statements in different natural languages, into logic formulas, and into algebraic expressions, to generate geometric diagrams automatically from the standard representation, and to prove the theorem.

The reasoner `theoremProver` needs several elements. For instance, there should be one or several effective procedures that can prove the class of geometric theorems in questions. We refer to such procedures as *provers*. For elementary geometry, one may implement advanced algebraic methods based on Wu–Ritt's characteristic sets [10], Buchberger's Gröbner bases [5, 7], and other triangularization techniques [8] or take the five provers available in GEOTHER [9]. Other elements of `theoremProver` are discussed briefly as follows.

- `theorem`: standard representation of the theorem to be proved, which is obtained from the input arguments by the `standardizer`.
- `subsidiaryCondition`: subsidiary conditions for the theorem to be true or meaningful (usually they are nondegeneracy conditions collected and analyzed by the `finder`).
- `standardizer`: transform different kinds of arguments into a standard representation of the theorem with hypothesis and conclusion.
- `translator`: translate the standard representation into geometric statements in natural languages, into logic formulas, or into algebraic expressions.
- `prover`: provers implemented on the basis of algebraic methods for proving the theorem.
- `finder`: procedures for handling (collecting, analyzing, and interpreting) nondegeneracy conditions.

Denote by `Theorem` the category of geometric theorems in standard representation and `[Relation2E]` the category of lists of geometric relations and algebraic equations and/or inequalities expressing geometric relations in plane Euclidean geometry. A specification of `theoremProver` looks like the following.

Reasoner `theoremProver`

```
theoremProver( args, arg-opt) :: Boolean {
     args :: Theorem | [Relation2E]
     arg-opt :: String = AutoW | AutoG | AutoGC | AutoT | Manual
     if standardizer( args )[0] == false then
       System.exit(Error, standardizer( args )[1])
     else
       theorem = standardizer( args )[1]
     subsidiaryCondition = [ ]
     Case ( arg-opt == AutoW )
       apply( Wprover, Wfinder )
```

```
    Case ( arg-opt == AutoG )
      apply( Gprover, Gfinder )
                    ⋮
    Case ( arg-opt == Manual )
      // use a prover selected by the user
    standardizer( arg ) :: Array {
      if args :: Theorem then return [true, args]
      else if args :: [Relation2E] then
           standardize( args )
           return [true, standard_representation]
      else return [false, wrong arguments]
    }
    apply( arg_prover, arg_finder ) {
      arg_prover = Wprover | Gprover | GCprover | Tprover | ···
      arg_finder = Wfinder | Gfinder | GCfinder | Tfinder | ···
      arg_prover( theorem )
      arg_finder()
    }
    finder = [Wfinder, Gfinder, GCfinder, Tfinder, ···]
      Wfinder() :: Void {
        collect()
        ascertain()
        handle()
        subsidiaryCondition.add( ··· )
      }
                    ⋮
    prover = [Wprover, Gprover, GCprover, Tprover, ...]
      Wprover( args_T ) :: Boolean {  ···  }
                    ⋮
    method = [Let, Coordinate, Translate, Algebraic, Logic, ...]
      Let( args_l ) :: Void {  ···  }
      Algebraic( args_c ) :: Void {  ···  }
                    ⋮
}
```

There are many technical issues for the implementation of geometric reasoners. We do not discuss those issues and refer to [2, 9] for more information about geometric theorem provers and their implementation.

4.4 Visualizers

Without diagrams and visualization one cannot do geometry geometrically. The capability of visualizing geometric objects is one of the key factors that can make Gool attractive. Recall that geometric objects under manipulation are usually symbolic and indefinite, while for their visualization one needs to take numeric values for the symbolic parameters in each concrete instance. The parametric data, information, and relations associated with the objects make it possible to

keep any diagram of these objects general and symbolic, in the sense that the diagram displayed on the screen may move dynamically according to the change of values of the symbolic parameters (but satisfies the geometric constraints). Changing the parametric values with mouse clicks and dragging thus allows the user to modify and animate the diagram very easily at the running time. It is obvious that visualization of indefinite geometric objects involves both numeric and symbolic computation with algebraic equations and inequalities (over the field of rational numbers, its algebraic extensions, and $\Re$). Usually one wishes to visualize only the real part of the geometric objects, so effective methods for computation over $\Re$ are indispensable and need be incorporated in any practical implementation of Gool. Without entering into the details of specification for visualizers, we list below some of the entries that should be contained in each visualizer.

- `geometricObject`: geometric objects to be visualized in one diagram, their types and categories, and the arguments with which the objects are defined.
- `constraint`: (algebraic and/or geometric) constraints on the geometric objects in the diagram (i.e., different known relations among the objects and possibly some unknown relations to be derived).
- `property`: global properties of the diagram such as its position, size, color, label, transparency, and font size).
- `method`: methods for modifying the diagram such as converting the constraints, adding or deleting geometric objects, and modifying global properties of the diagram.
- `operation`: operations on the diagram for modification and animation with mouse clicks and dragging.

5 Implementation and Experiments

To study and test the proposed language, we have implemented an experimental version of Gool. Although a considerable amount of effort has been made, the current version of the system is still very primitive. It is now possible to construct geometric objects such as circles, lines, and points in Euclidean plane, to modify the constructed objects (e.g., changing the values of their parameters), to perform basic operations such as intersection and calculation of area and perimeter, and to verify geometric relations such as contact and tangent between different objects. Some other capabilities such as acquiring geometric information and algebraic expressions have also been implemented. Figure 1 is the snapshot of a running session of the system.

Given the design specifications explained in the previous sections, the implementation of Gool is quite straightforward, but as usual it is a time-consuming task and requires special care to handle all the technical details at the programming level. In what follows, we discuss some general implementation issues. A few technical difficulties and challenges will be addressed in Section 6.2.

To implement Gool, we need a programming language that has convenient syntax structure, high efficiency of execution, multithreading support, ability to interact with external systems, and rich graphic primitives. Java is such a

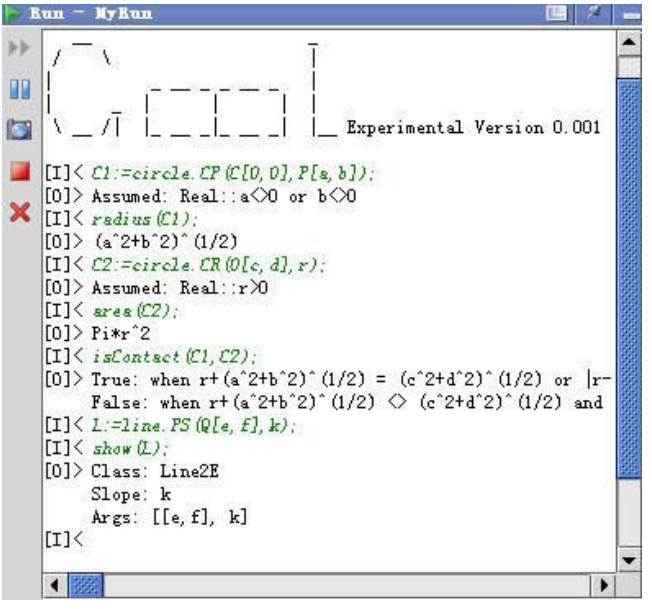


Fig. 1. Running session of experimental Gool

language that satisfies most of these requirements, so it has been chosen as our programming language. We have implemented a simple statement parser that can analyze input commands whose names are fixed but arguments are changeable. The user can interact with the system by means of this simple parser. The implementation of a more sophisticated statement parser is now in progress.

An interface with Maple has also been implemented to deal with symbolic and algebraic computation needed in Gool: computation requests are passed from Gool to Maple, and the results of computation carried out in Maple are returned to Gool. The performance of the interface is reasonable and can satisfy our requirements at this early stage, but sometimes it works in low efficiency. As frequent interaction between Gool and Maple is time-consuming, we have implemented a small module for simple algebraic computations such as $b + a - 2\,a = b - a$; thus only complicated computations are sent to Maple.

As symbolic computation is often complicated and time-consuming, we divide the processes in Gool into two groups: foreground processes and background processes (similar to a time-sharing operating system). Gool tasks are also divided into two kinds: active tasks and inactive tasks. The former are the main tasks that the user wants the system to carry out, and the latter are secondary tasks produced automatically by the former. For example, when a geometric object is being constructed, its construction is an active task, but computing its subobjects and derived objects is an inactive task. While the active tasks engross the shell, the inactive tasks remain in the task thread waiting for the idlesse of the shell. When the shell is idle, the inactive tasks are brought into the shell and executed. So tasks in Gool are well organized and harmonized.

To give the reader a clearer idea about Gool, we reproduce some of our experiments with its current version. For readability, the output shown below has been reformatted using LaTeX.

```
[I] < C1 := circle.CR(O[1,1], r);
[O] > Assumed r::Real > 0                                   C1::Circle2E

[I] < algebraic(C1);
[O] > (x-1)^2+(y-1)^2=r^2                                       ::PolyEq

[I] < C2 := circle.PPP(A[0,0], B[a,0], C[0,b]);
[O] > Assumed ab::Real ≠ 0                                  C2::Circle2E

[I] < show(C2);
[O] > Class: Circle2E
      Center: [a/2,b/2], Radius: √(a^2+b^2)/2
      Args: [[0,0],[a,0],[0,b]]                                  Void

[I] < perimeter(C2);
[O] > π√(a^2+b^2)                                               ::Real

[I] < L1 := line.PP(P1[a,b],P2[c,d]);
[O] > Assumed (c-a)^2+(d-b)^2::Real ≠ 0                     L1::Line2E

[I] < isTangent(C2, L1);
[O] > True: when [b(d-b)-a(c-a)]^2/(4[(c-a)^2+(d-b)^2]) = 0
      False: otherwise                                      ::Boolean

[I] < let(b = a);
[O] >                                                          Void

[I] < intersection(C1, C2);
[O] > C1 if a = 2 and r = √2                                ::Circle2E
      [(r^2-2±√D)/(2(a-2)), (r^2-2∓√D)/(2(a-2))] if D ≥ 0    ::Point2E
      ∅ otherwise                                           ::Null
      where D = -(r^2-2)(r^2-2a^2+4a-2)
```

The output results in Figure 1 are printed in one-dimensional format (similar to the Maple syntax) and it is not yet possible to display mathematical expressions like $\sqrt{a}$ graphically. It is expected that future versions of Gool will provide a graphic user interface with flexible command lines for input and output, dynamic visualization of geometric objects, and graphic display of mathematical symbols and expressions. We are also going to provide the possibility of outputting the results in LaTeX source format, so that they can be processed with LaTeX and transformed into HTML format automatically.

From the implementation of the Gool prototype, we see that almost all the specifications and techniques we proposed in Sections 3 and 4 are practical and

they (in particular the way to handle uncertainty) work quite well in the prototype. However, there are still some techniques, e.g., for consistency checking and advanced reasoning, that remain to be tested.

6 Difficulties and Challenges

In the previous sections, we have presented the capabilities and features of Gool as well as the specifications of some functors and preliminary experiments. Readers interested in the design and implementation of languages and systems for doing geometry might wish to know the difficulties and challenges we have encountered in the development of Gool. In this section, we discuss some of them at the design, programming, and computational levels.

6.1 Design Level

At this level, the main challenge we faced is how to formalize geometric construction, computation, and reasoning and what programming model should be chosen to realize a formalism on modern computer.

Formalizing Geometric Constructions with Case Distinction
As we have mentioned repeatedly, the main difficulty of dealing with symbolic geometric objects comes from their uncertainty and degeneracy; the latter has been widely known to the research community of automated geometric theorem proving. In order to ensure that every symbolic geometric object be constructed rigorously and completely, we have adopted a formalism that permits us to define the object by enumerating all distinct cases. This new formalism applies equally to the declaration of geometric relations and the operation among geometric objects and can handle the degeneracy issue satisfactorily.

To explain our formalism, let $C_1, \ldots, C_t$ be t (≥ 1) sets of geometric or algebraic conditions and for each i let O_i be a symbolic geometric object of type $\mathtt{T}_i$ under the condition C_i. A *composed geometric object* O of $O_1, \ldots, O_t$ is defined as follows:

$$O = \begin{cases} O_1 :: \mathtt{T}_1 & \text{if } C_1, \\ O_2 :: \mathtt{T}_2 & \text{if } C_2, \\ \quad\vdots & \quad\vdots \\ O_t :: \mathtt{T}_t & \text{if } C_t. \end{cases}$$

In the above definition, "if C_i" is understood as "if the condition C_i is satisfied." In this formalism, a symbolic geometric object is always given with type and condition. The condition may be omitted if it is a tautology. The definition of composed geometric objects is similar to that of piecewise functions and a composed object may consist of one or several symbolic geometric objects of different types under different conditions. We call O_i a *piece* of O and C_i the *side condition* of O for O_i ($1 \leq i \leq t$), and we refer to the formalism of dealing with geometric objects, relations, and computations by distinguishing different

cases as *case distinction*. The reader may have already noticed the use of this formalism in our specifications and experiments in the previous sections.

By means of case distinction, the results of basic operation (e.g., intersection) and computation (e.g., of the area or volume) of composed geometric objects are still composed geometric objects and we may also have *composed geometric relations* among composed geometric objects. Therefore, the remaining problem is how to manipulate composed geometric objects and relations effectively. It is clear that performing computation and reasoning with composed geometric objects requires powerful mechanisms and algorithms to deal with complicated side conditions and thus is substantially complex. However, the formalism of case distinction allows us to construct and manipulate geometric objects rigorously and thus provides a theoretical basis for our design and implementation of a robust software system for doing geometry. Some of the techniques from constraint logic programming [4] may be used to refine and implement our formalism.

Geometric Object-Oriented vs. Geometric Object-Based
Having adopted the formalism of case distinction, we had to chosen a basic programming model for specifying and implementing various modules, constructions, and functions for our system or language. The basic elements of the language are symbolic geometric objects and all operations, computations, and reasoning are oriented around these objects. Every symbolic geometric object need be strongly typed, should have an internal data structure for its representation, and may be instantiated to specialized geometric objects. These requirements lead to the notion of class representation of objects that occurs in generic OOL: we may define a symbolic geometric object by means of a class in OOL. It is therefore natural to choose OOL as the basic programming model for our language Gœl. The OOL paradigm also allows us to effectively implement the formalism of case distinction. Another reason for our choice is that generic OOL has the property of strong hierarchy that subobjects can be derived from the parent object, inheriting and keeping all the properties of the parent object. Subobjects may also have new properties, and all the inherited properties and the new properties can be inherited by subsubobjects of the subobjects. Geometry also has this hierarchy property: e.g., ellipses can inherit quadratic curves, and circles can be derived from ellipses. However, the hierarchy property of geometry has not been well understood in the context of formal specification and programming. We have been trying to identify and structure the hierarchy of geometric objects in two- and three-dimensional Euclidean space for the development of our system, but we have not yet worked out a reasonable frame. Further work on this aspect is in progress.

In fact, the reader may also consider the proposed language as geometric-object-based rather than geometric-object-oriented since all operations and manipulations are performed on the constructed geometric objects. However, our language share many features with generic OOL and we prefer to refer to it as a geometric-object-oriented language.

6.2 Programming Level

In Section 5, we have discussed several implementation issues which are more restricted to the programming aspects independent of geometry. Now we come to some issues that are more geometry-related.

Data Structure and Organization
To realize the formalism of case distinction, we have created a data structure, named `CondType`, that is represented by a triplet [`value, cond, anno`]. An instance CT of `CondType` means that CT.`value` is attained when CT.`cond` is satisfied. In general, the first two entries combined have strong geometric meanings, so the third entry is added to collect the meanings (represented in nature language, logic formulae, etc.) for further use. As an instance can represent only one component (one case), we build an array to cover all the cases. The reader may revisit the specification of the operator `intersection` in Section 4.1 and our experimental session in Section 5.

There are usually many external relations for a geometric object, so a good data structure should be used to ensure convenient storing and efficient searching and accessing. We have chosen to use a hash table to store the *externalRelation* entries of a geometric object O. The key of an entry in the table is a preserved relation predicate, and the corresponding element is an array — whose length is changeable (Vector in Java) — of geometric objects which have relations with O represented by the predicate. Types `Hashtable` and `Vector` are built in Java, and available miscellaneous operations on them make it easy to access the *externalRelation* entries.

Moreover, a geometric constraint may be represented by a logic formula, by algebraic expressions, by a predicate specification, or by a nature-language statement, and each representation has its own advantages and applications. Almost all of them will be used in different places for different purpose, so an ideal data structure for geometric constraints should contain all these representations. In the Gool prototype, a constraint is represented by a quaternion. This representation is overequipped and not convenient for frequent usage.

From Specification to Java Code
We chose Java as our programming language for Gool because of its many attractive advantages. These advantages have helped us to tackle several tough issues at the implementation level. For example, Java's multithreading support makes it feasible to take foreground and background processing heuristics into practice and flexible interaction with external systems makes our implementation of an interface with Maple easy. With the help of Java specialities, transformation from specification into Java code is quite straightforward. Here we only point out a couple of minor tricks to which one may need to pay attention at the programming level.

Usually, after an object is constructed with Java class, each of its entries is unchangeable. If one wants to modify any part of the constructed object, one has to reconstruct the object with new parameters. In the Gool prototype, we use real-time heuristics to avoid the reconstruction. When a geometric object is

constructed with some parameters, we create a symbolic link to the parameters for each entry without assigning any value. Based on the link and taking the real-time values of the parameters, we can get the value of any entry.

There may be many derived objects that need be constructed, but not all of them will be used in the following computations, so there is no need to construct all the derived object along with the construction of a target object. Here we use time-sharing heuristics: when a geometric object is being constructed, computations of its derived objects will be inactive processes with different priorities which are given on the basis of their significance waiting in the threads line. When the shell is idle, the process with highest priority will be brought in for computation. The user can bring inactive processes into the shell manually.

6.3 Computational Level

To make computations efficient, we need to develop sophisticated algorithms and use various heuristics to cope with problems caused by uncertainty and degeneracy. The heuristics may help reduce the practical cost that we have to pay for theoretical rigorousness and they work quite well in many cases.

Quantitative Complexity
The formalism of case distinction reduces the qualitative difficulty of dealing with symbolic geometric objects to the quantitative complexity of doing geometric and algebraic computation. The distinction of cases may take place in construction, operation, and computation and thus may introduce many pieces of geometric objects and relations with complicated side conditions. The manipulation of such composed geometric objects and relations will involve extensive verification and examination of the side conditions and may be very expensive computationally. For instance, for comparing two indefinite points, there are three cases to return: same, not same, and undetermined, and the last case consists of two cases: when the two points are the same and when they are not the same. The quantitative complexity of composed geometric objects and relations increases rapidly along with the manipulation.

Algorithmic Sophistication
To handle the quantitative complexity of composed geometric objects and relations with different representations for their pieces and side conditions, one must design and implement highly efficient algorithms for the involved computations. Such algorithms have to combine advanced techniques of handling algebraic relations and to incorporate well-studied heuristics. In general, side conditions in distinct cases may be expressed as implicit polynomial equations and inequalities. The problem of checking the consistency and verify the relations of side conditions may be reduced to the problem of quantifier elimination over $\Re$. The latter can be solved in principle, for example, by the method of cylindrical algebraic decomposition (CAD) proposed by Collins and improved by Hong and others [1]. However, in practice the CAD method is not efficient enough for dealing with side conditions of composed geometric objects and relations, and the output of the method is often too complicated to be subsequently used (according to our preliminary experiments).

Side conditions usually contain many relations, but each of them may be quite simple (e.g., linear or quadratic with a very few terms). Therefore, heuristic simplification of the conditions must be done before any advanced method is applied. Moreover, one should handle the equality relations by using more specialized methods such as an adapted version of the methods of triangular sets and Gröbner bases prior to the use of CAD-type methods. It is not difficult to imagine how sophisticated an efficient algorithm that integrates available techniques to deal with geometric constraints is. One of our next tasks is to design and implement such an algorithm; this will be discussed in Section 7.1.

Computational Difficulty

Even having an efficient algorithm implemented to deal with geometric constraints generated from side conditions, one still has to face the computational difficulty inherent in geometric problems themselves. It is well known that systems of polynomial equations and inequalities, called *semi-algebraic systems*, are difficult to solve. We believe that an adequate combination of the existing methods and tools will allow us to approach the involved computations for fundamental objects and relations in plane Euclidean geometry, where constraints are usually linear and quadratic. However, when working with geometry in high-dimensional space, non-Euclidean geometry, or geometric relations of high degree, we will encounter various difficulties for the involved computations. The existing methods must be improved and new techniques have to be introduced. Moreover, when coming to geometric diagram generation and animation, we will need to handle geometric constraints dynamically in real time and more computational problems will occur.

Finally, we take the famous Thébault–Taylor theorem as an example to illustrate the process of handling geometric constraints.

1. Construct geometric objects and declare relations:

```
ABC:=triangle.PPP(A,B,C); AB:=line.PP(A,B);
AC:=line.PP(A,C); BC:=line.PP(B,C); D:=point(); let(onLine(D,BC)=true);
AD:=line.PP(A,D); C0:=CircumCircle(ABC); C1:=circle.CR(C1,r1);
C2:=circle.CR(T2,r2); assume(T2≠T3); C3:=circle.CR(T3,r3);
let(tangent(C2,AD)=true,tangent(C2,BC)=true,contact(C2,C0)=true);
let(tangent(C3,AD)=true,tangent(C3,BC)=true,contact(C3,C0)=true);
let(tangent(C1,AB)=true,tangent(C1,BC)=true,tangent(C1,AC)=true);
areCollinear(T1,T3,T3);
```

2. Collect constraints and build the constraint set:

```
{not collinear(A,B,C), A≠B, B≠C, C≠A, A≠D, onLine(D,BC),
tangent(C1,AB), tangent(C1,BC), tangent(C1,AC), tangent(C2,AD),
tangent(C2,BC), contact(C2,C0), T2≠T3, r2>0, r3>0, tangent(C3,AD),
tangent(C3,BC), contact(C3,C0)}
```

3. Remove redundant conditions:

   Since `A≠B`, `B≠C` and `C≠A` are consequence of `not collinear (A,B,C)`, and `A≠D` is a consequence of `not collinear(A,B,C)` and `onLine(D, BC)`, they are removed.

4. Translate into algebraic expressions:

```
Reassign coordinates: A=[u2,u3], B=[-u1,0], C=[u1,0], D=[x1,y1], T2=[x2,y2],
                      T3=[x3,y3], T1=[x4,y4]
onLine(D,BC) -> y1=0
tangent(C1,AB) -> r1^2*(u3^2+(u1+u2)^2)-((x4-u2)*u3-(u1+u2)*(y4-u3))^2=0
tangent(C1,AC) -> r1^2*(u3^2+(u1-u2)^2)-((x4-u2)*u3+(u1-u2)*(y4-u3))^2=0
tangent(C1,BC) -> r1^2-y4^2=0
tangent(C2,AD) -> r2^2*((y1-u3)^2+(x1-u2)^2)-((y1-u3)*(x2-u2)-(x1-u2)*
                  (y2-u3))^2=0
tangent(C2,BC) -> r2^2-y2^2=0
contact(C2,C0) -> (r0+r2)^2-(x2-a)^2-(y2-b)^2=0 || (r0-r2)^2-(x2-a)^2-
                  (y2-b)^2 =0
tangent(C3,AD) -> r3^2*((y1-u3)^2+(x1-u2)^2)-((y1-u3)*(x3-u2)-(x1-u2)*
                  (y3-u3))^2=0
tangent(C3,BC) -> r3^2-y3^2=0
contact(C3,C0) -> (r0+r3)^2-(x3-a)^2-(y3-b)^2=0 || (r0-r3)^2-(x3-a)^2-
                  (y3-b)^2=0
not collinear(A,B,C) -> 2*u1*u3<>0
T2<>T3 -> x2-x3<>0 || y2-y3<>0
r2>0, r3>0
```

where the center [a, b] and radius r0 of C0 are known.

The theorem may be proved by deciding whether the conclusion relation is a formal consequence of the hypothesis constraints, a job that can be submitted to one of the algebraic provers.

7 Future Work

A complete implementation of the proposed system Gœl requires an enormous amount of effort and is our long-term project of research and development. Much work remains and a number of challenging problems have to be settled. We discuss some of these problems and the encountered difficulties that have not yet been completely overcome. Some of them have been mentioned previously and are on the top of our list of problems for future research.

