Proceedings of the DARPA Image Understanding Workshop, May, 1989, pp. 764–773.

Robust Geometric Computations for Vision and Robotics *

Victor J. Milenkovic
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

A new technique, the *hidden variable method*, is described for the creation of *robust* geometric algorithms: algorithms which can be implemented with absolute reliability using rounded finite precision arithmetic. The specific application given here is an algorithm for computing arrangements of line segments in two dimensions: determining how a set of line segments subdivides the plane into vertices, line segments, and polygons. Using *rubber band curves*, an implicit representation for the line segments, the algorithm can generate an arrangement with N bits of accuracy using rounded arithmetic with just $N+6$ bits of precision. Applications of this algorithm to problems in vision and robotics are discussed.

1 Introduction

Shape, position, orientation, and velocity are all geometric properties, and reasoning about these properties is an important part of vision and robotics. As much as possible, we wish to automate geometric reasoning by means of geometric programs. In order that these programs be efficient, we use rounded finite precision arithmetic. Unfortunately, when we write these programs we find that it is difficult to attain reasonable reliability and almost impossible to obtain absolute reliability: there always seems to be one more special case on which the program fails. Theoretical reasons are only now coming to light as to why geometric programs are so much more difficult to make reliable than purely numerical or purely symbolic programs, but this difficulty is very commonly experienced in practice.

We propose techniques for creating *robust* geometric programs: geometric programs with absolutely reliable rounded arithmetic implementations. This paper will focus on the domain of line segments in the plane, but the techniques presented here have broader applications. The techniques are based on two principles. First, one should use a malleable representation. Specifically, replace each line segment with a *rubber band curve* which can be modified as the computation proceeds to reconcile numerical error with the symbolic structure. Second, one should keep as much of the representation implicit as is possible. In the case of rubber band curves, their exact shapes are unknown at all stages of the computation. Put together, these two principles make up the *hidden variable method* which allows us to generate correct geometric information without the use of exact arithmetic. A certain amount of error is introduced by the use of rubber band curves, but this can be made to be a small fraction of the error arising from sensor noise and other measurement errors.

*This research was supported in part by DARPA, monitored by the Air Force Avionics Lab under contract F33615-87-C-1499. Victor J. Milenkovic was supported in part by an IBM Manufacturing Research Fellowship. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies. Author's current address: Aiken Computation Lab, Harvard University, Cambridge MA 02138.

1.1 Outline

Section 2 reviews the pitfalls of using rounded arithmetic to do geometry. The cost of using exact arithmetic instead is also considered. Section 3 formalizes the concept of single-precision arithmetic and explains why we expect robust single-precision geometric algorithms to exist. Section 4 gives a robust single-precision algorithm for reasoning about line segments in the plane. Given a set of line segments, we can determine how they intersect and what polygons they form using rounded arithmetic. This algorithm avoids the pitfalls discussed in Section 2, and it can be used for a variety of applications, such as modeling polygonal regions in the plane.

1.2 Background

In order to qualify as *robust* under the definitions in the paper, a geometric algorithm must be implementable using rounded $(N+c)$-bit arithmetic where N is the number of bits of accuracy in the output and c is constant or perhaps logarithmic in N. The implementation is not allowed to simulate high precision arithmetic, and it generates a solution for every well-formed input. The author has also created line arrangement and plane arrangement algorithms that satisfy this definition [10].

For a discussion of the more general problem of improving the reliability geometric computations using rounded arithmetic, see [4, 5]. Problems addressed include polygon [13, 9] and polyhedron [6, 7, 14] modeling and the calculation of convex hulls [12] and other problems in plane geometry [11]. Karasick [7] has the most practical result to date, an ultra-reliable polyhedral modeling system that is provably *safe*: either it generates a topologically consistent result or it indicates failure. Greene and Yao [3] have addressed the problem of rounding the output of an exact algorithms, and rounding can also be accomplished using the techniques in [10] and [11].

2 Pitfalls of Using Rounded Arithmetic

The problems that arise as a result of using rounded arithmetic to implement geometric algorithms fall into four areas: incomplete representation, ill-conditioned problems, incomplete information, and unexpected incidences. Problems arising from these areas cause either unreliable behavior and/or unbounded error. This section will consider each one in turn.

2.1 Incomplete Representation

Suppose we have a pair of line segments **AB** and **CD** whose endpoints are represented using standard single precision floating point numbers. In general, the intersection **I** of **AB** and **CD** will not be representable in single precision. It will in fact be a fraction, and if we carry out the division, the result is a repeating decimal.

One way out of this difficulty is to round **I** to the nearest representable point. Thus we replace **AB** by **AI** and **IB** and similarly for **CD**. This replacement can have two effects. First, it adds error which is ultimately proportional to the number of segments that **AB** intersects. Having error grow linearly with the problem size is usually not practical.

Rounding **I** may also introduce new intersections. In particular, **AI** and **IB** may intersect a segment **EF** which was not intersected by **AB**. Actually, the algorithm of Section 4 has this property. The point is that we must be careful to detect the consequences of the changes we make.

2.2 Ill-Conditioned Problems

If **AB** and **CD** intersect at a very small angle, it requires at least triple precision arithmetic (three times input precision) in order to calculate **I** to the same accuracy as **A**, **B**, **C** and **D**. As Section 3 will show, we should not need so much precision to generate a reasonable solution to the problem. However, from a naive point of view, a solution in which **I** is calculated with less accuracy than **A** is incorrect.

Small angle intersections are common in practice because **AB** and **CD** may be equal or overlapping segments which have been perturbed by measurement error or rounding.

