Computer-Aided Design, vol. 25, no. 9, September, 1993, pp. 546–566. Best paper of 1993.

Robust Polygon Modeling

Victor Milenkovic *
Harvard University
Center for Research in Computing Technology
Cambridge, MA 02138

Abstract

The article provides a set of algorithms for performing set operations on polygonal regions in the plane using standard floating point arithmetic. The algorithms are *robust*, guaranteeing both topological consistency and numerical accuracy. Each polygon edge is modeled as an implicit or explicit polygonal curve which stays within some distance β of the original line segment. If the curve is implicit, β is bounded by a small multiple of the rounding unit. If the curves are explicit, the bound on β may grow with the number of curves. One can mix implicit and explicit representations to suit the application.

Keywords: algorithms, polygons, geometric modelling

1 Introduction

The practical difficulties of implementing geometric algorithms are well-known to experienced developers. Geometric algorithms rely on the field axioms of real arithmetic, and they behave unpredictably when implemented in a naive fashion using rounded floating point arithmetic. Developers spend much time carefully choosing epsilons, adding fuzzy comparisons, and so forth, as the need arises, to improve the reliability of geometric programs. These modifications are made in an *ad hoc* fashion, and there is no general procedure for reliably implementing geometric algorithms. In the future, as geometric programs grow in number and complexity, the time and expertise required to fix numerical problems will become unsupportable. For some tasks, no amount of engineering can improve the reliability to the levels now needed. We conclude that if we are to increase reliability of programs, increase the productivity of our developers, and decrease the cost of implementing geometric algorithms, we must develop a theory of *robust geometry* whose aim is the creation of *robust algorithms*, geometric algorithms which can be proven correct even when implemented using rounded arithmetic.

There are two reasons why the current theory of computational geometry does not ordinarily address the issue of robustness. First, current theory is based on an analysis of asymptotic cost, and for any specific geometric construction, the cost of using unlimited precision rational arithmetic (or whatever is necessary to implement the algorithm without rounding) is only a constant factor greater than using rounded floating point arithmetic. Second, it is assumed (presumably by those who have never implemented a geometric algorithm), that round-off error poses only minor, easily surmountable difficulties. Experience teaches us that this is a false assumption. As far as the constant factors are concerned, there are at least three reasons why their size matters a great deal and should be addressed by theory. First, if operations are cascaded (rotations and translations are alternated with other operations on geometric objects), the "constant" factor for exact arithmetic can grow exponentially with the number of operations. In contrast, the extra cost of cascading rounded operations is only logarithmic. Second, constructions involving curved surfaces, which

*This research was funded by NSF grants CCR-90-09272 and CCR-91-157993.

are often required in industry, entail very large constant factors. Sugihara [22] estimates a factor of 80 for the intersection of quadratic surfaces. In order to intersect bi-cubic surface patches, Farouki [4] has determined that we would need to solve equations of degree 1000 or higher. The difference in cost between an approximate and an an exact solution to such equation is at least a factor of 1000.[1] Finally, the third reason is that a theoretically correct algorithm must compete with "near misses". Given a program with a certain residual (although unacceptable) amount of unreliability, a developer would be extremely reluctant to eliminate the unreliability by slowing down the program by a large factor. He would probably prefer to whittle away at the numerical problems by engineering, even though that wastes his expertise and time.

This author has devised a *strict robustness*[2] approach to creation of robust algorithms and has created *strictly robust* algorithms for a number of geometric constructions. These algorithms depend on the use of *implicit monotonic* curves called *mascs*,[3] which we believe to be the key to achieving robustness for a large variety of constructions. We describe here a strictly robust algorithm for performing set operations on polygonal regions in the plane. This algorithm will acquaint the reader with a number of definitions and techniques in the area of robust geometry. We hope that this exposition will help readers to understand other accounts of robust algorithms and to devise algorithms of their own.

Section 1.1 discusses the philosophy behind strict robustness. Section 2 discusses the types of numerical problems caused by a naive application of rounded arithmetic in geometric programs, and it describes current techniques for addressing these problems, including strict robustness. Sections 3 through 5 give strictly robust algorithms for performing set operations on polygonal regions bounded by masc segments. Section 3 gives low level primitives for robust operations on line segments. Section 4 defines mascs and gives an algorithm for finding all intersections among a set of masc segments. This algorithm is strictly robust through the use of implicit curves. If one desires, one can alter the algorithm to generate an *explicit* representation of the curves, but only at the cost of decreasing the overall accuracy. Section 5 gives an algorithm for performing set operations on poly-curves based on the robust intersection algorithm. Finally, Section 6 discusses how the same techniques might be applied to other domains.

1.1 Strict Robustness

We propose a classical approach to robust geometry. First, define a task to be performed. Second, devise an algorithm that performs the task. Third, seek the optimal algorithm for that task. For *strict robust geometry*, the task is the *accurate* construction of a *feasible* representation of some geometric object using *rounded arithmetic*. We do not accept any unreliability: an unreliable algorithm is not an algorithm.

It is an explicit assumption of the task that rounded B-bit floating point arithmetic is a primitive operation, and the algorithm must perform all its arithmetic computations using this arithmetic. It is not permitted to look into the internals of the floating point representation in order to simulate higher precision arithmetic. The algorithm is *accurate* if it introduces only a small constant number C of bits of error where C is independent of B and the input size n. The output is *feasible* if it is of the correct type. That is, if the task is to construct the intersection of a set of polygonal regions, then the algorithm constructs the intersection of *some* set of polygonal regions. Sections 2.4, 2.3, and 2.5 discuss accuracy and feasibility in more detail. Section 2.6 summarizes some of the current results in the area of robust geometry.

The output of a strictly robust algorithm is permitted to be *implicit*, where we use this term in the traditional sense. For example, the curve $\gamma(x) = \langle x, \sqrt{1-x^2}\rangle$ is an *explicit* representation for (part of) the unit circle because it tells us, for each coordinate x, what the corresponding coordinate y should be. The equation $x^2 + y^2 = 1$ is an *implicit* representation because it only allows us to test whether a given point $\langle x, y\rangle$ lies on the curve or not. The output of a strictly robust algorithm is implicit in the same sense, with two extra properties:

- using B-bit arithmetic, for each coordinate x, one can compute a coordinate y such that $\langle x, y\rangle$ lies very near to the implicit curve (error is a constant times 2^{-B});

[1] By an exact solution, we refer to the construction of rational intervals which contain isolated roots.
[2] Originally, the term was simply *robust* [13], but people now commonly use this to mean *very reliable*.
[3] Monotonic Adaptive Straight Curve Segments

- using sufficient precision, the coordinate y can be computed to any accuracy desired.

2 Sources of Error

This section describes the four types of numerical problems that arise when geometric algorithms are implemented naively: unexpected singularities, gross infeasibility, subtle infeasibility, and inaccuracy. These problems must be addressed in the design of robust algorithms. Section 2.1 discusses the numerical operations required to intersect line segments and construct polygons. The succeeding four sections illustrate the four numerical problems for this domain.

2.1 Polygons and Line Segments

Constructions of polygons and other operations involving line segments in the plane rely on a number of numerical tests: determine if a point lies inside or outside a polygon; determine whether two segments intersect; determine the order of two intersection points on the same segment (the *intersection-order* test); and so forth. In each case, the outcome of the test depends on the *sign* of an arithmetic expression. In other geometric domains also, the behavior of a geometric program is determined by the signs of expressions, and this partially accounts for the observed sensitivity to round-off error: the value of an expression may be stable but the sign of the value is not. If the expression is nearly zero, even a very small numerical error can change its sign.

The outcome of the third, *intersection-order*, test above depends on the sign of a fourth degree polynomial [16] in the vertex coordinates. If the coordinates are N-bit integers (in the range $[-2^N, 2^N]$), registers of size $4N+3$ bits suffice to compute the value of this polynomial exactly. For CAD/CAM applications, $N = 20$ (one part per million accuracy) and thus at least 83-bit arithmetic is needed. Since exact arithmetic of this precision usually is not available (in hardware), it is necessary to use rounded arithmetic to achieve practical efficiency.

We see, even for the simplest example, a four-fold inflation in precision is required in order to avoid rounding. Naturally, for more complex domains involving curves, planes in three dimensions, or curved surfaces, much more precisions is needed. If we do not sacrifice efficiency, we must allow rounding, and therefore we need robust geometry. The next three sections will describe the sort of problems that robust geometry must address.

2.2 Unexpected Singularities

A *singularity* occurs when the sign of an arithmetic expression is zero. The singularity manifests itself as an unexpected coincidence. For example, a zero value for the intersection-order test above implies that three segments are coincident at a single point. Other singular cases are overlapping line segments (segments which intersect in a segment instead of in a point), vertices with equal x or y coordinates, and parallel line segments. If there are no singularities, the set of segments is *simple* or in *general position*. A common source of unreliability in geometric programs is an *unexpected singularity*, a singular case for which the algorithm has no logic.

In general, published algorithms assume simplicity because there exist, in principle, a general techniques for transforming these algorithms into ones which do correctly treat singularities. This techniques, called *simulation of simplicity* [2] or *symbolic perturbation* [24], [3], remove these special cases by symbolically perturbing the input. These perturbation techniques cannot be applied in the case of rounded arithmetic. This is not a great loss, because this investigator has not found the technique to be particularly practical. The expansion of the fourth degree polynomial for the intersection-order test has 1152 terms for the perturbed input, and we must treat this polynomial differently depending on the order in which the vertices **A**, **B**, **C**, **D**, **E**, **F** were given to the algorithm, of which there are 720 possibilities. It is not clear how to overcome the combinatorial complexities of the technique. Furthermore, it is usually unreasonable to remove singularities because they are a meaningful aspect of geometric models. Any sort of physical contact or common boundary implies a singularity in the model. Thus, robust algorithms must include a methodical treatment of singular cases. For the algorithms listed in Section 2.6, the treatment is not particularly difficult. Section 5 shows how to treat singular cases for the domain of polygonal regions.

2.3 Gross Infeasibility

The *combinatorial structure* or *order type* of a set of geometric objects is the collection of properties that depends only on the signs of arithmetic expressions of the coordinates of the objects. For example, the fact that three lines are coincident at a single point is part of their combinatorial structure. The values of the angles they form is not. A combinatorial structure is *feasible* if there exists some set of values for the coordinates which give rise to that structure.

The structure generated by a set of lines must be *pseudo-linear* (no two lines intersect more than once) and *planar* (lines cannot cross without a represented point of intersection). A non-pseudo-linear or non-planar structure is clearly infeasible. We refer to these two types of infeasibility as *gross* because they can be detected by a polynomial-time test.

Figure 1 illustrates a grossly infeasible structure which might arise from a single incorrect intersection-order test. Vertex $\mathbf{I}_{12}$ lies before $\mathbf{I}_{13}$ and after $\mathbf{I}_{14}$ along line segment $\mathbf{A}_1\mathbf{B}_1$. This creates a duplicate $\mathbf{I}_{12}$ as illustrated and allows $\mathbf{A}_1\mathbf{B}_1$ and $\mathbf{A}_2\mathbf{B}_2$ to cross without an intersection. Tracing out the boundary of the quadrilateral $\mathbf{A}_3\mathbf{A}_4\mathbf{I}_{24}\mathbf{I}_{23}$ generates the following sequence of segments: $\mathbf{A}_3$, $\mathbf{A}_4$, $\mathbf{I}_{24}$, $\mathbf{I}_{23}$, $\mathbf{I}_{13}$, $\mathbf{I}_{14}$ (!), $\mathbf{B}_4$, $\mathbf{B}_3$, $\mathbf{I}_{23}$, $\mathbf{I}_{24}$, $\mathbf{I}_{14}$, $\mathbf{I}_{13}$, and (finally) back to $\mathbf{A}_3$. If we think of segments $\mathbf{A}_1\mathbf{B}_1$ and $\mathbf{A}_2\mathbf{B}_2$ as straight line cuts through region $\mathbf{A}_3\mathbf{A}_4\mathbf{B}_4\mathbf{B}_3$, the non-pseudo-linearity and non-planarity lead to the erroneous conclusion that two cuts have failed to separate the region into more than one piece. The program may crash, and it certainly fails to model reality.

Avoiding gross infeasibility is a major goal of robust geometry. The general principle in all work to date has been the maintenance of *consistency*. If a relationship can be deduced from previous calculations, it should be. Once we have calculated that $\mathbf{I}_{12}$ lies to the left of $\mathbf{I}_{13}$, we can deduce that it must also lie to the left of $\mathbf{I}_{14}$, and therefore we should not perform a computation that might generate a result inconsistent with this deduction. This technique has the limitation that it can only tractably avoid gross infeasibilities, ones which can be detected in polynomial time. The next section considers other types of infeasibility.

2.4 Subtle Infeasibility

Figure 2 depicts a pseudo-linear and planar structure which is nonetheless infeasible *for the domain of lines* because it violates Pappus's theorem, a standard theorem of projective geometry. (One of the "lines" is subtly curved). Mnev [18] has shown that determining whether a combinatorial structure is generated by some set of lines is as hard as solving the existential theory of the reals and therefore requires exponential time. Thus a polynomial-time algorithm will fail to detect some infeasible structures, and we refer to these as the *subtly infeasible* structures for that algorithm.

When we design a robust algorithm, we must show that the subtly infeasible structures for that algorithm are feasible for some broader domain. For the case of lines, we allow the algorithm to implicitly replace each line with a simple curve. Assuming that these curves are straight enough, the algorithm should be sufficient for modeling physical objects because, after all, there are no true straight lines in the physical world.

2.5 Inaccuracy

Actually, it is trivial to describe a perfectly reliable and feasible algorithm: generate the null output on all inputs. Unfortunately, the null algorithm has 100% inaccuracy. Thus it is essential to specify the accuracy of a robust algorithm. If the output structure of an algorithm is feasible (for some domain), then it is generated by some set of inputs I' which will generally not be the same as the actual set of inputs I. The *accuracy* of the algorithm is a measure of the similarity of I and I'. Conversely, the *numerical error* of an algorithm is the distance between I and I'.