7.1 Handling Geometric Constraints

From the previous discussions, we see that conditions and assumptions on geometric objects have to be considered in almost all the components of Gœl. It is essential to have an effective way to handle such conditions and assumptions, called *geometric constraints*. Not only powerful algebraic methods are needed for computation and reasoning with polynomial equations and inequalities, but also the geometric meanings of the constraints have to be conserved during the computation. In the current version of Gœl, a constraint is represented by a quaternion, but this representation is somewhat overequipped. So a better data structure, probably using hash tables, should be designed to represent the constraints both algebraically and geometrically with heuristic translation from one representation to the other.

Some assumptions, for example, $a > 0$ and $-a < 0$, may have the same meaning, but are in different forms. A simple, heuristic, and efficient simplification process should be undertaken regularly to remove redundancy in order to keep the set of constraints as compact as possible. In our current implementation of Gool, a set of criteria is used for constraint simplification in simple cases. More advanced techniques are required to deal with complicated cases.

One of the main problems, as mentioned before, is to check the consistency of (real) geometric constraints. This problem may be solved in principle by using the existing methods of semi-algebraic system solving over $\Re$. Those methods are known to have high computational complexity, so a key issue is practical efficiency. The numbers of geometric constraints and of variables in our case are usually large, but many of the constraints may be quite simple. It is therefore necessary to adapt the known methods with new and heuristic techniques for our purpose of efficient geometric constraint handling (simplification and consistency checking). We are taking an incremental approach: first use heuristic simplification, then try the methods of triangular sets and Gröbner bases to handle equality constraints, and finally apply the complete method of CAD.

7.2 Managing Geometric Knowledge

There is no essential difficulty but quantitative complexity for geometric knowledge management. An adequate data structure has to be designed to represent geometric knowledge including well-known facts, remarkable theorems, and literature information, so that the existing database technology may be used. Examples of questions we may ask are what type of knowledge and information should be collected, how to organize and structure them, and how to formalize geometric statements, so that they can interact effectively with other components in a uniform environment. We propose a mixture of documenting text and formal representation of geometric objects and statements, uniform with those in the declarers, reasoners, and visualizers, together with reference keys, indices, and URL links. This will allow easy cross-referencing, searching, sorting, translating, and reformatting. A logical framework may be identified to formalize and model the evolution of geometric knowledge.

It is expected that the system will be able to generate interactive documents automatically from the built-in geometric knowledge and the results of computation and reasoning. Any geometric fact, proposition, or theorem in the documents can be verified by the reasoners with a simple mouse click and automatically generated diagrams may be integrated into the documents. Web links will direct the user to the original sources of information.

7.3 Keeping Gool Geometric

The high interest and attraction of geometry lie largely in its intuition and figures. It is important that any geometric software system should maintain the usual intuition and features of geometry. As an advanced tool, the system should assist the user to do geometry more easily, more efficiently, and more

productively by using the computing and graphic power of modern computers. However, most of the computations in Gœol are based on algebraic methods, which may make geometric intuition and information lost. Therefore, we need to conserve the geometric meanings of the symbolic parameters and algebraic expressions as much as possible. For example, when "`r :: Real` >0" is assumed (as in the Gœol session at the end of Section 5), the information "`r` is the radius of the circle `C1`" should be stored in the system and may be displayed whenever needed.

In general, it is not possible to interpret the geometric meaning of a randomly generated algebraic expression. Nevertheless, the algebraic expressions occurring in the computation for a geometric problem are obtained successively from variables and initial expressions that have geometric meanings. By tracing the computational steps together with search and comparison, one can figure out the geometric meanings of meaningful algebraic expressions in many cases. For instance, the expression of the area of a triangle in terms of the coordinates of its vertices is rather large and it is difficult to know its geometric meaning only from the expression itself without any additional information. However, if the computational process (i.e., how the expression was obtained) is traced, then it is no longer a problem to know that the expression represents the area of the triangle.

The second author has used a number of heuristics for interpreting the geometric meanings of algebraic nondegeneracy conditions in GEOTHER [9]. These heuristics work quite well and may be generalized to deal with more complicated cases. We believe that a good heuristic imitation of what geometers usually do together with the support of algebraic methods and current computing technology will make Gœol a powerful and intuitive geometric system that may compete with expert geometers in terms of capability, productivity, and intelligence. We are highly motivated to work on the system and the problems involved.

Acknowledgments

The authors wish to thank one of the referees whose suggestion leads to the addition of Section 6. This work is supported partially by the National Key Basic Research Project 2004CB318000 of China.

References

1. Brown, C. W., Hong, H.: QEPCAD — Quantifier elimination by partial cylindrical algebraic decomposition. http://www.cs.usna.edu/~qepcad/B/QEPCAD.html (2004).
2. Chou, S.-C., Gao, X.-S., Liu, Z., Wang, D.-K., Wang, D.: Geometric theorem provers and algebraic equation solvers. In: *Mathematics Mechanization and Applications* (X.-S. Gao and D. Wang, eds.), pp. 491–505. Academic Press, London (2000).
3. Hilbert, D.: Grundlagen der Geometrie. Teubner, Stuttgart (1899).
4. Jaffar, J., Maher, M. J.: Constraint logic programming: A survey. *J. Logic Program.* **19/20**: 503–581 (1994).

5. Kapur, D.: Using Gröbner bases to reason about geometry problems. *J. Symb. Comput.* **2**: 399–408 (1986).
6. Kutzler, B.: *Algebraic Approaches to Automated Geometry Theorem Proving*. Ph.D. thesis, RISC-Linz, Johannes Kepler University, Austria (1988).
7. Kutzler, B., Stifter, S.: On the application of Buchberger's algorithm to automated geometry theorem proving. *J. Symb. Comput.* **2**: 389–397 (1986).
8. Wang, D.: Elimination procedures for mechanical theorem proving in geometry. *Ann. Math. Artif. Intell.* **13**: 1–24 (1995).
9. Wang, D.: GEOTHER 1.1: Handling and proving geometric theorems automatically. In: *Automated Deduction in Geometry* (F. Winkler, ed.), LNAI 2930, pp. 194–215. Springer-Verlag, Berlin Heidelberg (2004).
10. Wu, W.-t.: *Mechanical Theorem Proving in Geometries: Basic Principles* (translated from the Chinese by X. Jin and D. Wang). Springer-Verlag, Wien New York (1994).

Spatial Planning and Geometric Optimization: Combining Configuration Space and Energy Methods

Dmytro Chibisov, Ernst W. Mayr, and Sergey Pankratov

Institut für Informatik, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
chibisov@in.tum.de, mayr@in.tum.de, pankrato@in.tum.de

Abstract. In this paper, we propose a symbolic-numerical algorithm for collision-free placement and motion of an object avoiding collisions with obstacles. The algorithm is based on the combination of configuration space and energy approaches. According to the configuration space approach, the position and orientation of the geometric object to be moved or placed is represented as an individual point in a configuration space, in which each coordinate represents a degree of freedom in the position or orientation of this object. The configurations which, due to the presence of obstacles, are forbidden to the object, can be characterized as regions in this configuration space called configuration space obstacles. As will be demonstrated, configuration space obstacles can be computed symbolically using quantifier elimination over the reals and represented by polynomial inequalities. We propose to use the functional representation of semi-algebraic point sets defined by such inequalities, so-called R-functions, to describe nonlinear geometric objects in the configuration space. The potential field defined by R-functions can be used to "move" objects in such a way as to avoid collisions. Introducing the additional function, which forces the object towards the goal position, we reduce the problem of finding collision free path to a solution of the Newton's equations, which describes the motion of a body in the field produced by the superposition of "attractive" and "repulsive" forces. These equations can be solved iteratively in a computationally efficient manner. Furthermore, we investigate the differential properties of R-functions in order to construct a suitable superposition of attractive and repulsive potentials.

1 Introduction

Many practical geometric problems for industrial applications deal with placing and moving an object without colliding with nearby objects. The intricate nature of such problems manifests itself in enhanced computational complexity, whereas in industrial applications these problems must often be solved in real time. In the present paper, we consider two main types of spatial planning problems in a common framework:

H. Hong and D. Wang (Eds.): ADG 2004, LNAI 3763, pp. 156–168, 2006.

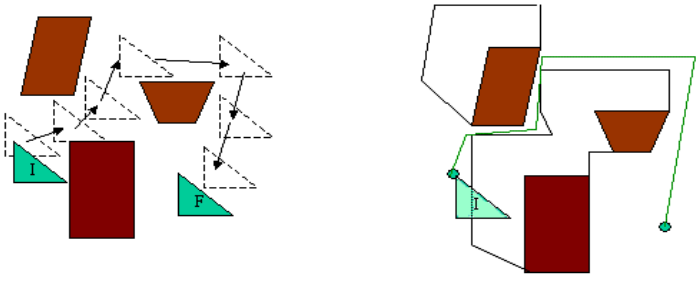


Fig. 1. Configuration space approach - enlarging obstacles: a problem of motion planning is reduced to finding a curve in the configuration space

- **FindSpace:** optimal placement of geometric objects, for example, maximizing the number of objects of similar shape that can be cut out from a piece of material, minimizing the quantity of material needed to produce certain shapes, various packing problems, etc. (see, for example, [5]);
- **FindPath:** finding a collision-free motion path of an object amidst some obstacles of a particular shape, for example, an automatic assembly using an industrial robot, which requires grasping objects, moving them without collisions, and ultimately bringing them together.

The position and orientation of the geometric object to be moved or placed in the real space may be manifested as an individual point in a configuration space, in which each coordinate represents a degree of freedom in the position or orientation of this object ([8]). The configurations which, due to the presence of obstacles, are forbidden to the object can be characterized as regions in the configuration space called *configuration space obstacles* (see Fig. 1). The algorithm which solves the translational and rotational collision-free motion or safe placement problem when the objects are polygons or polyhedra was first presented in [8]. This algorithm computes configuration space obstacles using the notion of the Minkowski sum. After the configuration space obstacles have been calculated, the problem of motion planning is reduced to finding a path in the so-called *visibility graph*. In the presence of rotational motion, the induced configuration space obstacles may be represented as nonlinear constraints, which can be approximated by linear constraints. As noted in [9], the fundamental difficulty is that an exponential number of linear constraints would be required to approximate even a quadratic surface within an accuracy of 2^{-n}, resulting in an exponential time algorithm.

The exact computation of configuration space obstacles can be done with the aid of real quantifier elimination methods, as will be discussed in Section 2. The configuration space obstacles are semi-algebraic sets and the task of collision-free motion planning is then reduced to the problem of constructing a semi-algebraic curve between two points, such that the intersection of this curve with the interior of semi-algebraic set is empty. This purely geometric problem has been solved in [14] using Cylindrical Algebraic Decomposition ([4]) of semi-algebraic

sets. The latter algorithm can be performed in time polynomial in the number of polynomials as well as their maximal degree and double exponential in the number of variables. More efficient algorithms for the path calculation are presented in [1], [2] and have single exponential bounds in the number of variables. One of disadvantages of the mentioned algorithms is that they follow the boundary of configuration space and may produce paths, which touch obstacles. To calculate paths with maximal clearance from obstacles several methods based on Voronoi diagrams have been proposed (see [2], [15]).

In contrast to all these approaches, the objective of the present work is the generalization of finding a geometric path in order to

- find paths, which guarantee a certain minimum clearance from obstacles,
- provide the possibility to incorporate nonholonomic motion constraints (velocities, acceleration, etc.).

For this purpose, we shall describe a family of analytic functions with the property to rise in the vicinity of obstacles of arbitrary shape in the direction towards them. Using such "obstacle functions" (sometimes called "distance function"), we shall show how a "goal function" (sometimes called "target function") can be constructed, which decreases monotonously along some path from the initial to the final position, if and only if the path does not intersect any obstacle. Combining obstacle and goal function, we shall obtain a scalar-valued "navigation function" such that the problem of motion planning can be reduced to the task of following the gradient of the navigation function.

To our knowledge, the idea of using scalar valued functions for the obstacle avoidance was pioneered in [6]. The author proposed the navigation functions for the case the obstacles are a parallelepiped, a finite cylinder, and a cone. However, these geometric primitives do not form a sufficient set to describe the images of obstacles in the configuration space. The first construction of a general analytic navigation function is due to [11]. The authors show how a smooth navigation function can be constructed for the case when obstacles are smooth manifolds. In the present paper, we describe the construction of a more general family of navigation functions for arbitrary semi-algebraic objects. For this purpose, we shall use the functional representation of semi-algebraic point sets defined by so-called R-functions ([12], [16]) and reduce the problem of path finding to the solution of the Newton's equations of motion in a field of forces that can be done numerically. The obstacle and goal functions play the role of repulsive and attractive forces that push the object away from obstacles and pull it towards the goal position. As will be shown, the R-functions exhibit a wide range of differential properties, which can be used for the purpose of nonholonomic motion control. The implementation of our approach and computational examples will be presented.

2 Description of Geometric Objects Using R-Functions

The theory of R-functions ([12],[16]) provides the methodology of constructing an implicit functional representation for any semi-algebraic set using logical (set-theoretical) operations. In this section we shall briefly introduce this concept and

some results from the theory of R-functions, which shall be used in Section 3 for the purpose of the collision-free motion planning.

Let $F(X_1, ..., X_n)$ be a Boolean function with truth value 1 and false value 0 built using logical operations $\wedge$, $\vee$ and $\neg$. A real valued function $f(x_1, ..., x_n)$ is called an R-function if its sign is completely determined by the signs of its arguments. More precisely, f is an R-function if there exists a Boolean function F such that

$$sign(f(x_1, ..., x_n)) = F(sign(X_1), ..., sign(X_n)). \tag{1}$$

In other words, f works as a Boolean switching function, changing its sign only when its arguments change their signs. For example, logical operations on Boolean variables X_1, X_2 may be performed on real-valued variables x_1, x_2 such that (1) is satisfied using the following rules:

$$\begin{aligned} &x_1 \wedge\ x_2 \equiv x_1 + x_2 - \sqrt{x_1^2 + x_2^2} \\ &x_1 \vee x_2 \equiv x_1 + x_2 + \sqrt{x_1^2 + x_2^2} \\ &\neg x_1 \equiv -x_1. \end{aligned} \tag{2}$$

Consider, e.g., the Boolean function defined by

$$F(X_1, X_2, X_3, X_4) = X_1 \wedge X_2 \wedge X_3 \wedge X_4 \wedge X_5.$$

The corresponding real valued function f may be defined recursively according to (2):

$$\begin{aligned} &f_1(x_1, x_2) = x_1 + x_2 - \sqrt{x_1^2 + x_2^2} \\ &f_2(x_3, x_4) = x_3 + x_4 - \sqrt{x_3^2 + x_4^2} \\ &f(x_1, x_2, x_3, x_4, x_5) = f_1 + f_2 - \\ &\sqrt{{f_1}^2 + {f_2}^2} + x_5 - \sqrt{\left(f_1 + f_2 - \sqrt{{f_1}^2 + {f_2}^2}\right)^2 + {x_5}^2} \end{aligned} \tag{3}$$

This R-function can be used to describe point sets bounded by four arbitrary polynomials:

$$\begin{aligned} R(x, y) = \{(x, y) | &\phi_1(x, y) \geq 0 \wedge \phi_2(x, y) \geq 0 \wedge \\ &\phi_3(x, y) \geq 0 \wedge \phi_4(x, y) \geq 0 \wedge \phi_5(x, y) \geq 0\} \end{aligned} \tag{4}$$

For example, let four lines in the plane be given by the roots of the polynomials ϕ_i, $i = 1...4$,

$$\begin{aligned} &\phi_1(x, y) = x \\ &\phi_2(x, y) = x - 4 \\ &\phi_3(x, y) = y \\ &\phi_4(x, y) = y - 4 \end{aligned}$$

and a circle be given by

$$\phi_5(x, y) = (x - 2)^2 + (y - 2)^2 - 1.$$

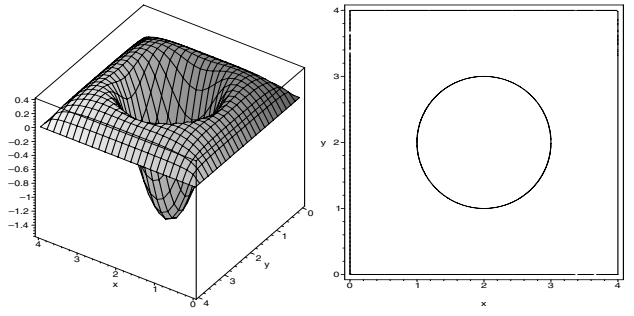


Fig. 2. O(x,y) and its roots

The object shown in Fig. 2 can be described as semi-algebraic set (4) or, alternatively, with the help of analytic function $f(\phi_1, \phi_2, \phi_3, \phi_4, \phi_5)$, that is equal to zero on the boundary of the object, positive inside and negative outside the object. In this way, any complicated semi-algebraic object can be constructed from primitive algebraic objects. Thus, R-functions enable one to write easily an equation for an object of arbitrary shape in the same way as one forms geometric objects by logical or set theoretic operations ([10]). Therefore R-functions are helpful in describing complicated semi-algebraic objects as an analytic function having the following sign property

$$\begin{array}{ll} f(\mathbf{x}) > 0 & \text{if } \mathbf{x} \text{ is inside the object} \\ f(\mathbf{x}) = 0 & \text{if } \mathbf{x} \text{ is on the boundary of the object} \\ f(\mathbf{x}) < 0 & \text{if } \mathbf{x} \text{ is outside the object} \end{array}$$

Alternatively to (2), the following rules described in [12] can be used to form union and intersection of geometric objects:

$$R_\alpha : \frac{1}{1+\alpha}(x_1 + x_2 \pm \sqrt{x_1^2 + x_2^2 - 2\alpha x_1 x_2}),$$

where $\alpha(x_1, x_2)$ is an arbitrary symmetric function such that $-1 < \alpha(x_1, x_2) < 1$.

$$R_0^m : (x_1 + x_2 \pm \sqrt{x_1^2 + x_2^2})(x_1^2 + x_2^2)^{\frac{m}{2}},$$

where m is any even positive integer.

$$R_p : x_1 + x_2 \pm (x_1^p + x_2^p)^{(\frac{1}{p})},$$

for any even positive integer p.

In each case above, choosing the $+/-$ sign determines the type of an R-function: $(+)$ corresponds to R-disjunction and $(-)$ sign gives the R-conjunction. The given families of R-functions exhibit a wide range of differential properties, which are studied in [17]. The change of parameters α, m and p leads to different characteristics of the navigation function, which will be described in Section 4 and allows to control the velocity or acceleration of the object.

The following theorem about the derivative of R_α-functions at the boundary has been proven in [12]. It states that the absolute value of the derivative of an R-function at the boundary point $\mathbf{p}$ in the given vector direction is equal to the absolute value of the derivative of the polynomial ϕ_i, which describes this part of boundary, provided the boundary part ϕ_i does not intersect with any other boundary boundary part ϕ_j in $\mathbf{p}$. The sign of the derivative is determined by the number of logical negations of x_i, called inversion degree.

Theorem 1 (Rvachev [12], [16]). *Let $f(x_1, ..., x_N)$ be such R_α-function that argument x_i appears in f only once and has the inversion degree m. Suppose the functions $\phi_1, ..., \phi_N$ and f are continuously differentiable and satisfy the following condition at point $\mathbf{p}$:*

$$\phi_i(\mathbf{p}) = 0; \phi_j(\mathbf{p}) \neq 0, i \neq j;$$
$$f(\phi_1, ..., \phi_N)|_{\mathbf{p}} = 0.$$

Then, for any vector direction l, the following equality holds

$$\left.\frac{\partial f(\phi_1, ..., \phi_N)}{\partial l}\right|_{\mathbf{p}} = (-1)^m \left.\left(\frac{\partial \phi_i}{\partial l}\right)\right|_{\mathbf{p}.}$$

For example, for any point $\mathbf{p}$ on the boundary part ϕ_i, $i = 1...5$, shown in Fig. 2, the following condition is satisfied

$$\left.\frac{\partial f(\phi_1, ..., \phi_5)}{\partial l}\right|_{\mathbf{p}} = \left.\left(\frac{\partial \phi_i}{\partial l}\right)\right|_{\mathbf{p}.}$$

This condition allows one to use the gradient of R-functions to predict the presence of obstacles and avoid collisions, as will be described in Section 4.

3 Computing Configuration Space Obstacles

As mentioned above, an important part in our approach to motion planning is a configuration space method ([8]). We propose to use the following two solutions:

- exact computation of configuration space obstacles based on quantifier elimination methods ([7]);
- approximation of configuration space obstacles by nonlinear constraints, which can be calculated in a more efficient manner ([13]).

In the following paragraphs we shall briefly describe both approaches.

Exact computation of configuration space obstacles. This algorithmic problem can be formulated as a decision problem for the first-order theory of real fields. The real numbers constitute an ordered field, which is closed under addition and multiplication. The formulas in the first-order theory of reals, defined by A. Tarski in 1930 and called the Tarski formulas, are composed from equalities and

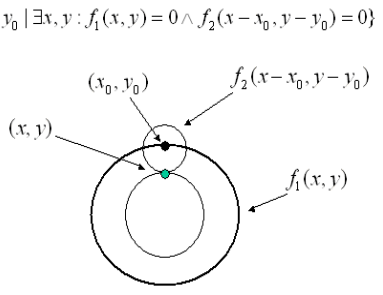


Fig. 3. Calculation of configuration space obstacles by quantifier elimination

inequalities. Such formulas may be constructed by introducing logical connectives (conjunction, disjunction and negation) and the universal and existential quantifiers to the atomic formulas.

For example, let some geometric object representing obstacle O be bounded by roots of finitely many polynomials $O_{i,j}(x, y, z)$. The inequalities $O_{i,j}(x, y, z) \geq 0$ and $O_{i,j}(x, y, z) \leq 0$ can be used to describe the exterior and interior of the object. Choosing the suitable sign of the polynomials we may use only "$\geq$" to describe any geometric object. The Tarski formula describing the set of points, which belong to the object, can be written as follows:

$$O(x, y, z) \equiv \bigvee_i \bigwedge_j O_{i,j}(x, y, z) \geq 0$$

The object P to be moved is given by roots of polynomials $P_i(x, y, z)$ and can be described with a Tarski formula in the same way. A shift of the object by x_0, y_0, z_0 units can be written as:

$$P(x, y, z) \equiv \bigvee_i \bigwedge_j P_{i,j}(x - x_0, y - y_0, z - z_0) \geq 0$$

As can be seen in Fig. 1 and 3, in the two-dimensional case the configuration space obstacle corresponding to O can be calculated for a particular orientation of P by contacting P with O and moving P along the boundary of O keeping them in contact. The resulting geometric object is the configuration space obstacle O_ϕ^{Conf} corresponding to the orientation ϕ of P.

The contact of O and P can be expressed in terms of common roots of bounding polynomials. Thus, O_ϕ^{Conf} corresponds to such shifts x_0, y_0, z_0 of P where some of polynomials P_i and O_i have common roots. This can be formalized with a Tarski sentence as follows:

$$\{(x_0, y_0, z_0) | \exists x, y, z : P(x - x_0, y - y_0, z - z_0) = 0 \wedge O(x, y, z) = 0\}$$

Eliminating $\exists$-quantifiers with existing methods ([3]) produces the semi-algebraic set that corresponds to O_ϕ^{Conf}. The latter quantity can also be described with

the aid of R-functions, as explained in Section 2. In Section 4 we shall show how such description can be used to predict collisions.

Approximate computation of configuration space obstacles. The configuration space obstacles can be calculated using the notion of the Minkowski sum. An algorithm for the approximation of the Minkowski sum with the help of R-functions has been proposed in [13]. Suppose P and O are defined by R-functions $P(x_1, ..., x_d) \geq 0$ and $O(x_1, ..., x_d) \geq 0$, respectively. The intersection of P shifted by $s_1, ..., s_d$ units and O can be written as

$$F(x_1, ..., x_d, s_1, ..., s_d) = P(x_1 - s_1, ..., x_d - s_d) \wedge O(x_1, ..., x_d)$$

As explained above, O_ϕ^{Conf} consists exactly of such shifts $s_1, ..., s_d$, which produce the contact between P and O. The contact of P and O means that their intersection is not empty. In this case $F \geq 0$, otherwise, if P does not touch O, $F < 0$.