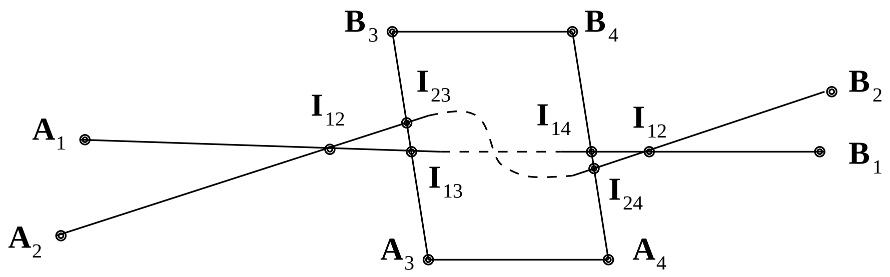

Figure 1: Global Contradiction

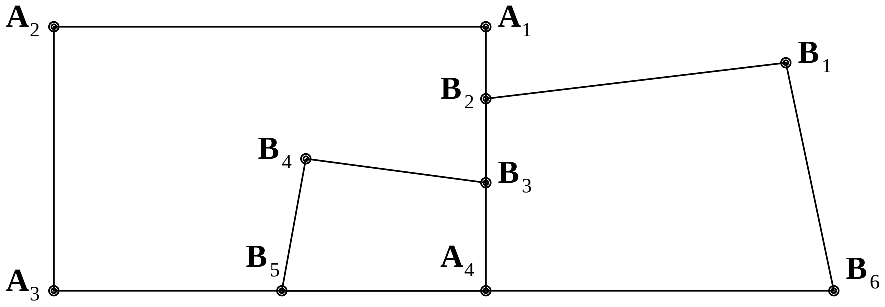

Figure 2: Unexpected Incidence of $\mathbf{A}_3\mathbf{A}_4$ and $\mathbf{B}_5\mathbf{B}_6$

2.3 Incomplete Information

Three segments $\mathbf{A}_1\mathbf{B}_1$, $\mathbf{A}_2\mathbf{B}_2$, and $\mathbf{A}_3\mathbf{B}_3$ can intersect in essentially three different ways: $\mathbf{I}_{12}$ (the intersection of $\mathbf{A}_1\mathbf{B}_1$ and $\mathbf{A}_2\mathbf{B}_2$) lies left of, right of, or on $\mathbf{A}_3\mathbf{B}_3$. In order to correctly determine which case holds, we must use more than six times the input precision for our calculations. If we do not use this much precision, we are left with situations in which we cannot know the correct answer, and thus one consequence of rounding is incomplete information.

Let us see what can happen when we make an arbitrary choice in the case that we do not know the correct answer. Figure 1 shows what happens when two *local* choices, $\mathbf{I}_{12}$ lies to the left of $\mathbf{A}_3\mathbf{B}_3$ and to the right of $\mathbf{A}_4\mathbf{B}_4$, lead to a global contradiction. If we attempt to trace out the boundary of the quadrilateral $\mathbf{A}_3\mathbf{A}_4\mathbf{I}_{24}\mathbf{I}_{23}$, we end up with the following sequence: $\mathbf{A}_3$, $\mathbf{A}_4$, $\mathbf{I}_{24}$, $\mathbf{I}_{23}$, $\mathbf{I}_{13}$, $\mathbf{I}_{14}$ (!), $\mathbf{B}_4$, $\mathbf{B}_3$, $\mathbf{I}_{23}$, $\mathbf{I}_{24}$, $\mathbf{I}_{14}$, $\mathbf{I}_{13}$, and (finally) back to $\mathbf{A}_3$. This is clearly an erroneous solution resulting from that fact that the local choices we have made have caused an unacknowledged intersection between segments $\mathbf{I}_{23}\mathbf{I}_{24}$ and $\mathbf{I}_{13}\mathbf{I}_{14}$. If we are to avoid this error, we must either avoid causing extra intersections or we must detect them.

2.4 Unexpected Incidences

In Figure 2 we see two polygons $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4$ and $\mathbf{B}_1\mathbf{B}_2\mathbf{B}_3\mathbf{B}_4\mathbf{B}_5\mathbf{B}_6$. Suppose we attempt to take the union using a *covered edge* rule: remove any edge covered by the other polygon. Edge $\mathbf{A}_4\mathbf{B}_5$ is duplicated, and if we naively apply the rule, either we remove both copies or we remove neither. Neither solution is satisfactory.

This problem arises because the rule does not cover a special case: overlapping edges. This failure, which we refer to as an *unexpected incidence*, can occur whether we are using exact or rounded arithmetic. For the domain of exact arithmetic, there is a general technique called *simulation of simplicity* [2, 15] for removing these special cases. In the domain of rounded arithmetic, we have no choice but to use careful programming.

3 Accuracy and Precision

In this section we will discuss the issues of precision and accuracy. We define single-precision arithmetic and show that it is sufficient to generate accurate solutions to numerical problems. We will also see why it is hard to reliably compute single-precision solutions to geometric problems.

3.1 Definitions

Definition 1 (Accuracy) *Suppose $a \in \Re$ is an approximate input to a numerical computation. If $a_{correct}$ is the correct value of a, we say that a has N bits of* accuracy *if,*

$$|a - a_{correct}| \leq 2^{-N} a.$$

A number $\xi \in \Re$ is small *with respect to a if $|\xi| \leq 2^{-N} a$.*

Definition 2 (P-bit Arithmetic) *An arithmetic has P bits of* precision *if any single operation has P bits of accuracy. For example,*

$$|(a + b)_{\mathrm{R}} - (a + b)| \leq 2^{-P} |a + b|,$$

where the subscript R denotes the rounded operation.