The measure of error is related to the architecture of rounded floating point arithmetic. A computer provides a floating point representation with a B-bit mantissa. The result of each binary operation on two numbers is rounded to B bits, thus introducing a maximum *relative error* of $\epsilon = 2^{-B}$,

$$|(a \mathbf{\ op\ } b)_B - (a \mathbf{\ op\ } b)| \leq \epsilon |a \mathbf{\ op\ } b|,$$

where **op** represents any of the four basic arithmetic operations and where the subscript B indicates the use of rounded B-bit arithmetic. It is usually convenient to assume that the constructed object lies in some region of radius M about the origin. In this case, we can express *absolute errors* in terms of $\mu = \epsilon M$. For example, if $|a|, |b| \leq M$,

$$|(a+b)_B - (a+b)| \leq 2\mu.$$

In the case of intersecting line segments, the input is a set of line segments $\{\mathbf{A}_1\mathbf{B}_1,\ \mathbf{A}_2\mathbf{B}_2, \ldots,\ \mathbf{A}_n\mathbf{B}_n\}$. A robust intersection algorithm may not generate an output structure corresponding to these or any set of line segments; instead, it may correspond to a set of simple curves,

$$\{\gamma_1(t), \gamma_2(t), \ldots, \gamma_n(t) | t \in [0,1]\},$$

such that

$$\gamma_i(0) = \mathbf{A}_i, \qquad \gamma_i(1) = \mathbf{B}_i, \qquad i = 1, 2, \ldots, n.$$

Suppose that γ_i never strays farther than $k\mu$ from its corresponding line segment $\mathbf{A}_i\mathbf{B}_i$. Then we say that the algorithm has a maximum error of $\log k$ bits and a guaranteed accuracy of $N = B - \log k$ bits.

We will say that a robust algorithm is *accurate* if $\log k$ is a small constant. Typical values for B are 21 (IEEE single precision floating point) and 53 (double precision). For practical purposes, $N \geq 20$, which implies one part in a million accuracy, is sufficient. Hence, for practical purposes $\log k \leq 53 - 20 = 23$ is sufficient. For the robust algorithm given in this paper $\log k < 7$.

2.6 Current Results

This author first focused on the task of constructing *line arrangements*: the set of vertices, line segments, and polygons generated by a set of lines in the plane. In order to attain feasibility, it was necessary to assume that lines were very straight simple curves. Section 2.4 explains why this was necessary. Strictly robust algorithms were devised for the construction of arrangements of lines [13], line segments [15], planes [13], and algebraic curves [14]. With Li [17], this author also devised a strictly robust convex hull algorithm for points in two dimensions. Fortune [5] has also given a strictly robust algorithm for constructing convex hulls.

Hoffman and Hopcroft [8] show how to prove a strong form of feasibility for the intersection of no more than two polygonal regions. For this construction, the Pappus configuration (Section 2.4) cannot arise, and thus it is not necessary to "bend" the line segments to prove feasibility. Hopcroft and Kahn [9] do the same for intersections of *convex* polyhedra, another situation in which there is no subtle infeasibility.

A number of algorithms set aside strict accuracy in favor of good error bounds in all but pathological cases. This author [12] gives such an algorithm for constructing unions and intersections of polygonal regions. Segal and Sequin [20] have a similar result. Fortune [5] gives a robust algorithm for maintaining point-set triangulations which has optimal time in all cases and good error bounds in non-pathological cases. Fortune and this author [6] give optimal running time algorithms for constructing arrangements of lines and prove an error bound linear in the number of input lines.

Some algorithms are based on a notion of *consistency* instead of *feasibility.* Karasick [10] gives an algorithm for computing set operations on polyhedral objects, and Suguihara and Iri [23] shows how to compute Voronoi diagrams using rounded arithmetic. More recently, Fortune and Van Wyk [7].

Guibas, Stolfi, and Salesin [19] and Segal and Sequin [21] have given techniques for automating error analysis. These guarantee robustness, but it is difficult to prove good error bounds, even in typical cases.

Finally, Karasick, Lieber, and Nackman [11] demonstrate an exact arithmetic technique that uses only as much precision as is needed. They show that for the task of constructing Delaunay triangulations, the average cost of using exact arithmetic can be much smaller than the worst case.

3 Numerical Operations on Segments

This section describes a number of low level operation on line segments in the plane. The first two operations act on points and line segments: measuring the distance from a point to a line segment and *classifying* the point with respect the line segment as lying above, on, or below the segment. The remaining operations determine the intersection of a line segment with a different types of objects: horizontal or vertical line segments (parallel to the x or y axis), *axis-parallel rectangles* (see below), and other line segments. For each operation we will indicate explicit error bounds. The error analyses appear in Appendix A

3.1 Point-Segment Distance and Classification

This section gives a definition for the bounding rectangle of a line segment $\mathbf{AB}$ and the classification of a point $\mathbf{C}$ with respect to a line segment $\mathbf{AB}$. These definitions are followed by algorithms for computing the distance from $\mathbf{C}$ to $\mathbf{AB}$ and classifying $\mathbf{C}$ with respect to $\mathbf{AB}$.

3.1.1 Definitions

Definition 1 *An* axis-parallel 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). Define* $\mathrm{R}(\mathbf{AB})$, *the* bounding rectangle *of* $\mathbf{AB}$, *to be the axis-parallel rectangle which has* $\mathbf{AB}$ *as one of its diagonals. In interval notation,*

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

Definition 2 (Classification: Above, Below, On) *If* $\mathbf{AB}$ *is a non-vertical line segment* ($\mathbf{A}_x \neq \mathbf{B}_x$)*, and if* $\mathbf{C}$ *is any point such that* $\mathbf{C}_x \in [\mathbf{A}_x, \mathbf{B}_x]$, *then* $\mathbf{C}$ *has a* classification *with respect to* $\mathbf{AB}$*:* above, below, *or* on. *Let* $\mathbf{P}$ *be the point of* $\mathbf{AB}$ *such that* $\mathbf{P}_x = \mathbf{C}_x$.

$$\textit{Point } \mathbf{C} \textit{ lies} \left\{ \begin{array}{c} \text{above} \\ \text{below} \\ \text{on} \end{array} \right\} \mathbf{AB} \textit{ if and only if } \mathbf{C}_y \left\{ \begin{array}{c} > \\ < \\ = \end{array} \right\} \mathbf{P}_y.$$

3.1.2 Distance from Point to Line Segment

Input: Point $\mathbf{C}$ and line segment $\mathbf{AB}$.

Precondition: $\mathbf{C} \in \mathrm{R}(\mathbf{AB})$. It is possible to compute the distance when $\mathbf{C}$ is outside the bounding rectangle, but this is never required for the algorithms in this article.

Output: The signed distance $\delta(\mathbf{C}, \mathbf{AB})$ from $\mathbf{C}$ to $\mathbf{AB}$.

Algorithm:

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

where,

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

This quantity is called the *circulation* of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. There are other ways to calculate the circulation, but this is the most accurate.

Error Bounds: If we 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, then the error in calculating $\delta(\mathbf{C}, \mathbf{AB})$ using B-bit arithmetic is bounded by $\alpha = 6\sqrt{2}\epsilon\mathrm{M}$, where $\epsilon = 2^{-B}$. In other words,

$$|\delta(\mathbf{C}, \mathbf{AB})_B - \delta(\mathbf{C}, \mathbf{AB})| \leq \alpha,$$

where $\delta(\mathbf{C}, \mathbf{AB})_B$ is the the value of $\delta(\mathbf{C}, \mathbf{AB})$ calculated using B-bit arithmetic. The proofs of this and all other error bounds in Section 3 appear in Appendix A.

The ultimate error bound on the robust polygon union algorithm of Section 5 will be 11α. This is the small multiple β of the rounding unit μ promised in the abstract of this paper. Since $\log_2 11 \cdot 6\sqrt{2} < 7$, the ultimate error is about 7 bits.

3.2 Classification Algorithm

Input: Point $\mathbf{C}$ and line segment $\mathbf{AB}$.

Precondition: $\min(\mathbf{A}_x, \mathbf{B}_x) \leq \mathbf{C}_x \leq \max(\mathbf{A}_x, \mathbf{B}_x)$.

Output: Classification of $\mathbf{C}$ with respect to $\mathbf{AB}$.

Algorithm: If $\mathbf{C}$ lies in the bounding rectangle of $\mathbf{AB}$, then the classification of $\mathbf{C}$ depends on the sign of $\delta(\mathbf{C}, \mathbf{AB})$ (which is the same as the sign of $\mathrm{circ}(\mathbf{A}, \mathbf{B}, \mathbf{C})$). However, if rounded arithmetic is used to compute $\delta(\mathbf{C}, \mathbf{AB})$, then the classification is ambiguous when $|\delta(\mathbf{C}, \mathbf{AB})| \leq \alpha$.

The following algorithm is based on the assumption that $\mathbf{A}_x < \mathbf{B}_x$. If $\mathbf{A}_x > \mathbf{B}_x$, then the sign of $\delta(\mathbf{C}, \mathbf{AB})$ is reversed.

> /* special cases */
> if $\mathbf{C} = \mathbf{A}$ or $\mathbf{C} = \mathbf{B}$ then return *on*
> if $\mathbf{A}_y = \mathbf{B}_y = \mathbf{C}_y$ then return *on*
> if $\mathbf{C}_y \geq \max(\mathbf{A}_y, \mathbf{B}_y)$ then return *above*
> if $\mathbf{C}_y \leq \min(\mathbf{A}_y, \mathbf{B}_y)$ then return *below*
>
> /* $\mathbf{C} \in \mathrm{R}(\mathbf{AB})$ */
> if $|\delta(\mathbf{C}, \mathbf{AB})_B| \leq \alpha$ (alternatively $|\mathrm{circ}(\mathbf{A}, \mathbf{B}, \mathbf{C})_B| \leq \alpha|\mathbf{A} - \mathbf{B}|$)
> then Print("Warning: answer is ambiguous.")
> if $\mathrm{circ}(\mathbf{A}, \mathbf{B}, \mathbf{C})_B = 0$ then return *on*
> if $\mathrm{sgn}(\mathbf{B}_x - \mathbf{A}_x)\mathrm{circ}(\mathbf{A}, \mathbf{B}, \mathbf{C})_B > 0$ then return *above*
> if $\mathrm{sgn}(\mathbf{B}_x - \mathbf{A}_x)\mathrm{circ}(\mathbf{A}, \mathbf{B}, \mathbf{C})_B < 0$ then return *below*

Error Bounds: Since this algorithm takes into account the error bounds for the distance computation, it correctly emits an "ambiguous" warning when the classification is uncertain.

3.3 Intersections

This section describes how to compute the intersection of a line segment $\mathbf{AB}$ with a variety of geometric objects. If these computations are performed using rounded arithmetic, they are inaccurate. We establish bounds on the error. We know *a priori* that the actual intersection of $\mathbf{AB}$ with any object must lie in the bounding rectangle $\mathrm{R}(\mathbf{AB})$. The intersection algorithms of this section are designed to assure that the approximate intersection also satisfies this property. This property will be essential for establishing the correctness of the higher level algorithms in Sections 4 and 5.

3.3.1 Intersection with Horizontal or Vertical Line

Input: Line segment $\mathbf{AB}$, and horizontal line $y = c$ or a vertical line $x = c$.

Precondition: Line intersects bounding box R(**AB**).

Output: Intersection point $\langle X, Y\rangle$.

Algorithm: For a horizontal line $y = c$, $c \in [\mathbf{A}_y, \mathbf{B}_y]$, compute

$$X = \mathbf{A}_x + \frac{\mathbf{B}_x - \mathbf{A}_x}{\mathbf{B}_y - \mathbf{A}_y}(c - \mathbf{A}_y), \qquad Y = c.$$

(Vertical line is analogous.) If X_B (X calculated using B-bit arithmetic) lies outside the interval $[\mathbf{A}_x, \mathbf{B}_x]$, then move it to the nearest endpoint.

Error Bounds: Clearly $\langle X_B, Y_B\rangle$ lies on the line $Y = c$. Also, $|\delta(\langle X_B, Y_B\rangle, \mathbf{AB})| < 2\alpha$.

3.3.2 Intersection with a Horizontal or Vertical Line Segment

Input: Line segments **AB** and **CD**, where **CD** is horizontal ($\mathbf{C}_y = \mathbf{D}_y$) or vertical ($\mathbf{C}_x = \mathbf{D}_x$).

Precondition: Segment **CD** must intersect the bounding box R(**AB**) (**C** and **D** may or may not lie inside the box).

Output: Intersection **I** of **CD** with **AB** or the nearest endpoint of **CD** to **AB** if they do not intersect.

Algorithm: Using the algorithm of the previous section, compute the intersection of **AB** with the line $y = \mathbf{C}_y$ (if $\mathbf{C}_y = \mathbf{D}_y$) or with the line $x = \mathbf{C}_x$ (if $\mathbf{C}_x = \mathbf{D}_x$). If the resulting point does not lie on segment **CD**, move it to the nearest endpoint. Call the resulting point **I** ($\mathbf{I}_B$ when calculated using rounded arithmetic).

Error Bounds: If **I** is not **C** or **D**, then $|\delta(\mathbf{I}_B, \mathbf{AB})| \leq 2\alpha$. If **I** equals **C** or **D**, then either $\delta(\mathbf{I}_B, \mathbf{AB})| \leq 2\alpha$ and **I** maximizes $\delta(\mathbf{P}, \mathbf{AB})$ for $\mathbf{P} \in \mathbf{CD}$ or $\delta(\mathbf{I}_B, \mathbf{AB})| \geq -2\alpha$ and **I** minimizes $\delta(\mathbf{P}, \mathbf{AB})$ for $\mathbf{P} \in \mathbf{CD}$. These error bounds follows directly from that of the previous section.