Thus, the projection of $F(x_1, ..., x_d, s_1, ..., s_d)$ must be calculated: find such $s_1, ..., s_d$ so that there exist some $x_1, ..., x_d$ with $F(x_1, ..., x_d, s_1, ..., s_d) \geq 0$. As shown in [13], this projection can be computed by solving the following maximization problem:

$$O_\phi^{Conf}(s_1, ..., s_d) = \mathbf{max}\{F_3(x_1, ..., x_d, s_1, ..., s_d)\}.$$

The necessary condition for a point $(x_1, ..., x_{2d})$ where the maximum is attained:

$$\frac{\partial F_3}{\partial s_i} = 0, i = 1...d;$$

These equations can be solved numerically, for example, with the help of the Newton's method. In this manner, the configuration space obstacles can be represented as R-functions and used to predict collisions with obstacles.

4 Navigation in the Configuration Space

As mentioned above, the calculated configuration space obstacles can be represented with the help of R-functions. It follows from Theorem 1 that in the vicinity of obstacles the R-function increases towards them (see Fig. 4). Such "obstacle function" is therefore useful in order to predict collisions and determine the direction of the motion in order to avoid obstacles.Apart from the "obstacle function" O, we introduce the "goal function" G, which is decreasing monotonously along the path π that connects the initial position $\mathbf{s} = (s_1, ..., s_N)$ and the target position $\mathbf{g} = (g_1, ..., g_N)$. The goal function is required to have only one minimum value in $\mathbf{g}$. As we shall describe below, the sum of both functions defines the potential field U, which is used for motion planning (see Fig. 5):

$$U(x_1, ..., x_N) = O(x_1, ..., x_N) + G(x_1, ..., x_N). \quad (5)$$

Different functions with only one minimum value in the goal position and different differential properties can be used. In general, the following conditions must be satisfied:

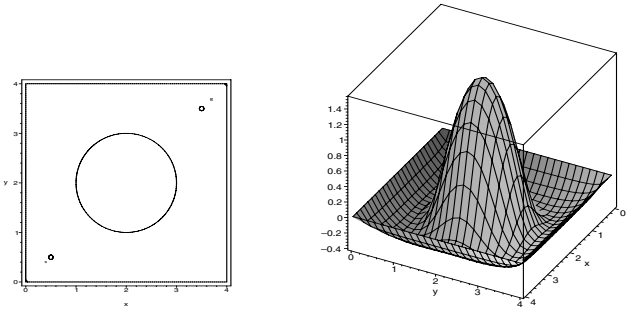


Fig. 4. The region with obstacles (colored black) and its obstacle function. The path from s to g must be calculated.

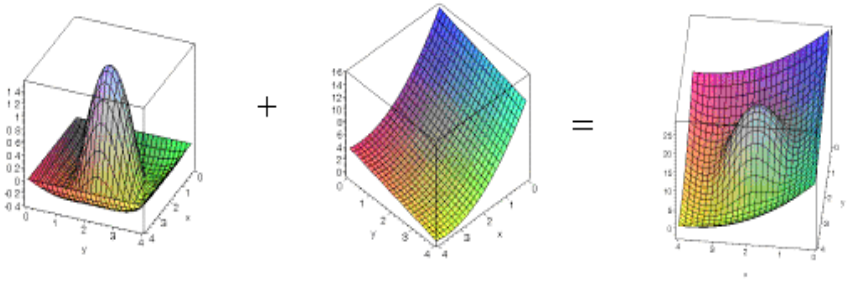


Fig. 5. Addition of the obstacle and the goal functions

– G is decreasing monotonously along the shortest path π connecting $\mathbf{s}$ with $\mathbf{g}$, e.g. the sign of the derivatives should be constant:

$$sign\left(\left.\frac{\partial G(x_1,...,x_N)}{\partial x_i}\right|_{\pi}\right) = const. \tag{6}$$

– U is decreasing monotonously in all points $\mathbf{p} \in \pi$, which lie not too close to any obstacle ($|O(\mathbf{p})| < \epsilon$). From (5), (6) and from the fact that $sign(\frac{\partial O}{\partial x_i}) \neq const$, it follows that the derivatives of G should be greater than those of O:

$$|O(\mathbf{p})| < \epsilon \Leftrightarrow \left|\left.\frac{\partial G(x_1,...,x_N)}{\partial x_i}\right|_{\mathbf{p}}\right| > \left|\left.\frac{\partial O(x_1,...,x_N)}{\partial x_i}\right|_{\mathbf{p}}\right| \tag{7}$$

– U has a minimum value in some point $\mathbf{p} \in \pi$ in the vicinity of an obstacle ($|O(\mathbf{p})| \geq \epsilon$) and increases towards the obstacle:

$$|O(\mathbf{p})| \geq \epsilon \Leftrightarrow \left|\left.\frac{\partial G(x_1,...,x_N)}{\partial x_i}\right|_{\mathbf{p}}\right| \leq \left|\left.\frac{\partial O(x_1,...,x_N)}{\partial x_i}\right|_{\mathbf{p}}\right| \tag{8}$$

The following functions, which have only one minimum value in the goal position $\mathbf{g}$, can be used as goal functions:

$$G_0(x_1, ..., x_N) = \alpha(\epsilon)\sqrt{|g_1 - x_1| + ... + |g_N - x_N|},$$

$$G_d(x_1, ..., x_N) = \alpha(\epsilon)((g_1 - x_1)^{2d} + ... + (g_N - x_N)^{2d})^{\frac{1}{2d}},$$

where d and $\alpha(\epsilon)$ are the parameters to be chosen in order to satisfy the conditions (6)-(8). Using this function, the potential field U can be constructed according to (5). The collision-free path from the initial to the final position corresponds to the direction of the gradient of U. In other words, we must simply follow the gradient of U. In this way, the purely geometric problem of path calculation can be reduced to the physical problem formulated with the help of the Newton's equations, which describe the motion of an object in the field of some forces $\mathbf{F}$:

$$m\mathbf{a} + \lambda\mathbf{v} = \mathbf{F},$$

where m is a mass of the object to be moved, $\mathbf{a}$ and $\mathbf{v}$ are acceleration and velocity, respectively, and λ is a so-called dissipation coefficient. Large values of λ correspond to the motion in a highly viscous environment. To describe the motion we may write the following differential equation:

$$m\frac{d^2x_i(t)}{dt^2} + \lambda\frac{dx_i(t)}{dt} = -\frac{\partial U(x_1, ..., x_d)}{\partial x_i} \tag{9}$$

(for simplicity, we do not consider curvilinear coordinates here). The force due to the environment "resistance" in our model is taken to be $\mathbf{R} = -\lambda\mathbf{v}$. However, other models, in particular those that account for the resistance increasing with velocity, can also be formulated, e.g. $\mathbf{R} = -C|\mathbf{v}|\mathbf{v}$ or, in the component form, $R^j = -(Cg_{ik}\dot{x}^i\dot{x}^k)^{\frac{1}{2}}\dot{x}^j$. Here g_{ik} is a metric tensor and C is the drag coefficient, which in general depends on the object's geometry and on the Reynolds number. The first term in (9) corresponds to the inertial motion. In our primary example, we assume this term to be small as compared to the dissipative term, which impedes the object's when the object approaches the obstacle. This is justifiable when the inertia coefficient m is small compared to $\lambda\tau_0$ where τ_0 is the characteristic time of object motion.

The equations

$$\lambda\frac{dx_i(t)}{dt} = -\frac{\partial U(x_1, ..., x_d)}{\partial x_i} \tag{10}$$

can be solved numerically, e.g. using the finite difference techniques. Numerical methods of solution of the motion equations are mostly based on evaluating the fist derivatives as

$$\frac{df(x)}{dx} = \frac{f(x + \Delta x) - x(x)}{\Delta x} + O(\Delta x).$$

Such discretization of (10) leads to

$$\lambda x_i(t_{j+1}) = \lambda x_i(t_j) - \frac{\Delta t}{\Delta x_i}(U(x_1, ..., x_i + \Delta x_i, ..., x_n) - U(x_1, ..., x_i, ..., x_n)). \tag{11}$$

According to (11), the object position $x(t_{j+1})$ at the time step t_{j+1} can be calculated from the previous position x_i at the time step t_j and the approximation

of the gradient of U at $x(t_j)$. Initial values $x_i(0)$ designate the initial positions of an object. Solving the equations

$$x_{j+1} = \lambda x_j - \frac{\Delta t}{\Delta x}(U(x_j + \Delta x, y_j) - U(x_j, y_j))$$
$$y_{j+1} = \lambda y_j - \frac{\Delta t}{\Delta y}(U(x_j, y_j + \Delta y) - U(x_j, y_j)) \tag{12}$$

leads to the motion shown in Fig. 6, 7. An example with three degrees of freedom demonstrated in Fig. 8 can be produced by solving

$$x_{j+1} = \lambda x_j - \frac{\Delta t}{\Delta x}(U(x_j + \Delta x, y_j, \phi_j) - U(x_j, y_j, \phi_j))$$

$$y_{j+1} = \lambda y_j - \frac{\Delta t}{\Delta y}(U(x_j, y_j + \Delta y, \phi_j) - U(x_j, y_j, \phi_j))$$

$$\phi_{j+1} = \lambda \phi_j - \frac{\Delta t}{\Delta \phi}(U(x_j, y_j, \phi_j + \Delta \phi) - U(x_j, y_j, \phi_j)).$$

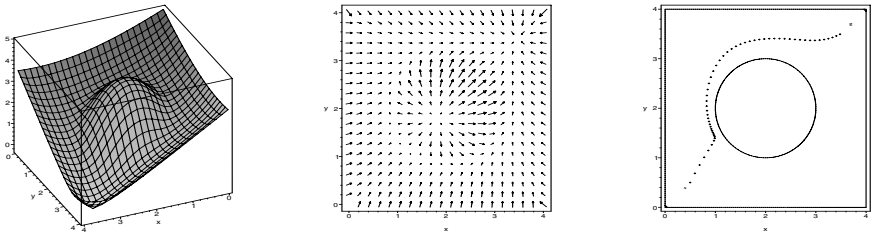


Fig. 6. The potential field (left), its gradient field (in the middle) and the path (right) calculated by following the gradient according to (12). $\frac{\Delta t}{\Delta x} = 0.1$; number of time steps: 54; computational time (using Maple): 0.121 sec.

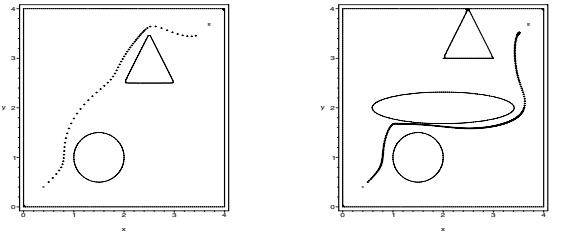


Fig. 7. Examples of calculated paths from the initial to final positions. Left: $\frac{\Delta t}{\Delta x} = 0.1$; number of time steps: 60; computational time (using Maple): 0.251 sec. Right: $\frac{\Delta t}{\Delta x} = 0.035$; number of steps: 300; computational time (using Maple): 1.562 sec.

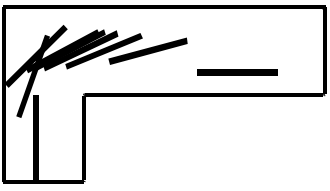


Fig. 8. An example of calculated path with three degrees of freedom: x and y translations and rotation

5 Conclusion

Our approach to spatial planning and associated geometrical problems is based on the object motion representation in a configuration space, which has a dimensionality equal to the number of independent coordinates describing the object position and orientation in the real space. The advantage of such a method is due to the fact that in the configuration space an object's motion corresponds to that of a fictitious material point moving in a potential field combined with viscous (dissipative) forces. This allows one to employ powerful numerical algorithms to compute collision-free trajectories. The potential field configuration is defined with the help of R-function techniques, which seems to be a convenient method for the functional (analytical) representation of complex geometries. The potential force field defined by R-functions has an attractive and a repulsive parts whose competition determines the goal function and the obstacle function, respectively. As it is typical of such situations, certain extremal properties arise defining the optimal path. The future work will be devoted to the extremal properties of the obstacle and goal functions.

Possible applications of presented techniques, apart from robot motion planning, may include medical kinesiology, biomechanics of human motion, rendering of human body positions, velocities and accelerations, joint simulations - all being modeled with the help of motion equations.

Acknowledgments

The authors are grateful to Prof. Hoon Hong and Prof. Christopher Brown for helpful suggestions concerning the presentation of this work as well as for valuable recommendations on further development of the described method.

References

1. Basu, S., Pollack, R., Roy, M.-F.: Computing Roadmaps of Semi-algebraic Sets on a Variety, In Foundations of Computational Mathematics, F. Cucker and M. Shub (eds.), 1-15, Springer-Verlag, 1997
2. Canny, J.: The Complexity of Robot Motion Planning, MIT Press, 1987

3. Caviness B.F., Johnson J.R. (eds.): Quantifier Elimination and Cylindrical Algebraic Decomposition, Springer-Verlag, 1998
4. Collins, G.E.: Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition, Lect. Notes in Comp. Sci. (33), 515-532, Springer-Verlag, 1975
5. Croft T.H., Falconer K. J., Guy R. K.: Unsolved Problems in Geometry, Springer-Verlag, 1991
6. Khatib, O.: Real time obstacle avoidance for manipulators and mobile robots, Internat. J. Robotics, 5, 90-99, 1986
7. Latombe, J.-C.: Robot Motion Planning, Kluwer Academic Publishers, 1991
8. Lozano-Perez, T.: Spatial Planning: A Configuration Space Approach, IEEE Transactions on Computers, C-32 (2), 108-120, 1983
9. Reif, J. H.: Complexity of the Generalized Mover's Problem, In Planing, Geometry and Complexity of Robot Motion, T. Schwartz, M. Sharir, J. Hopcroft (eds.), 267-281, Ablex Publishing Corporation, 1987
10. Requicha, A.: Representations for Rigid Solids: Theory, Methods, and Systems, ACM Computing Surveys (CSUR) Archive, 12 (4), 437-464, ACM Press, 1980
11. Rimon, E., Koditschek, D.E.: The Construction of Analytic Diffeomorphisms for Exact Robot Navigation on Star Worlds, Transactions of the AMS, 327 (1), 71-116, 1991
12. Rvachov, V.L.: Methods of Logic Algebra in Mathematical Physics, Naukova Dumka, Kiev, 1974 (in Russian)
13. Pasko, A., Okunev O., Savchenko V.: Minkowski sum of point sets defined by inequalities, Computers and Mathematics with Applications, Elsevier Science, 45 (10/11), 1479-1487, 2003
14. Schwartz, J., Sharir, M.: On the Piano Movers Problem II. General Techniques to Computing Topological Properties of Real Algebraic Manifolds, Advances in Applied Mathematics, 4, 298-351, 1983
15. Schwartz, J., Yap C. K.: Advances in Robotics , Lawrence Erlbaum associates, Hillside New Jersey, 1986
16. Shapiro, V.: Theory and Applications of R-Functions: A primer, Technical Report, Cornel University, 1991
17. Shapiro, V., Tsukanov, I.: Implicit Functions With Guaranteed Differential Properties, In Proceedings of the Fifth Symposium on Solid Modeling "SOLID MODELING'99", 258-269, Ann Abor, Michigan, ACM Press, 1999

nD Polyhedral Scene Reconstruction from Single 2D Line Drawing by Local Propagation*

Hongbo Li

Mathematics Mechanization Key Laboratory,
Academy of Mathematics and Systems Science,
Chinese Academy of Sciences,
Beijing 100080, China

Abstract. In this paper, we study the problem of reconstructing the polyhedral structures and geometric positions of a general nD polyhedral scene from a single 2D line drawing. With the idea of local construction and propagation, we propose several powerful techniques for structural reconstruction (i.e. face identification) and geometric reconstruction (i.e. realizability and parametrization). Our structural reconstruction algorithm can handle 3D solids of over 10,000 faces efficiently, outperforming any other existing method. Our geometric reconstruction algorithm can lead to amazing simplification in symbolic manipulation of the geometric data, and can be used to find linear construction sequences for non-spherical polyhedra.

Keywords: Polyhedra, Structural Reconstruction, Geometric Reconstruction, Grassmann-Cayley Algebra, Local Propagation.

1 Introduction

Representing and perceiving an nD object has been a very fascinating problem in both science and art [18]. For $n = 3$, the simplest representation is a line drawing which is the 2D projection of the wireframe of the object, like drafting in geometric design and mathematical diagram. To perceive an nD object one needs to rebuild the nD structure from its 2D projection.

How can the n dimensions be recovered from a representation in which almost all dimensions are lost? To start with, let us analyze how a solid in 3D space is perceived. No one can direct his eyesight to pierce through the solid. The only perceived object is the boundary of the solid, which is a 2D *closed manifold*. It is the closedness that allows us to fill the boundary with solid content to achieve one more dimension. When we watch a line drawing of the wireframe of a solid, which is essentially one dimensional, we extract each cycle of edges, either fill it by a plane or by some other surface to improve its dimension by one. Then we detect if any closed manifold is formed by the planes and surfaces, and if so, gain one more dimension by filling the closed manifold with solid content. The

* Supported partially by NSFC Grant 10471143 and 973 Project 2004CB318001.

H. Hong and D. Wang (Eds.): ADG 2004, LNAI 3763, pp. 169–197, 2006.

closedness of a manifold and a pattern to fill it are the two essential things in our 3D perception from low dimensional data.

From the topological point of view, a closed manifold is a *closed chain* in homology, and the boundary of a non-closed manifold is a *boundary chain*. While all boundary chains are closed, the converse is not true. For any dimension r, the quotient of the closed r-chains over the boundary r-chains is called the *r-th homology* of the object. Determining the boundary chains from the closed ones is equivalent to determining the homology.

The *wireframe model* of an nD object consists of (1) a set of *edges* connecting a finite set of points, called *vertices* of the object, (2) a subset of closed r-chains for $0 < r < n$, called *boundary r-chains*, which are the boundaries of the $(r+1)$D pieces of the object, (3) a set of filling patterns, each for a boundary r-chain.

Example 1. Line drawings of a tetrahedron and a torus (also a 3D sphere). Figure 1(a) shows a triangle being decomposed into three smaller ones. When all the four triangles are interpreted as polyhedral faces in 3D, the four faces form the boundary of a tetrahedron.

Figure 1(b) has 6 triangular cycles and 9 square cycles of edges. If all the 15 cycles are interpreted as polyhedral faces, then the 2D faces form 6 cycles of faces which are the boundaries of 6 triangular columns. If the 6 cycles of faces are interpreted as triangular columns then they form a cycle of 3D faces, which is the boundary of a 4D ball, i.e., the cycle of 3D faces forms a 3D sphere. If the object is required to be a 2D manifold, then the line drawing has a unique interpretation: it represents a torus in which the 6 triangular cycles are holes instead of boundaries of filled faces.

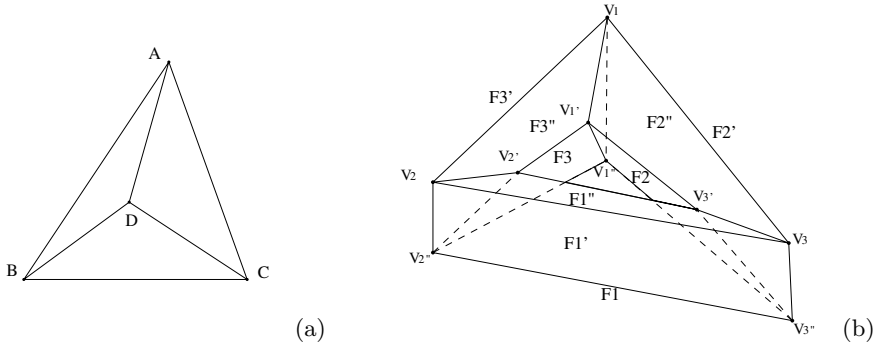


Fig. 1. (a): a tetrahedron; (b): a torus in 3D space (also a 3D sphere in 4D space)

For several decades, the study of wireframe representation and reconstruction has been focused on the real-world case $n = 3$. The reconstruction consists of two steps: structural reconstruction (i.e., face identification) and geometric reconstruction (i.e., realizability and parametrization). Structural reconstruction is to find those closed cycles which are boundaries of faces of a prescribed filling pattern. Geometric reconstruction is to determine whether a 2D line drawing of an nD scene is realizable in the space, and if so, give a parametrization of the space of all possible realizations.

The reconstruction can be based on either one or several projections of the same object. The projection can be either perspective or parallel. The filling patterns can be either transparent or opaque, either curved or polyhedral. In this paper, we focus on reconstructing an nD polyhedral wireframe in an mD ($m \geq n$) affine space from a single 2D line drawing, by assuming that the fillings be *completely transparent*, so that all the edges and vertices are visible, and that no two vertices are projected onto the same spot.

The structural reconstruction of an 3D object has been an active research topic in computer-aided design [1], [6], computer graphics [3], [7], [27] and computer vision [9], [17], but the case is completely open for $n > 3$. In the literature, all algorithms for 3D reconstruction consist of two steps: searching for all the cycles in the wireframe which are face candidates, and then identifying faces from the candidates. In searching for cycles, all the algorithms are *global* in that the searching is within the whole wireframe. A consequence is that the number of face candidates found is usually much larger than the number of real faces. Since identifying faces from the candidates is an NP-complete problem, the fastest algorithm in the literature can only handle wireframes of about 30 faces [15].

In the first part of this paper, we extend the study of polyhedral structural reconstruction from 3D to nD, under the most general assumption that neither the dimension n of the object nor the dimension m of its surrounding space is given, and whether or not the object is a manifold is unknown. This study is at least valuable in scientific visualization and high dimensional animation in entertainment industry: scientists and artists may be very much excited to find that their conceptual and spiritual nD object can be readily embodied in and perceivable from a single 2D line drawing.

We then propose and implement a very efficient structural reconstruction algorithm, which is valid for any n and m, no matter if they are known or not. In the classical case $m = n = 3$, our algorithm outperforms all other algorithms for face identification in both speed and range of application. For all the examples in [14], [15], [21], our algorithm can generate all the solutions for ambiguous wireframes, and do not produce or produce much fewer redundant cycles which are not real faces. Surprisingly, our algorithm can handle complicated 3D objects of over 10,000 faces [13].

There are several key ideas in our algorithm:

(1) For a general object in an unknown environment, although any compatible face identification is a solution to the reconstruction problem, the goal should be to find the most plausible solution that would be identified by a human observer. A human tends to choose a face identification in which there are as many edges as possible participating in as many faces as possible, which is the guideline for the algorithms in [14] and [21]. For nD face identification, the most important goal should be to find the highest dimension n, and for this purpose the above guideline may not prove to be helpful. In this paper, we propose a *new guideline* catering to this goal.

(2) To improve the speed for finding the first acceptable solution, it is very important to arrange the face candidates in such an order that the most plausible ones come first. We classify the cycles according to their *rigidity* so that they have different *levels of priority* in face identification.

(3) Initially to speed up the finding of the highest dimension, we propose to search for the cycles *locally* in the wireframe, then propagate the local wireframe to construct more cycles. This localization technique proves to be very efficient also in reducing the number of redundant cycles.

(4) While all other methods do not allow neighboring faces to be coplanar, in this paper we do. To further control the number of face candidates, we propose two techniques, *deletion* and *blocking*, to reduce the scope of cycle searching by deleting or blocking the branches that do not produce any new face candidates.

Once the polyhedral structure (also called *incidence structure*) of an nD polyhedral scene is determined, the next task is to determine the geometric positions of the faces in the mD surrounding space. This is the problem of *geometric reconstruction*. It is generally assumed that $m = n$ and that the nD scene is an $(n-1)$D manifold, or equivalently, an nD manifold with boundary. In this paper, we focus on the geometric reconstruction from a single 2D line drawing, by assuming that the center (or direction) of the perspective (or parallel) projection from nD to 2D is given, and that no 2D face of the polyhedron is projected into an image line. The wireframe is *realizable* if there exists an $(n-1)$D manifold whose projection is the line drawing. The equalities that must be satisfied by the 2D coordinates for the wireframe to be realizable are called the *realizability conditions*. The geometric positions of the faces cannot be unique, but can be *parametrized by free parameters*. Thus geometric reconstruction is also referred to *realizability and parametrization*.