For a standard floating point arithmetic, P is the number of bits in the mantissa. We will say that a computation uses *single-precision* arithmetic if it solves problems with N bits of accuracy using P-bit arithmetic, where $P \approx N$.

3.2 Non-Geometric Numerical Computation

As an example of a non-geometric numerical computation, consider the problem of solving a system of D linear equations in D variables. Using Gaussian elimination and single-precision arithmetic, we can safely and efficiently generate a reasonable solution.

What do we mean by "safe"? The order in which Gaussian elimination eliminates variables depends on what appears to be the largest coefficient in a column. Round-off error may increase the value of a coefficient and thus alter the order in which variables are eliminated. In other words, Gaussian elimination acts on the basis of incomplete information. Fortunately, no matter how the order comes out, the algorithm generates a reasonable result.

What is efficient? If the coefficients have N bits of accuracy, arithmetic with $P = N + 2\log D$ bits suffices. Since $\log D$ is small, this is basically "single" precision, $P \approx N$. In order to solve the system exactly, we would require at least $P = DN$ bits, D-tuple precision. Furthermore, the rounded solution uses no more operations than the exact solution, and obviously, single-precision operations are much cheaper than D-tuple precision operations. We cannot hope to generate a solution with N bits of accuracy using arithmetic with $P < N$, and therefore the rounded solution is as efficient as possible.

What is reasonable? In the case of an ill-conditioned problem, the single-precision implementation of Gaussian elimination may generate a solution point that is very far from the correct solution. However, one can prove that if the approximate single-precision solution is $\bar{\mathbf{x}} = \langle c_1, c_2, \ldots, c_D \rangle$, then there exists some small (Definition 1) perturbation of the coefficients such that $\bar{\mathbf{x}}$ is the exact infinite-precision solution to the perturbed system. Thus, given the accuracy to which we know the input coefficients, $\bar{\mathbf{x}}$ may in fact be the correct solution.

3.3 Geometric Problems

Our goal is to safely and efficiently generate reasonable solutions to geometric problems using single-precision arithmetic. Arbitrary decisions in the face of incomplete information do not seem to have a significant effect on Gaussian elimination. Unfortunately, arbitrary decisions do have a strong effect on geometric algorithms, as we have seen in Section 2. Because of topological constraints, the decisions are highly interdependent.

Because geometric reasoning is so sensitive to arbitrary decisions, we must be careful how we define a reasonable solution. In the case of line segments, one might think that we should allow each endpoint to vary within a small disk, but this still appears to be too constrained. Recent theoretical work in computational geometry [1] has shown that determining whether a particular topology can be generated by a set of segments is a very hard problem.

Instead, we will fix the endpoints and allow the interior of the segment to vary into the shape of an arbitrary monotonic curve. As we will see in the next section, we can still constrain the shape of the approximation to be roughly linear. This type of solution is reasonable because we can constrain the points of the curve to approximate the points of the segment to N bits of accuracy. The solution is physically meaningful because straight lines do not occur in the natural world. A sufficiently straight curve is as useful a model as a straight line segment for most purposes.

4 Robust Reasoning about Line Segments

This section describes a technique for reasoning about line segments using an implicit approximation called the *rubber band curve*. After a few basic definitions, we will describe a straightforward means for storing topological information called a SIDE database. By imposing certain consistency conditions on the arbitrary decisions we store in the database we can assure that there exist a set of rubber band curves consistent with the decisions we have made. Next, we see how to detect implicit intersections among these curves by examining the values in the database. This section ends with an algorithm that creates new vertices and moves the intersection points of the rubber band curves to these explicit vertex locations.

4.1 Definitions

Definition 3 *A* coordinate rectangle *is a rectangle whose sides are aligned with the coordinate axes. Let* $\mathbf{AB}$ *be a line segment in the plane (with floating point endpoints). Let* $\mathrm{R}(\mathbf{AB})$ *be the coordinate rectangle which has* $\mathbf{AB}$ *as one of its diagonals. In interval notation,*

$$\mathrm{R}(\mathbf{AB}) = [\mathbf{A}_x, \mathbf{B}_x] \times [\mathbf{A}_y, \mathbf{B}_y].$$

If a vertex $\mathbf{C}$ lies outside $\mathrm{R}(\mathbf{AB})$, then we can easily classify $\mathbf{C}$ with respect to $\mathbf{AB}$. For example if $\mathbf{C}_x \in [\mathbf{A}_x, \mathbf{B}_x]$ and $\mathbf{C}_y > \max(\mathbf{A}_y, \mathbf{B}_y)$, then $\mathbf{C}$ is clearly above $\mathbf{AB}$.

If $\mathbf{C} \in \mathrm{R}(\mathbf{AB})$, we must look at the signed distance $\delta(\mathbf{C}, \mathbf{AB})$ from $\mathbf{C}$ to $\mathbf{AB}$,

$$\delta(\mathbf{C}, \mathbf{AB}) = \frac{(\mathbf{B} - \mathbf{A}) \times (\mathbf{C} - \mathbf{A})}{|\mathbf{B} - \mathbf{A}|},$$

where,

$$(\mathbf{B} - \mathbf{A}) \times (\mathbf{C} - \mathbf{A}) = (\mathbf{B}_x - \mathbf{A}_x)(\mathbf{C}_y - \mathbf{A}_y) - (\mathbf{B}_y - \mathbf{A}_y)(\mathbf{C}_x - \mathbf{A}_x).$$