3.3.3 Intersection with an Axis-Parallel Rectangle

Input: **R**, an axis-parallel rectangle $[X_1, X_2] \times [Y_1, Y_2]$ where $X_1 < X_2$ and $Y_1 < Y_2$. This rectangle has vertices,

$$\mathbf{P}_{ij} = \langle X_i, Y_j\rangle, \qquad i = 1, 2 \text{ and } j = 1, 2.$$

For simplicity, we treat only the case in which $\mathbf{A}_x < \mathbf{B}_x$ and $\mathbf{A}_y < \mathbf{B}_y$. The other case ($\mathbf{A}_y > \mathbf{B}_y$) is analogous.

Precondition: $\mathbf{R} \subseteq$ R(**AB**).

Output: An *entry* point $\mathbf{I} \in \mathbf{P}_{11}\mathbf{P}_{12} \cup \mathbf{P}_{11}\mathbf{P}_{21}$ and an *exit* point $\mathbf{O} \in \mathbf{P}_{12}\mathbf{P}_{22} \cup \mathbf{P}_{21}\mathbf{P}_{22}$. If there is no intersection, then output $\mathbf{I} = \mathbf{O}$, and both are equal to the point of **R** closest to **AB**.

Algorithm:

> set **I** equal to the intersection of **AB** with the line $y = Y_1$
> if $\mathbf{I}_x > X_2$ then set $\mathbf{I} = \mathbf{O} = \mathbf{P}_{21}$
> else if $\mathbf{I}_x < X_1$ then
>   set **I** equal to the intersection of **AB** with the line $x = X_1$.
>   if $\mathbf{I}_y > Y_2$ then set $\mathbf{I} = \mathbf{O} = \mathbf{P}_{12}$
>   else if $\mathbf{I}_y < Y_1$ then set $\mathbf{I} = \mathbf{P}_{11}$
>
> if **O** has been calculated then return
>
> set **O** equal to the intersection of **AB** with the line $y = Y_2$
> if $\mathbf{O}_x < X_1$ then set $\mathbf{I} = \mathbf{O} = \mathbf{P}_{12}$
> else if $\mathbf{O}_x > X_2$ then
>   set **O** equal to the intersection of **AB** with the line $x = X_2$.

if $\mathbf{O}_y < Y_1$ then set $\mathbf{I} = \mathbf{O} = \mathbf{P}_{21}$
else if $\mathbf{O}_y > Y_2$ then set $\mathbf{I} = \mathbf{P}_{22}$

Error Bounds: If $\mathbf{I} \neq \mathbf{O}$, then

$$|\delta(\mathbf{I}, \mathbf{AB})|, |\delta(\mathbf{O}, \mathbf{AB})| \leq 2\alpha.$$

If $\mathbf{I} = \mathbf{O}$, then both equal either $\mathbf{P}_{12}$ or $\mathbf{P}_{21}$. If they equal $\mathbf{P}_{12}$, then $\delta(\mathbf{P}_{12}, \mathbf{AB}) \leq 2\alpha$ and $\mathbf{P}_{12}$ maximizes $\delta(\mathbf{P}, \mathbf{AB})$ for $\mathbf{P} \in \mathrm{R}$. If they equal $\mathbf{P}_{21}$, then $\delta(\mathbf{P}_{21}, \mathbf{AB}) \geq -2\alpha$ and $\mathbf{P}_{21}$ minimizes $\delta(\mathbf{P}, \mathbf{AB})$ for $\mathbf{P} \in \mathrm{R}$. These error bounds follows directly from those of the previous section.

3.3.4 Intersection with a Line Segment

Input: Line segments **AB** and **CD**.

Precondition: **AB** and **CD** intersect. (At the end of this section, we give one way one prove this precondition.)

Algorithm: Compute t by solving the linear equation,

$$t(\mathbf{B} - \mathbf{A}) \times (\mathbf{D} - \mathbf{C}) - (\mathbf{C} - \mathbf{A}) \times (\mathbf{D} - \mathbf{C}) = 0.$$

Set $\mathbf{I} = \mathbf{A} + t(\mathbf{B} - \mathbf{A})$.

The value of **I** computed using rounded arithmetic is called $\mathbf{I}_B$. If $\mathbf{I}_B$ lies outside $\mathrm{R}(\mathbf{AB}) \cap \mathrm{R}(\mathbf{CD})$, move it to the nearest point on the boundary of $\mathrm{R}(\mathbf{AB}) \cap \mathrm{R}(\mathbf{CD})$.

Error Bounds: $|\delta(\mathbf{I}_B, \mathbf{AB})| < 6\alpha$ and $|\delta(\mathbf{I}_B, \mathbf{CD})| < 9\alpha$.

This procedure depends on our *a priori* knowledge that segments **AB** and **CD** actually intersect. One way we might know this is if the points **A**, **B**, **C**, and **D** happen to lie on the boundary of some axis-parallel rectangle in a particular order. If, for example, we walk around the rectangle and encounter these points in the order **A**, **C**, **B**, **D**, then we know the segments must intersect. Testing the order can be done by performing comparisons on the x and y coordinates of the points. These comparisons are not subject to round-off error. This technique is in fact what we use in Section 4.4.2.

4 Masc Segments

In order to perform operation on polygons it is necessary to calculate intersections among a set of line segments. However, because of problems of reliability in floating point, we cannot do this directly. Instead, we will replace each input line segment with a close approximation called a *masc* segment. This section defines masc segments and gives an algorithm for computing all intersections among a set of masc segments. This algorithm is an integral part of the robust polygon intersection algorithm in Section 5.

The following section defines masc segments. Section 4.2 gives a high level description of an algorithm for computing intersections of masc segments. Sections 4.3 and 4.4 provide the working parts of the algorithm: updating masc segments, detecting intersection, and computing potential intersection points.

4.1 Definition

Instead of working with line segments we will work with *masc* (*monotonic adaptive straight curve*) segments. Like all good acronyms, *masc* is descriptive of the nature of mascs (or masks), namely that they hide details from our view. For each masc, we know certain combinatorial information, but we can never see the curve itself unless we use exact arithmetic to calculate it. Strictly speaking, the following defines a *masc segment with accuracy* β. As Sections 4.4.2 and 4.4.3 below show, $\beta = 11\alpha$ is the accuracy that can be maintained when intersecting masc segments (see Section 3.1.2 on Page 6 for the definition of α). Specifically, 11α is the sum of the error 2α arising from the intersection of a line segment with a rectangle (Section 3.3.3) and the error 9α arising from the intersection of a line segment with another segment (Section 3.3.4).

Definition 3 (MASC Segment) *A* masc *consists of five parts:*

- *a line segment* basis $\mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}$*;*
- endpoints $\mathbf{D}^{\text{orig}}, \mathbf{D}^{\text{dest}} \in \mathrm{R}(\mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}})$*;*
- *a set A of sites (distinguished points in the plane), called the* above-set*, such that* $\forall \mathbf{A} \in A$, ${\mathbf{D}^{\text{orig}}}_x < \mathbf{A}_x < {\mathbf{D}^{\text{dest}}}_x$*;*
- *a set B of sites, called the* below-set*, such that* $\forall \mathbf{B} \in B$, ${\mathbf{D}^{\text{orig}}}_x < \mathbf{B}_x < {\mathbf{D}^{\text{dest}}}_x$*;*
- *a continuous monotonic (increasing or decreasing) function $f(x)$,*

$$f : [{\mathbf{D}^{\text{orig}}}_x, {\mathbf{D}^{\text{dest}}}_x] \to [{\mathbf{D}^{\text{orig}}}_y, {\mathbf{D}^{\text{dest}}}_y].$$

The curve part of a masc, $\gamma(x) = \langle x, f(x)\rangle$ *for* $x \in [{\mathbf{D}^{\text{orig}}}_x, {\mathbf{D}^{\text{dest}}}_x]$*, is defined to have certain properties.*

- *It joins* $\mathbf{D}^{\text{orig}}$ *to* $\mathbf{D}^{\text{dest}}$*:* $f({\mathbf{D}^{\text{orig}}}_x) = {\mathbf{D}^{\text{orig}}}_y$ *and* $f({\mathbf{D}^{\text{dest}}}_x) = {\mathbf{D}^{\text{dest}}}_y$*.*
- *It lies* below *its above-set A:* $\forall \mathbf{A} \in A,\ f(\mathbf{A}_x) < \mathbf{A}_y$*.*
- *It lies* above *its below-set B,* $\forall \mathbf{B} \in B,\ f(\mathbf{B}_x) > \mathbf{B}_y$*.*
- *It lies within β of line segment* $\mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}$*,*

$$\forall x \in [{\mathbf{D}^{\text{orig}}}_x, {\mathbf{D}^{\text{dest}}}_x] : \ |\delta(\langle x, f(x)\rangle, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}| < \beta.$$

  (This is what is meant by a straight *curve.)*

Two degenerate cases are permitted in which $\mathbf{D}^{\text{orig}}\mathbf{D}^{\text{dest}}$ *is a horizontal or vertical line segment (parallel to the x or y axis).*

A masc has certain properties; for example, as Figure 3 illustrates, if $f(x)$ is an increasing function and if $\mathbf{A} \in$ above-set, then every point of $\langle x, f(x)\rangle$ is excluded from the set $\{\langle x, y\rangle | x \le \mathbf{A}_x \text{ and } y \ge \mathbf{A}_y\}$ (shaded area). Observations such as these are summed up in the following lemma.

Lemma 1 *Let* $\langle \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}, \mathbf{D}^{\text{orig}}, \mathbf{D}^{\text{dest}}, A, B, f\rangle$ *be a masc segment.*

- **(monotonicity)** *If $f(x)$ is* $\left\{ \begin{array}{c} \textit{increasing} \\ \textit{decreasing} \end{array} \right\}$*, then* $\forall \mathbf{A} \in A$ *and* $\forall \mathbf{B} \in B$ *the following three conditions hold:*
  - $\mathbf{A}_y > \min({\mathbf{D}^{\text{orig}}}_y, {\mathbf{D}^{\text{dest}}}_y)$*;*
  - $\mathbf{B}_y < \max({\mathbf{D}^{\text{orig}}}_y, {\mathbf{D}^{\text{dest}}}_y)$*;*
  - *either* $\mathbf{A}_y > \mathbf{B}_y$ *or* $\mathbf{A}_x \left\{ \begin{array}{c} < \\ > \end{array} \right\} \mathbf{B}_x$*.*
- **(accuracy)** *The following three bounds hold:*
  - $\forall \mathbf{A} \in A$*:* $\delta(\mathbf{A}, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}) > -\beta$*;*
  - $\forall \mathbf{B} \in B$*:* $\delta(\mathbf{B}, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}) < \beta$*;*
  - $|\delta(\mathbf{D}^{\text{orig}}, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}})|, |\delta(\mathbf{D}^{\text{dest}}, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}})| < \beta$*.*

The proof of this lemma and the other lemmas and theorems of this section appear in Appendix B.

Now we note that the conclusions of Lemma 1, the properties we call *monotonicity* and *accuracy*, are properties of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}$, $\mathbf{D}^{\text{orig}}$, and $\mathbf{D}^{\text{dest}}$, but *not* $f(x)$ explicitly. These properties are a necessary consequence of the existence of f. The following theorem shows that these properties are sufficient to ensure that f exists. We call this the *Hidden Variable Theorem* because it shows that certain observable properties, the monotonicity and accuracy of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}$, $\mathbf{D}^{\text{orig}}$, and $\mathbf{D}^{\text{dest}}$, imply the existence of an unobservable quantity, the shape of f.

Theorem 1 (First Hidden Variable Theorem) *If* $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}$, $\mathbf{D}^{\text{orig}}$, *and* $\mathbf{D}^{\text{dest}}$ *satisfy the conclusions of Lemma 1, there exists a monotonic function* $f(x)$ *such that* $\langle \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}, \mathbf{D}^{\text{orig}}, \mathbf{D}^{\text{dest}}, A, B, f\rangle$ *is a masc.*

This theorem also expresses what is meant by f being *adaptive*: whenever the combinatorial information satisfies a few simple rules, a function f exists. If we add another site to the above-set or below-set, the function f can change shape to accommodate the new site. The proofs of this and all other theorems of Section 4 are given in Appendix B.

4.2 Intersection Algorithm

Our robust intersection algorithm for a set of line segments consists of two parts. First we convert the line segments into masc segments. Then we compute the intersections among these masc segments.

Converting: Let us suppose we have a list of line segments:

$$\mathbf{C}_1^{\text{orig}}\mathbf{C}_1^{\text{dest}}, \quad \mathbf{C}_2^{\text{orig}}\mathbf{C}_2^{\text{dest}}, \quad \ldots, \quad \mathbf{C}_n^{\text{orig}}\mathbf{C}_n^{\text{dest}}.$$

Put all the endpoints into a *universe* of sites $U = \{\mathbf{C}_i^{\text{orig}}, \mathbf{C}_i^{\text{dest}} | i = 1, 2, \ldots, n\}$. For $i = 1, 2, \ldots, n$, to create a masc segment $\langle \mathbf{C}_i^{\text{orig}}\mathbf{C}_i^{\text{dest}}, \mathbf{D}_i^{\text{orig}}, \mathbf{D}_i^{\text{dest}}, A_i, B_i, f_i\rangle$, set $\mathbf{D}_i^{\text{orig}}$ and $\mathbf{D}_i^{\text{dest}}$ equal to $\mathbf{C}_i^{\text{orig}}$ and $\mathbf{C}_i^{\text{dest}}$, respectively. Initialize A_i and B_i to $\emptyset$. Step through the sites $\mathbf{P} \in U$ sequentially. If ${\mathbf{D}_i^{\text{orig}}}_x < \mathbf{P}_x < {\mathbf{D}_i^{\text{dest}}}_x$, add $\mathbf{P}$ to A_i or B_i using the update algorithm of Section 4.3.1.