Example 2. Truncated pyramid.
This is a classical example of 3D geometric reconstruction [5]. In Figure 2 (left) there are 2 triangular faces and 3 square ones. It represents a truncated pyramid formed by the 5 faces. However, the truncated pyramid can be realized in 3D if and only if the three image lines in Figure 2 (right) concur, otherwise it can only lie in a spatial plane, i.e., is flat.

In the literature of structural reconstruction, all attention is focused on the real-world case $n = 3$, and in most cases, on the realizability problem. Being a

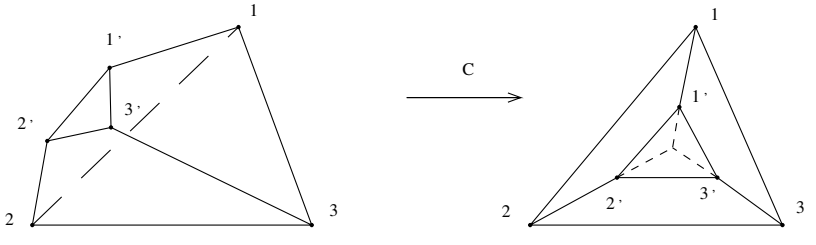


Fig. 2. 3D truncated pyramid and its 2D projection

classical problem in both computer vision and combinatorial geometry, it has been studied from both numerical computation and symbolic computation aspects. In a series of papers [23], [24], [25], Sugihara formulated the problem as a set of equations called *fundamental equations*, and proved that the realizability is equivalent to the existence of solutions for a linear programming problem. The problem is much more difficult to solve if the geometric data are in symbolic form, and has been an active research topic in combinatorial geometry for over 20 years [5], [22], [30], [31]. In [5], Crapo proposed to reformulate the problem as a set of equations called *syzygy equations*, which is much smaller in number. Still the equations are very difficult to solve symbolically.

In the second part of this paper, we extend the study of the geometric reconstruction of polyhedra from 3D to nD. We propose a simple and effective method to solve simultaneously the realizability and parametrization problems symbolically. The method can be used to find *linear construction sequences* for the 3D geometric configurations of a class of polyhedra with nonzero genus, a result that has never been obtained before [12].

There are three key ideas in our algorithm:

(5) Using Grassmann-Cayley algebra, we decompose the fundamental equations into two systems: the *system of extensors* and the *system of heights*. We only need to solve the former system, for which we can employ the powerful *vectorial equation-solving technique* [10].

(6) In solving the equations of extensors, we *introduce new parameters locally*, propagate the local solutions to obtain more parametrized ones. Essentially this is a procedure of transforming the system of extensors into a *system of new parameters*.

(7) To simplify the system of new parameters, algebraic factorization is necessary but very difficult. For $n = 3$, we propose an efficient method called *calotte factorization*, to factor the system of new parameters *geometrically*.

This paper is arranged as follows. In Section 2 we analyze some properties of a general nD wireframe object and its 2D line drawing, and lay down the foundation of high-dimensional reconstruction. In Section 3 and 4 we propose some powerful techniques for nD structural and geometric reconstructions.

2 nD Wireframe Object and 2D Line Drawing

We consider the wireframe models in which the filling patterns are affine flats. This pattern has a strong constraint, called the rD *face adjacency constraint* [21]: if two rD faces share two $(r-1)$D faces which are in different $(r-1)$D affine planes, then the two rD faces must be in the same rD affine plane, i.e., are *coplanar*.

We assume that the 2D line drawing of an nD wireframe object is obtained by a perspective or parallel projection from an mD affine space to the image plane, such that all the edges and vertices are revealed, and if three vertices are not collinear in the mD space, nor should their images. So in the line drawing,

– if two edges are collinear, so are they in the mD scene;
– if two edges cross not at a vertex, they do not intersect in the mD scene.

2.1 nD Perspective and Parallel Projections

A *perspective projection* from mD to 2D is the composition of $m-2$ successive perspective projections whose projective centers are linearly independent vectors, and at least one center is an affine point. A *parallel projection* from mD to 2D is the composition of $m-2$ successive parallel projections whose projective centers are linearly independent directions.

As is well known, both projective and affine geometries can be efficiently represented by Grassmann-Cayley algebra and bracket algebra [22] in a coordinate-free manner. Below we give a brief introduction of this algebra.

Let $\mathcal{V}^{m+1}$ be an $(m+1)$D vector space representing the mD affine space. The *outer product* is the unique associative, multilinear and anticommutative product defined among the vectors of $\mathcal{V}^{m+1}$. The outer product of r vectors is called an *r-extensor*. Any linear combination of r-extensors is called an *r-vector*. The *Grassmann space* $G(\mathcal{V}^{m+1})$ generated by $\mathcal{V}^{m+1}$ is the graded vector space of r-vectors for r from 0 to n. The *Grassmann-Cayley algebra* of $G(\mathcal{V}^{m+1})$ is the Grassmann space together with two products among its elements: the outer product, denoted by juxtaposition, and the *meet product*, denoted by "$\wedge$", which is the dual of the outer product.

The space of $(m+1)$D-vectors is one dimensional. Let I be a basis of this space, then the *bracket* of an $(m+1)$-vector x with respect to I is its coordinate

$$[x] = x/I. \tag{2.1}$$

Bracket algebra refers to the ring of brackets.

In Grassmann-Cayley algebra, an r-extensor represents the $(r-1)$D projective space spanned by the r vectors. So in the mD projective space, a point is represented by a nonzero vector, which is unique up to scale. The line passing through points $\mathbf{1}, \mathbf{2}$ is represented by *bivector* $\mathbf{12}$. The rD plane passing through points $\mathbf{C}_1, \ldots, \mathbf{C}_{r+1}$ is represented by the r-extensor $\mathbf{C}_1\mathbf{C}_2\cdots\mathbf{C}_{r+1}$. The intersection of an rD plane A and an sD plane B, if $r+s > m+1$, is represented by the meet $A \wedge B$. A perspective (or parallel) projection from mD to 2D is represented by [11]

$$\mathbf{P} \mapsto X\mathbf{P}, \quad \text{for all } \mathbf{P} \in \mathcal{R}^{m+1} \tag{2.2}$$

where X is the $(m-2)$-extensor representing the center (or direction) of the projection. The space $\{X\mathbf{P} \,|\, \mathbf{P} \in \mathcal{R}^{m+1}\}$ is a 3D vector space representing the image plane. Any decomposition of X into the outer product of $m-2$ vectors induces a decomposition of the projection into $m-2$ "classical" projections in which the perspective center is a point or point at infinity.

2.2 Constraints for Structural Reconstruction

Structural reconstruction is to recover for r from 2 up to n the rD polyhedral structure, i.e., incidence structure, i.e., boundary $(r-1)$-chains. Since it is seldom

possible to judge if a single cycle should be a boundary chain, in general we need to find a number of cycles which are *face candidates*, and then select from them a subset satisfying the properties of real faces. Structural reconstruction is also called *face identification*.

In a wireframe model, a 0D *face* is a vertex, a 0D *cycle* is the two vertices of an edge, and a 1D *face* is an edge. For $r > 0$, an rD *cycle* is a set of rD faces such that (1) if two faces intersect, the intersection belongs to their iD faces for $0 \leq i < r$, (2) any $(r-1)$D face of one rD face is shared by exactly one other rD face in the set. For $r > 1$, an rD *face* is an $(r-1)$D cycle filled by the rD affine flat surrounded by it.

Some geometric constraints must be satisfied for an $(r-1)$D cycle to become an rD face. In previous work on face identification, it is generally assumed that any two neighboring faces are not coplanar. We feel that this is too strong a constraint to include many interesting models, so we discard it.

If two rD faces share at least two $(r-1)$D faces which are in different $(r-1)$D planes, then the two rD faces must be in the same rD affine plane, i.e., *coplanar*. The union of a set of rD coplanar faces is called an rD *polyface*, and the faces are said to be *merged* together. An advantage of this concept is that usually we do not need to decompose a polyface into non-overlapping faces.

By a perspective projection, a 2D face is projected onto a 2D region of the image plane whose boundaries or boundary do not intersect. The 2D *non-self-intersection constraint* [9] says that if two edges intersect not at a vertex, then they cannot be in the same 1D cycle.

If there are at least three vertices which are collinear in the mD space, then the 2D projection of the line passing through the vertices is called a *line* in the line drawing. The edges within a line are the *real parts*, and the virtual connections between real parts are the *virtual parts*. The 2D *non-interior-intersection constraint* [15] says that if two 1D cycles intersect at only two vertices and the line segment between the two vertices intersects both the enclosed regions of the 1D cycles in the image plane, then either the two cycles form a polyface, i.e., are coplanar, or at most one can be assigned as a face. This is because if both cycles are faces and are non-coplanar, then the virtual part between the two vertices must be the projection of a real part of the line of intersection, contradicting the assumption that all edges are visible by the projection.

In a 2D line drawing it is impossible to distinguish between the interior and the exterior of an rD object for $r > 2$. So the 2D non-interior-intersection constraint has no high dimensional generalization.

If an rD face F is not in an rD cycle C but its intersection with C is exactly the boundary of F, then F is called a *chord* of C. A face with chord can always be decomposed along its chord into two coplanar faces. On the other hand, if a cycle with chord is not assigned as a face, then it can be decomposed into two cycles by adding the chord into it, and by assigning both cycles to be faces they do not need to be coplanar. To gain more degree of freedom in the reconstruction, an rD cycle with chord is not assigned as a face. This is the rD *chordless constraint*.

Let F be an sD face for $s \geq 0$. Any rD face containing F is called an rD *F-face*. When $r > s$, an rD *F-cycle* refers to a set of rD F-faces in which any $(r-1)$D F-face of one rD F-face is shared by exactly one other rD F-face, and not all rD F-faces are in the same rD plane.

In many applications it is required that the nD polyhedral object be a manifold with boundary, i.e., the boundary is an $(n-1)$D *manifold*. The formal definition is that at every point of the boundary the intersection of the boundary with a sufficiently small mD disk centered at the point is homeomorphic to an $(n-1)$D disk. In general, the following constraint, called $(n-1)$D *manifold constraint*, suffices to guarantee that an $(n-1)$D cycle C satisfying all previous constraints in this subsection be an $(n-1)$D manifold: at any vertex $\mathbf{V}$ of the cycle, there is at most one $(n-1)$D $\mathbf{V}$-cycle in C. This constraint has never occurred in the literature of face identification before.

2.3 mD Geometric Reconstruction

Let the perspective or parallel projection from mD to 2D be induced by the center $X = \mathbf{C}_3\mathbf{C}_4 \cdots \mathbf{C}_m$, where the $\mathbf{C}$'s are vectors in the $(m+1)$D vector space of homogeneous coordinates. Each $\mathbf{C}$ represents a point or direction (point at infinity). The geometric reconstruction from 2D to mD can be realized step by step from $(r-1)$D to rD, for r from 3 to m. The lift from $(r-1)$D to rD is the reverse procedure of the projective from rD to $(r-1)$D centered at $\mathbf{C}_r$. In this subsection we always assume that $r \geq 3$.

Assume that the line drawing represents an mD manifold with boundary. Further assume that both $\mathbf{C}_r$ and the $(r-1)$D image plane $\mathcal{I}_{r-1}$ are given and are *in generic positions* in the rD affine space $\mathcal{I}_r$. In fact,

$$\mathcal{I}_r = \mathcal{I}_{r-1}\mathbf{C}_r = \mathcal{I}_{r-2}\mathbf{C}_{r-1}\mathbf{C}_r = \cdots = \mathcal{I}_2\mathbf{C}_3 \cdots \mathbf{C}_r. \tag{2.3}$$

The input of the lift from $(r-1)$D to rD is (1) the $(r-1)$D coordinates of the image vertices, (2) the incidence structure between the $(r-2)$D and $(r-1)$D faces. The output is (1) the realizability conditions of the *rD faces* satisfied by the the *2D coordinates* of the image vertices, (2) the parametrized solution space of the vertices and $(r-1)$D faces in $\mathcal{I}_r$. Since the perspective center is generic, the equality constraints on the $(r-1)$D coordinates are reduced to equalities on the 2D coordinates of the vertices.

Let $\mathcal{V}^{r+1}$ be the homogeneous coordinate space of $\mathcal{I}_r$. Let $\{\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}_2\}$ be a basis of $\mathcal{V}^3$ such that $\mathbf{C}_1, \mathbf{C}_2$ are directions of the image plane $\mathcal{I}_2$. Then $\{\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_r\}$ is a basis of $\mathcal{V}^{r+1}$. Let the homogeneous coordinates of a vertex $\mathbf{V}^r$ in $\mathcal{I}_r$ be $(h^0, h^1, \ldots, h^r)$, where h^r is called the *height* of the vertex with respect to $\mathbf{C}_r$. Let the homogeneous coordinates of an $(r-1)$D face F^{r-1} in $\mathcal{I}_r$ be $(a_0, a_1, \ldots, a_{r-1}, -1)$. Then vertex $\mathbf{V}^r$ is in face F^r if and only if

$$\sum_{i=0}^{r-1} a_i h^i = h^r. \tag{2.4}$$

When the vertices and $(r-1)$D faces range over the whole $(r-1)$D incidence structure, we obtain a set of linear homogeneous equations in the a's and h^r's,

called *Sugihara's rD fundamental equations*. Solving (2.4) for the realizability conditions if the h_i take symbolic values for $1 \leq i < r$, is a very difficult task.

For any basis $\{\mathbf{C}_i \,|\, 1 \leq i \leq r+1\}$ of $\mathcal{V}^{r+1}$, its *dual basis* in the Grassmann space generated by $\mathcal{V}^{r+1}$ is the r-extensors $\{\mathbf{C}_j^* \,|\, 1 \leq j \leq r+1\}$ such that $[\mathbf{C}_i\mathbf{C}_j^*] = \delta_{ij}$. In fact,

$$\mathbf{C}_j^* = (-1)^{j-1}\mathbf{C}_1\mathbf{C}_2\cdots\check{\mathbf{C}}_j\cdots\mathbf{C}_{r+1}, \tag{2.5}$$

where $\check{\mathbf{C}}_j$ denotes that $\mathbf{C}_j$ does not occur in the product.

The $(r-1)$-extensor

$$\mathbf{B}^{r-1} = \sum_{i=0}^{r-1} a_i\mathbf{C}_i^* \tag{2.6}$$

is called the *inhomogeneous coordinates* of face F^{r-1}. Two faces are coplanar if and only if their inhomogeneous coordinates are identical. Now (2.4) can be written as an equality in the bracket algebra generated by $\mathcal{I}_{r-1}$:

$$h^r = [\mathbf{V}^{r-1}\mathbf{B}^{r-1}], \tag{2.7}$$

where $\mathbf{V}^{r-1} = \sum_{i=0}^{r-1} h^i\mathbf{C}_i$ is the projection of $\mathbf{V}^r$ in $\mathcal{I}_{r-1}$.

For any $(r+1)$-tuple of vertices $\mathbf{V}_1^r, \mathbf{V}_2^r, \ldots, \mathbf{V}_{r+1}^r$ in face F^{r-1}, their projections $\mathbf{V}_1^{r-1}, \mathbf{V}_2^{r-1}, \ldots, \mathbf{V}_{r+1}^{r-1}$ in the image plane $\mathcal{I}_{r-1}$ satisfy the following *Grassmann-Plücker relation* [22]:

$$\sum_{i=1}^{r+1}(-1)^{i-1}[\mathbf{V}_i^{r-1}\mathbf{B}^{r-1}][\mathbf{V}_1^{r-1}\mathbf{V}_2^{r-1}\cdots\check{\mathbf{V}}_i^{r-1}\cdots\mathbf{V}_{r+1}^{r-1}] = 0. \tag{2.8}$$

When F^{r-1} ranges over all $(r-1)$D faces with more than r vertices, we obtain a set of equations linear in the h^r's by substituting (2.7) into (2.8):

$$\sum_{i=1}^{r+1}(-1)^{i-1}h_i^r[\mathbf{V}_1^{r-1}\mathbf{V}_2^{r-1}\cdots\check{\mathbf{V}}_i^{r-1}\cdots\mathbf{V}_{r+1}^{r-1}] = 0. \tag{2.9}$$

They are called *Crapo's rD syzygy equations*.

On one hand, the fundamental equations and the syzygy equations have the same solutions for the h^r's. On the other hand, the number of syzygy equations is much smaller than the number of fundamental equations. The syzygy equations can also be derived by eliminating the $\mathbf{B}$'s from the fundamental equations (2.7).

Notice that for $r > 3$, both systems have a lot of redundant equations. Since an $(r-1)$D face F^{r-1} is in an rD face F^r if and only if there is an r-tuple of vertices of F^{r-1} incident to F^r but not incident to any $(r-2)$D affine plane, in the case that F^{r-1} has more than r vertices, the syzygy equations of its $(r+1)$-tuples of vertices are all equivalent to each other. To remove the redundancy, we require the incidence structure between the $(r-2)$D and $(r-1)$D faces be in the input of rD lift, instead of only the incidence structure between the vertices and $(r-1)$D faces.

2.4 Rigidity

By the definition of a cycle, the supporting affine plane of an $(r-1)$D cycle has dimension at least r. By the geometric reconstruction from rD to $(r+1)$D, for a given $(r-1)$D cycle in the rD affine plane, if the dimension of the configuration space of the lifted cycle (which is defined to be the maximal number of free parameters in the parametrization) is $k+r+1$, then the *rigidity* of the cycle is defined to be $-k$, and the *flexibility* of the cycle is defined to be k. A cycle of rigidity 0, or -1, or < -1 is said to be *rigid*, or *elastic*, or *plastic* respectively.

For example if $r = 2$, a 1D cycle of $k+3$ vertices has rigidity $-k$, because when lifting from 2D to 3D along the perspective center, the cycle has $k+3$ free parameters which are exactly the heights of all the vertices. Thus rigid 1D cycles are triangular and elastic 1D cycles are square ones.

The *principle of rigidity* in structural reconstruction states that (1) rigid cycles are always identified as faces, (2) elastic cycles are more likely to be faces than plastic ones. The explanation is as follows:

1. Rigid cycles are natural object delimiters in the wireframe, they are either real or interior faces of the object. If they are assigned as faces, they never force any two faces of different planes to be coplanar, i.e., they do not cause any geometric incompatibility.
2. If the object is not assumed to be a manifold, then taking all rigid cycles as real faces conforms to the *principle of psychological selection* to be introduced in Section 3, that more faces can be produced which pass through more lower dimensional ones and do not reduce the number of higher dimensional ones.
3. If the object is required to be a manifold, then taking all rigid cycles as real faces induce a decomposition of the object into smaller ones of the same dimension, and by the manifold assembly algorithm to be introduced in Section 3, all interior faces can be removed.
4. Elastic cycles are next to rigid ones in simplicity. Experiments show that they are the next most plausible face candidates.

On one hand, the rigidity of a general cycle can be determined only by geometric reconstruction. On the other hand, the rigidity is to be employed in structural reconstruction, far before geometric reconstruction starts. This paradox can be resolved as follows: The principle of rigidity serves only as a heuristic rule in ordering the face candidates. It is not a prerequisite that all cycles be ordered strictly by their rigidity in face identification. In practice we use the following algorithm to search for a class of rD rigid cycles for $r > 1$, called *rooted rigid cycles*:

Two non-coplanar rD faces sharing one $(r-1)$D face are called *neighbors*. For every pair of rD neighbors, do the following:

(1) Initially let set C contains only the pair.
(2) Repeatedly put into C all the rD faces and polyfaces sharing with C at least $r+1$ vertices which are not in the same $(r-1)$D plane.
(3) If C is an rD cycle then it is rigid.

Example 3. For a 2D cycle, if at each vertex there are at most three edges, then the cycle must be rooted rigid. Figure 1(a) and Figure 2 are such examples.

3 Structural Reconstruction

Even for the classical case $m = n = 3$, structural reconstruction is very difficult. The number of cycles in a graph is generally exponential in the number of vertices. Given that all the cycles which are potential faces are found, the number of their combinations is exponential in the number of the cycles. So this is a problem of double-exponential complexity even for $m = n = 3$.

The *guideline* in designing an efficient algorithm should not be to find a polynomial-time algorithm for all optimal solutions. We propose that it should be to *reduce the time for finding the first optimal solution.*

The what is an "optimal solution"? For different purposes there are different criteria. We propose the following *sequence of criteria* according to their precedence in our general-purpose algorithm:

1. The highest dimension n of the object should be found and reached.
2. The solution should be most likely identified by a human.
   Our *principle of psychological selection* is that for $r > 1$, the rD face identification should make as many $(r-1)$D faces as possible participating in as many rD faces as possible, such that the sequence of numbers of non-coplanar iD faces is maximal lexicographically for i from n down to r.
3. If the solution is required to be a manifold of a given dimension, then any such solution is optimal.
4. If there are several optimal solutions, then as many of them as possible should be found.

Below we propose six powerful techniques for structural reconstruction based on the above guideline and criteria. Without loss of generality, we only describe the classical case of finding 1D cycles for 2D face identification. By the following correspondences, the techniques and concepts therein (except for Subsection 3.5) can be readily generalized to rD cycle-finding for $(r+1)$D face identification, for $r > 1$:

$$\begin{aligned} \text{vertex (0D face)} &\longrightarrow (r-1)\text{D face},\\ \text{edge (1D face)} &\longrightarrow r\text{D face},\\ \text{cycle (1D cycle)} &\longrightarrow r\text{D cycle},\\ \text{face (2D face)} &\longrightarrow (r+1)\text{D face}. \end{aligned}$$

3.1 Localization

It is well recognized that to search for the real faces one does not need to find all cycles. We further recognized that to speed up the finding of the first optimal solution ones can simply search for face candidates *locally.*

Let C be a wireframe model. A *local wireframe model* of C is a subset of the vertices of C together with all the higher dimensional faces formed by the

vertices. A *localization filter*, or *localization*, of C is a sequence of local wireframe models $S_1 \subset S_2 \subset \ldots \subset S_k = C$ in which each successor introduces more vertices than its predecessor. With the introduction of new vertices, all the edges among them and the existing vertices are introduced.

Localization is often realized by *propagation through edges*. Starting from a vertex called the *origin*, we localize the wireframe by considering only the subgraph of the origin and its neighboring vertices. Within the local wireframe we identify the faces. Then we set the origin to be the current local wireframe, and repeat the localization and identification procedure. By introducing new vertices according to the closeness of their relations with the existing ones, the complexity of cycle searching can be reduced. Below are some typical localizations.

The first is the *descendent localization*: (1) Start from a set C containing only one vertex called the *origin*, put all the neighboring vertices of C into C. (2) Put all the vertices collinear with at least two vertices of C into C. Repeat until C no longer changes. (3) Put all the neighboring vertices of C into C. (4) Repeat Steps 2 and 3 until all vertices are in C. Each repeat is a round of localization.

The second is the *singleton localization*: Steps 1, 2, 4 are the same with those in descendent localization. Step 3 is as follows: (3) Put into C those vertices called *singletons*, which are common neighbors of at least two vertices of C, and if there is no singleton at all then put into C its neighboring vertices, called *twins*. This localization makes much easier the generation of new cycles.

The third is the *rigidity localization*: Steps 1, 2, 4 are the same with those in the neighbor localization. Step 3 is as follows: (3) Put into C those singletons each forming a rigid or elastic cycle with some vertices in C, and if there is no such vertex at all then use Step 3 of the singleton localization. Locally, rigid cycles are constructed first, followed by elastic and plastic ones. This localization proves to be the most efficient.

Example 4. In Figure 3(a) there is no collinearity constraint prescribed. The three localizations are identical:

$$\{1\} \subset \{1,2,3,4\} \subset \{1,2,3,4,5,6,7,8,9,0\} \subset \{1,2,3,4,5,6,7,8,9,0,a,b,c\},$$

which proceeds till all vertices are included.

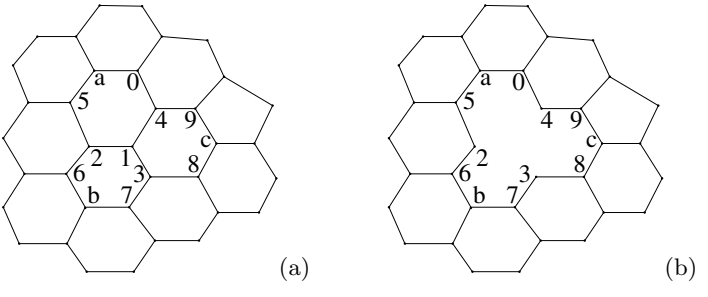


Fig. 3. Localization and deletion

3.2 Deletion

Two techniques accompany each round of localization: deletion and path blocking. The purpose of deletion is to reduce the size of a local wireframe by cutting off those edges and vertices which no longer contribute to face identification.