Suppose we use arithmetic with P bits of precision. How accurate is $\delta(\mathbf{C}, \mathbf{AB})_{\mathrm{R}}$? (The subscript R refers to the use of rounded arithmetic.) It can be shown that for $\mathbf{C}$ close to $\mathbf{AB}$,

$$|\delta(\mathbf{C}, \mathbf{AB})_{\mathrm{R}} - \delta(\mathbf{C}, \mathbf{AB})| \leq 3\epsilon|\mathbf{C} - \mathbf{A}|,$$

where $\epsilon = 2^{-P}$. We will assume that all reasoning occurs inside some bounding square $[-\mathrm{M}, \mathrm{M}] \times [-\mathrm{M}, \mathrm{M}]$ where M is the maximum magnitude of any coordinate. Thus $|\mathbf{C} - \mathbf{A}| \leq 2\sqrt{2}\mathrm{M}$, and the error in calculating $\delta(\mathbf{C}, \mathbf{AB})$ is bounded by $6\sqrt{2}\epsilon\mathrm{M}$. We will refer to this constant as α.

4.2 Handling Incomplete Information

The fact that $\alpha \neq 0$ indicates that for $|\delta(\mathbf{C}, \mathbf{AB})_{\mathrm{R}}| \leq \alpha$, we do not know whether $\mathbf{C}$ lies above, below or on $\mathbf{AB}$. We make arbitrary decisions and store them in a SIDE database: for each segment $\mathbf{AB}$ and each vertex $\mathbf{C} \in \mathrm{R}(\mathbf{AB})$, $\mathrm{SIDE}(\mathbf{C}, \mathbf{AB})$ equals ABOVE, BELOW or ON. Since $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ are vertices we are given or ones we create, the database stores only a finite amount of information. The decisions we make may be incorrect, but by imposing some consistency conditions, we can still generate useful results.

Definition 4 (Numerical Consistency) *The SIDE database is* numerically consistent *if*

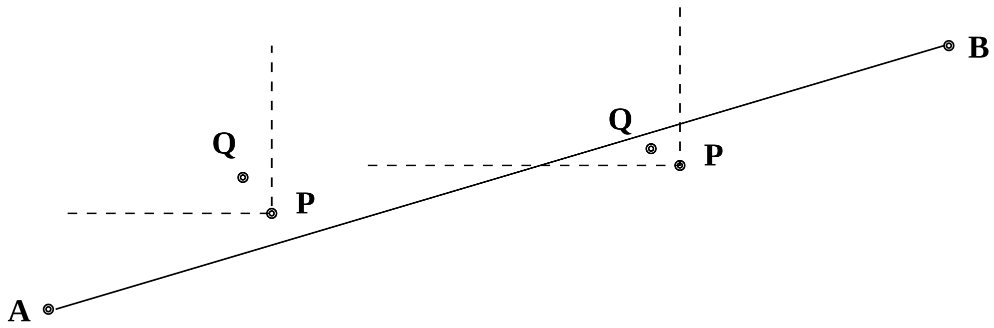

Figure 3: Examples of XY-Consistency

$$\mathrm{SIDE}(\mathbf{C},\mathbf{AB}) = \left\{ \begin{array}{c} \mathrm{ABOVE} \\ \mathrm{BELOW} \\ \mathrm{ON} \end{array} \right\} \textit{ implies that } \left\{ \begin{array}{c} \delta(\mathbf{C},\mathbf{AB}) \geq -5\alpha \\ \delta(\mathbf{C},\mathbf{AB}) \leq 5\alpha \\ |\delta(\mathbf{C},\mathbf{AB})| \leq 5\alpha \end{array} \right\}.$$

This definition states that the decisions may be incorrect, but not very incorrect. The reason we use a multiple 5α will become apparent later. The value 5α represents about six bits of error.

As we know from Section 2, imposing arbitrary decisions in the face of incomplete information is a risky business. Therefore we enforce another consistency condition on SIDE.

Definition 5 (XY-Consistency: Prototype Definition) *Suppose we have $\mathbf{A}_x < \mathbf{B}_x$ and $\mathbf{A}_y < \mathbf{B}_y$ and $\mathbf{P},\mathbf{Q} \in \mathrm{R(AB)}$. If $\mathrm{SIDE}(\mathbf{P},\mathbf{AB}) = \mathrm{ABOVE}$ and if $\mathbf{Q}_x \leq \mathbf{P}_x$ and $\mathbf{Q}_y \geq \mathbf{P}_y$, then $\mathrm{SIDE}(\mathbf{Q},\mathbf{AB}) = \mathrm{ABOVE}$ (see Figure 3).*

Imagine putting the origin at $\mathbf{P}$. The fact that $\mathbf{P}$ is ABOVE $\mathbf{AB}$ forces everything in the second quadrant to be ABOVE also. Even if we are wrong about $\mathrm{SIDE}(\mathbf{P},\mathbf{AB})$, we will still insist on xy-consistency. The following is the complete definition.