Computing Intersections: Section 4.4.1 gives an effective procedure for determining if two mascs $\gamma_f = \langle x, f(x)\rangle$ and $\gamma_g = \langle x, g(x)\rangle$ intersect. One simply runs through the current universe of sites looking for sites $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{P}$ is evidence that $f(\mathbf{P}_x) > g(\mathbf{P}_x)$ and $\mathbf{Q} \in U$ is evidence that $f(\mathbf{Q}_x) < g(\mathbf{Q}_x)$. Of course, if one does this search naively, it has cost $n(n-1)|U|(|U|-1)/4$, since we would have to look at all pairs of segments and all pairs of points in U. However, we expect that outside of rare pathological cases, one could find all intersections by a sweepline algorithm using $\mathrm{O}(n \log n + k \log n)$ time, where k is the number of intersect points. The pathological cases may increase the running time, but they do not spoil the correctness of the algorithm. Deciding on the most efficient intersection algorithm[4] is beyond the scope of this paper.

Each time we uncover evidence of an intersection between curves γ_f and γ_g, the algorithms in Sections 4.4.2 through 4.4.4 provide an approximate intersection point $\mathbf{I}^{\text{approx}}$. We first attempt to split γ_f and γ_g at $\mathbf{I}^{\text{approx}}$ using the algorithm in Section 4.3.2. If some already existing site $\mathbf{P}^{\text{block}}$ prevents either curve from being split at $\mathbf{I}^{\text{approx}}$, we throw away $\mathbf{I}^{\text{approx}}$ and split that curve at $\mathbf{P}^{\text{block}}$. In this way we always make progress: either we split both curves at a new site $\mathbf{I}^{\text{approx}}$, or we split one curve at a previously existing site without increasing the number of sites. If we successfully split both curves at $\mathbf{I}^{\text{approx}}$, then we add $\mathbf{I}^{\text{approx}}$ to U and update all above-sets or below-sets accordingly using the algorithm of Section 4.3.1.

Termination: A masc segment is nearly a straight line segment. Nominally two masc segments intersect at most once. In pathological cases where many segments are nearly parallel, it may happen that two masc segments intersect more than once. In any case, each time we attempt to compute the intersection of two segments, either we successfully split both segments or we split one of the segments at an already existing site. Thus the number of splits is bounded by nk, and thus the algorithm terminates. Except in pathological cases, the number of splits should be in $\mathrm{O}(k)$.

Correctness: In the output, each input masc segment γ has been split at each point it intersects some other masc segment. Thus the curve is split into a sequence of curves $\gamma_1, \gamma_2, \ldots, \gamma_{n_i}$. As Figure 4 depicts, each of these is a monotonic curve or a horizontal or vertical line segment. These curves all stay within 11α of the original segment $\mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}$, and therefore they are masc segments with accuracy $\beta = 11\alpha$.

It is not possible using rounded arithmetic to convert each masc segment back into a sequence of straight line segments. However, for the purposes of graphical display, calculation of area, or some other type of calculation, the line segment $\mathbf{D}_i^{\text{orig}}\mathbf{D}_i^{\text{dest}}$ is a good approximation to a masc segment $\langle \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}, \mathbf{D}_i^{\text{orig}}, \mathbf{D}_i^{\text{dest}}, A_f, B_f, f\rangle$.

[4] Another possibility is bucketing of regions of the plane.

Since $\mathbf{D}_i^{\text{orig}}$, $\mathbf{D}_i^{\text{dest}}$, and the masc segment all stay within 11α of line segment $\mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}$, the line segment $\mathbf{D}_i^{\text{orig}}\mathbf{D}_i^{\text{dest}}$ therefore stays within 22α of the masc segment.

If we choose to use *explicit* updating and splitting algorithms as given in Section 4.3.3, we can explicitly calculate the shapes of the masc segments. However the output of the explicit version is not necessarily accurate because it allows segments to drift farther than 11α from their original positions; in fact, the accuracy bound β may grow with the number of input curves.

4.3 Updating and Splitting Masc Segments

A masc segment is shaped only by the sites in its above and below sets. It changes its shape (adapts) as new sites are added to these sets–if these new sites satisfy the premises of the first Hidden Variable Theorem. Sometimes these premises, monotonicity and accuracy, will constrain our actions and prevent us from adding a site to either the above or below set. This section give algorithms for updating masc segments which maintain the validity of the above and below sets. The *update* algorithm in the next section describes how a masc segment adapts to a new site, assuming we have the freedom to put the new site into either the above or below set. Sometimes we do not have this freedom. In particular, if $\mathbf{I}$ is an intersection point we have calculated for two masc segments, we desire to *split* each of these segments at the point $\mathbf{I}$. Splitting a masc involves partitioning the above and below sets each into two parts. Section 4.3.2 gives a complete algorithm for splitting.

If we prefer that our masc segments be explicit, true line segments, we can perform an operations called *explicit updating* and *explicit splitting*. These operations are not strictly robust in the sense that they do not have the accuracy bounds that the implicit version do. Section 4.3.3 gives the explicit operations.

4.3.1 Updating the Above and Below Sets

This section describes a set of rules which can be used to update masc segments correctly. These rules will be for the case of an increasing masc segment. The decreasing case is the same, except that comparisons of y-coordinates are reversed.

Let $\langle \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}, \mathbf{D}^{\text{orig}}, \mathbf{D}^{\text{dest}}, A, B, f\rangle$ be a masc, and let $\mathbf{P}$ be a point such that ${\mathbf{D}^{\text{orig}}}_x < \mathbf{P}_x < {\mathbf{D}^{\text{dest}}}_x$.

Rule 1: If $\exists \mathbf{A} \in A$ such that $\mathbf{P}_x \leq \mathbf{A}_x$ and $\mathbf{P}_y \geq \mathbf{A}_y$ (see Figure 5), then $\mathbf{P}$ *must* be added to the above-set A.

Rule 2: If $\exists \mathbf{B} \in B$ such that $\mathbf{P}_x \geq \mathbf{B}_x$ and $\mathbf{P}_y \leq \mathbf{B}_y$, then $\mathbf{P}$ *must* be added to the below-set B.

Rule 3: Otherwise, if we can show that $\delta(\mathbf{P}, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}) > -11\alpha$, then we *may* add $\mathbf{P}$ to A, if we choose. Similarly, if we can show that $\delta(\mathbf{P}, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}) < 11\alpha$, then we *may* add $\mathbf{P}$ to B.

In general, if Rules 1 or 2 do not hold, then we use the classification algorithm of Section 3.2 to classify $\mathbf{P}$ with respect to $\mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}$ using rounded arithmetic. This algorithm can return an incorrect classification only when $|\delta(\mathbf{P}, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}})| < \alpha$ which is well within the 11α of Rule 3.

Definition 4 *Let U be a set of sites in the plane. A masc segment $\langle \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}, \mathbf{D}^{\text{orig}}, \mathbf{D}^{\text{dest}}, A, B, f\rangle$ is* adapted *to U if*

$$A \cup B \cup \{\mathbf{C}^{\text{orig}}, \mathbf{C}^{\text{dest}}, \mathbf{D}^{\text{orig}}, \mathbf{D}^{\text{dest}}\} \subseteq U \qquad \textit{and} \qquad A \cup B = \{\mathbf{P} \in U | {\mathbf{D}^{\text{orig}}}_x < \mathbf{P}_x < {\mathbf{D}^{\text{dest}}}_x\}.$$

In our algorithm, all masc segments are adapted to the same universe U of sites, and therefore every time we create a new site, we must update each segment with respect to that site.

4.3.2 Splitting

Let $\langle \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}, \mathbf{D}^{\text{orig}}, \mathbf{D}^{\text{dest}}, A, B, f\rangle$ be a masc segment. Section 4.4 shows how to detect when this segment intersects some other segment and gives an algorithm for computing an approximate point of intersection.

This intersection point $\mathbf{I}$ is guaranteed to lie in the rectangle $\mathrm{R}(\mathbf{D}^{\mathrm{orig}}, \mathbf{D}^{\mathrm{dest}})$, and it lies within 11α of $\mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}$, $|\delta(\mathbf{I}, \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}})| < 11\alpha$. This section gives an algorithm for splitting the masc segment $\gamma(x) = \langle x, f(x)\rangle$ at the point $\mathbf{I}$. Nominally, this is a simple task; however, sometimes it will not be possible to split γ at $\mathbf{I}$. In this case, we split γ at some already existing site in U. As in the previous section we consider only the case in which f is an increasing function of x. As before, the decreasing case is the same, except that comparisons of y-coordinates are reversed.

Nominal Case: Splitting is very similar to updating as described in the previous section, and we start by updating γ with $\mathbf{I}$. If Rules 1 and 2 do not force us to put $\mathbf{I}$ in A or B, then we simply replace $\langle \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}, \mathbf{D}^{\mathrm{orig}}, \mathbf{D}^{\mathrm{dest}}, A, B, f\rangle$ by $\langle \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}, \mathbf{D}_1^{\mathrm{orig}}, \mathbf{D}_1^{\mathrm{dest}}, A_1, B_1, f_1\rangle$ and $\langle \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}, \mathbf{D}_2^{\mathrm{orig}}, \mathbf{D}_2^{\mathrm{dest}}, A_2, B_2, f_2\rangle$, where

$$\begin{array}{ll} \mathbf{D}_1^{\mathrm{orig}} = \mathbf{D}^{\mathrm{orig}}, & \mathbf{D}_2^{\mathrm{orig}} = \mathbf{I}, \\ \mathbf{D}_1^{\mathrm{dest}} = \mathbf{I}, & \mathbf{D}_2^{\mathrm{dest}} = \mathbf{D}^{\mathrm{dest}}, \\ A_1 = \{\mathbf{A} \in A | {\mathbf{D}_1^{\mathrm{orig}}}_x < \mathbf{A}_x < {\mathbf{D}_1^{\mathrm{dest}}}_x\}, & A_2 = \{\mathbf{A} \in A | {\mathbf{D}_2^{\mathrm{orig}}}_x < \mathbf{A}_x < {\mathbf{D}_2^{\mathrm{dest}}}_x\}, \\ B_1 = \{\mathbf{B} \in B | {\mathbf{D}_1^{\mathrm{orig}}}_x < \mathbf{B}_x < {\mathbf{D}_1^{\mathrm{dest}}}_x\}, & B_2 = \{\mathbf{B} \in B | {\mathbf{D}_2^{\mathrm{orig}}}_x < \mathbf{B}_x < {\mathbf{D}_2^{\mathrm{dest}}}_x\}. \end{array}$$

Note that $\gamma_1 = \langle x, f_1(x)\rangle$ and $\gamma_2 = \langle x, f_2(x)\rangle$ are still approximations to (subsegments of) segment $\mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}$.

It can happen that after performing a nominal split, $\mathbf{D}_1^{\mathrm{orig}}\mathbf{D}_1^{\mathrm{dest}}$ or $\mathbf{D}_2^{\mathrm{orig}}\mathbf{D}_2^{\mathrm{dest}}$ ends up being parallel to the x or y axis. For this reason, the definition of a masc segment (Definition 3) permitted horizontal or vertical line segments as degenerate cases.

The following two paragraphs deal with the non-nominal cases.

Special Case 1: If Rule 1 applies to the point $\mathbf{I}$, then it is not possible for γ to adapt itself to pass through $\mathbf{I}$. We say that the site $\mathbf{A} \in A$ whose existence is implied by Rule 1 *blocks* $\mathbf{I}$ from splitting γ. Of course, $\mathbf{A}$ might not be the only site which blocks $\mathbf{I}$. We choose the site $\mathbf{A}^{\mathrm{block}} \in A$ that blocks $\mathbf{I}$ from splitting γ and that has largest x-coordinate ${\mathbf{A}^{\mathrm{block}}}_x$. If more than one site has the same x-coordinate, then choose the one with minimum y-coordinate.

Special Case 2: In an analogous fashion, if Rule 2 applies, we say that the site $\mathbf{B} \in B$ whose existence is implied by this rule *blocks* $\mathbf{I}$. We choose the site $\mathbf{B}^{\mathrm{block}} \in B$ that blocks $\mathbf{I}$ and that has smallest x-coordinate ${\mathbf{B}^{\mathrm{block}}}_x$. If more than one site has the same x-coordinate, then choose the one with maximum y-coordinate.

If either of the special cases apply, we cannot split f at $\mathbf{I}$, but we can split it at $\mathbf{A}^{\mathrm{block}}$ or $\mathbf{B}^{\mathrm{block}}$, whichever the case may be, and this is what we do.

Lemma 2 *If Special Case 1 holds, splitting f at $\mathbf{A}^{\mathrm{block}}$ generates two valid masc segments. Similarly, if Special Case 2 holds, splitting f at $\mathbf{B}^{\mathrm{block}}$ generates two valid masc segments.*

4.3.3 Explicit Updating and Splitting

This section gives *explicit* versions of the updating and splitting operations described in the previous two sections. These explicit versions have poorer accuracy bounds than the implicit versions, but they are useful if it is necessary to have explicit representations for the segments.