When $n = 3$, the idea of deleting edges in cycle searching can be found in [15]. There the object is a 2D manifold in which any edge occurs in exactly two faces, so if two faces containing the edge have been found, the edge can be deleted. At any time, those vertices through which there is only one edge can always be deleted.

For a general object, the criterion for an edge to be deletable should be that *the deletion does not influence the final identification of 2D faces and polyfaces*. By our principle of psychological selection, an edge E in the procedure of localization can be deleted if by assigning any new cycle through it as a face, (1) the number of non-coplanar faces does not increase, (2) if any new vertex $\mathbf{V}$ is to be added into the polyface containing the face, there is always a cycle of edges in the polyface that passes through $\mathbf{V}$ but not E, i.e., deleting E does not prevent $\mathbf{V}$ from joining the polyface via a cycle of edges.

Deletion Theorem. *In cycle searching, any saturated edge outside lines, and any saturated real part of a line, can be deleted.*

We shall explain the term "saturated" but omit the proof of the theorem. An edge is said to be *saturated* if all the edges protruding from its vertices are already in faces passing through the edge. A connected real part of a line is said to be *saturated* if every edge of it is saturated.

In Figure 3(a), if three cycles $125a04$, $138c94$ and $126b73$ are already assigned as faces, then edges $12, 13, 14$ are saturated and can be deleted. Then vertex 1 is no longer connected to any other vertex and can be deleted. See Figure 3(b).

3.3 Path Blocking

In Figure 3(b), the vertices in the localization form a big cycle $25a049c837b6$. This cycle cannot be a face, otherwise all three constructed faces have to be merged. In searching for more face candidates passing through a fixed vertex, those faces having been identified can block off some search branches by preventing identified faces from merging, according to our principle of psychological selection. This technique is extremely useful in reducing the number of branches in cycle searching.

For a general object, if a branch of edges intersects a face in at least three vertices which are not collinear, then the branch is blocked by the face. The block is called a *face block*. If a branch meets two different face blocks, then it is blocked permanently. If along a branch there is only one face block, then the branch of edges can be merged with the face to form a polyface.

In Figure 3(b), suppose we want to find a new cycle passing through vertex 2. From 2 to 6, the path is blocked by face $62137b$. From 2 to 5, the path is blocked by face $04125a$. The two different face blocks permanently block any new cycle from passing through branch 625.

In [15], the models are 2D manifolds in which no two neighboring faces are coplanar. If a cycle is identified as a face then no other branch passing through two edges of it can generate a face. Here one face suffices to block off the branch permanently. For a general object, this blocking does not work.

There are other types of blocks. The 2D non-self-intersection constraint can block some branches permanently. These blocks are called *intersection blocks*. The 2D non-interior-intersection constraint can block some branches from including the interior of an existing face. These blocks are called *interior blocks*. The 2D chordless constraint can permanently block some branches from generating cycles with chords. These blocks are called *chord blocks*.

3.4 Local Optimization

Assume that all the constructed cycles satisfy the constraints in Subsection 2.2. For a group of cycles, if when assigning all of them to be new faces, either two of them merge, or one face merges with an existing face, then the group of cycles is said to be *degenerate*. For a degenerate group of cycles, by the principle of psychological selection, generally not all the cycles are assigned as faces.

An optimal selection of a subset of cycles within the group should maximize the number of non-coplanar faces, or equivalently, minimize the number of merges. If there are several optimal selections, then the algorithm bifurcates and the *state-space tree* [15] is generated, or extended if it has been generated. Each optimal selection is to be explored to find all solutions.

In rigidity localization, the above *local optimization* can be greatly simplified. Recall that in each round of localization, there are two kinds of new vertices: singletons and twins. In face identification, elastic cycles have priority over plastic cycles, and singletons have priority over twins. Thus, there are four *levels of priority* for non-rigid cycles. Our strategy is to restrict the local optimization to cycles of the same level of priority. The details can be read from the following example.

Example 5. A diagonal is drawn in a cube (Figure 4), which makes the face identification very difficult to reach dimension three [21]. By the local optimization within rigidity localization, the unique 3D explanation can be easily obtained.

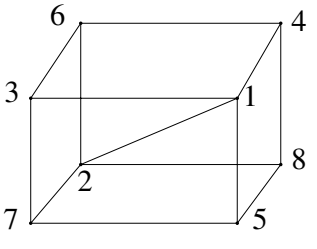


Fig. 4. Local optimization

The following new vertex series is produced by rigidity localization:

$$1 \longrightarrow 2, 3, 4, 5 \longrightarrow 6, 7, 8. \tag{3.1}$$

Only in the last step do the cycles appear. The singletons generated by elastic cycles are

$$6 : \{1264, 1364\}; \quad 7 : \{1275, 1273, 1375\}; \quad 8 : \{1284, 1485\}. \tag{3.2}$$

The other elastic cycles are $2637, 2648, 2758$. For each singleton, its generating elastic cycles are degenerate. The 3 groups of cycles form 12 combinations, among which only the combination $(6 : 1364,\ 7 : 1375,\ 8 : 1485)$ is optimal. No state-space tree is generated.

3.5 Repair and Assembly

Example 6. In Figure 5, there is a line 1563 with virtual part 56. In the literature, usually two solutions are found, one with faces 123675 and 415863, the other with faces 123685 and 415763. The latter is geometrically impossible, because the 2D non-interior-intersection constraint is violated.

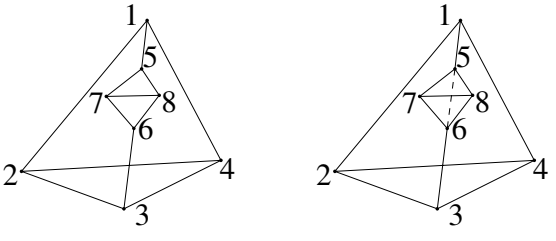


Fig. 5. Repair and assembly

We introduce a simple technique called *repair and assembly*, to avoid producing the fake solution. It is also very helpful in simplifying cycle searching, although it only applies to 2D face identification. The technique is composed of three steps:

(1) If there is any virtual part of a line, *repair* the line drawing by connecting the virtual part. The new edges are said to be *virtual*.

(2) Make 2D face identification in the repaired line drawing. In Figure 5, two tetrahedra are obtained.

(3) The *second assembly principle* states that if two 3D faces intersect at an edge E but not at any face, then they can be assembled at the edge if and only if within the line drawing, a face F_1 at E in one 3D face is within the 2D region of a face F_2 at E in the other 3D face. If there are no two such faces then one 3D face must be deleted; else the assembling is realized by replacing the two faces by their difference in the polyface generated by them.

Assemble the faces containing the virtual edges according to this principle.

(4) Remove the virtual edges to recover the original wireframe.

3.6 Manifold Assembly

In many applications it is required that the object be a 2D manifold. For this special purpose, there are two alternatives to revise our general-purposed structural reconstruction algorithm:

Anterior Approach: Employ the apriori constraints of a 2D manifold in the general algorithm from the start, by revising the deletion, path blocking and local optimization techniques, similar to the algorithm in [15].

Posterior Approach: Neglect the given information and use the general algorithm to produce a set of 3D faces which are themselves 2D manifolds. Assemble the 3D faces into a single 2D manifold. This approach, called *manifold assembly*, appears to be more efficient than the previous one.

Example 7. Figure 6(a) shows a torus in 3D space. Without the requirement that the output be a 2D manifold, it will be shown in the next subsection that our structural reconstruction algorithm produces a 4D-face explanation in which the boundary is composed of 6 triangular columns (3D faces), with side faces $F_1F_{1'}F_{1''}$, $F_2F_{2'}F_{2''}$, $F_3F_{3'}F_{3''}$, $F_1F_2F_3$, $F_{1'}F_{2'}F_{3'}$, $F_{1''}F_{2''}F_{3''}$ respectively. Denote the columns by their side faces. We need to assemble them to obtain the torus. Below we use this example to explain our manifold assembly technique.

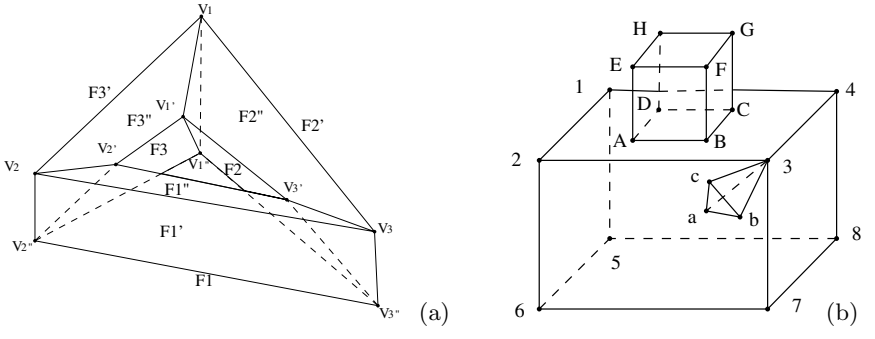


Fig. 6. 2D manifold assembly

The input is a set of 3D faces whose boundaries are 2D manifolds, denoted by $\mathcal{S}$. The output is a single 2D manifold. There are several steps:

(1) Compute the pairwise intersections of the 3D faces. Divide $\mathcal{S}$ into different connected components. In each component $\mathcal{C}$, count the *degree* of each edge in the 2D intersections, with respect to the 2D faces in $\mathcal{S}$.

For a set of rD faces, the *degree* of an $(r-1)$D face with respect to the set is the number of elements in the set passing through the $(r-1)$D face.

In Figure 6(a), the intersection between any two columns is a face. The columns form a single connected component. Each edge in the graph has degree 3. If face $F_{1''}$ is deleted, then the degree of every edge of $F_{1''}$ is reduced to 2.

(2) Now the assembling starts formally. Choose a 3D face called the *origin*, which has the maximal number of 3D neighbors in the component. Use $\mathcal{O}$ to denote the origin, and remove it from $\mathcal{C}$.

In Figure 6(a), any column has 5 neighbors. Choose any of them, say $F_1F_{1'}F_{1''}$, as the origin.

(3) The *first assembly principle* states that if two 3D faces intersect at a face whose edges are of degree greater than two, then the 3D faces can be assembled at the face by removing it; if one edge has degree two, then one 3D face must be deleted.

Use this principle repeatedly, update $\mathcal{C}$ and $\mathcal{O}$ accordingly, and move the assembled result from $\mathcal{C}$ to $\mathcal{O}$, until no 3D face in $\mathcal{C}$ has 2D intersection with $\mathcal{O}$.

In Figure 6(a), we can assemble columns $F_1F_{1'}F_{1''}$ and $F_{1''}F_{2''}F_{3''}$, and denote the result by $\mathcal{O}$. Then face $F_{1''}$ is deleted, and its neighbors $F_1, F_{1'}, F_{2''}, F_{3''}$ each have a degree-2 edge. The four neighbors disallow the other four columns to be annexed to $\mathcal{O}$. As a result, vertex $\mathbf{V}_{1''}$ is absent from $\mathcal{O}$. Although $\mathcal{O}$ itself is a 2D manifold, it does not provide the whole line drawing with such an explanation.

On the other hand, if we assemble $F_1F_{1'}F_{1''}$ and $F_2F_{2'}F_{2''}$, or start from $F_{1''}F_{2''}F_{3''}$ and assemble it with $F_{1'}F_{2'}F_{3'}$, we can get the torus explanation in two different manners.

(4) Use the second assembly principle repeatedly, update $\mathcal{C}$ and $\mathcal{O}$ accordingly, and move the assembled result from $\mathcal{C}$ to $\mathcal{O}$, until no 3D face in $\mathcal{C}$ has 1D intersection with $\mathcal{O}$.

If $\mathcal{O}$ is not a manifold and has an edge at whose endpoints $\mathbf{V}_1, \mathbf{V}_2$ the 2D manifold constraints are violated, then $\mathcal{O}$ can be assembled at the edge if and only if its $\mathbf{V}_1$-cycles of faces can be assembled at the edge by the second assembly principle. If $\mathcal{O}$ cannot be assembled at the edge then the whole assembling exits, else use the principle repeatedly until $\mathcal{O}$ no longer changes.

In Figure 5 (right), there are two tetrahedra sharing a common edge 56. The pair of faces 12365 and 567 are merged into a polyface and then replaced by face 123675. The pair of faces 15634 and 568 are merged and then replaced by face 158634. However, edge 56 is NOT deleted, although it is no longer in any face.

(5) The *third assembly principle* states that if two 3D faces intersect at a vertex $\mathbf{V}$ but not at any face or edge, then they can be assembled at the vertex if and only if within the line drawing, a face F_1 at $\mathbf{V}$ in one 3D face is within the 2D region of a face F_2 at $\mathbf{V}$ in the other 3D face. If no such two faces then one 3D face must be deleted; else the assembling is realized by replacing the pair of faces by their difference in the polyface generated by them.

Use this principle repeatedly, update $\mathcal{C}$ and $\mathcal{O}$ accordingly, and move the assembled result from $\mathcal{C}$ to $\mathcal{O}$, until $\mathcal{C}$ is empty.

If $\mathcal{O}$ is not a manifold, then if it has a vertex $\mathbf{V}$ where the 2D manifold constraint is violated, then $\mathcal{O}$ can be assembled at $\mathbf{V}$ if and only if its $\mathbf{V}$-cycles of faces can be assembled at the vertex by the third assembly principle. If $\mathcal{O}$ cannot be assembled at the vertex then the whole assembling exits, else use the principle repeatedly until $\mathcal{O}$ no longer changes.

In Figure 6(b) there are three 3D faces: two cubes and a tetrahedron. Vertex 3 is shared by the tetrahedron and a cube. Face 2376 can be merged with either face 3ab or face 3ac, leading to two different manifold structures.

(6) The *fourth assembly principle* states that if two 3D faces do not intersect, then they can be assembled at two 2D faces if and only if one 2D face is within the 2D region of the other 2D face. To assemble the two 3D faces is to replace the pair of faces by their difference in the polyface generated by them.

Now each set $\mathcal{C}$ has been replaced by the corresponding set $\mathcal{O}$. Check if each $\mathcal{O}$ is a manifold, and if not then the whole assembling exits. Use the fourth assembly principle repeatedly among the $\mathcal{O}$'s until the result is stable.

In Figure 6(b), the two cubes can be assembled at either the pair of faces $(1234, ABCD)$, or the pair $(1584, ABCD)$, leading to two different manifolds.

(7) If no manifold is constructed or more solutions are needed, then changing the order of 3D faces in the assembling sequence, including changing the origin, may lead to different results.

Remark. (a) rD assembling differs from rD merging in that in merging we do not delete anything, but in assembling we not only delete $(r-1)$D and rD faces, but also generate new $(r-1)$D and rD faces.

(b) If we employ the knowledge that Figure 6(a) represents a 3D manifold, then before the assembling we can simply delete one of the six columns, because its 3D content must be the algebraic union of the 3D contents of the other five.

3.7 The Main Algorithm

Input: (1) A 2D line drawing composed of vertices and edges. A vertex is represented by its 2D coordinates, an edge by two vertices.
(2) A set of lines. A line is represented by a sequence of collinear edges.
(3) A vertex as the origin of localization. The default is a vertex contained in a maximal number of edges.

Output: Objects of dimension > 1: faces and polyfaces.

Initialization: (1) Find all pairs of edges intersecting not at a vertex. (2) Repair lines with virtual parts.

Step 1. Localization start: Start from the origin, use rigidity localization to generate a set of new vertices.

Step 2. Cycle searching: Generate faces by the depth-first cycle searching strategy, together with the deletion, path blocking and local optimization techniques.

Step 3. Dimension upgrading: Start from the 2D local wireframe constructed so far, construct higher dimensional faces in a hierarchical order, following a procedure similar to Steps 1 to 3.

Step 4. Assembling: This occurs if there is any repair in the local wireframe, or the result is required to be a manifold. The assembling is also local.

Step 5. Localization end: Go back to Step 1 for another round of localization. Terminate after all vertices are included.

Step 6. More solutions: Explore the state-space tree or change the origin to get more solutions.

Remark. Although a specific set of 2D coordinates are given in the input, they are used only to test the inequalities occurring in the 2D non-self-intersection constraint, the 2D non-interior-intersection constraint and the 3D assembly. A solution based on these coordinates is acceptable as long as the polyhedral structures are compatible, no matter if the coordinates satisfy the realizability conditions for the geometric reconstruction.

Example 8. In Figure 6(a), the 4D interpretation is obtained as follows:

From 1D to 2D: The sequence of new vertices in the localization is:

$$\mathbf{V}_1 \text{ (origin)} \longrightarrow \mathbf{V}_2, \mathbf{V}_3, \mathbf{V}_{1'}, \mathbf{V}_{1''} \longrightarrow \mathbf{V}_{2'}, \mathbf{V}_{2''}, \mathbf{V}_{3'}, \mathbf{V}_{3''}. \tag{3.3}$$

In the first round, two rigid cycles are found:

$$G = \mathbf{V}_1\mathbf{V}_2\mathbf{V}_3, \quad G_1 = \mathbf{V}_1\mathbf{V}_{1'}\mathbf{V}_{1''},$$

no edge is deleted. In the second round, the 4 new vertices are singletons constructed by 4 elastic cycles:

$$\begin{aligned} &\mathbf{V}_{2'} : F_{3''} = \mathbf{V}_1\mathbf{V}_2\mathbf{V}_{1'}\mathbf{V}_{2'}, && \mathbf{V}_{2''} : F_{3'} = \mathbf{V}_1\mathbf{V}_2\mathbf{V}_{1''}\mathbf{V}_{2''},\\ &\mathbf{V}_{3'} : F_{2''} = \mathbf{V}_1\mathbf{V}_3\mathbf{V}_{1'}\mathbf{V}_{3'}, && \mathbf{V}_{3''} : F_{2'} = \mathbf{V}_1\mathbf{V}_3\mathbf{V}_{1''}\mathbf{V}_{3''}. \end{aligned}$$

Then the 4 edges passing through $\mathbf{V}_1$ are deleted, so is $\mathbf{V}_1$ itself. The 4 new vertices also generate 4 rigid cycles and 5 elastic cycles in which they are twins:

$$\begin{aligned} &G_2 = \mathbf{V}_2\mathbf{V}_{2'}\mathbf{V}_{2''}, && G_3 = \mathbf{V}_3\mathbf{V}_{3'}\mathbf{V}_{3''}, && G' = \mathbf{V}_{1'}\mathbf{V}_{2'}\mathbf{V}_{3'},\\ &G'' = \mathbf{V}_{1''}\mathbf{V}_{2''}\mathbf{V}_{3''}, && F_1 = \mathbf{V}_{2'}\mathbf{V}_{3'}\mathbf{V}_{2''}\mathbf{V}_{3''}, && F_2 = \mathbf{V}_{1'}\mathbf{V}_{3'}\mathbf{V}_{1''}\mathbf{V}_{3''},\\ &F_3 = \mathbf{V}_{1'}\mathbf{V}_{2'}\mathbf{V}_{1''}\mathbf{V}_{2''}, && F_{1'} = \mathbf{V}_2\mathbf{V}_3\mathbf{V}_{2''}\mathbf{V}_{3''}, && F_{1''} = \mathbf{V}_2\mathbf{V}_3\mathbf{V}_{2'}\mathbf{V}_{3'}. \end{aligned}$$

After this all edges are deleted, so no more cycles are constructed.

From 2D to 3D: It happens in every local wireframe of (3.3) but produces results only in the last one. The sequence of new faces (and edges therein) in the localization is

$$\begin{aligned} \mathbf{V}_1\mathbf{V}_2 \text{ (origin)} \longrightarrow F_{3'}, F_{3''}, G \longrightarrow F_3, G_1, G_2, F_{1'}, F_{2'}, G'', F_{1''}, F_{2''}, G'\\ \longrightarrow \mathbf{V}_{3'}\mathbf{V}_{3''}, F_1, F_2, G_3. \end{aligned} \tag{3.4}$$

The second local wireframe of (3.4) is generated by three rigid 2D cycles:

$$C_3 = G_1F_3F_{3'}F_{3''}G_2, \quad C' = GF_{1'}F_{2'}F_{3'}G'', \quad C'' = GF_{1''}F_{2''}F_{3''}G'.$$

Then faces $F_{3'}, F_{3''}, G$ and hence edge $\mathbf{V}_1\mathbf{V}_2$ are deleted. The last local wireframe of (3.4) is generated by three 2D rigid cycles which introduce edge $\mathbf{V}_{3'}\mathbf{V}_{3''}$:

$$C = G'F_1F_2F_3G'', \quad C_1 = G_2F_1F_{1'}F_{1''}G_3, \quad C_2 = G_1F_2F_{2'}F_{2''}G_3.$$

After this all faces are deleted.

From 3D to 4D: Only in the last wireframe of (3.4) does it produce any result. The localization is

$$G \text{ (origin)} \longrightarrow C', C'' \longrightarrow C, C_1, C_2, C_3. \tag{3.5}$$

The last wireframe is generated by the rigid cycle through C', C''. Thus the final result is a 3D rigid cycle, or equivalently, a 4D face.

We have implemented all the algorithms in VC++ 6.0, have tested all the examples in [1], [14], [15], [21], in addition to higher dimensional ones made by ourselves. We We have made a comparison between our algorithm and the existing fastest algorithm for face identification of 2D manifolds in [15]. It appears that our algorithm can handle complex manifold objects of over 10,000 faces by an IBM PC of Intel 2.60GHz CPU and 248MB RAM within one and a half hours, while their algorithm has to be stopped after running many more hours.

4 Geometric Reconstruction

The geometric reconstruction from $(r-1)$D to rD can be formulated as solving either Sugihara's fundamental equations (2.7) for the unknowns h's and $\mathbf{B}$'s, or Crapo's syzygy equations (2.9) for the unknowns h's. The problem is obviously linear for numerical data.

If the 2D coordinates are symbolic, then they must satisfy some unknown realizability conditions for the geometric reconstruction. After they are found, the realizability conditions must be further *triangulated* [32] in order to obtain a sequence of explicit geometric constructions for the 2D line drawing. From this aspect, all 2D coordinates are unknowns and the equations are highly nonlinear with a large number of unknowns. This explains the difficulty in symbolic geometric reconstruction.

Without loss of generality, we set $r = 3$ in this section and focus on the 2D to 3D lift. The syzygy equations are obtained by eliminating the $\mathbf{B}$'s from the fundamental equations. To further eliminate the h's from the system, and then triangulate the equations of the 2D coordinates, is usually very difficult.

How about eliminating the h's from the fundamental equations, and then further eliminating the $\mathbf{B}$'s? The fundamental equations (2.7) can be trivially split into two systems. The first system, called the $\mathbf{B}$-*system*, or *bivector system*, is that the height of any vertex $\mathbf{V}_i$ computed from any of its incident faces $F_1, \ldots, F_k$ are the same:

$$[\mathbf{iB}_1] = [\mathbf{iB}_2] = \cdots = [\mathbf{iB}_k], \text{ for any } 1 \leq j \leq k. \tag{4.1}$$

Here $\mathbf{i}$ denotes the 2D coordinates of vertex $\mathbf{V}_i$. The second system, called the *h-system*, or *height system*, is that one face F_1 is sufficient to characterize the height of its vertex $\mathbf{V}_i$:

$$h_i = [\mathbf{iB}_1]. \tag{4.2}$$

The $\mathbf{B}$-*system* appears to be much easier to solve than the syzygy equations, although it has more unknowns and equations.

The $\mathbf{B}$-*system* is the starting point of our geometric reconstruction. We can use the powerful *vectorial equation-solving* [10] method to solve it. The following three formulas will used in this paper, whose proofs are easy and are omitted.

Type B.0.

$$\begin{cases} [\mathbf{V}_1\mathbf{B}] = [\mathbf{V}_1\mathbf{B}'] \\ [\mathbf{V}_2\mathbf{B}] = [\mathbf{V}_2\mathbf{B}'] \\ \mathbf{V}_1\mathbf{V}_2 \neq 0 \end{cases}$$

Solution: $\mathbf{B} = \mathbf{B}' + \omega_{12}\mathbf{V}_1\mathbf{V}_2$ for new parameter ω_{12}.