Definition 6 (XY-Consistency: Complete Definition) *Suppose we have $\mathbf{A}_x < \mathbf{B}_x$ and $\mathbf{A}_y < \mathbf{B}_y$ and $\mathbf{P},\mathbf{Q} \in \mathrm{R}(\mathbf{AB})$.*

$$\textit{If } \mathrm{SIDE}(\mathbf{P},\mathbf{AB}) = \left\{ \begin{array}{c} \mathrm{ON} \textit{ or } \mathrm{ABOVE} \\ \mathrm{ON} \textit{ or } \mathrm{BELOW} \end{array} \right\} \textit{ and } \left\{ \begin{array}{c} \mathbf{Q}_x \leq \mathbf{P}_x \\ \mathbf{Q}_x \geq \mathbf{P}_x \end{array} \right\} \textit{ and } \left\{ \begin{array}{c} \mathbf{Q}_y \geq \mathbf{P}_y \\ \mathbf{Q}_y \leq \mathbf{P}_y \end{array} \right\}$$

$$\textit{then } \mathrm{SIDE}(\mathbf{Q},\mathbf{AB}) = \left\{ \begin{array}{c} \mathrm{ON} \textit{ or } \mathrm{ABOVE} \\ \mathrm{ON} \textit{ or } \mathrm{BELOW} \end{array} \right\}.$$

If either of these two conditions holds, $\mathrm{SIDE}(\mathbf{Q},\mathbf{AB})$ can equal ON only in the case that

$$\mathrm{SIDE}(\mathbf{P},\mathbf{AB}) = \mathrm{ON} \qquad \textit{and} \qquad (\mathbf{Q}_x = \mathbf{P}_x \textit{ or } \mathbf{Q}_y = \mathbf{P}_y).$$

Finally, if $\mathbf{A}_y > \mathbf{B}_y$ the definition is the same except that comparisons of y-coordinates are reversed.

4.3 Rubber Band Curves

The following theorem indicates that there is some reasonable curve satisfying the set of decisions we have made.

Theorem 1 (Hidden Variable Theorem) *Let $\gamma : [0,1] \to \Re^2$ be the shortest rectifiable (actually polygonal) curve satisfying $\gamma(0) = \mathbf{A}$, $\gamma(1) = \mathbf{B}$, and for all vertices $\mathbf{P} \in \mathrm{R}(\mathbf{AB})$,*

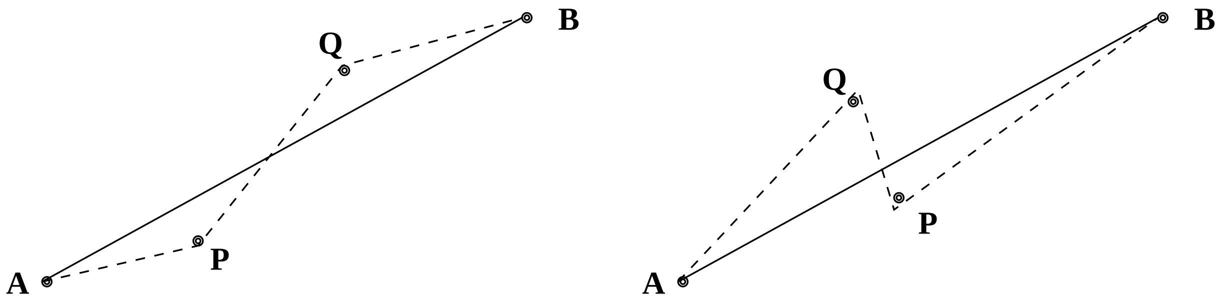

Figure 4: Rubber Band Curves

$$\mathrm{SIDE}(\mathbf{P}, \mathbf{AB}) = \left\{ \begin{array}{c} \mathrm{ABOVE} \\ \mathrm{BELOW} \\ \mathrm{ON} \end{array} \right\} \textit{ implies that } \mathbf{P} \textit{ is } \left\{ \begin{array}{c} \textit{above or on} \\ \textit{below or on} \\ \textit{on} \end{array} \right\} \gamma.$$

Then γ *is monotonic and close to* $\mathbf{AB}$:

$$\forall t \in [0,1]: \quad \gamma'(t) \cdot (\mathbf{B} - \mathbf{A}) > 0 \quad \textit{and} \quad |\delta(\gamma(t), \mathbf{AB})| \leq 5\alpha.$$

The tangent $\gamma'(t)$ *may be undefined at a finite number of points.*

For a proof, see [10]. We call the curve γ the *rubber band curve* rbc($\mathbf{AB}$). Figure 4 gives two examples in which $\mathbf{P}$ is ABOVE $\mathbf{AB}$ and $\mathbf{Q}$ is BELOW. The instance on the right shows what happens when xy-monotonicity is violated; the rubber band curve (dashed) is not monotonic.

4.4 Detecting Intersections

Rubber band curves are implicit. In the examples in Figure 4 we cannot know if $\mathbf{P}$ and $\mathbf{Q}$ actually deflect rbc($\mathbf{AB}$) unless we know the signs of $\delta(\mathbf{P}, \mathbf{AB})$ and $\delta(\mathbf{Q}, \mathbf{AB})$, which we do not in the case of incomplete information. Despite this implicitness, we can always tell whether or not two rubber band curves intersect simply by looking at the SIDE database. This section shows how to accomplish this.

Definition 7 (Subsegment) $\mathbf{P}_i\mathbf{P}_{i+1}$ *is a (minimal)* subsegment *of segment* $\mathbf{AB}$ *if*

- $\mathrm{SIDE}(\mathbf{P}_i, \mathbf{AB}) = \mathrm{SIDE}(\mathbf{P}_{i+1}, \mathbf{AB}) = \mathrm{ON}$*;*
- $\mathbf{P}_i$ *and* $\mathbf{P}_{i+1}$ *are* neighbors: *for every other vertex* $\mathbf{P} \in \mathrm{R}(\mathbf{P}_i\mathbf{P}_{i+1})$, $\mathrm{SIDE}(\mathbf{P}, \mathbf{AB}) \neq \mathrm{ON}$.

Define also $\mathrm{rbc}_{\mathbf{AB}}(\mathbf{P}_i\mathbf{P}_{i+1})$ *to be the portion of* rbc($\mathbf{AB}$) *joining* $\mathbf{P}_i$ *to* $\mathbf{P}_{i+1}$.