We first describe explicit splitting. This is the same as implicit splitting with the following change. When we split a masc segment $\langle \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}, \mathbf{D}^{\mathrm{orig}}, \mathbf{D}^{\mathrm{dest}}, A, B, f\rangle$ into two, we also split the basis segment $\mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}$. Thus the output is two masc segments,

$$\langle \mathbf{C}_1^{\mathrm{orig}}\mathbf{C}_1^{\mathrm{dest}}, \mathbf{D}_1^{\mathrm{orig}}, \mathbf{D}_1^{\mathrm{dest}}, A_1, B_1, f_1\rangle \text{ and } \langle \mathbf{C}_2^{\mathrm{orig}}\mathbf{C}_2^{\mathrm{dest}}, \mathbf{D}_2^{\mathrm{orig}}, \mathbf{D}_2^{\mathrm{dest}}, A_2, B_2, f_2\rangle,$$

where $\mathbf{D}_1^{\mathrm{orig}}$,$\mathbf{D}_1^{\mathrm{dest}}$,$A_1$,$B_1$, $\mathbf{D}_2^{\mathrm{orig}}$,$\mathbf{D}_2^{\mathrm{dest}}$,$A_2$, and B_2 are computed as before and

$$\mathbf{C}_1^{\mathrm{orig}} = \mathbf{D}_1^{\mathrm{orig}}, \quad \mathbf{C}_2^{\mathrm{orig}} = \mathbf{D}_2^{\mathrm{orig}}, \quad \mathbf{C}_1^{\mathrm{dest}} = \mathbf{D}_1^{\mathrm{dest}}, \quad \mathbf{C}_2^{\mathrm{dest}} = \mathbf{D}_2^{\mathrm{dest}}.$$

Explicit updating is a modification of implicit updating. If we have a point $\mathbf{P}$ that does not satisfy Rules 1 or 2 and if $|\delta(\mathbf{P}, \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}})_B| \le \alpha$ (subscript B implies use of B-bit rounded arithmetic), we explicitly split the masc segment which we are updating at the site $\mathbf{P}$.

If we use explicit operations in all cases, then each masc segment

$$\langle \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}, \mathbf{D}^{\mathrm{orig}}, \mathbf{D}^{\mathrm{dest}}, A, B, f\rangle$$

is explicitly equal to its basis line segment $\mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}$.

4.4 Computing Intersections

This section gives algorithms for effectively detecting the fact that two masc segments intersect and for approximately calculating this point of intersection. Section 4.4.1 gives a Second Hidden Variable Theorem that shows we can effectively detect when two masc segments intersect using only set operations on their above and below sets and comparisons of coordinates (which are not subject to round-off error). However, we cannot, without incurring round-off error, explicitly calculate the exact intersection point $\mathbf{I}^{\mathrm{exact}}$. Instead, we must settle for an approximate intersection point $\mathbf{I}^{\mathrm{approx}}$. This point lies within the bounding rectangles of both curves and it lies within 11α of the basis segments of the curves (Definition 3). Sections 4.4.2 through 4.4.4 give algorithms for computing $\mathbf{I}^{\mathrm{approx}}$ in all cases.

4.4.1 Evidence

By looking at the above-sets and below-sets of masc segments, we can learn some information about how their shapes are related. This section shows how to determine if two masc segments intersect.

Let $\langle \mathbf{C}_f^{\mathrm{orig}}\mathbf{C}_f^{\mathrm{dest}}, \mathbf{D}_f^{\mathrm{orig}}, \mathbf{D}_f^{\mathrm{dest}}, A_f, B_f, f\rangle$ and $\langle \mathbf{C}_g^{\mathrm{orig}}\mathbf{C}_g^{\mathrm{dest}}, \mathbf{D}_g^{\mathrm{orig}}, \mathbf{D}_g^{\mathrm{dest}}, A_g, B_g, g\rangle$ be masc segments such that

$$(\mathbf{D}_f^{\mathrm{orig}}{}_x, \mathbf{D}_f^{\mathrm{dest}}{}_x) \cap (\mathbf{D}_g^{\mathrm{orig}}{}_x, \mathbf{D}_g^{\mathrm{dest}}{}_x) \neq \emptyset.$$

Definition 5 (Evidence) *Let $\mathbf{P} \in U$ be a site such that*

$$\mathbf{P}_x \in (\mathbf{D}_f^{\mathrm{orig}}{}_x, \mathbf{D}_f^{\mathrm{dest}}{}_x) \cap (\mathbf{D}_g^{\mathrm{orig}}{}_x, \mathbf{D}_g^{\mathrm{dest}}{}_x).$$

If $\mathbf{P} \in A_f$ and if $\mathbf{P} \in B_g$ then we say that $\mathbf{P}$ is evidence *that $f(\mathbf{P}_x) < g(\mathbf{P}_x)$.*

If $\mathbf{D}_f^{\mathrm{orig}}{}_x \in [\mathbf{D}_f^{\mathrm{orig}}{}_x, \mathbf{D}_f^{\mathrm{dest}}{}_x] \cap [\mathbf{D}_g^{\mathrm{orig}}{}_x, \mathbf{D}_g^{\mathrm{dest}}{}_x]$ and

$$\mathbf{D}_f^{\mathrm{orig}} \in \left\{ \begin{array}{c} A_g \\ B_g \end{array} \right\} \qquad or \qquad \left(\mathbf{D}_f^{\mathrm{orig}}{}_x = \mathbf{D}_g^{\mathrm{orig}}{}_x \text{ and } \mathbf{D}_f^{\mathrm{orig}}{}_y \left\{ \begin{array}{c} > \\ < \end{array} \right\} \mathbf{D}_g^{\mathrm{orig}}{}_y \right)$$

then $\mathbf{D}_f^{\mathrm{orig}}$ is evidence that $f(\mathbf{D}_f^{\mathrm{orig}}{}_x) \left\{ \begin{array}{c} > \\ < \end{array} \right\} g(\mathbf{D}_f^{\mathrm{orig}}{}_x)$. Similarly for $\mathbf{D}_f^{\mathrm{dest}}$.

Lemma 3 *If $\mathbf{P} \in U$ is evidence that $f(\mathbf{P}_x) > g(\mathbf{P}_x)$ and if $\mathbf{Q} \in U$ is evidence that $f(\mathbf{Q}_x) < g(\mathbf{Q}_x)$, the curves $\gamma_f(x) = \langle x, f(x)\rangle$ and $\gamma_g(x) = \langle x, g(x)\rangle$ must intersect at some point $\mathbf{I}$ such that $\mathbf{I}_x \in (\mathbf{D}_f^{\mathrm{orig}}{}_x, \mathbf{D}_f^{\mathrm{dest}}{}_x) \cap (\mathbf{D}_g^{\mathrm{orig}}{}_x, \mathbf{D}_g^{\mathrm{dest}}{}_x)$.*

Proof: This is a fundamental property of continuous functions.

Lemma 3 gives a testable condition that suffices to show that two masc segments intersect. The following theorem shows that the condition is also necessary.

Theorem 2 (Second Hidden Variable Theorem) *If $\gamma_f(x)$ and $\gamma_g(x)$ intersect in their interiors–in the range $x \in (\mathbf{D}_f^{\mathrm{orig}}{}_x, \mathbf{D}_f^{\mathrm{dest}}{}_x) \cap (\mathbf{D}_g^{\mathrm{orig}}{}_x, \mathbf{D}_g^{\mathrm{dest}}{}_x)$–then there exists sites $\mathbf{P}, \mathbf{Q} \in U$ that are evidence that $f(\mathbf{P}_x) > g(\mathbf{P}_x)$ and $f(\mathbf{Q}_x) < g(\mathbf{Q}_x)$.*

4.4.2 Computing Intersections: Opposite Slope Sign

This section describes how to compute an approximate intersection $\mathbf{I}^{\rm approx}$ for two masc segments $\gamma_f = \langle \mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest}, \mathbf{D}_f^{\rm orig}, \mathbf{D}_f^{\rm dest}, A_f, B_f, f\rangle$ and $\gamma_g = \langle \mathbf{C}_g^{\rm orig}\mathbf{C}_g^{\rm dest}, \mathbf{D}_g^{\rm orig}, \mathbf{D}_g^{\rm dest}, A_g, B_g, g\rangle$ such that f is increasing and g is decreasing. Obviously, the same algorithm will work if f is decreasing and g is increasing. The following section will treat the cases in which both are increasing or both decreasing.

The Second Hidden Variable Theorem tells us that if two masc segments intersect, there must be sites $\mathbf{P}, \mathbf{Q} \in U$ that are evidence that $f(\mathbf{P}_x) < g(\mathbf{P}_x)$ and $f(\mathbf{Q}_x) > g(\mathbf{Q}_x)$. Since f is increasing, and g is decreasing, they can have at most one intersection. Therefore, $\mathbf{D}_f^{\rm orig}$ and $\mathbf{D}_f^{\rm dest}$ lie on opposite side of the masc segment γ_g and *vice versa*. We can assume $\mathbf{P}$ and $\mathbf{Q}$ are endpoints of the two masc segments, in particular, $\mathbf{P} \in \{\mathbf{D}_f^{\rm orig}, \mathbf{D}_g^{\rm orig}\}$ and $\mathbf{Q} \in \{\mathbf{D}_f^{\rm dest}, \mathbf{D}_g^{\rm dest}\}$.

Let $\mathbf{R} = {\rm R}(\mathbf{D}_f^{\rm orig}\mathbf{D}_f^{\rm dest}) \cap {\rm R}(\mathbf{D}_g^{\rm orig}\mathbf{D}_g^{\rm dest})$. The intersection lies somewhere within this rectangle $\mathbf{R}$. Using the algorithm of Section 3.3.3, we compute the intersections $\mathbf{I}_f$ and $\mathbf{O}_f$ of line segment $\mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest}$ with rectangle $\mathbf{R}$. Similarly we compute the intersections $\mathbf{I}_g$ and $\mathbf{O}_g$ of line segment $\mathbf{C}_g^{\rm orig}\mathbf{C}_g^{\rm dest}$ with rectangle $\mathbf{R}$.

Claim 1 *If* $\mathbf{I}_f \neq \mathbf{O}_f$, *then*

$$|\delta(\mathbf{I}_f, \mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest})|, |\delta(\mathbf{O}_f, \mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest})| \leq 2\alpha.$$

If $\mathbf{I}_f = \mathbf{O}_f$, *then*

$$|\delta(\mathbf{I}_f, \mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest})|, |\delta(\mathbf{O}_f, \mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest})| < 11\alpha.$$

Analogous bounds hold for $\mathbf{I}_g$ *and* $\mathbf{O}_g$.

Proof: The first bound is proved in Section 3.3.3. The second case only occurs when segment $\mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest}$ fails to intersect rectangle $\mathbf{R}$. In this case, $\mathbf{I}_f$ and $\mathbf{O}_f$ are set equal to the point of $\mathbf{R}$ closest to $\mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest}$. However, we know that $\gamma_f(x) = \langle x, f(x)\rangle$ and $\gamma_g(x) = \langle x, g(x)\rangle$ intersect inside $\mathbf{R}$ at some point $\mathbf{I}^{\rm exact}$. Since $\mathbf{I}^{\rm exact}$ lies on γ_f, it lies within 11α of $\mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest}$. Therefore, $\mathbf{I}_f$ and $\mathbf{O}_f$ also lie within 11α of $\mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest}$.

Continuing the algorithm, in order to compute an approximate intersection $\mathbf{I}^{\rm approx}$ between γ_f and γ_g, we temporarily replace γ_f with the polygonal path $\mathbf{D}_f^{\rm orig}\mathbf{I}_f\mathbf{O}_f\mathbf{D}_f^{\rm dest}$ and we replace γ_g with the polygonal path $\mathbf{D}_g^{\rm orig}\mathbf{I}_g\mathbf{O}_g\mathbf{D}_g^{\rm dest}$. Like γ_f and γ_g, these paths intersect somewhere inside or on the boundary of $\mathbf{R}$, and we know from Claim 1 that these polygonal paths stay within 11α of corresponding basis segments $\mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest}$ and $\mathbf{C}_g^{\rm orig}\mathbf{C}_g^{\rm dest}$, respectively. We will compute the intersection of these two paths, approximately, and then show that this approximate intersection $\mathbf{I}^{\rm approx}$ is sufficiently accurate.

Nominal Case: in the nominal case, the two paths intersect in the interior of $\mathbf{R}$ at the intersection of segments $\mathbf{I}_f\mathbf{O}_f$ and $\mathbf{I}_g\mathbf{O}_g$. We can determine that this nominal case holds by looking at the positions of $\mathbf{I}_f$, $\mathbf{O}_f$, $\mathbf{I}_g$,and $\mathbf{O}_g$ along the boundary of $\mathbf{R}$. If these two segments form an "X" topologically, then we compute their intersection $\mathbf{I}_B$ inside the rectangle $\mathbf{R}$ using the algorithm in Section 3.3.4. The nominal case clearly holds only if $\mathbf{I}_f \neq \mathbf{O}_f$ and $\mathbf{I}_g \neq \mathbf{O}_g$. Therefore, we have

$$|\delta(\mathbf{I}_f, \mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest})|,\ |\delta(\mathbf{O}_f, \mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest})|,\ |\delta(\mathbf{I}_g, \mathbf{C}_g^{\rm orig}\mathbf{C}_g^{\rm dest})|,\ |\delta(\mathbf{O}_g, \mathbf{C}_g^{\rm orig}\mathbf{C}_g^{\rm dest})| \leq 2\alpha.$$

Section 3.3.4 and Appendix A.2.2 prove the bounds,

$$|\delta(\mathbf{I}_B, \mathbf{I}_f\mathbf{O}_f)|, |\delta(\mathbf{I}_B, \mathbf{I}_g\mathbf{O}_g)| < 9\alpha.$$

Combining these bounds,

$$|\delta(\mathbf{I}_B, \mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest})|, |\delta(\mathbf{I}_B, \mathbf{C}_g^{\rm orig}\mathbf{C}_g^{\rm dest})| < 11\alpha.$$

Thus, we can set $\mathbf{I}^{\rm approx}$ equal to $\mathbf{I}_B$.