Type B.1.

$$\begin{cases} [\mathbf{V}_1\mathbf{B}] = [\mathbf{V}_1\mathbf{B}'] \\ [\mathbf{V}_2\mathbf{B}] = [\mathbf{V}_2\mathbf{B}'] \\ [\mathbf{V}_3\mathbf{B}] = [\mathbf{V}_3\mathbf{B}'] \\ [\mathbf{V}_1\mathbf{V}_2\mathbf{V}_3] \neq 0 \end{cases} \tag{4.3}$$

Solution: $\mathbf{B} = \mathbf{B}'$.

Type B.2.

$$\begin{cases} [\mathbf{V}_1\mathbf{B}] = [\mathbf{V}_1\mathbf{B}'] \\ [\mathbf{V}_2\mathbf{B}] = [\mathbf{V}_2\mathbf{B}'] \\ \ldots \quad\quad \ldots \\ [\mathbf{V}_k\mathbf{B}] = [\mathbf{V}_k\mathbf{B}'], \text{ where } k > 3 \\ [\mathbf{V}_1\mathbf{V}_2\mathbf{V}_3] \neq 0 \end{cases} \tag{4.4}$$

Solution: An expression of $\mathbf{B}$ by $\mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3$ and $k-3$ syzygy equations.

$$\begin{cases} [\mathbf{V}_1\mathbf{V}_2\mathbf{V}_3]\,\mathbf{B} = [\mathbf{V}_1\mathbf{B}']\mathbf{V}_2\mathbf{V}_3 - [\mathbf{V}_2\mathbf{B}']\mathbf{V}_1\mathbf{V}_3 + [\mathbf{V}_3\mathbf{B}']\mathbf{V}_1\mathbf{V}_2, \\ [\mathbf{V}_1\mathbf{B}'][\mathbf{V}_2\mathbf{V}_3\mathbf{V}_j] - [\mathbf{V}_2\mathbf{B}'][\mathbf{V}_1\mathbf{V}_3\mathbf{V}_j] + [\mathbf{V}_3\mathbf{B}'][\mathbf{V}_1\mathbf{V}_2\mathbf{V}_j] \\ \qquad\qquad -[\mathbf{V}_j\mathbf{B}'][\mathbf{V}_1\mathbf{V}_2\mathbf{V}_3] = 0, \text{ for } 3 < j \leq k. \end{cases}$$

To avoid bifurcation in vectorial equation-solving, some inequalities are necessary. In this section, we adopt the following **Practical Assumption**:

In the line drawing, no three neighboring vertices of a face are collinear.

4.1 Parametric Propagation

To solve the **B**-system, using elimination methods usually leads to complicated bifurcations, in which most branches are geometrically meaningless. We need a method to control the bifurcation, by *transforming the system into one with a minimal number of unknowns and equations.*

We propose a technique called *parametric propagation* to make the transformation. The idea is to solve the **B**-system *locally* by introducing suitable parameters, like the solving of type-**B.0** equations. The *propagation* is similar to the localization in structural reconstruction: First we choose a **B** as the "origin", and solve for other **B**'s neighboring to it when taken as faces, by introducing a minimal number of new parameters. The solved **B**'s are then put into the origin, and the propagation continues. In the end, the **B**-system is transformed into two subsystems, one with the **B**'s explicitly expressed by the new parameters, the other with the parameters only. The latter is called the *system of parameters.* The details can be read from the following example.

Example 9. The torus in Figure 6(a) is called the *Sugihara torus* [25]. Its **B**-system is as follows.

$$\begin{cases} [\mathbf{1B}_{2'}] = [\mathbf{1B}_{2''}] = [\mathbf{1B}_{3'}] = [\mathbf{1B}_{3''}] \\ [\mathbf{2B}_{1'}] = [\mathbf{2B}_{1''}] = [\mathbf{2B}_{3'}] = [\mathbf{2B}_{3''}] \\ [\mathbf{3B}_{2'}] = [\mathbf{3B}_{2''}] = [\mathbf{3B}_{1'}] = [\mathbf{3B}_{1''}] \\ [\mathbf{1'B}_{2}] = [\mathbf{1'B}_{2''}] = [\mathbf{1'B}_{3}] = [\mathbf{1'B}_{3''}] \\ [\mathbf{2'B}_{1}] = [\mathbf{2'B}_{1''}] = [\mathbf{2'B}_{3}] = [\mathbf{2'B}_{3''}] \\ [\mathbf{3'B}_{1}] = [\mathbf{3'B}_{1''}] = [\mathbf{3'B}_{2}] = [\mathbf{3'B}_{2''}] \\ [\mathbf{1''B}_{2}] = [\mathbf{1''B}_{2'}] = [\mathbf{1''B}_{3}] = [\mathbf{1''B}_{3'}] \\ [\mathbf{2''B}_{1}] = [\mathbf{2''B}_{1'}] = [\mathbf{2''B}_{3}] = [\mathbf{2''B}_{3'}] \\ [\mathbf{3''B}_{1}] = [\mathbf{3''B}_{1'}] = [\mathbf{3''B}_{2}] = [\mathbf{3''B}_{2'}] \end{cases} \tag{4.5}$$

Parametric propagation:

Round 0: Choose face $F_{3''}$ as the origin. Algebraically this is equivalent to setting $\mathbf{B}_{3''} = 0$.

Round 1: Propagate towards a neighbor of the origin, say F_3. Algebraically this is equivalent to solving a system of type **B.0**:

$$[\mathbf{1'B}_3] = 0,\ [\mathbf{2'B}_3] = 0 \xrightarrow{B.0} \mathbf{B}_3 = \omega_{1'2'}\mathbf{1'2'}.$$

Round 2: Now only $\mathbf{B}_{3'}$ can be solved without introducing new parameters, i.e., only $F_{3'}$ can be constructed geometrically. Propagation towards $F_{3'}$.

$$[\mathbf{1B}_{3'}] = 0,\ [\mathbf{2B}_{3'}] = 0,\ [\mathbf{1''B}_{3'}] = \omega_{1'2'}[\mathbf{1'2'1''}],\ [\mathbf{2''B}_{3'}] = \omega_{1'2'}[\mathbf{1'2'2''}]$$
$$\xrightarrow{B.2} [\mathbf{121''}]\mathbf{B}_{3'} = \omega_{1'2'}[\mathbf{1'2'1''}]\mathbf{12}, \quad \omega_{1'2'}\mathbf{12} \wedge \mathbf{1'2'} \wedge \mathbf{1''2''} = 0.$$

A Cayley factorization [29] has been carried out in the solution.

Round 3: To construct any new face, at least one new parameter must be introduced. Propagation towards $F_{1''}$.

$$[\mathbf{2B}_{1''}] = 0,\ [\mathbf{2'B}_{1''}] = 0 \xrightarrow{B.0} \mathbf{B}_{1''} = \omega_{22'}\mathbf{22'}.$$

Round 4: Propagation towards $\mathbf{B}_{2''}$, i.e., construct $F_{2''}$ in Figure 6(a).

$$[\mathbf{1B}_{2''}] = 0,\ [\mathbf{1'B}_{2''}] = 0,\ [\mathbf{3B}_{2''}] = -\omega_{22'}[\mathbf{232'}],\ [\mathbf{3'B}_{2''}] = \omega_{22'}[\mathbf{22'3'}]$$
$$\xrightarrow{B.2} [\mathbf{131'}]\mathbf{B}_{2''} = \omega_{22'}[\mathbf{232'}]\mathbf{11'}, \quad \omega_{22'}\mathbf{11'} \wedge \mathbf{22'} \wedge \mathbf{33'} = 0.$$

Rounds 5-8: Propagations towards $\mathbf{B}_1$, $\mathbf{B}_2$, $\mathbf{B}_{1'}$, $\mathbf{B}_{2'}$. They are similar to Round 4.

Solution: By setting $\mathbf{B}_{3''} = 0$ and using new parameters $\omega_{1'2'}, \omega_{22'}$, the original **B**-system is changed into

$$\begin{cases} \mathbf{B}_{3''} = 0, \\ \mathbf{B}_3 = \omega_{1'2'}\mathbf{1'2'}, \\ [\mathbf{121''}]\mathbf{B}_{3'} = \omega_{1'2'}[\mathbf{1'2'1''}]\mathbf{12}, \\ \mathbf{B}_{1''} = \omega_{22'}\mathbf{22'}, \\ [\mathbf{131'}]\mathbf{B}_{2''} = \omega_{22'}[\mathbf{232'}]\mathbf{11'}, \\ [\mathbf{2'3'2''}]\mathbf{B}_1 = \omega_{1'2'}[\mathbf{1'2'2''}]\mathbf{2'3'} - \omega_{22'}[\mathbf{22'3'}]\mathbf{2'2''}, \\ [\mathbf{1'3'1''}]\mathbf{B}_2 = \omega_{1'2'}[\mathbf{1'2'1''}]\mathbf{1'3'} - \omega_{22'}[\mathbf{22'3'}]\mathbf{1'1''}, \\ [\mathbf{232''}]\mathbf{B}_{1'} = \omega_{1'2'}[\mathbf{1'2'2''}]\mathbf{23} + \omega_{22'}[\mathbf{232'}]\mathbf{22''}, \\ [\mathbf{131''}]\mathbf{B}_{2'} = \omega_{1'2'}[\mathbf{1'2'1''}]\mathbf{13} + \omega_{22'}[\mathbf{232'}]\mathbf{11''}, \end{cases} \tag{4.6}$$

together with

$$\begin{cases} \omega_{22'}\mathbf{11'} \wedge \mathbf{22'} \wedge \mathbf{33'} = 0, \\ \omega_{1'2'}\mathbf{12} \wedge \mathbf{1'2'} \wedge \mathbf{1''2''} = 0, \\ (\omega_{1'2'}[\mathbf{1'2'3'}] - \omega_{22'}[\mathbf{22'3'}])\mathbf{1'1''} \wedge \mathbf{2'2''} \wedge \mathbf{3'3''} = 0, \\ (\omega_{1'2'}[\mathbf{1'2'2''}] - \omega_{22'}[\mathbf{22'2''}])\ \mathbf{23} \wedge \mathbf{2'3'} \wedge \mathbf{2''3''} = 0, \\ \omega_{22'}([\mathbf{11''3''}][\mathbf{232'}][\mathbf{2'3'2''}] + [\mathbf{131''}][\mathbf{22'3'}][\mathbf{2'2''3''}]) \\ = \omega_{1'2'}([\mathbf{131''}][\mathbf{1'2'2''}][\mathbf{2'3'3''}] - [\mathbf{133''}][\mathbf{1'2'1''}][\mathbf{2'3'2''}]). \end{cases} \tag{4.7}$$

By parametric propagation, we have successfully reduced the 27 equations in (4.5) to the 5 equations in (4.7), and reduced the 8 bivector unknowns to 2 scalar unknowns.

The next task is to solve for the ω's from (4.7) so that the 2D faces given by (4.6) constitute a 2D manifold in 3D.

4.2 Calotte Factorization

In (4.7), the first 4 equations are in factored form, so each can be decomposed into two equations, one with the parameters and the other without. This is a great simplification. Although the last equation in (4.7) itself cannot be factored, it becomes so if the other 4 equations are used.

The algebraic factorization for the last equation is very delicate. Below we propose a very simple technique called *calotte factorization*, to produce solutions in factored form for equations of type **B.2**, using the *local geometric information*. It is a geometric method for algebraic factorization. Although this technique only applies to the reconstruction from 2D to 3D, it is general enough and very effective in simplifying the system of parameters.

The idea is inspired by Crapo's work [5], where it is suggested that for bracket polynomials produced in geometric computation, the geometric background can help finding rational Cayley factorization. Let F be a face of m vertices, called

the *center*. A *regular m-calotte* centered at F [5] is defined as a 2D F-cycle. The faces of the cycle are the *circumfaces*. Let $1, 2, \ldots, m$ be the vertices of the center, and let $11', 22', \ldots, mm'$ be the edges shared by neighboring circumfaces. Then the calotte can be denoted by $(12\ldots m,\ 1'2'\ldots m')$. In a *general calotte*, the center may not be a face.

Crapo's Theorem. (*cf.* [5]) *Let* $(12\ldots m,\ 1'2'\ldots m')$ *be a regular m-calotte, let the inhomogeneous coordinates of the center and the circumfaces be* $\mathbf{B}_0$ *and* $\mathbf{B}_i$ *for* $1 \le i \le m$. *If the center and* $m-1$ *circumfaces have been constructed, then the last circumface can be constructed if and only if*

$$(\mathbf{B}_i - \mathbf{B}_0)c = 0, \quad \textit{or equivalently,} \quad (\mathbf{B}_{i+1} - \mathbf{B}_i)c = 0 \quad \textit{for some} \quad 1 \le i \le m, \tag{4.8}$$

where

$$\begin{aligned} c = &-[\mathbf{1'12}][\mathbf{2'23}]\cdots[(\mathbf{m-1})'(\mathbf{m-1})\mathbf{m}][\mathbf{m'm1}] \\ &+[\mathbf{2'12}][\mathbf{3'23}]\cdots[\mathbf{m'}(\mathbf{m-1})\mathbf{m}][\mathbf{1'm1}] \end{aligned}$$

is the Crapo binomial. The equation in (4.8) is called the calotte equation.

For a general 3-calotte, no matter if its center is a face or not, the calotte equation is

$$(\mathbf{B}_2 - \mathbf{B}_1)\mathbf{11'} \wedge \mathbf{22'} \wedge \mathbf{33'} = 0. \tag{4.9}$$

Calotte factorization is to replace the syzygy equation in the solution of type-**B.2** equations by a calotte equation, using Crapo's Theorem in the procedure of parametric propagation. If several calotte equations are available, then any of them will do.

Example 10. In the parametric propagation of Example 9, the calottes are produced along with the construction of new faces:

$$\begin{aligned} \mathbf{B}_{3''} &\longrightarrow \mathbf{B}_3 \\ &\longrightarrow \mathbf{B}_{3'},\ \text{calotte } (11'1'', 22'2'') \\ &\longrightarrow \mathbf{B}_{1''}, \mathbf{B}_{2''},\ \text{calotte } (123, 1'2'3') \\ &\longrightarrow \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_{1'}, \mathbf{B}_{2'},\ \text{calottes} \\ &\qquad (1'2'3', 1''2''3''), (22'2'', 33'3''), (11'1'', 33'3''), (123, 1''2''3''). \end{aligned}$$

The equations of the first 4 calotte are exactly the first 4 equations in (4.7) when applying (4.6). The last equation in (4.7) is a syzygy equation obtained by solving for $\mathbf{B}_{2'}$, i.e., constructing $F_{2'}$, in the last round of propagation. By Crapo's Theorem, this equation can be replaced by the calotte equation of $(11'1'', 33'3'')$. The result after applying (4.6) is

$$\begin{aligned} (\omega_{1'2'}[\mathbf{131'}][\mathbf{1'2'1''}]\mathbf{1'3'} - \omega_{22'}([\mathbf{131'}][\mathbf{22'3'}]\mathbf{1'1''} + [\mathbf{1'3'1''}][\mathbf{232'}]\mathbf{11'})) \\ \mathbf{13} \wedge \mathbf{1'3'} \wedge \mathbf{1''3''} = 0. \end{aligned} \tag{4.10}$$

4.3 Solving the System of Parameters

Now we are ready to solve the system of parameters. We illustrate this by solving (4.7). Since the reconstruction result is a manifold, both $\omega_{1'2'}$ and $\omega_{22'}$ must be nonzero.

Lemma 1. *If $\omega_{1'2'}, \omega_{22'}$ are both nonzero, then at most one of the following three equalities holds:*

$$\omega_{1'2'}[\mathbf{131'}][\mathbf{1'2'1''}]\mathbf{1'3'} - \omega_{22'}([\mathbf{131'}][\mathbf{22'3'}]\mathbf{1'1''} + [\mathbf{1'3'1''}][\mathbf{232'}]\mathbf{11'}) = 0 \quad (4.11)$$

$$\omega_{1'2'}[\mathbf{1'2'2''}] - \omega_{22'}[\mathbf{22'2''}] = 0 \quad (4.12)$$

$$\omega_{1'2'}[\mathbf{1'2'3'}] - \omega_{22'}[\mathbf{22'3'}] = 0 \quad (4.13)$$

After deleting nonzero factors, the system of parameters is decomposed into three subsystems:

$$\begin{cases} \mathbf{11'} \wedge \mathbf{22'} \wedge \mathbf{33'} = 0 \\ \mathbf{1'1''} \wedge \mathbf{2'2''} \wedge \mathbf{3'3''} = 0 \\ \mathbf{12} \wedge \mathbf{1'2'} \wedge \mathbf{1''2''} = 0 \\ \mathbf{23} \wedge \mathbf{2'3'} \wedge \mathbf{2''3''} = 0 \end{cases}, \begin{cases} \mathbf{11'} \wedge \mathbf{22'} \wedge \mathbf{33'} = 0 \\ \mathbf{12} \wedge \mathbf{1'2'} \wedge \mathbf{1''2''} = 0 \\ \mathbf{13} \wedge \mathbf{1'3'} \wedge \mathbf{1''3''} = 0 \\ \mathbf{23} \wedge \mathbf{2'3'} \wedge \mathbf{2''3''} = 0 \end{cases}, \begin{cases} \mathbf{11'} \wedge \mathbf{22'} \wedge \mathbf{33'} = 0 \\ \mathbf{1'1''} \wedge \mathbf{2'2''} \wedge \mathbf{3'3''} = 0 \\ \mathbf{12} \wedge \mathbf{1'2'} \wedge \mathbf{1''2''} = 0 \\ \mathbf{13} \wedge \mathbf{1'3'} \wedge \mathbf{1''3''} = 0 \end{cases} \quad (4.14)$$

Lemma 2. *If*

$$\begin{cases} \mathbf{12} \wedge \mathbf{1'2'} \wedge \mathbf{1''2''} = 0 \\ \mathbf{23} \wedge \mathbf{2'3'} \wedge \mathbf{2''3''} = 0 \\ \mathbf{11'} \wedge \mathbf{22'} \wedge \mathbf{33'} = 0 \\ \mathbf{1'1''} \wedge \mathbf{2'2''} \wedge \mathbf{3'3''} = 0 \end{cases} \quad (4.15)$$

then $\mathbf{13} \wedge \mathbf{1'3'} \wedge \mathbf{1''3''} = 0,\ \mathbf{11''} \wedge \mathbf{22''} \wedge \mathbf{33''} = 0$. *If*

$$\begin{cases} \mathbf{12} \wedge \mathbf{1'2'} \wedge \mathbf{1''2''} = 0 \\ \mathbf{13} \wedge \mathbf{1'3'} \wedge \mathbf{1''3''} = 0 \\ \mathbf{23} \wedge \mathbf{2'3'} \wedge \mathbf{2''3''} = 0 \\ \mathbf{11''} \wedge \mathbf{22''} \wedge \mathbf{33''} = 0 \end{cases} \quad (4.16)$$

then $\mathbf{11'} \wedge \mathbf{22'} \wedge \mathbf{33'} = 0,\ \mathbf{1'1''} \wedge \mathbf{2'2''} \wedge \mathbf{3'3''} = 0$ *under the condition* $[\mathbf{1'2'3'}] \neq 0$.

By this lemma, in the first and third subsystems of (4.14), the six tuples

$$\begin{gathered} (\mathbf{12}, \mathbf{1'2'}, \mathbf{1''2''}),\ (\mathbf{13}, \mathbf{1'3'}, \mathbf{1''3''}),\ (\mathbf{23}, \mathbf{2'3'}, \mathbf{2''3''}), \\ (\mathbf{11'}, \mathbf{22'}, \mathbf{33'}),\ (\mathbf{11''}, \mathbf{22''}, \mathbf{33''}),\ (\mathbf{1'1''}, \mathbf{2'2''}, \mathbf{3'3''}) \end{gathered} \quad (4.17)$$

are all concurrent lines. The corresponding configuration is called the *Triple Desargues Configuration*. By Desargues Theorem, the six intersections $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{A'}$, $\mathbf{B'}, \mathbf{C'}$ of the six tuples lie on two lines, with each line passing through three intersections. In this configuration, both $\omega_{1'2'}, \omega_{22'}$ are free of any equality constraint, and according to (4.6), the line drawing can be lifted to a spatial torus.

For the second subsystem of (4.14), if it is not the Triple Desargues Configuration, then $[\mathbf{1''2''3''}] = 0$ and $\omega_{1'2'}[\mathbf{1'2'3'}] = \omega_{22'}[\mathbf{22'3'}]$. By (4.6), $\mathbf{B}_1 = \mathbf{B}_2 = \mathbf{B}_3$, the solution is not a manifold.

Theorem on Sugihara Torus. *A line drawing of Sugihara torus with vertices $\mathbf{i}, \mathbf{i'}, \mathbf{i''}$ for $1 \leq i \leq 9$ is realizable in 3D if and only if it is a Triple Desargues Configuration, i.e., if and only if the following are concurrent 3-tuples of image lines:*

$$(\mathbf{ij}, \mathbf{i'j'}, \mathbf{i''j''}),\quad (\mathbf{ii'}, \mathbf{ii''}, \mathbf{i'i''}),\quad \textit{for } 1 \leq i < j \leq 3.$$

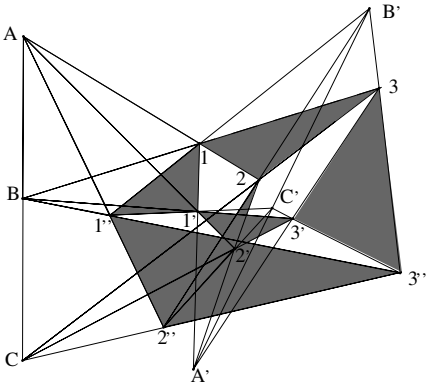


Fig. 7. Triple Desargues configuration

The configuration space of all Sugihara tori allows global parametrization. Let $\mathbf{2}, \mathbf{3}, \mathbf{1}', \mathbf{2}', \mathbf{3}', \mathbf{1}'', \mathbf{2}''$ *be free points in the plane, let* $\mathbf{B}_{3''}$ *be a free bivector in the image plane and let* $\omega_{1'2'}, \omega_{22'}$ *be free parameters, then the following is a global parametrization.*

$$\left\{\begin{array}{rl}
\mathbf{1} &= \mathbf{2}(\mathbf{1}'\mathbf{2}' \wedge \mathbf{1}''\mathbf{2}'') \wedge \mathbf{1}'(\mathbf{22}' \wedge \mathbf{33}') \\
\mathbf{3}'' &= (\mathbf{23} \wedge \mathbf{2}'\mathbf{3}')\mathbf{2}'' \wedge (\mathbf{1}'\mathbf{1}'' \wedge \mathbf{2}'\mathbf{2}'')\mathbf{3}' \\
\mathbf{B}_3 &= \mathbf{B}_{3''} + \omega_{1'2'}\mathbf{1}'\mathbf{2}' \\
\mathbf{B}_{3'} &= \mathbf{B}_{3''} + \omega_{1'2'}([\mathbf{1}'\mathbf{2}'\mathbf{1}'']/[\mathbf{121}''])\mathbf{12} \\
\mathbf{B}_{1''} &= \mathbf{B}_{3''} + \omega_{22'}\mathbf{22}' \\
\mathbf{B}_{2''} &= \mathbf{B}_{3''} + \omega_{22'}([\mathbf{232}']/[\mathbf{131}'])\mathbf{11}' \\
\mathbf{B}_1 &= \mathbf{B}_{3''} + \omega_{1'2'}([\mathbf{1}'\mathbf{2}'\mathbf{2}'']/[\mathbf{2}'\mathbf{3}'\mathbf{2}''])\mathbf{2}'\mathbf{3}' - \omega_{22'}([\mathbf{22}'\mathbf{3}']/[\mathbf{2}'\mathbf{3}'\mathbf{2}''])\mathbf{2}'\mathbf{2}'' \\
\mathbf{B}_2 &= \mathbf{B}_{3''} + \omega_{1'2'}([\mathbf{1}'\mathbf{2}'\mathbf{1}'']/[\mathbf{1}'\mathbf{3}'\mathbf{1}''])\mathbf{1}'\mathbf{3}' - \omega_{22'}([\mathbf{22}'\mathbf{3}']/[\mathbf{1}'\mathbf{3}'\mathbf{1}''])\mathbf{1}'\mathbf{1}'' \\
\mathbf{B}_{1'} &= \mathbf{B}_{3''} + \omega_{1'2'}([\mathbf{1}'\mathbf{2}'\mathbf{2}'']/[\mathbf{232}''])\mathbf{23} + \omega_{22'}([\mathbf{232}']/[\mathbf{232}''])\mathbf{22}'' \\
\mathbf{B}_{2'} &= \mathbf{B}_{3''} + \omega_{1'2'}([\mathbf{1}'\mathbf{2}'\mathbf{1}'']/[\mathbf{131}''])\mathbf{13} + \omega_{22'}([\mathbf{232}']/[\mathbf{131}''])\mathbf{11}'' \\
h_1 &= [\mathbf{1}\mathbf{B}_{3''}], \quad h_2 = [\mathbf{2}\mathbf{B}_{3''}], \quad h_3 = [\mathbf{3}\mathbf{B}_{1''}] \\
h_{1'} &= [\mathbf{1}'\mathbf{B}_{3''}], \; h_{2'} = [\mathbf{2}'\mathbf{B}_{3''}], \; h_{3'} = [\mathbf{3}'\mathbf{B}_{1''}] \\
h_{1''} &= [\mathbf{1}''\mathbf{B}_3], \; h_{2''} = [\mathbf{2}''\mathbf{B}_3], \; h_{3''} = [\mathbf{3}''\mathbf{B}_1]
\end{array}\right. \tag{4.18}$$

(4.18) is in fact a *linear construction sequence* of the torus, i.e., a triangular form in which the polynomials are linear with respect to their leading variables. Algebraically, all the properties of resolvable sequences [26] needed in application are occupied by linear construction sequences.