If a subsegment of $\mathbf{AB}$ crosses a subsegment of $\mathbf{CD}$ we expect to find evidence of the fact in the SIDE database.

Definition 8 (Evidence Vertices) *Let* $\mathbf{P}_i\mathbf{P}_{i+1}$ *be a* subsegment *of segment* $\mathbf{AB}$ *and let* $\mathbf{Q}_j\mathbf{Q}_{j+1}$ *be a* subsegment *of segment* $\mathbf{CD}$ *such that* $\mathrm{R}(\mathbf{P}_i\mathbf{P}_{i+1}) \cap \mathrm{R}(\mathbf{Q}_j\mathbf{Q}_{j+1})$ *is non-empty. Let* $\mathbf{E}$ *be a vertex such that* $\mathbf{E}_x \in [\mathbf{P}_{i,x}, \mathbf{P}_{i+1,x}] \cap [\mathbf{Q}_{j,x}, \mathbf{Q}_{j+1,x}]$.

Vertex $\mathbf{E}$ *is an* evidence vertex *if* $\mathrm{SIDE}(\mathbf{E}, \mathbf{AB}) \neq \mathrm{SIDE}(\mathbf{E}, \mathbf{CD})$. *In particular* $\mathbf{E}$ *is* evidence *that* $\mathrm{rbc}_{\mathbf{AB}}(\mathbf{P}_i\mathbf{P}_{i+1})$ *is partially above* $\mathrm{rbc}_{\mathbf{CD}}(\mathbf{Q}_j\mathbf{Q}_{j+1})$ *if*

$$\mathrm{SIDE}(\mathbf{E}, \mathbf{AB}) = \mathrm{BELOW} \qquad \textit{or} \qquad \mathrm{SIDE}(\mathbf{E}, \mathbf{CD}) = \mathrm{ABOVE}.$$

Otherwise it is evidence *that* $\mathrm{rbc}_{\mathbf{AB}}(\mathbf{P}_i\mathbf{P}_{i+1})$ *is partially below* $\mathrm{rbc}_{\mathbf{CD}}(\mathbf{Q}_j\mathbf{Q}_{j+1})$.

Lemma 1 *Let* $\mathbf{P}_i\mathbf{P}_{i+1}$ *and* $\mathbf{Q}_j\mathbf{Q}_{j+1}$ *be subsegments of* $\mathbf{AB}$ *and* $\mathbf{CD}$*, respectively. Curve* $\mathrm{rbc}_{\mathbf{AB}}(\mathbf{P}_i\mathbf{P}_{i+1})$ *crosses curve* $\mathrm{rbc}_{\mathbf{CD}}(\mathbf{Q}_j\mathbf{Q}_{j+1})$ *if and only if there is an evidence vertex* $\mathbf{E}_{\mathrm{a}}$ *that indicates* $\mathrm{rbc}_{\mathbf{AB}}(\mathbf{P}_i\mathbf{P}_{i+1})$ *is partially above* $\mathrm{rbc}_{\mathbf{CD}}(\mathbf{Q}_j\mathbf{Q}_{j+1})$ *and there is an evidence vertex* $\mathbf{E}_{\mathrm{b}}$ *that indicates* $\mathrm{rbc}_{\mathbf{AB}}(\mathbf{P}_i\mathbf{P}_{i+1})$ *is below* $\mathrm{rbc}_{\mathbf{CD}}(\mathbf{Q}_j\mathbf{Q}_{j+1})$ *(see Figure 5).*

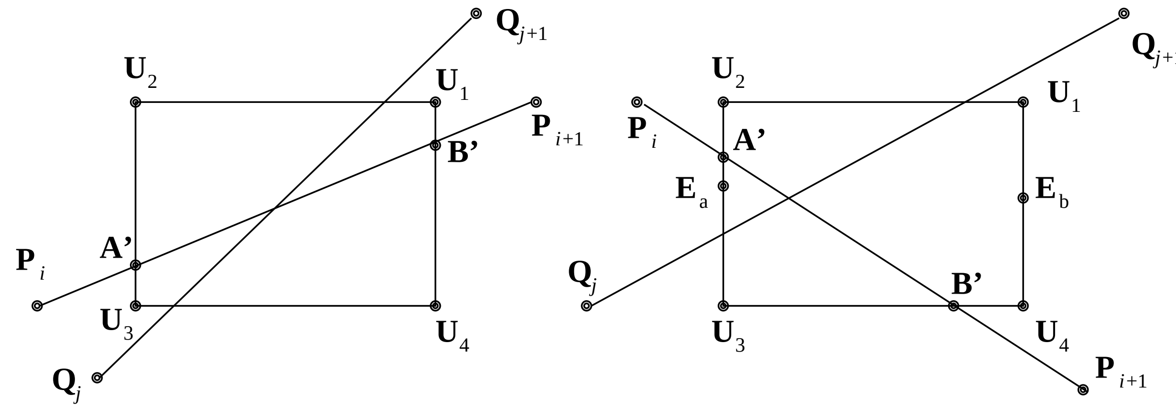

Figure 5: Evidence for Intersection of Subsegments $\mathbf{P}_i\mathbf{P}_{i+1}$ and $\mathbf{Q}_j\mathbf{Q}_{j+1}$

Thus if two subsegments cross, we can detect the fact looking only at the values in the SIDE database. In the left half of Figure 5 $\mathbf{E}_\mathrm{a}$ equals $\mathbf{U}_3$ and $\mathbf{E}_\mathrm{b}$ equals $\mathbf{U}_1$. This always happens if the slopes of **AB** and **CD** have the same sign. For a proof of the lemma, see [10].