Special Case: We know that the two paths $\mathbf{D}_f^{\rm orig}\mathbf{I}_f\mathbf{O}_f\mathbf{D}_f^{\rm dest}$ and $\mathbf{D}_g^{\rm orig}\mathbf{I}_g\mathbf{O}_g\mathbf{D}_g^{\rm dest}$ must intersect on the boundary of $\mathbf{R}$. In this case, we can find a point $\mathbf{I}^{\rm approx}$ which is in common to both paths by examining the vertices of the rectangle, the vertices of the paths, and at most two horizontal or vertical line segments. This can be done using no arithmetic operations other than comparisons. Figure 6 illustrates an example in which $\mathbf{I}_f$ is the vertex in common to both paths. By the claim above, the point $\mathbf{I}^{\rm approx}$ we generate in this manner lies within 11α of segments $\mathbf{C}_f^{\rm orig}\mathbf{C}_f^{\rm dest}$ and $\mathbf{C}_g^{\rm orig}\mathbf{C}_g^{\rm dest}$.

4.4.3 Computing Intersections: Same Slope Sign

This section describes how to compute an approximate intersection $\mathbf{I}^{\text{approx}}$ for two masc segments $\langle \mathbf{C}_f^{\text{orig}}\mathbf{C}_f^{\text{dest}}, \mathbf{D}_f^{\text{orig}}, \mathbf{D}_f^{\text{dest}}, A_f,$ and $\langle \mathbf{C}_g^{\text{orig}}\mathbf{C}_g^{\text{dest}}, \mathbf{D}_g^{\text{orig}}, \mathbf{D}_g^{\text{dest}}, A_g, B_g, g\rangle$ such that both f and g are increasing. Obviously, the same algorithm will work if both functions are decreasing.

As we reasoned in the previous section, if two masc segments intersect, there must be sites $\mathbf{P}, \mathbf{Q} \in U$ that are evidence that $f(\mathbf{P}_x) < g(\mathbf{P}_x)$ and $f(\mathbf{Q}_x) > g(\mathbf{Q}_x)$. In the case of two increasing functions, we *cannot* assume that $\mathbf{P}$ and $\mathbf{Q}$ are endpoints of the masc segments. However, it is true that $\mathrm{R}(\mathbf{PQ}) \subseteq \mathrm{R}(\mathbf{D}_f^{\text{orig}}\mathbf{D}_f^{\text{dest}}) \cap \mathrm{R}(\mathbf{D}_g^{\text{orig}}\mathbf{D}_g^{\text{dest}})$, and therefore the intersection of γ_f and γ_g lies in $\mathrm{R}(\mathbf{PQ})$.

As in the previous section, we temporarily replace γ_f with a polygonal path from $\mathbf{D}_f^{\text{orig}}$ to $\mathbf{D}_f^{\text{dest}}$. We compute the intersection $\mathbf{I}_f$ and $\mathbf{O}_f$ of line segment $\mathbf{C}_f^{\text{orig}}\mathbf{C}_f^{\text{dest}}$ with rectangle $\mathrm{R}(\mathbf{PQ})$, using the algorithm in Section 3.3.3. Then we replace γ_f with the path $\mathbf{D}_f^{\text{orig}}\mathbf{I}_f\mathbf{O}_f\mathbf{D}_f^{\text{dest}}$. However, if $\mathbf{I}_{f,x} = \mathbf{P}_x$ (and thus $\mathbf{I}_{f,y} \geq \mathbf{P}_y$), then we replace the segment $\mathbf{D}_f^{\text{orig}}\mathbf{I}_f$ with the path $\mathbf{D}_f^{\text{orig}}\mathbf{P}\mathbf{I}_f$. Similarly, if $\mathbf{O}_{f,x} = \mathbf{Q}_x$, then we replace $\mathbf{O}_f\mathbf{D}_f^{\text{dest}}$ with $\mathbf{O}_f\mathbf{Q}\mathbf{D}_f^{\text{dest}}$. We denote the resulting path by $\mathbf{D}_f^{\text{orig}}(\mathbf{P})\mathbf{I}_f\mathbf{O}_f(\mathbf{Q})\mathbf{D}_f^{\text{dest}}$, where the parentheses indicate that the points $\mathbf{P}$ and $\mathbf{Q}$ may or may not be vertices of the path.

Claim 2 *Let* $\mathbf{P}' = \langle \mathbf{P}_x, \mathbf{Q}_y\rangle$ *and* $\mathbf{Q}' = \langle \mathbf{Q}_x, \mathbf{P}_y\rangle$. *The path* $\mathbf{D}_f^{\text{orig}}(\mathbf{P})\mathbf{I}_f\mathbf{O}_f(\mathbf{Q})\mathbf{D}_f^{\text{dest}}$ *enters* $\mathrm{R}(\mathbf{PQ})$ *through segment* $\mathbf{PQ}'$ *and leaves through segment* $\mathbf{Q}'\mathbf{P}$. *It stays with* 11α *of segment* $\mathbf{C}_f^{\text{orig}}\mathbf{C}_f^{\text{dest}}$.

Proof: The claim about entering and leaving are clear from the construction. In particular, we add $\mathbf{P}$ to the path when $\mathbf{I}_f$ lies on segment $\mathbf{PP}'$ instead of segment $\mathbf{PQ}'$, its nominal location. The previous section showed that the distance constraint holds for the path $\mathbf{D}_f^{\text{orig}}\mathbf{I}_f\mathbf{O}_f\mathbf{D}_f^{\text{dest}}$. We have to show that when we add $\mathbf{P}$, $|\delta(\mathbf{P}, \mathbf{C}_f^{\text{orig}}\mathbf{C}_f^{\text{dest}})| < 11\alpha$. First, we know that $\mathbf{P} \in A_f$ (Definition 5), and thus $\delta(\mathbf{P}, \mathbf{C}_f^{\text{orig}}\mathbf{C}_f^{\text{dest}}) >= -11\alpha$ (Lemma 1). We add $\mathbf{P}$ to the path when $\mathbf{I}_{f,x} = \mathbf{P}_x$ and $\mathbf{I}_{f,y} \geq \mathbf{P}_y$. For fixed x, δ is a monotonic function of y, and thus

$$\delta(\mathbf{P}, \mathbf{C}_f^{\text{orig}}\mathbf{C}_f^{\text{dest}}) \leq \delta(\mathbf{I}_f, \mathbf{C}_f^{\text{orig}}\mathbf{C}_f^{\text{dest}}) < 11\alpha.$$

Similarly, we can show that when $\mathbf{Q}$ is part of the path, $|\delta(\mathbf{Q}, \mathbf{C}_f^{\text{orig}}\mathbf{C}_f^{\text{dest}})| < 11\alpha$.

By switching the roles of x and y, we construct a path by $\mathbf{D}_g^{\text{orig}}(\mathbf{P})\mathbf{I}_g\mathbf{O}_g(\mathbf{Q})\mathbf{D}_g^{\text{dest}}$ that stays within 11α of segment $\mathbf{C}_g^{\text{orig}}\mathbf{C}_g^{\text{dest}}$ and that enters rectangle $\mathrm{R}(\mathbf{PQ})$ through segment $\mathbf{PP}'$ and leaves through segment $\mathbf{Q}'\mathbf{Q}$.

The two paths we created must intersect either inside or on the outside of rectangle $\mathrm{R}(\mathbf{PQ})$ because one enters through segment $\mathbf{PQ}'$ and leaves through segment $\mathbf{Q}'\mathbf{P}$ and the other enters through $\mathbf{PP}'$ and leaves through segment $\mathbf{Q}'\mathbf{Q}$. The construction of point $\mathbf{I}^{\text{approx}}$ is as in the previous section.

4.4.4 Curve-Segment Intersection

In Section 4.3.2 we expanded the notion of masc segment to include horizontal and vertical line segments. Therefore, for the sake of completeness, we must consider the possibility that one or both of the masc segments we are intersecting may be a horizontal or vertical line segment. This section shows how to detect and compute intersections in these cases.

The case when both are line segments is very simple. Let $\mathbf{D}_f^{\text{orig}}\mathbf{D}_f^{\text{dest}}$ be a horizontal masc segment (${\mathbf{D}_f^{\text{orig}}}_y = {\mathbf{D}_f^{\text{dest}}}_y$), and let $\mathbf{D}_g^{\text{orig}}\mathbf{D}_g^{\text{dest}}$ be a vertical masc segment (${\mathbf{D}_g^{\text{orig}}}_x = {\mathbf{D}_g^{\text{dest}}}_x$). These two segments intersect if

$${\mathbf{D}_g^{\text{orig}}}_y < {\mathbf{D}_f^{\text{orig}}}_y < {\mathbf{D}_g^{\text{dest}}}_y \qquad \text{and} \qquad {\mathbf{D}_f^{\text{orig}}}_x < {\mathbf{D}_g^{\text{orig}}}_x < {\mathbf{D}_f^{\text{dest}}}_x.$$

Their intersection is the point $\langle {\mathbf{D}_g^{\text{orig}}}_x, {\mathbf{D}_f^{\text{orig}}}_y\rangle$.

The case when only one masc is a horizontal or vertical line segment is almost as simple. Suppose that $\langle \mathbf{C}_f^{\text{orig}}\mathbf{C}_f^{\text{dest}}, \mathbf{D}_f^{\text{orig}}, \mathbf{D}_f^{\text{dest}}, A_f, B_f, f\rangle$ is a true masc segment and $\mathbf{D}_g^{\text{orig}}\mathbf{D}_g^{\text{dest}}$ is a horizontal or vertical line segment. The line segment intersects the masc segment if and only if $\mathbf{D}_g^{\mathrm{orig}} \in A_f$ and $\mathbf{D}_g^{\mathrm{dest}} \in B_f$ (or vice versa).

To compute the intersection, first compute the intersection between the line segment $\mathbf{D}_g^{\mathrm{orig}}\mathbf{D}_g^{\mathrm{dest}}$ and the axis-parallel rectangle $\mathrm{R}(\mathbf{C}_f^{\mathrm{orig}}\mathbf{C}_f^{\mathrm{dest}})$. The intersection will be a line segment $\mathbf{PQ}$. To compute $\mathbf{I}^{\mathrm{approx}}$, use the algorithm of Section 3.3.4 to compute the intersection between segment $\mathbf{C}_f^{\mathrm{orig}}\mathbf{C}_f^{\mathrm{dest}}$ and segment $\mathbf{PQ}$ using rounded arithmetic. It is easy to show that this segment lies within 11α of $\mathbf{C}_f^{\mathrm{orig}}\mathbf{C}_f^{\mathrm{dest}}$.

At this point we have described all the operations necessary to implement the masc segment intersection algorithm of Section 4.2.

5 Set Operations on Polygons

This section gives a strictly robust algorithm for performing set operations on planar regions bounded by line segments. As we know, it is not possible to apply rounded arithmetic to this problem directly. Therefore we will approximate the line segments by mascs. Fortunately, all the tricky numerical calculations involved in performing robust calculations on mascs have been done in the previous two sections. The algorithm described in this section has no numerical operations; it is purely symbolic.

It is easy to apply the operations of the previous section to the problem of modeling planar regions. In Figure 7 we see a rectangle $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4$ and a triangle $\mathbf{B}_1\mathbf{B}_2\mathbf{B}_3$. By approximating the line segments by masc segments and by applying the masc segment intersection algorithm, we obtain the four intersection points $\mathbf{I}_1$, $\mathbf{I}_2$, $\mathbf{I}_3$, and $\mathbf{I}_4$. Using information generated by the intersection algorithm, namely the above-sets and below-sets of the masc segments, it is possible to determine that masc segments[5] $\mathbf{I}_1\mathbf{I}_2$, $\mathbf{I}_2\mathbf{I}_3$, $\mathbf{I}_3\mathbf{I}_4$, and $\mathbf{I}_4\mathbf{I}_1$ should be removed to generate the set union of the polygonal regions as shown on the right. This section gives an algorithm for determining which segments to remove.

5.1 Polygon Definitions

This section defines polygons and polygonal regions. These can be bounded by any continuous simple curve segments, but for the purposes of this paper, they will be bounded by masc segments. We disallow the regions as on the left of Figure 8 in which two segments are intersecting other than at an endpoint. However, we do allow a regions as shown on the right in which vertices have degree higher than two.

Definition 6 (Graph of Segment Set) *Let M be a set of masc segments adapted to a common universe U of sites (Definition 4). Define* $\mathrm{Graph}(M)$ *to be the graph whose vertices are the sites in U and whose edges are the segments in M.*

Theorem 3 *Let M be a set of masc segments adapted to a common universe U of sites. If* $\mathrm{Graph}(M)$ *is the planar embedding of a planar graph (meaning that segments in M intersect only at their endpoints) and if each vertex in* $\mathrm{Graph}(M)$ *has even degree, then the union of the segments in M is the boundary of a closed region of the plane.*

Proof: It is a standard result of graph theory that the set of faces of an embedding of a planar graph is two-colorable, if every vertex has even degree. The union of the set of faces with one color is a region of the plane whose boundary is M.

Corollary 1 *If M is the boundary of one closed planar region, then it is also the boundary of exactly two closed regions, one bounded, the other unbounded.*

Definition 7 *A set M of masc segments is a* polygon *if it satisfies the conditions of Theorem 3. In this case, M is the boundary of two* polygonal regions*, one unbounded and one bounded.*

[5] In the following discussion, we denote masc segments by their endpoints.

One can represented a polygonal region by a set P of mascs which bounds it plus a single bit to indicate whether the region is bounded or unbounded.

5.2 Unions of Bounded Polygonal Regions

Let P and Q be polygons. The following two sections give an algorithm for computing the union of the bounded polygonal regions defined by P and Q. Section 5.2.3 extends the algorithm to perform any set operation on any two of the polygonal regions defined by P and Q.

5.2.1 Comparing Masc Segments

The concept of *evidence* introduced in Section 4.4.1 (page 14) allows us to determine information about the relative positions of two masc segments merely by examining their above and below sets. This fact is formalized in the following definitions.