Although a Sugihara torus is composed of three triangular columns as in Figure 2, the condition that the three columns can be reconstructed into manifolds does not guarantee that the torus can. What is more useful is the 4D structure, or equivalently, the 3D manifold structure of Figure 6(a). There are 6

triangular columns which form a cycle of 3D faces. In 3D one column is trivial and can be removed. That the torus can be reconstructed in 3D is equivalent to that the 5 columns can be respectively reconstructed in 3D, following Lemma 1 and 2. Further connection between high dimensional structural reconstruction and geometric reconstruction is a topic of our future research.

4.4 Higher Dimensional Case

For geometric reconstruction from 2D to nD where $n > 3$, the number of nontrivial incidence relations grows rapidly, and as a consequence, the system of parameters is much more complicated. By *trivial incidence relations* we mean those that can be deduced from lower dimensional ones.

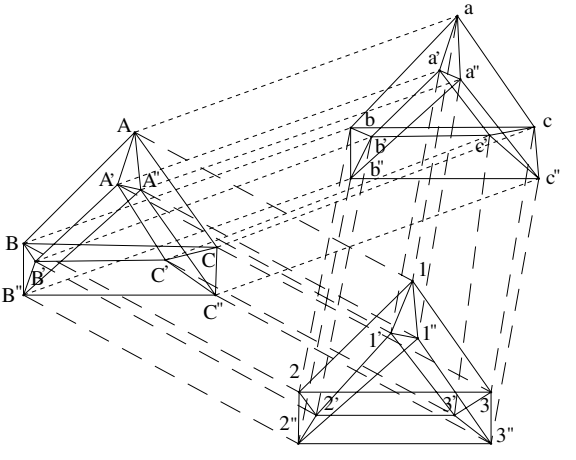


Fig. 8. 3D Sugihara torus

Example 11. Since $T^r = \overbrace{S^1 \times S^1 \times \cdots \times S^1}^{r}$, when representing S^1 with a triangular cycle, we obtain a wireframe of T^r, called *rD Sugihara torus*.

For $r = 3$, there are 81 nontrivial 2D coplanarity constraints, and 3 nontrivial 3D coplanarity constraints. In Figure 8, the 3D coplanarities are that $1231'2'3'1''2''3''$, $abca'b'c'a''b''c''$ and $ABCA'B'C'A''B''C''$ are respectively 3D.

In geometric reconstruction from 2D to 3D, 4 new parameters are introduced, and are all free ones in manifold parametrization. From 3D to 4D, 2 new parameters are introduced, and are both free ones in manifold parametrization.

The parametric propagation and calotte factorization algorithms have been implemented with Maple 8 and tested by more than 20 examples.

5 Conclusion

In this paper, we study the fascinating problem of nD polyhedral scene reconstruction from a single 2D line drawing, from both structural and geometric

aspects. We propose a number of very efficient techniques for the reconstructions, and test them by common examples for $n = 3$, showing the superiority of our algorithms. We believe our algorithms are the first for $n > 3$, and have a lot of room for further improvement despite their current efficiency.

References

1. S. C. Agarwel and W. Waggenspack. Decomposition method for extracting face topologies from wireframe models. *Computer Aided Design* **24**(3): 123–140, 1992.
2. J. Bokowski and B. Sturmfels. *Computational Synthetic Geometry*, **LNM 1355**, Springer, Berlin, Heidelberg, 1989.
3. S. Courter and J. Brewer. Automated conversion of curvilinear wireframe models to surface boundary models. *Comput. Graph.* **20**(4): 171–178, 1986.
4. H. Crapo and W. Whiteley. Stresses on Frameworks and Motions of Panel Structures: a Projective Geometric Introduction. *Structural Topology* **6**: 42-82, 1982.
5. H. Crapo. Invariant-Theoretic Methods in Scene Analysis and Structural Mechanics. *J. of Symbolic Computation* **11**: 523-548, 1991.
6. M. Ganter and J. Uicker. From wireframe to solid geometric: Automated conversion of data representations. *Computer in Mechanical Eng.* **2**(2): 40–45, 1983.
7. P. Hanrahan. Creating volume models from edge-vertex Graphs. *Computer Graphics* **16**(3): 77–84, 1982.
8. D. Huffman. Realizable Configurations of Lines in Pictures of Polyhedra, *Machine Intelligence* **8**: 493-509, 1977.
9. Y. Leclerc and M. Fischler. An optimization based approach to the interpretation of single line drawings as 3d wireframes. *International J. Computer Vision* **8(2)**: 113–136, 1992.
10. H. Li. Vectorial Equation-Solving for Mechanical Geometry Theorem Proving. *J. Automated Reasoning* **25**: 83-121, 2000.
11. H. Li and G. Sommer. Coordinate-free projective geometry for computer vision. In: *Geometric Computing with Clifford Algebra*, G. Sommer (ed.), Springer Berlin Heidelberg, pp. 415-454, 2001.
12. H. Li and L. Zhao. A symbolic approach to polyhedral scene analysis by parametric calotte propagation, *MM Research Preprints* **22**: 194-230, 2003. Available: http://www.mmrc.iss.ac.cn/pub/mm22.pdf/13.pdf.
13. H. Li, Q. Wang, L. Zhao, Y. Chen and L. Huang. nD object representation and detection from single 2D line drawing, *MM Research Preprints* **23**: 178-195, 2004. Available: http://www.mmrc.iss.ac.cn/pub/mm23.pdf/hli-2.pdf
14. J. Liu and Y. Lee. A graph-based method for face identification from a single 2d line drawing. *IEEE Trans. on PAMI* **23(10)**: 1106–1119, 2001.
15. J. Liu, Y. Lee and C. Cham. Identifying faces in a 2d line drawing representing a manifold object. *IEEE Trans. on PAMI* **24(12)**: 1579–1593, 2002.
16. A. Mackworth. Interpreting Pictures of Polyhedral Scenes, *Artificial Intelligence* **4**: 121-137, 1973.
17. T. Marill. Emulation the human interpretation of line drawings as 3d objects. *International J. of Computer Vision* **6**(2): 147–161, 1991.
18. K. Miyazaki. *An Adventure in Multidimensional Space: The Art and Geometry of Polygons, Polyhedra, and Polytopes*. Wileyinterscience Publ, 1983.
19. J. Richter-Gebert. *Realization Spaces of Polytopes*. **LNM 1643**, Springer, Wien, 1996.

20. L. Ros and F. Thomas. Overcoming Superstrictness in Line Drawing Interpretation. *IEEE Trans. PAMI* **24(4)**: 456-466, 2002.
21. M. Shpitalni and H. Lipson. Identification of faces in a 2d line drawing projection of a wireframe object. *IEEE Trans. on PAMI* **18(10)**: 1000–1012, 1996.
22. B. Sturmfels. *Algorithms in Invariant Theory*. Springer, Wien, 1993.
23. K. Sugihara. Mathematical Structures of Line Drawings of Polyhedrons – Towards Man-Machine Communication by Means of Line Drawings. *IEEE Trans. on PAMI* **4**: 458-469, 1982.
24. K. Sugihara. An Algebraic Approach to Shape-from-Image Problems, *Artificial Intelligence* **23**: 59-95, 1984.
25. K. Sugihara. *Machine Interpretation of Line Drawings*, MIT Press, Cambridge, Mass, 1986.
26. K. Sugihara. Resolvable Representation of Polyhedra. *Discrete Computational Geometry* **21(2)**: 243-255, 1999.
27. A. Turner, D. Chapman and A. Penn. Sketching space. *Computers and Graphics* **24**, 869–879, 2000.
28. N. White and W. Whiteley. The Algebraic Geometry of Stresses in Frameworks, *SIAM J. Alg. Discrete Math.* **4**: 481-511, 1983.
29. N. White. Multilinear Cayley Factorization. *J. of Symbolic Computation* **11**: 421-438, 1991.
30. W. Whiteley. From a Line Drawing to a Polyhedron. *J. Math. Psych.* **31**: 441-448, 1987.
31. W. Whiteley. Matroids and Rigid Structures. In *Matroid Applications*, N. White *ed.*, Encyclopedia of Mathematics and Its Applications **40**, pp. 1-53, Cambridge University Press, Cambridge, 1992.
32. W.-T. Wu. *Mathematics Mechanization*, Science Press and Kluwer Academic Publishers, Beijing 2000.

Planar Generalized Stewart Platforms and Their Direct Kinematics

Gui-Fang Zhang[1,⋆] and Xiao-Shan Gao[2,⋆⋆]

[1] School of Sciences, Beijing Forestry University, Beijing 100083, China
gfzhang@bjfu.edu.cn
[2] Key Laboratory of Mathematics Mechanization,
Institute of Systems Science, AMSS,
Academia Sinica, Beijing 100080, China
xgao@mmrc.iss.ac.cn

Abstract. In this paper, we introduce the concept of *planar generalized Stewart platform* (GSP) consisting of two rigid bodies connected with three constraints between three pairs of geometric primitives in the two rigid bodies respectively. This problem can be treated as a special but important class of geometric constraint solving problems. We show that there exist sixteen forms of planar GSPs. We also obtain the closed-form solutions of the direct kinematics for the planar GSPs. For a class of GSPs with two distance and one angular constraints, we may give pure geometric solutions based on ruler and compass constructions.

Keywords: Planar generalized Stewart platform, geometric constraint solving, direct kinematics, closed-form solution.

1 Introduction

The Stewart platform, originated from the mechanism designed by Stewart for flight simulation [22] and the mechanism designed by Gough for tire test [10], is a spatial parallel manipulator consisting of two rigid bodies: a moving platform, or simply a platform, and a base. The position and orientation (pose) of the base are fixed. The base and platform are connected with six extensible legs. For a set of given lengths of the six legs, the pose of the platform could generally be determined. The Stewart platform has been studied extensively in the past twenty years and has many applications. Comparing to serial mechanisms, the main advantage of the Stewart platform is its inherent stiffness and high load/weight ratio. For more information on the platform, please consult [2, 4, 13, 15, 18, 19]. A large portion of the work on Stewart platform is focused on the *direct kinematics*[13, 15, 18, 19].

On the other hand, *geometric constraint solving* is the central topic in much of the current work of developing intelligent CAD systems [5, 11, 12, 14, 20]. It

⋆ Partially supported by a NSFC grant (No. 60225016).
⋆⋆ Partially supported by a National Key Basic Research Project of China.

H. Hong and D. Wang (Eds.): ADG 2004, LNAI 3763, pp. 198–211, 2006.

also has applications in molecular modelling, linkage design, computer vision and computer aided instruction. Geometric constraint solving algorithms accept the declarative description of geometric diagrams or engineering drawings as the input and output a drawing procedure. In [6, 7], as a special class of geometric constraint problems, we introduce the spatial generalized Stewart platform (GSP) consisting of two rigid bodies connected with six distance and/or angular constraints between six pairs of points, lines and/or planes on the base and the moving platform respectively, which could be considered as the most general form of parallel manipulators with six DOFs in certain sense. We prove that there exist 3850 possible forms of GSPs which could provide more practical six DOFs parallel manipulators. The original Stewart platform is one of the GSPs in [6], where the six constraints are distance constraints between points.

While a majority of the work on Stewart platform focuses on the spatial case, several people also considered the planar Stewart platform which consists a moving platform and a base connected with three extensible legs. In [21], Pennock and Kanssner proved that the the upper bound of the number of solutions for the direct kinematics of the planar Stewart platform is six. Gosselin and Merlet developed robust solving schemes and established sharper bounds for special planar Stewart platforms [9]. Other interesting work on the planar Stewart platform could be found in [1, 3, 16, 17].

In this paper, we introduce the *planar generalized Stewart platform* which could be considered as the most general form of planar parallel manipulators with three DOFs in certain sense. A planar *GSP* consists of a base and a moving platform connected with three distance or/and angular constraints between three pairs of points and/or lines on the base and platform respectively. We show that there exist sixteen forms of planar GSPs. The planar Stewart platform considered in previous work such as [21, 9] is a planar GSP where the three constraints are three distance constraints among three pairs of points.

The direct kinematics is to solve an algebraic equation system. The characteristic set method is a convenient and powerful tool to deal with such equations[25]. Using the characteristic set method, we could reduce the solving of an equation system into the solving of equations in triangular form and hence the solving of univariate equations. It should be noticed that these univariate polynomial equations are in "cascade" form, that is, the coefficients of an equation involve the roots of the previous equations. These equations in triangular form are called *closed-form solutions* in this paper. We show that closed-form solutions to the direct kinematics of all planar GSPs could be found with the characteristic set method [25]. With these closed-form solutions, upper bounds for the number of solutions of the direct kinematics in the general cases can also be given. For a class of GSPs involving an angular constraint, we provide a solution to the direct kinematics based on ruler and compass constructions.

The rest of the paper is organized as follows. In Section 2, we define the planar GSP. In Section 3, we give the solutions to direct kinematics for the planar GSPs. In Section 4, conclusions are given. The results presented in this paper were reported in the un-published technical report [8].

2 Geometric Constraint Solving and Generalized Stewart Platform

In this section, we will introduce the generalized Stewart platform as a special class of geometric constraint problems.

2.1 A General Method of Geometric Constraint Solving

We consider two types of *geometric primitives*: points and lines in the two dimensional Euclidean plane and two types of *geometric constraints*: the distance constraint between point/point, point/line and the angular constraint between line/line. A *geometric constraint problem* is to find all the possible solutions of a set of geometric primitives satisfying a set of geometric constraints.

In [7], we proposed a geometric constraint solving method. As shown in Figure 1, to solve a geometric constraint problem, we first use the C-tree decomposition algorithm to reduce the problem to general construction sequences, and then reduce the solving of general construction sequences to the solving of basic merge patterns, which are the smallest problems we have to solve in order to solve the original problem.

Let $\mathcal{B}$ and $\mathcal{U}$ be two sets of geometric primitives. A *basic merging pattern* is to determine the position of $\mathcal{U}$ assuming that the position of $\mathcal{B}$ are known and there exists a set of geometric constraints among geometric primitives in $\mathcal{B}$ and $\mathcal{U}$. We further assume that a basic merge pattern $(\mathcal{B},\mathcal{U})$ has the following properties.

1. $\mathcal{B}$ and $\mathcal{B}\cup\mathcal{U}$ are rigid bodies. Here, by a rigid body, we mean a structurally well-constrained problem [7].
2. There is no subset V of $\mathcal{U}$ such that $\mathcal{B}\cup V$ is a rigid body.

As shown in Figure 1, there are three classes of basic merge patterns. The type of explicit constructions means to construct one geometric primitive, that is, $\mathcal{U}$

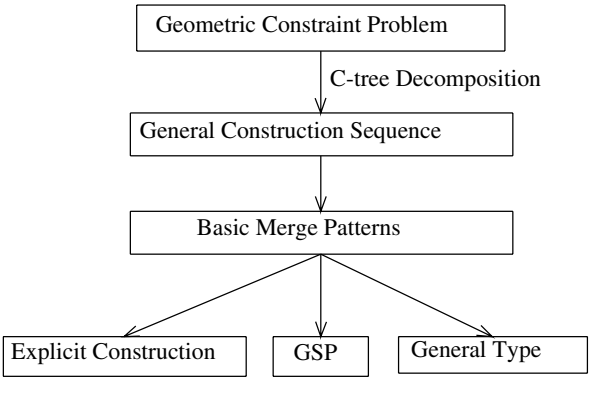


Fig. 1. Solving a constraint problem

consists of one geometric primitive. Explicit constructions are generally easy to solve. The next easy case is the general Stewart platform (GSP), where both $\mathcal{B}$ and $\mathcal{U}$ are rigid bodies and the problem is to determine the relative position of two rigid bodies according to three constraints. In the general case, $\mathcal{U}$ is not a rigid body and we need to determine the position of $\mathcal{U}$ using the constraints between primitives in $\mathcal{B}$ and $\mathcal{U}$ and constraints between primitives inside $\mathcal{U}$. In this paper, we will give closed-form solutions to the 2D GSPs.

2.2 Planar Generalized Stewart Platform

A rigid body in the plane has three DOFs. Therefore, to determine its position and orientation, we need three geometric constraints. This leads to the following definition.

Definition 1. *A planar* generalized Stewart platform *consists of two rigid bodies connected with three geometric constraints. One of the rigid bodies called* base *is fixed and the other rigid body called* platform *is movable. The position and orientation of the platform are determined by the values of the three constraints.*

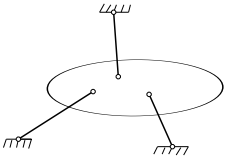


Fig. 2. Planar GSP

The planar GSP can be divided into two classes:

DDA. The GSP has two distance and one angular constraints.
DDD. The GSP has three distance constraints.

We cannot have more than one angular constraints due to the fact that a rigid body in the plane has one rotational DOF and the rotational DOF can generally be determined by one angular constraint.

Proposition 1. *If we assume that the geometric primitives in the base and platform are distinct, there are 6 types of* **DDA** *planar GSPs and 10 types of* **DDD** *planar GSPs. Totally, there are 16 types of planar GSPs.*

Proof. Let $d_i(a_i)$ be the number of possible ways to assign i distance(angular) constraints between the platform and the base. There is one type of angular constraint: line/line. For the point/line constraint, we need to consider two cases: line/point and point/line meaning that the line is on the platform and the base respectively. So we need only to consider three types of distance constraints: point/point, point/line and line/point.

The number of possible ways to select m objects from n types of objects is C_{m+n-1}^{m}. Then the number of possible types of GSPs with j distance constraints and i angular constraints is:

$$a_i d_j = C_{i+1-1}^{i} \cdot C_{j+3-1}^{j} = C_i^i \cdot C_{j+2}^{j} = C_{j+2}^{j}.$$

Then the number of DDA GSPs is: $d_2 = C_{2+2}^{2} = 6$ and the number of DDD GSPs is $d_3 = C_{3+2}^{3} = 10$. ∎

3 Closed-Form Solutions to the Direct Kinematics of Planar GSPs

The *direct kinematics* of a GSP $(\mathcal{B},\mathcal{U})$ is to find the position and direction of $\mathcal{U}$ relative to $\mathcal{B}$ assuming that the position and direction of $\mathcal{B}$ is fixed and the values for the three constraints between $\mathcal{B}$ and $\mathcal{U}$ are given.

3.1 The Characteristic Set Method

In what follows, we will use Ritt-Wu's characteristic set method [25, 23] to find the closed-form solutions of the direct kinematics of a GSP. Let V be a set of parameters, $x_i, i = 1, \ldots, p$ the variables to be determined, and PS $= 0$ a set of polynomial equations in V and the x_i. The method could be used to find a set of equations in *triangular form*, that is, an equation system

$$\mathrm{CS} = \begin{cases} f_1(V, x_1) = I_1(V)x_1^{d_1} + R_1(V, x_1) \\ f_2(V, x_1, x_2) = I_2(V, x_1)x_2^{d_2} + R_2(V, x_1, x_2) \\ \vdots \\ f_p(V, x_1, \ldots, x_p) = I_p(V, x_1, \ldots, x_{p-1})x_p^{d_p} + R_p(V, x_1, \ldots, x_p) \end{cases} \tag{1}$$

where $deg_{x_i} R_i(V, x_1, \ldots, x_i) < d_i (i = 1, \ldots, p)$. Variable x_i is called the *leading variable* of f_i. I_i is called the *initial* of f_i. For a set of values of the parameters V, we may solve x_i with the univariate equation $f_p(V, x_1, \ldots, x_i) = 0$ recursively under the condition $I_i \neq 0$. These univariate equations could be solved with either numerical methods or symbolic methods such as methods of real root isolation. It is clear that in order to solve a set of equations in triangular form, we need only to solve univariate equations.

For a set of polynomials PS and a polynomial D, let Zero(PS $/D$) be the set of solutions for all $P \in$ PS which are not solutions of $D = 0$. With Ritt-Wu's characteristic set method, we may decompose the solution set Zero(PS) as the union of the zero sets of several triangular sets:

$$\mathrm{Zero}(\mathrm{PS}\ ) = \cup_{i=1}^{m} \mathrm{Zero}(\mathcal{A}_i / J_i) \tag{2}$$

where each $\mathcal{A}_i$ is a triangular set and J_i is the product of the initials of the polynomials in $\mathcal{A}_i$. In this paper, when we say that the *closed-form solutions* of an equations system PS $= 0$ are given, we mean that we have reduced the PS $= 0$ to the solutions of triangular sets.

3.2 The DDA Planar GSPs

For planar **DDA** GSPs, we may solve the direct kinematic problem in two steps. First, we impose an angular constraint to determine the rotational DOF of the platform. Then we impose the distance constraints without breaking the angular constraint imposed previously. In this way, we generate a solution to the direct kinematic problem based on the ruler and compass construction.

1. Imposing Angular Constraint
Let $\mathcal{B}$ and $\mathcal{U}$ be the base and the platform of the GSP. After an angular constraint is imposed between $\mathcal{B}$ and $\mathcal{U}$, we need only to find a rotational matrix $\mathbf{R}$ such that $\mathbf{R}\mathcal{U}$ satisfies the angular constraint. We need only to consider angular constraints between two unit vectors on $\mathcal{B}$ and $\mathcal{U}$ respectively. Let s_1 be a unit vector on the base and s_2 a unit vector on the platform. Without loss of generality, we may further assume that $s_1 = s_2$. Let $\mathbf{R} = (r_{ij})_{2\times 2}$ be the rotational matrix. The angular constraint is imposed as follows. We assume that the platform is at some known place at the beginning. After imposing the angular constraint, the platform moves to the correct position by a rotation represented by the rotational matrix $\mathbf{R}$. So the angular constraint can be represented by

$$\cos(\angle(s_1, \mathbf{R}s_2)) = d.$$

Let $s_2 = s_1 = (l_1, m_1)$ where $l_1^2 + m_1^2 = 1$. We can obtain the following equation system:

$$\begin{cases} \mathbf{R^T R} = \mathbf{I} \\ det(\mathbf{R}) = 1 \\ s_1 \cdot \mathbf{R}s_2 = d \\ l_1^2 + m_1^2 = 1 \end{cases} \tag{3}$$

Applying Ritt-Wu's characteristic set method [24, 25] to equations (3) under the variable order $r_{11} > r_{22} > r_{12} > r_{21} > d > l_1 > m_1$, we have

$$\text{Zero}((3)) = \text{Zero}(\text{CS})$$

where CS is given below.

$$\text{CS} = \begin{cases} l_1^2 + m_1^2 = 1 \\ {r_{21}}^2 - 1 + d^2 = 0 \\ r_{12} + r_{21} = 0 \\ r_{22} - r_{11} = 0 \\ r_{11} - d = 0. \end{cases} \tag{4}$$

Proposition 2. *After imposing an angular constraint, the number of real solutions for the direction of the platform is at most two and this bound can be reached. Furthermore, the equations CS in triangular form provide closed-form solutions to the problem.*

Proof. Since equation system (4) consists of one quadratic equation and three linear equations in the variables $r_{i,j}$, the direct kinematics problem has at most two solutions. Furthermore, the problem has two real solutions if and only if $1 - d^2 > 0$ which is possible since $d = \cos(\angle(s_1, \mathbf{R}s_2))$. ∎

2. Imposing Distance Constraints
As mentioned in Section 2, there exist three kinds of distance constraints:

DPP: the distance constraint between two points,
DLP: the distance constraint between a line on the platform and a point on the base, and
DPL: the distance constraint between a point on the platform and a line on the base.