4.5 Creating Intersections

When two subsegments cross we must create an explicit vertex for this crossing and then add it to the SIDE database. Eventually, we will find all intersections, and as the lemma of the previous section indicates, by scrutinizing the SIDE database, we can be sure that we have not missed any intersections. When we have finished, the SIDE database will contain all the topological information we need to know about the rubber band curves. This section describes the intersection algorithm.

Lemma 1 implies the existence of evidence vertices $\mathbf{E}_\mathrm{a}$ and $\mathbf{E}_\mathrm{b}$ whenever two subsegments $\mathbf{P}_i\mathbf{P}_{i+1}$ and $\mathbf{Q}_j\mathbf{Q}_{j+1}$ intersect. The specific task of the intersection algorithm is to create a new vertex **I** such that $\mathbf{E}_{a,x} \leq \mathbf{I}_x \leq \mathbf{E}_{b,x}$ and such that it is numerically consistent and xy-consistent (Definitions 4 and 6) to set $\mathrm{SIDE}(\mathbf{I}, \mathbf{AB})$ and $\mathrm{SIDE}(\mathbf{I}, \mathbf{CD})$ equal to ON. In order to be xy-consistent, it is necessary that,

$$\mathbf{I} \in \mathbf{R} = ([\mathbf{E}_{\mathrm{a},x}, \mathbf{E}_{\mathrm{b},x}] \times [-\infty, \infty]) \cap \mathrm{R}(\mathbf{P}_i\mathbf{P}_{i+1}) \cap \mathrm{R}(\mathbf{Q}_j\mathbf{Q}_{j+1}).$$

We call **R** the *bounding rectangle for the intersection* and denote its vertices by $\mathbf{U}_1$, $\mathbf{U}_2$, $\mathbf{U}_3$, and $\mathbf{U}_4$.

The intersection algorithm is based on the following idea: if there are no vertices in the interior of **R**, then it is possible to explicitly determine the shapes of the rubber band curves $\mathrm{rbc}(\mathbf{AB})$ and $\mathrm{rbc}(\mathbf{CD})$ inside **R**. Unfortunately, there may be vertices inside **R**. To get around this problem, we will create a temporary database which ignores these points.

We create a constant-sized temporary database SIDE$'$,

$$\mathrm{SIDE}' : \{\mathbf{E}_\mathrm{a}, \mathbf{E}_\mathrm{b}, \mathbf{P}_i, \mathbf{P}_{i+1}, \mathbf{Q}_j, \mathbf{Q}_{j+1}, \mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3, \mathbf{U}_4\} \times \{\mathbf{AB}, \mathbf{CD}\} \to \{\mathrm{ABOVE}, \mathrm{BELOW}, \mathrm{ON}\}.$$

Initially, SIDE$'$ agrees with SIDE on $\{\mathbf{E}_\mathrm{a}, \mathbf{E}_\mathrm{b}, \mathbf{P}_i, \mathbf{P}_{i+1}, \mathbf{Q}_j, \mathbf{Q}_{j+1}\} \times \{\mathbf{AB}, \mathbf{CD}\}$. Otherwise, we choose values that are xy-consistent and numerically consistent. Next, we change as many values to ON as possible: for any vertex **P** in the domain of SIDE$'$,

$$\text{if } \mathrm{SIDE}'(\mathbf{P}, \mathbf{AB}) = \left\{ \begin{array}{c} \mathrm{ABOVE} \\ \mathrm{BELOW} \end{array} \right\} \text{ and } \delta(\mathbf{P}, \mathbf{AB})_\mathrm{R} \left\{ \begin{array}{c} \leq 4\alpha \\ \geq -4\alpha \end{array} \right\} \text{ then set } \mathrm{SIDE}'(\mathbf{P}, \mathbf{AB}) = \mathrm{ON}.$$

Do the same thing for **CD**.

It may happen that there are **P** and **Q** in the domain of SIDE$'$ such that **PQ** is a subset of the boundary of **R** and such that

$$\mathrm{SIDE}'(\mathbf{P}, \mathbf{AB}) = \mathrm{BELOW} \qquad \text{and} \qquad \mathrm{SIDE}'(\mathbf{Q}, \mathbf{AB}) = \mathrm{ABOVE}.$$

In this case we simply compute the intersection of this segment with $\mathbf{AB}$ using rounded arithmetic and insert the new vertex into the SIDE$'$ database.

If after we do all this, there is some point $\mathbf{I} \in \mathbf{R}$ in the domain SIDE$'$ such that

$$\mathrm{SIDE}'(\mathbf{I}, \mathbf{AB}) = \mathrm{SIDE}'(\mathbf{I}, \mathbf{CD}) = \mathrm{ON},$$

then $\mathbf{I}$ is the desired intersection. Otherwise, there will be a subsegment $\mathbf{A}'\mathbf{B}'$ of $\mathbf{AB}$ which intersects the interior of $\mathbf{R}$. In this case, the desired vertex $\mathbf{I}$ is the intersection of $\mathbf{A}'\mathbf{B}'$ (see Figure 5) with $\mathbf{CD}$ as calculated using rounded arithmetic. This vertex will have accuracy 5α (Definition 4).