Definition 8 (Comparing Masc Segments) *Let* $\langle \mathbf{C}_f^{\mathrm{orig}} \mathbf{C}_f^{\mathrm{dest}}, \mathbf{D}_f^{\mathrm{orig}}, \mathbf{D}_f^{\mathrm{dest}}, A_f, B_f, f\rangle$ *and* $\langle \mathbf{C}_g^{\mathrm{orig}} \mathbf{C}_g^{\mathrm{dest}}, \mathbf{D}_g^{\mathrm{orig}}, \mathbf{D}_g^{\mathrm{dest}}, A_g, B_g, g\rangle$ *be masc segments adapted to the same universe such that* $\gamma_f = \langle x, f(x)\rangle$ *and* $\gamma_g = \langle x, g(x)\rangle$ *either have disjoint interiors or are identical (they intersect only at their endpoints) and such that*

$$(\mathbf{D}_f^{\mathrm{orig}}{}_x, \mathbf{D}_f^{\mathrm{dest}}{}_x) \cap (\mathbf{D}_g^{\mathrm{orig}}{}_x, \mathbf{D}_g^{\mathrm{dest}}{}_x) \neq \emptyset.$$

There are three possibilities:

- $\mathbf{D}_f^{\mathrm{orig}} = \mathbf{D}_g^{\mathrm{orig}}$, $\mathbf{D}_f^{\mathrm{dest}} = \mathbf{D}_g^{\mathrm{dest}}$, $A_f = A_g$, *and* $B_f = B_g$, *in which case we say* $\gamma_f = \gamma_g$*;*
- $\exists \mathbf{P} \in U$ *that is evidence (Definition 5) that* $f(\mathbf{P}_x) > g(\mathbf{P}_x)$, *in which case we say* $\gamma_f > \gamma_g$*;*
- $\exists \mathbf{P} \in U$ *that is evidence that* $f(\mathbf{P}_x) < g(\mathbf{P}_x)$, *in which case we say* $\gamma_f < \gamma_g$.

Definition 9 *Let* P *be a polygon with universe* U. *We say a value* X *is in* general position *if it does not equal the x-coordinate of any site in* U. *For each* X *in general position, define* $P(X)$ *to be the set of masc segments* $\langle \mathbf{C}_f^{\mathrm{orig}} \mathbf{C}_f^{\mathrm{dest}}, \mathbf{D}_f^{\mathrm{orig}}, \mathbf{D}_f^{\mathrm{dest}}, A_f, B_f, f\rangle \in P$ *such that* $X \in (\mathbf{D}_f^{\mathrm{orig}}{}_x, \mathbf{D}_f^{\mathrm{dest}}{}_x)$ *(*X *is in the x-interval of the segment).*

Definition 10 *Let* P *be a polygon with universe* U, *and let* γ *be a masc segment with the same universe that does not cross any* $\gamma_f \in P$ *(*γ *may or may not be an element of* P*). Let* X *be a real value in general position in the x-interval of* γ.

The index *of* γ *in* $P(X)$, *denoted* $\mathrm{Index}(\gamma, P(X))$, *is the number of curves* $\gamma_f \in P(X)$ *such that* $\gamma < \gamma_f$ *under Definition 8.*

One can think of the index as follows. Take the point on γ with x-coordinate X and cast a ray in the vertical (positive y) direction. The number of segments of P that this ray crosses is $\mathrm{Index}(\gamma, P(X))$. However, as Definition 8 indicates, the index can be computed using purely symbolic operations, and we consider it to be a function available to the union algorithm below.

For any masc γ, the value of $\mathrm{Index}(\gamma, P(X))$ varies with X. However, if γ crosses no segment of P, the *parity* (evenness or oddness) of $\mathrm{Index}(\gamma, P(X))$ is independent of X. This is easy to show from the fact that every vertex in $\mathrm{Graph}(P)$ has even degree.

5.2.2 Union Algorithm

This section gives the algorithm for taking the union of bounded polygonal regions defined by polygons P and Q. This consists of several steps. First, use the intersection algorithm of Section 4 to compute all intersections among the masc segments in $P \cup Q$. This results in new sets P' and Q' of masc segments. Second, "clean up" P' and Q' by eliminating multiple occurrences of masc segments in each of these sets. Finally, eliminate more segments from $R = P' \cup Q'$ to generate the boundary of the union of the bounded polygonal regions defined by P and Q.

Intersection: During the intersection stage, each segment in P is split at the points of intersection with segments in Q using the algorithms described in Sections 4.4.2 and 4.4.3. The resulting sets of segments P' and Q' are adapted to the same universe.

Cleaning: All or part of two distinct segments in P may be "pinched" together to result in multiple identical segments appearing in P'. Fortunately, as Definition 8 indicates, we can detect identical segments by comparing endpoints, above sets, and below sets. For each class of equal segments in P', the algorithm *cleans* the class by deleting the largest possible even number of segments from the class: the result is zero or one segment depending on whether the class had an even or odd number of identical segments, respectively. The same procedure is applied to Q'. Once P' and Q' have been *cleaned* this manner, they satisfy Definition 7.

Marking Embedded Segments: Now, for each $\gamma_g \in Q'$ which is not a vertical line segment and which is not identical to a segment in P', choose $X \in (\mathbf{D}^{\mathrm{orig}}_{g\ \ x}, \mathbf{D}^{\mathrm{dest}}_{g\ \ x})$ in general position. If $\mathrm{Index}(\gamma_g, P'(X))$ is odd, mark γ_g for deletion. Do the same for each $\gamma_f \in P'$ with respect to Q'.

Marking Identical Segments: For each $\gamma_f \in P'$ and $\gamma_g \in Q'$ such that $\gamma_f = \gamma_g$, choose $X \in (\mathbf{D}^{\mathrm{orig}}_{f\ \ x}, \mathbf{D}^{\mathrm{dest}}_{f\ \ x})$ (which equals $(\mathbf{D}^{\mathrm{orig}}_{g\ \ x}, \mathbf{D}^{\mathrm{dest}}_{g\ \ x})$) in general position. If $\mathrm{Index}(\gamma_f, P'(X))$ and $\mathrm{Index}(\gamma_g, Q'(X))$ have opposite parity (one odd and the other even), mark both for deletion; otherwise, mark one of them for deletion.

Putting in Vertical Segments Delete all marked segments from $R = P' \cup Q'$. The set R is almost a polygon. We just have to deal with vertical line segments. Remove all vertical line segments from R. For each X that is *not* in general position, let $\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_{n(X)}$ be the list of vertices with x-coordinate X, sorted by y-coordinate

$$\mathbf{D}_{1,y} < \mathbf{D}_{2,y} < \cdots < \mathbf{D}_{n(X),y}.$$

Apply the following algorithm to R:

for $i = 1$ to $d - 1$
    if D_i has odd degree in Graph(R) then
        add segment $D_i D_{i+1}$ to R.

This completes the algorithm for the union of two bounded polygonal regions.

5.2.3 General Set Operations

The algorithm in the previous section only works for taking the union of bounded polygonal regions. The general algorithm is based on the following observation. Define special values $-\infty$ and $+\infty$ which are less than and greater than every other value, respectively. If P is a polygon, then the unbounded region defined by P is the bounded region defined by

$$P \cup \left\{ \begin{array}{ll} \langle -\infty, -\infty \rangle \langle +\infty, -\infty \rangle, & \langle +\infty, -\infty \rangle \langle +\infty, +\infty \rangle, \\ \langle +\infty, +\infty \rangle \langle -\infty, +\infty \rangle, & \langle -\infty, +\infty \rangle \langle -\infty, -\infty \rangle \end{array} \right\},$$

which is the union of P with the "square at infinity." We can use this observation to take the complement of any region and to take the union of any two unbounded regions. All set operations can be reduced to set complement and union. Adding or removing the square at infinity requires only constant cost, and thus any binary set operation, such as intersection or difference, can be computed as fast as the union.

5.3 Cleaning Up

The algorithm of the previous section is strictly robust, but it may generate unexpected answers in certain pathological cases. This section describes a more complex clean-up algorithm than the one described in

Section 5.2.2. The clean up algorithm in that section removed multiple occurrences of masc segments in P' and Q', where P' and Q' are the result of finding all intersections among the segments of two polygons P and Q. The clean up algorithm in this section removes or alters other masc segments which correspond to "non-physical" situations.[6]

Figure 7 (page 30) shows the nominal intersection of a rectangle $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4$ and a triangle $\mathbf{B}_1\mathbf{B}_2\mathbf{B}_3$. In pathological cases, which can only occur if segments $\mathbf{A}_1\mathbf{A}_2$ and $\mathbf{A}_3\mathbf{A}_4$ lie within a small multiple of α, segments $\mathbf{A}_1\mathbf{A}_2$ and $\mathbf{A}_3\mathbf{A}_4$ might intersect. In Figure 9, the masc segment intersection algorithm has generated two intersection points $\mathbf{I}_5$ and $\mathbf{I}_6$ for these segments. The algorithm of Section 5.2.2 generates the union of these two regions as shown on the right.

This union is somewhat non-physical in appearance. If the original segments $\mathbf{A}_1\mathbf{A}_2$ and $\mathbf{A}_3\mathbf{A}_4$ meet, they should "cancel" each other, as shown on the left of Figure 10. This corresponds more closely to a notion of a rectangle of physical material which is drying up like a puddle. We describe here an algorithm for this type of *physical cleaning.* The resulting union is shown on the right.

In the notation of Section 5.2.2,

$$\begin{aligned} P &= \{\mathbf{A}_1\mathbf{A}_2, \mathbf{A}_2\mathbf{A}_3, \mathbf{A}_3\mathbf{A}_4, \mathbf{A}_4\mathbf{A}_1\}, \\ P' &= \{\mathbf{A}_1\mathbf{I}_5, \mathbf{I}_5\mathbf{I}_1, \mathbf{I}_1\mathbf{I}_6, \mathbf{I}_6\mathbf{I}_3, \mathbf{I}_3\mathbf{A}_2, \mathbf{A}_2\mathbf{A}_3, \mathbf{A}_3\mathbf{I}_4, \mathbf{I}_4\mathbf{I}_6, \mathbf{I}_6\mathbf{I}_2, \mathbf{I}_2\mathbf{I}_5, \mathbf{I}_5\mathbf{A}_4, \mathbf{A}_4\mathbf{A}_1\}. \end{aligned}$$

We presume that the records for segments $\mathbf{I}_1\mathbf{I}_6$ and $\mathbf{I}_4\mathbf{A}_3$ in P' contain pointers back to the original segments $\mathbf{A}_1\mathbf{A}_2$ and $\mathbf{A}_3\mathbf{A}_4$, respectively, which spawned them. For each value X in general position (Definition 10) such that $X \in (\mathbf{I}_{1,x}, \mathbf{I}_{6,x})$, we define $\mathrm{OldIndex}(\mathbf{I}_1\mathbf{I}_6, P(X))$ to be the index of segment $\mathbf{A}_1\mathbf{A}_2$ in $P(X)$. For the example shown, $\mathrm{OldIndex}(\mathbf{I}_1\mathbf{I}_6, P(X)) = 1$ (independent of X). Note, however that the index of $\mathbf{I}_1\mathbf{I}_6$ in $P'(X)$ is 2. By looking at the old and new indices, we can determine which curves should be removed from P'.

Here is the general algorithm applied to P and P'. It should also be applied to Q and Q', in both cases as a substitute for the original cleaning algorithm.

For each X in general position,[7] do the following. First, assign indices to the curves in $P'(X)$ in a manner consistent with the "$<$" relation:

$$P'(X) = \{\gamma_1, \gamma_2, \ldots, \gamma_{n(X)}\}$$

such that

$$\forall 1 \leq i < j \leq n(X),\ \gamma_i < \gamma_j \text{ or } \gamma_i = \gamma_j.$$

Next, assign each $\gamma_i \in P'(X)$ a *reference count* of zero. For each $i = 1, 2, \ldots, n(X)$ do the following: let $j \geq i$ be the largest index such that $\mathrm{OldIndex}(\gamma_j, P(X)) \leq \mathrm{OldIndex}(\gamma_i, P(X))$. Increment the reference count of γ_j. If $j \neq i$, mark γ_i for deletion. After performing this operation for each i, mark for deletion every curve with an even reference count. After performing this operation for all X, remove all marked curves from P'. Finally, apply the original clean up function of Section 5.2.2 to P' to eliminate multiple identical segments.

Thus, in Figure 10, segment $\mathbf{I}_2\mathbf{I}_6$ is removed because its index is less than that of $\mathbf{I}_1\mathbf{I}_6$ and its old index is greater. Segment $\mathbf{I}_1\mathbf{I}_6$ is removed because it ends up having a reference count equal to two. Figures 11 and 12 show the result of the cleaning algorithm on other pathological cases.

6 Conclusion

In this paper, we have considered the problem of constructing unions and intersections of polygonal regions in the plane using rounded finite precision arithmetic. Section 2 determined that the best we can hope to accomplish is to perform set operations on regions bounded by curves, not necessarily straight line segments, because straightness is a property not detectable by a polynomial-time rounded arithmetic algorithm.

[6] This is the author's opinion. Others who have viewed Figures 9 through 12 have perceived the output of the previous algorithm to be more "physical."

[7] It suffices to look at a set of X values which covers the set of segments.

Sections 3 through 5 give a strictly robust algorithm for performing the desired set operations, and this algorithm generates the best output possible: regions bounded by monotonic curves which are straight to within a small multiple of the rounding unit of the machine arithmetic.

This final section summarizes our robust polygon set operation algorithm. Section 6.1 gives reasons why the output of the robust algorithm is practically applicable even though it replaces straight line segments with curves. Section 6.2 describes techniques which might be used to implement the robust algorithm efficiently. Finally, Sections 6.3 and 6.4 discuss the *comparison principle* and the *hidden variable method*, techniques used by our algorithm, and how they might be applied in other geometric domains.

6.1 Practical Applicability of the Robust Algorithm

The robust algorithm in Sections 3 through 5 can be used by applications which require set operations on polygonal regions. Section 5 uses the example of the union of a rectangle and a triangle. The robust algorithm generates a good approximation to the correct polygon, an approximate polygon bounded by masc segments, and this approximate polygon can be used as input to the same algorithm. All numerical calculations used by the robust algorithm are performed using rounded B-bit arithmetic, and no matter how many operations are performed, the error bound depends only on B, not on the number of operations.