Definition 2. *For each distance constraint, say* $C =$**DPL**, *the locus of the corresponding geometric element* e *on the platform under the angular constraint and this distance constraint is called the* locus induced by this constraint, *and is denoted by* $\mathcal{L}_C$ *or* $\mathcal{L}_{DPL}$.

Proposition 3. *Let* D *be a distance constraint between a geometric element* e *on the platform and a geometric element on the base. If the direction of the platform is fixed, then the locus of* e, *that is* $\mathcal{L}_D$, *could be a circle or two lines.*

Proof. We use $\mathrm{DIS}(e_1, e_2)$ to denote the distance between a geometric element e_1 on the platform and a geometric element e_2 on the base. The loci induced by the three distance constraints can be determined as follows.

$\mathcal{L}_{DPP}$**.** For constraint $\mathrm{DIS}(p_1, p_2) = d$, the locus of point p_1 is a circle with center p_2 and radius d.
$\mathcal{L}_{DLP}$**.** For constraint $\mathrm{DIS}(l, p) = d$, if we only consider the distance constraint, then line l could be all the tangent lines of a circle with center p and radius d. If we further assume that the direction of line l is fixed, then the locus of l is two lines l_1 and l_2 which are parallel to the line l and with distance d to p.
$\mathcal{L}_{DPL}$**.** For constraint $\mathrm{DIS}(p, l) = d$, the locus of point p is two lines l_1 and l_2 which are parallel to the line l and with distance d to l.

In Figure 3, the circle in diagram (a) is the locus of **DPP**; the bold line l tangent to the circle in diagram (b) represents the line on the platform and the lines l_1 coincident to l and l_2 parallel to l is the locus of **DLP**; and two thin lines parallel to line l in diagram (c) is the locus of **DPL**. ∎

Proposition 4. *Let* D *be a distance constraint between a geometric element* e *on the platform and a geometric element on the base. If the direction of the platform is fixed, then the locus of any given point on* e *is* $\mathcal{L}_D$.

Proof. If e is a point, D must be either **DPP** or **DPL**. In this case, the statement is obviously valid. Otherwise, e is a line. From the above discussion, we know

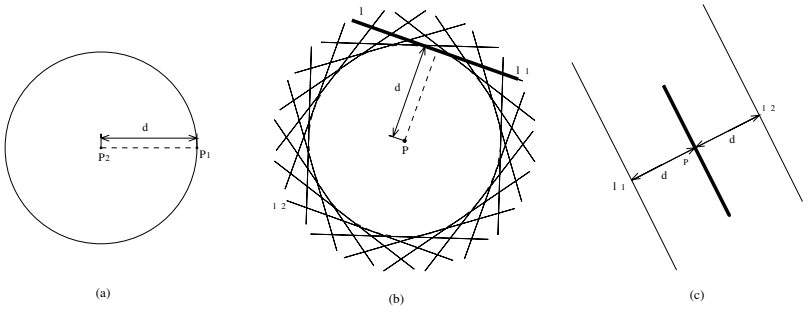


Fig. 3. Loci of distance constraints

that the collection of points on e is $\mathcal{L}_D$. Hence $\mathcal{L}_D$ could be considered as the locus for a given point on e. ∎

After the angular constraint is imposed, the direction of the platform is fixed. To find the position of the platform, we need only to find the position of a point on the platform.

Algorithm 1. *The input includes two distance constraints $D_i, i = 1, 2$ between geometric elements on the platform and the base. We further assume that the directions of the platform and hence the directions of the lines on the platform are fixed. The output is a new position for the platform such that the two distance constraints are satisfied.*

1. Determine the equations $E_i(x, y) = 0, i = 1, 2$ for the loci $\mathcal{L}_{D_i}$ as shown in Proposition 3.
2. Let D_i be a constraint between a geometric element e_i on the platform and f_i on the base. If e_i is a point, let $p_i = e_i$. Otherwise e_i is a line. Select an arbitrary fixed point on e_i as p_i. Let $p_i = (x_i, y_i)$.
3. By Proposition 4, after imposing the distance constraint D_i, point p_i is on the locus $\mathcal{L}_{D_i}$. Furthermore, since the direction of the platform is fixed, when imposing the constraint D_2, point p_1 must also be on the locus $\mathcal{L}'_2$ which is the translation of $\mathcal{L}_{D_2}$ at the direction $p_1 - p_2$. Then after imposing the two distance constraints, the new position p'_1 for point p_1 must be the intersection of two equations:

$$\begin{aligned} &E_1(x, y) = 0, \\ &E_2(x - x_1 + x_2, y - y_1 + y_2) = 0. \end{aligned} \tag{5}$$

4. By Proposition 3, (5) are equations for lines or circles. Then we need only to find the intersections of pairs of lines and circles, which are very easy to be solved. We generally could have two or four solutions.
5. Move the platform along the translation vector $t = p'_1 - p_1$, it will satisfy the two distance constraints.

So for the DDA case, we have the following conclusions.

1. To impose the angular constraint, we usually have two solutions.
2. To impose the two distance constraints, the problem is reduced to the intersection of a pair of lines/a circle which has four real solutions; a pair of lines/a pair of lines which has four real solutions; circle/circle which has two real solutions.

As a consequence, we have proved the following result.

Theorem 2. *We generally could have four or eight real solutions for a* **DDA** *problem depending on the types of the constraints imposed on it. Furthermore, these solutions can be obtained by rotating the platform and taking intersections between line/line, line/circle, or circle/circle.*

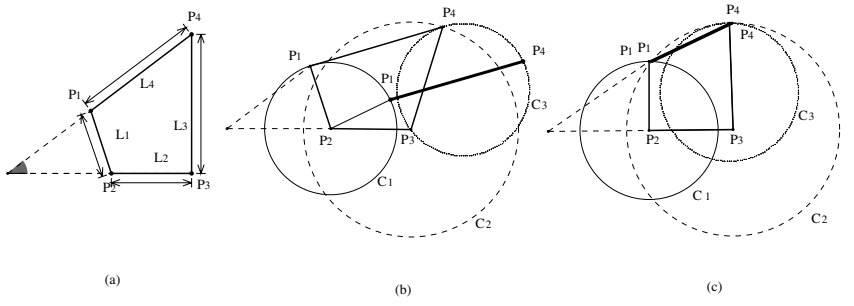


Fig. 4. A **DDA** geometric constraint problem and its geometric solution

Note that the solution given above is pure geometric. Let us illustrate this with the example in Figure 4. We may consider this as a **DDA** GSP by considering $p_1p_4l_4$ as the platform and $p_2p_3l_2$ as the base. We may solve this problem as follows.

1. Rotate line p_1p_4 so that the angle between line p_1p_4 and line p_2p_3 is the given angle.
2. Let c_1 be the circle with p_2 as center and $|p_2p_1|$ as the radius, c_2 the circle with p_3 as center and $|p_3p_4|$ as the radius, and c_3 the translation of c_1 along vector $p_4 - p_1$. The correct position for p_4 is the intersection of c_2 and c_3. Denote this intersection as p_4'.
3. The position for p_1' is $p_4' + p_1 - p_4$. The problem could have either one or two solutions as shown in (c) and (b) of Figure 4.

3.3 The DDD GSPs

A problem is called ruler and compass constructible, or *RC-constructible*, if the coordinates of its geometric elements can be found by solving univariate linear or quadratic equations.

Theorem 3. *As mentioned in section 3.2, there exist three kinds of distance constraints:* **DPP**, **DLP** *and* **DPL**. *The* **DDD** *GSPs can be divided into ten different sub-cases shown below. We use a new notation to represent these ten cases. For instance,* **PPP-LLL** *represents the GSP where the three distance constraints are between three points on the platform and three lines on the base respectively.*

1. *Direct kinematics for* **PPP-LLL** *and* **LLL-PPP** *can be reduced to the solving of one quadratic and three linear equations in the general case. Hence these GSPs are RC-constructible. Considering the fact that the distance constraint between a point and a line has two forms ($|pl| = \pm d$), each of* **PPP-LLL** *and* **LLL-PPP** *has at most 8 solutions.*
2. *Direct kinematics for* **PPP-LLP**, **LLP-PPP**, **LPP-PLL** *and* **LLP-PPL** *can be reduced to the solving of one quartic and three linear equations in the general case. We use the method in [5] to decide that the problems are not RC-constructible. Each of* **PPP-LLP** *and* **LLP-PPP** *has at most 16 solutions. Each of* **LPP-PLL** *and* **LLP-PPL** *has at most 32 solutions.*
3. *Direct kinematics for* **PPP-LPP**, **LPP-PPP**, **LPP-PLP** *and* **PPP-PPP** *can be reduced to the solving of one equation of degree six and three linear equations in the general case. The polynomials of degree six in these cases are irreducible. Then it is obvious that the problems are not RC-constructible. Each of* **PPP-LPP** *and* **LPP-PPP** *has at most 12 solutions.* **LPP-PLP** *has at most 24 solutions and* **PPP-PPP** *has at most six solutions.*

Proof. Let us consider the case **LPP-PLP**, which is to impose three distance constrains: **DPP**, **DPL** and **DLP** simultaneously. It is obvious that we can always get three non collinear points on the base and on the platform, respectively. If the primitive involved is a line, we can take a point on it.

Let the three points on the base be B_1, B_2 and B_3. Assuming that B_1 is the origin of the fixed coordinate system on the base, B_1B_2 the x-axis. The coordinates of three points on the base are $B_1 = (0, 0)$, $B_2 = (b_1, 0)$ and $B_3 = (b_2, b_3)$. Let the three points on the platform be D_1, D_2 and D_3. Assuming that point p is the origin of the moving coordinate system on the platform. The coordinate of point p in the fixed coordinate system is $p = (x_3, x_4)$, and point p is the foot of perpendicular line from point D_3 to line D_1D_2. Let $\angle(B_1B_2, D_1D_2) = \theta$, $x_1 = \cos\theta, x_2 = \sin\theta$. The moving coordinates of the three points on the platform are $D_1 = (-h_1, 0)$, $D_2 = (h_2, 0)$, $D_3 = (0, h_3)$, where h_1, h_2, h_3 are three nonnegative parameters. D_1D_2 is the x-axis of the moving coordinate system. The coordinates of D_1, D_2, D_3 in the fixed coordinate system are $D_{11} = (-h_1x_1+x_3, -h_1x_2+x_4)$, $D_{22} = (h_2x_1+x_3, h_2x_2+x_4)$ and $D_{33} = (-h_3x_2+x_3, h_3x_1 + x_4)$.

Let the parametric equation of the line l on the base be $p = B_3 + u_1s_1$, where $s_1 = (l_1, m_1)$ and $|s_1| = 1$. Let the parametric equation of line l_0 on the platform in the moving coordinate system be $P = D_2 + u_2s_2$ where $s_2 = (l_2, m_2)$ and $|s_2| = 1$. Then the parametric equation of line l_0 in the fixed coordinate system is $p = D_{22} + u_2s_{22}$, where $|s_{22}| = 1$ and $s_{22} = (l_2x_1 - m_2x_2, l_2x_2 + m_2x_1)$.

Let the three constraints be $|B_1D_{11}| = t$, $|B_2l_0| = t_1$ and $|D_{33}l| = t_2$, we have

$$\begin{array}{l} x_1^2 + x_2^2 - 1 = 0 \\ (-h_1x_1 + x_3)^2 + (-h_1x_2 + x_4)^2 - t^2 = 0 \\ (l_2x_2 + m_2x_1)(h_2x_1 + x_3 - b_1) - (l_2x_1 - m_2x_2)(h_2x_2 + x_4) - d_1 = 0 \\ m_1(-h_3x_2 + x_3 - b_2) - l_1(h_3x_1 + x_4 - b_3) - d_2 = 0 \end{array} \tag{6}$$

where $d_1 = \pm t_1$ and $d_2 = \pm t_2$.

Equation system (6) can be reduced to the following triangular form with Ritt-Wu's characteristic set method under the variable order $x_1 < x_2 < x_3 < x_4$.

$$\begin{array}{l} z_{41}\mathbf{x_1}^6 + z_{42}x_1^5 + z_{43}x_1^4 + z_{44}x_1^3 + z_{45}x_1^2 + z_{46}x_1 + z_{47} = 0 \\ (z_{31}x_1^2 + z_{32}x_1 + z_{33})\mathbf{x_2} + z_{34}x_1^3 + z_{35}x_1^2 + z_{36}x_1 + z_{37} = 0 \\ ((-m_1m_2 - l_1l_2)x_2 + (-l_1m_2 + m_1l_2)x_1)\mathbf{x_3} + m_1h_3x_2^2m_2 + ((-m_1h_3l_2 \\ +l_1h_3m_2)x_1 + l_1l_2b_1 - l_1b_3m_2 - d_2m_2 + m_1b_2m_2)x_2 - l_1h_3x_1^2l_2 + (d_2l_2 \\ +l_1b_1m_2 - m_1b_2l_2 + l_1b_3l_2)x_1 - l_1m_2h_2 - l_1d_1 = 0 \\ -l_1\mathbf{x_4} - m_1h_3x_2 + m_1x_3 - l_1h_3x_1 - m_1b_2 + l_1b_3 + d_2 = 0 \end{array} \tag{7}$$

where z_{ij} are the polynomials in the parameters l_i, m_j, and h_k, which may be found in the technical report [8]. The equations in (7) give the solution to the GSP in the generic case and hence the platform has at most six solutions. Considering the fact that $d_1 = \pm t_1$ and $d_2 = \pm t_2$, the problem could have twenty four solutions. For the other nine planar DDD GSPs, the proofs are quite similar. Details could be found in the technical report [8]. ∎

Example 1. The problem in Figure 5 can be reduced into merging two rigid bodies $p_1p_2p_3p_4$ and $p_5p_6p_7p_8$. We take $p_5p_6p_7p_8$ as the the base object and $p_1p_2p_3p_4$ the dependent object. The constraints are $|l_1p_4| = 0$, $|l_2p_3| = 0$ and $|p_5l_3| = 0$, which is an **LPP-PLL** GSP. Let $p_7 = (0, 0)$. The parametric equations for lines l_1, l_2 are $p = (0, 0) + u_1(0, 1)$ and $p = (0, 0) + u_2(1, 0)$. Let point p_3 be the origin of the moving coordinate system. Then $p_3 = (x_3, x_4)$. Let $|p_6p_7| = b_2$, $|p_5p_6| = b_3$ and $|p_3p_4| = h_3$. Thus the coordinates for points p_4 and p_5 are

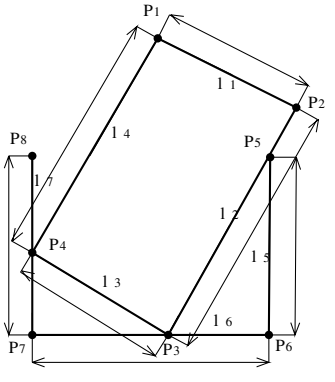


Fig. 5. An example of planar **DDD** GSP

$p_4 = (-x_2h_3 + x_3, x_1h_3 + x_4)$ and $p_5 = (b_2, b_3)$. The parametric equation of line l_3 is $p = (x_3, x_4) + u_3(x_1, x_2)$. The equation system is

$$\begin{cases} x_1^2 + x_2^2 - 1 = 0 \\ |x_2(b_2 - x_3) - x_1(b_3 - x_4)| = 0 \\ | - h_3x_2 + x_3| = 0 \\ |x_4| = 0 \end{cases} \tag{8}$$

Applying Ritt-Wu's characteristic set method to equation system (8) under the variable order $x_3 > x_2 > x_1 > b_2 > b_3 > h_3$, we obtain the following decomposition:

$$\text{Zero}((8)) = \cup_{i=1}^{6}\text{Zero}(\text{CS}_i/J_i)$$

where CS_i and J_i are given below.

$\text{CS}_1 = [b_2\mathbf{x_3} + h_3^2\mathbf{x_1}^2 - \mathbf{x_1}b_3h_3 - h_3^2, b_2\mathbf{x_2} + h_3\mathbf{x_1}^2 - \mathbf{x_1}b_3 - h_3, h_3^2\mathbf{x_1}^4 - 2b_3h_3\mathbf{x_1}^3 + (b_2^2 + b_3^2 - 2h_3^2)\mathbf{x_1}^2 + 2\mathbf{x_1}b_3h_3 + h_3^2 - b_2^2],\ J_1 = b_2h_3.$
$\text{CS}_2 = [\mathbf{x_3}, -\mathbf{x_2}b_2 + \mathbf{x_1}b_3, b_2^2\mathbf{x_1}^2 - b_2^2 + \mathbf{x_1}^2b_3^2, h_3], J_2 = b_2.$
$\text{CS}_3 = [\mathbf{x_3}, \mathbf{x_2} - 1, \mathbf{x_1}, b_2, h_3],\ J_3 = 1.$
$\text{CS}_4 = [\mathbf{x_3}, \mathbf{x_2} + 1, \mathbf{x_1}, b_2, h_3], J_4 = 1.$
$\text{CS}_5 = [\mathbf{x_3}, \mathbf{x_1}^2 + \mathbf{x_2}^2 - 1, b_2, b_3, h_3],\ J_5 = 1.$
$\text{CS}_6 = [h_3\mathbf{x_2} - \mathbf{x_3}, h_3\mathbf{x_2}^2 + \mathbf{x_1}b_3, h_3\mathbf{x_1}^2 - \mathbf{x_1}b_3 - h_3, b_2],\ J_6 = h_3.$

With the above zero decomposition, the solutions of (8) are reduced to the solutions of $\text{CS}_i = 0, i = 1, \ldots, 6$.

From the structure of these triangular sets, we could solve equation (8) as follows.

1. If $h_3 \neq 0, b_2 \neq 0$, we will use $\text{CS}_1 = 0$ to find the solutions.
2. If $h_3 \neq 0, b_2 = 0$, we will use $\text{CS}_6 = 0$ to find the solutions.
3. If $h_3 = 0, b_2 = 0, b_3 = 0$, we will use $\text{CS}_5 = 0$ to find the solutions.
4. If $h_3 = 0, b_2 = 0, b_3 \neq 0$, we will use $\text{CS}_3 = 0, \text{CS}_4 = 0$ to find the solutions.
5. If $h_3 = 0, b_2 \neq 0$, we will use $\text{CS}_2 = 0$ to find the solutions.

If we take $b_2 = \frac{1}{2}$, $b_3 = 0$ and $h_3 = 1$, we obtain four real solutions from $\text{CS}_1 = 0$, which are $(\frac{\sqrt{3}}{2}, 0, 0, 0)$, $(-\frac{\sqrt{3}}{2}, 0, 0, 0)$, $(1, 0, 0, 0)$ and $(-1, 0, 0, 0)$. So the problem has four real solutions at most.

4 Conclusions

A generalization of the planar Stewart platform is introduced by considering all possible geometric constraints between three pairs of geometric primitives on the base and the platform respectively. This gives 16 types of planar GSPs. The purpose of introducing these new types of planar Stewart platforms is to find new and better parallel mechanisms. We give closed-form solutions to the direct kinematics of these GSPs. For the six GSPs with two distance constraints and

one angular constraint, we are able to give a pure geometric solution based on ruler and compass constructions.

Acknowledgment. We want to thank the anonymous referees for valuable suggestions.

References

1. H.R.M. Daniali, P.J. Zsombor-Murray and J. Angles. Singularity Analysis of Planar Parallel Manipulators. Mech. Mach. Theorey, **30**(5), 665-678, 1995.
2. B. Dasgupta and T.S. Mruthyunjaya. The Stewart Platform Manipulator: a Review. Mech. Mach. Theorey, **35**(1), 15-40, 2000.
3. J. Duffy. Statics and Kiematics with Applications to Robotics. Cambridge University Press, 1996.
4. J.C. Faugere and D. Lazard. Combinatorial Classes of Parallel Manipulators. Mech. Mach. Theorey, **30**(6), 765-776, 1995.
5. X.S. Gao and S.C. Chou. Solving Geometric Constraint Systems II. A Symbolic Approach and Decision of Rc-constructibility. Computer-Aided Design, **30**(2), 115-122, 1998.
6. X.S. Gao, D. Lei, Q. Liao and G. Zhang. Generalized Stewart Platforms and Their Direct Kinematics. IEEE Trans. Robotics, **21**(2), 141-151, 2005.
7. X.S. Gao and G. Zhang. Geometric Constraint Solving via C-tree Decomposition. ACM SM03, 45-55, Seattle, USA, ACM Press, New York, 2003.
8. X.S. Gao and G. Zhang. Classification and Solving of Merge Patterns in Geometric Constraint Solving. MM-Preprints, **21**, 77-93, Inst. of Sys. Sci, 2002.
9. G.M. Gosselin and J.P. Merlet. The Direct Kinematics of Planar Parallel Manipulators: Special Architectures and Number of Solutions. Mech. Mach. Theorey, **29**(8), 1083-1097, 1994.
10. V.E. Gough. Automobile Stability, Control, and Tyre Performance. Proc. Automobile Division, Inst. Mech. E., 392-394, 1956.
11. C.M. Hoffmann and P.J. Vermeer. Geometric Constraint Solving in R^2 and R^3. In *Computing in Euclidean Geometry*, D. Z. Du and F. Huang (eds), World Scientific, Singapore, 266-298, 1995.
12. C.M. Hoffmann, A. Lomonosov and M. Sitharam. Decomposition Plans for Geometric Constraint Systems, I: Performance Measures for CAD, **31**(4), 367-408; II: New Algorithms, **31**(4), 409-427, J. of Symbolic Computation, 2001.
13. D. Kapur. Automated Geometric Reasoning: Dixon Resultants, Gröbner bases and Characteristic Sets. In *Automated Deduction in Geometry*, D. Wang(eds), LNAI 1360, 1-36, Springer, Berlin, 1998.
14. H. Lamure and D. Michelucci. Qualitative Study of Geometric Constraints. In *Geometric Constraint Solving and Applications*, 234-258, Springer, Berlin, 1998.
15. D. Lazard. Generalized Stewart Platform: How to Compute with Rigid Motions? In IMACS Symp. on. Symbolic Computation, 85-88, Lille, June, 14-17, 1993.
16. D. Lazard and J.P. Merlet. The (true) Stewart Platform has 12 Configurations. Proc. IEEE Int. Conf. on Robotics and Automation, 2160-2165, San Diego, May, 8-13, 1994.
17. H.L. Liu, T.Z. Zhang and H.S. Ding. Forward Solution of the 3-RPR Planar Parallel Mechanism with Wu's Method (in Chinese). Journal of Beijing Institute of Technology, **20**(5), 565-569, 2000.

18. J.P. Merlet. Parallel Robots. Kluwer, Dordrecht, 2000.
19. B. Mourrain. Enumeration Problems in Geometry, Robotics and Vision. Algorithms in Algebraic Geometry and Applications, eds. L. Gonzalez and T. Recio, **143**, Prog. in Math., 285-306. Birkhäuser, 1996.
20. J.C. Owen. Algebraic Solution for Geometry from Dimensional Constraints. In *ACM Symp., Foundation of Solid Modeling*, 397-407, ACM Press, New York, 1991.
21. G.R. Pennock and D.J. Kassner. Kinematic Analysis of a Planar Eightbar Linkage: Application to a Platform-type Robot. Trans. ASME, J. Mech. Des., **114**(1), 87-95, 1992.
22. D. Stewart. A Platform with Six Degrees of Freedom. Proc. Inst. of Mech. Eng., London, **180**(1), 371-386, 1965.
23. D. Wang. Elimination Methods. Springer-Verlag, Wien, New York, 2001.
24. D.K. Wang. Wsolve, http://www.mmrc.iss.ac.cn/~ dwang/wsolve.
25. W.T. Wu. Mechanical Theorem Proving in Geometries: Basic Principles. Springer-Verlag, Wien, New York, 1994.

Author Index