4.6 Updating SIDE

The previous section described how to generate a vertex $\mathbf{I}$ such that

$$\mathrm{SIDE}'(\mathbf{I}, \mathbf{AB}) = \mathrm{SIDE}'(\mathbf{I}, \mathbf{CD}) = \mathrm{ON}.$$

Unfortunately, the SIDE$'$ database was constructed ignoring any vertices lying inside $\mathbf{R}$. What do we do if there is a vertex $\mathbf{P}$ such that $\mathrm{SIDE}(\mathbf{P}, \mathbf{AB}) = \mathrm{ON}$ and $\mathbf{I}_x \leq \mathbf{P}_x$ and $\mathbf{I}_y \geq \mathbf{P}_y$ (see Definition 6)? In this case we cannot set $\mathrm{SIDE}(\mathbf{I}, \mathbf{AB})$ equal to ON. What we do is that we throw away $\mathbf{I}$ and set $\mathrm{SIDE}(\mathbf{P}, \mathbf{AB})$ equal to ON.

If we are forced to throw away $\mathbf{I}$, we do not add to the number of vertices. We also convert an ABOVE to an ON or a BELOW to an ON, which can happen only a finite number of times. Thus if we keep trying to generate intersections, eventually we will succeed in generating an $\mathbf{I}$ that works.

5 Conclusion

In conclusion, we would like to answer some questions about the algorithm described above. What sort of applications does it have? Can its complexity be decreased? Are there other approaches?

5.1 Applications

The main application of the segment algorithm is polygon modeling. We define a polygonal region in the plane by listing the set of line segments that form its boundary. If wish to take a union, intersection, or difference of two polygonal regions, the first step is to apply the segment algorithm to the sets of segments. Given the completed SIDE database, generating the desired regions is purely a symbolic, not numerical, process.

There are many applications of polygon modeling in vision and robotics. For example, given a polyhedral scene, we can generate its projection into the image plane using polygon modeling. If we have extracted a region from an image, we can compare it to the projection of a hypothesized model by applying the symmetric set difference. In the field of robotics, we can represent the map used by a mobile robot as a union of polygonal regions. Determining if the robot can fit in a certain location is a matter of taking a polygon intersection. Many other applications surely exist.

5.2 Complexity

The segment algorithm is somewhat complicated. However, if a developer attempts to naively use floating point to solve a geometric problem, he dooms himself to a large period of testing and patching. His final product will be more complicated and less reliable. Correctness more than makes up for the extra complexity.

Rubber band curves are not the only way to approximate segments, and the hidden variable method is not the only way to create robust programs. The real lesson is that there are techniques for creating rounded arithmetic geometry programs that are theoretically well founded. One can also use exact arithmetic even though it appears to be very expensive. Karasick, Lieber and Nackman [8] have recently demonstrated an exact Delaunay triangulation algorithm with empirical cost well below the worst case. In any case, a considerable amount of time and effort can be saved by the small investment of time it takes to investigate these possibilities.

5.3 Acknowledgments

The author thanks Takeo Kanade for his comments, suggestions and guidance on both style and content. Thanks also to Barbara A. Jones for her corrections and suggestions.

References

[1] Jürgen Bokowski, Jürgen Richter, and Bernd Sturmfels. Nonrealizability proofs in computational geometry. *Journal of Discrete and Computational Geometry*, 1988, (to appear).

[2] Herbert Edelsbrunner and Ernst Peter Mucke. *Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms.* Technical Report UIUCDCS-R-87-1393, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, December 1987.

[3] Daniel H. Greene and F. Frances Yao. Finite-resolution computational geometry. In *27th Annual Symposium on the Foundations of Computer Science*, pages 143–152, IEEE, October 1986.

[4] C. Hoffmann and J. Hopcroft. Towards implementing robust geometric computations. In *Proceedings of the Symposium on Computational Geometry*, ACM, 1988.

[5] Christoph M. Hoffmann. *The Problem of Accuracy and Robustness in Geometric Computation.* Technical Report CSD-TR-771, Computer Sciences Department, Purdue University, April 1988.

[6] Christoph M. Hoffmann, John E. Hopcroft, and Michael S. Karasick. *Robust Set Operations on Polyhedral Solids.* Technical Report 87-875, Department of Computer Science, Cornell University, Ithaca, New York 14853-7501, October 1987.

[7] Michael Karasick. *On the Representation and Manipulation of Rigid Solids.* PhD thesis, McGill University, August 1988.

[8] M. Karsick, D. Lieber, and L. Nackman. *Efficient Delaunay Triangulation using Rational Arithmetic.* IBM Research Report RC 14455, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, To appear in 1989.

[9] Victor Milenkovic. Verifiable implementations of geometric algorithms using finite precision arithmetic. *Artificial Intelligence*, 37:377–401, 1988.

[10] Victor J. Milenkovic. *Verifiable Implementations of Geometric Algorithms using Finite Precision Arithmetic.* Technical Report CMU-CS-88-168, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, July 1988.

[11] Thomas Ottmann, Gerald Thiemt, and Christian Ullrich. Numerical stability of geometric algorithms. In *Proceedings of the Symposium on Computational Geometry*, ACM, 1987.

[12] D. Salesin, J. Stolfi, and L. Guibas. Epsilon geometry: building robust algorithms from imprecise computations. In *Proceedings of the Symposium on Computational Geometry*, ACM, 1989.

[13] Mark Segal and Carlo H. Sequin. Consistent calculations for solids modeling. In *Proceedings of the Symposium on Computational Geometry*, pages 29–37, ACM, June 1985.

[14] Mark Segal and Carlo H. Sequin. Partitioning polyhedral objects into non-intersecting parts. *IEEE Computer Graphics and Applications*, 8(1), January 1988.

[15] C. Yap. A geometric consistency theorem for a symbolic perturbation theorem. In *Proceedings of the Symposium on Computational Geometry*, ACM, 1988.