We can describe the accuracy of the robust algorithm in terms of backwards error analysis. The approximate polygon it generates can be *thought of* as being generated by two steps:

- replace each line segment of the input polygons by monotonic curves which do not stray farther than $\beta = 66\sqrt{2}\mu$ distant from the segments they replace ($\mu = 2^{-B}\mathrm{M}$ is the absolute rounding unit);
- perform the desired set operation on the altered inputs *exactly*.

If the output becomes the input to another set operation, no additional error is added.

The "thought-construction" shows that the output of the robust algorithm is indeed practically applicable. Let us suppose the input polygons are known only to a certain accuracy β' representing B' bits of precision ($\beta' = 2^{-B'}\mathrm{M}$), as is generally the case in any physical application ($B' = 20$ corresponds to one part in a million accuracy). In other words, the actual shapes of the input polygons may differ by as much as β' from the shapes we perceive them to have. If B is sufficiently large, $B > B' + 7$, so that $\beta < \beta'$, then the first step in our "thought-construction" above may introduce no error at all: it is possible the β-perturbation of the input is in fact the actual shape of the input, and therefore the output of the robust algorithm is the correct polygon. Of course, it is also possible that the robust algorithm generates an incorrect solution, but the same holds true for an exact algorithm using infinite precision arithmetic: it too may or may not generate the correct polygon. The important observation is that the robust solution and the exact solution are equally valid from the point of view of practical applications.

Since the robust solution requires less precision to compute, we should use it instead of the exact solution which requires arithmetic with more than $4B'$ bits of precision. In particular, the robust algorithm requires only 27-bit arithmetic which is easily provided by standard 53-bit floating point arithmetic. The exact algorithm requires more than 80-bit arithmetic, which would generally have to be implemented in software.

6.2 Implementation of Robust Polygon Set Operations

Sections 3 through 5 do not immediately lead to a computer program for performing set operations on polygonal regions. Instead, they reduce the difficult problem of reasoning about round-off error to a few basic operations on sets of points in the plane. There are well-understood techniques for implementing these operations, and it remains to discover which one leads to the most practically efficient program. We can only briefly summarize here one technique, a sweepline construction, and discuss ways in which it can be made efficient.

We need a method for maintaining the sets and lists necessary for implementing the robust segment intersection algorithm. We describe a modification of the Bentley-Ottmann [1] sweepline technique to demonstrate how the robustness techniques can be added to a commonly used algorithm. Similar adaptions can be made to other techniques, such as bucketing.

In the sweepline algorithm, we sweep a vertical line from left to right, putting anticipated intersections into a priority queue. Above and below lists are represented implicitly in the current state.

For each segment $\mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}$ with positive slope, we must save a *highest below-site* $\mathbf{B}$, the site in the below set with the greatest y-value. (Analogously, we save the *lowest above-site* for each segment with negative slope.) This is the only extra information needed over that provided by the Bentley-Ottmann algorithm. Maintaining the highest below-site for each $\mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}$ is potentially expensive because every time we add a new site, it may become the highest below-site for a large number of segments. However, we can employ the following efficiency technique. If $\delta(\mathbf{B}, \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}})_B < -12\alpha$ (recall that the subscript B refers to the use of rounded B-bit arithmetic), then $\delta(\mathbf{B}, \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}) < -11\alpha = \beta$ (see Definition 3 on page 10), and thus it cannot affect the shape of the masc segment $\mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}$. Therefore we can safely leave $\mathbf{B}$ out of the below-set of $\mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}$, considerably reducing the cost of maintaining the below-set.

This efficiency technique works well unless many sites are near segments, a case which generally causes non-robust algorithms to fail. Difficult or pathological cases may increase the running time of our robust algorithm, but they do not affect its reliability.

6.3 New Techniques

Our robust polygon intersection algorithm is based on a number of new techniques, in particular a *comparison principle* and a *hidden variable method.* The *comparison* principle is the observation that comparisons of representable floating point numbers can be carried out without round-off error. The *hidden variable* method is based on the observation that by keeping certain portions of a geometric representation implicit or *hidden*, it is generally possible to greatly improve the error bounds.

We heavily use the comparison principle in the design of the low-level numerical operations on points and line segments in Section 3. Definition 2, classifying a point with respect to a line segment, is based on the assumption that we can classify a point with respect to an axis-parallel rectangle. The algorithms for intersecting a segment with a horizontal or vertical line or line segment (Sections 3.3.1 and 3.3.2) require us to move a point to the nearest endpoint of a line segment. This operation can be performed using comparisons alone if the segment is parallel to a coordinate axis. The general algorithm for intersecting two line segments (Section 3.3.3) requires us to move a point to the nearest point on the boundary of an axis-parallel rectangle. This operation can also be performed using comparisons alone.

A number of the operations in the higher level segment intersection of Section 4 also depend on the comparison principle. In particular, the update procedure for masc segments (Section 4.3.1) and the technique for detecting intersections of masc segments (Section 4.4.1) depend on comparisons of floating point numbers. When we compute an intersection point (Sections 4.4.2 and 4.4.3), we use comparisons to determine if it lies on the boundary of an axis-parallel rectangle, and we use additional comparisons to compute its location if it does.

The *hidden variable* method is closely related to the comparison principle. Theorem 1, the first hidden variable theorem, implies the existence of a masc function if certain premises are satisfied. Most of these premises are based on comparisons of floating point numbers, and all of these premises are testable using rounded B-bit arithmetic. Actually constructing the curve explicitly would require much higher precision. In a similar fashion, Theorem 2, the second hidden variable theorem, implies the existence of an intersection point based on testable premises, again without requiring that it be computed explicitly. These two theorems make it possible to work with implicit curves, which in turn makes it possible to bound the maximum error of the intersection algorithm by a constant multiple of the rounding unit. Explicit representation of the curves, as described in Section 4.3.3, is possible, but it does not have a good provable accuracy bound.

6.4 Future Work

We have already shown that the hidden variable method leads to strictly robust algorithms for computing intersections of algebraic curves in two dimensions [14] and planes in three dimensions [13]. Both this author [17] and Fortune [5] have devised strictly robust algorithms for constructing convex hulls of points in the plane. These algorithms rely heavily on the comparison principle.

In the immediate future, we plan to extend our work to the intersection of spline curves in the plane and to the construction of convex hulls in higher dimensions. In the case of spline curves, we plan to develop a hidden variable theory that allows us to work with curves with unspecified control points. By doing so, we hope to obtain the same high accuracy we have obtained for the intersection of line segments.

Beyond that point, we plan to work on developing robust algorithms for the domains of polyhedra and objects bounded by curved surfaces. These pose difficult problems, but we hope that the techniques we have devised for the two dimensional domain will generalize to three dimensions.

A Error Analysis: Numerical Operations on Segments

This appendix proves error bounds for the operations given in Section 3 when they are implemented using arithmetic with B bits of precision (Section 2.5).

A.1 Point-Segment Distance and Classification

Section 3.1.2 gives a formula for $\delta(\mathbf{C}, \mathbf{AB})$, the signed distance from a point $\mathbf{C}$ to a line segment $\mathbf{AB}$. As a precondition, $\mathbf{C}$ lies inside the bounding box $\mathrm{R}(\mathbf{AB})$. The claimed error bound is $|\delta(\mathbf{C}, \mathbf{AB})_B - \delta(\mathbf{C}, \mathbf{AB})| \leq \alpha$, where $\delta(\mathbf{C}, \mathbf{AB})_B$ is the value of $\delta(\mathbf{C}, \mathbf{AB})$ computed using rounded B-bit arithmetic.

The error arising from the circulation computation dominates. Using elementary error analysis and the Cauchy inequality, one can show,

$$\begin{aligned} |\mathrm{circ}(\mathbf{A}, \mathbf{B}, \mathbf{C})_B - \mathrm{circ}(\mathbf{A}, \mathbf{B}, \mathbf{C})| &= |((\mathbf{A}-\mathbf{C}) \times (\mathbf{B}-\mathbf{C}))_B - (\mathbf{A}-\mathbf{C}) \times (\mathbf{B}-\mathbf{C})| && (1) \\ &\leq 3\epsilon|\mathbf{A}-\mathbf{C}|\,|\mathbf{B}-\mathbf{C}| + \epsilon|(\mathbf{A}-\mathbf{C}) \times (\mathbf{B}-\mathbf{C})|, && (2) \end{aligned}$$

where $\epsilon = 2^{-B}$. For $\mathbf{C}$ close to $\mathbf{AB}$, the second term can be neglected, and thus

$$|\delta(\mathbf{C}, \mathbf{AB})_B - \delta(\mathbf{C}, \mathbf{AB})| \leq 3\epsilon \frac{|\mathbf{A}-\mathbf{C}|\,|\mathbf{B}-\mathbf{C}|}{|\mathbf{A}-\mathbf{B}|}.$$

For $\mathbf{C} \in \mathrm{R}(\mathbf{AB})$,

$$|\mathbf{A}-\mathbf{C}|,\ |\mathbf{B}-\mathbf{C}| \leq |\mathbf{C}-\mathbf{A}|,$$

and therefore,

$$|\delta(\mathbf{C}, \mathbf{AB})_B - \delta(\mathbf{C}, \mathbf{AB})| \leq 3\epsilon|\mathbf{C}-\mathbf{A}| \leq 6\sqrt{2}\epsilon M,$$

where M is an upper bound on the magnitude of any coordinate.

A.2 Intersections

A.2.1 Intersection with Horizontal or Vertical Line

Section 3.3.1 gives a formula for the intersection of a line segment $\mathbf{AB}$ with a horizontal line $Y = c$. Error analysis[8] shows that if we use rounded arithmetic to compute X,

$$|X_B - X| \leq \epsilon|X| + 5\epsilon \left| \frac{\mathbf{B}_x - \mathbf{A}_x}{\mathbf{B}_y - \mathbf{A}_y}(c - \mathbf{A}_y) \right|.$$

[8] For example, the factor of 5ϵ arises from the three subtractions, one division, and one multiplication.

If X_B lies outside the interval $[\mathbf{A}_x, \mathbf{B}_x]$, then we move it to the nearest endpoint. This will always decrease the error.

For fixed y-coordinate, a unit change in x changes the distance to line **AB** by $(\mathbf{B}_y - \mathbf{A}_y)/|\mathbf{B}-\mathbf{A}|$, which is always less than unity. Therefore,

$$|\delta(\langle X_B, Y_B\rangle, \mathbf{AB})| \leq \epsilon |X| \frac{|\mathbf{B}_y - \mathbf{A}_y|}{|\mathbf{B}-\mathbf{A}|} + 5\epsilon |c - \mathbf{A}_y| \frac{|\mathbf{B}_x - \mathbf{A}_x|}{|\mathbf{B}-\mathbf{A}|} \leq 11\epsilon \mathrm{M} < 2\alpha.$$

A.2.2 Intersection with a Line Segment

Let **AB** and **CD** be line segments such that we know, *a priori*, that they intersect. Section 3.3.4 gives a procedure for computing this intersection **I** approximately using rounded arithmetic. The resulting approximate intersection point $\mathbf{I}_B$ lies in $\mathrm{R}(\mathbf{AB}) \cap \mathrm{R}(\mathbf{CD})$.

We have,

$$t_B = \left(\frac{(\mathbf{C}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C})}{(\mathbf{B}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C})}\right)_B.$$

A single division introduces at most a relative error of size ϵ,

$$\left| t_B - \frac{((\mathbf{C}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}))_B}{((\mathbf{B}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}))_B} \right| \leq \epsilon \left| \frac{((\mathbf{C}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}))_B}{((\mathbf{B}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}))_B} \right|,$$

and hence,

$$|t_B((\mathbf{B}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}))_B - ((\mathbf{C}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}))_B| \leq \epsilon |((\mathbf{C}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}))_B| \leq \epsilon |\mathbf{C}-\mathbf{A}|\,|\mathbf{D}-\mathbf{C}|.$$

We can write this as $\exists \epsilon_1 \in [-\epsilon, \epsilon]$ such that

$$t_B((\mathbf{B}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}))_B - ((\mathbf{C}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}))_B = \epsilon_1 |\mathbf{C}-\mathbf{A}|\,|\mathbf{D}-\mathbf{C}|. \tag{3}$$

From Equation 2, it follows that $\exists \epsilon_2, \epsilon_3 \in [-\epsilon, \epsilon]$ such that

$$(\mathbf{B}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}) - ((\mathbf{B}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}))_B = 4\epsilon_2 |\mathbf{B}-\mathbf{A}|\,|\mathbf{D}-\mathbf{C}| \tag{4}$$

$$(\mathbf{C}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}) - ((\mathbf{C}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}))_B = 4\epsilon_3 |\mathbf{C}-\mathbf{A}|\,|\mathbf{D}-\mathbf{C}| \tag{5}$$

Combining Equations 3, 4, and 5,

$$\begin{aligned}
\delta(\mathbf{A} + t_B(\mathbf{B}-\mathbf{A}), \mathbf{CD}) &= \frac{t_B(\mathbf{B}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C}) - (\mathbf{C}-\mathbf{A}) \times (\mathbf{D}-\mathbf{C})}{|\mathbf{D}-\mathbf{C}|} \\
&= \frac{\epsilon_1 |\mathbf{C}-\mathbf{A}|\,|\mathbf{D}-\mathbf{C}| + t_B(4\epsilon_2 |\mathbf{B}-\mathbf{A}|\,|\mathbf{D}-\mathbf{C}|) - 4\epsilon_3 |\mathbf{C}-\mathbf{A}|\,|\mathbf{D}-\mathbf{C}|}{|\mathbf{D}-\mathbf{C}|} \\
&= \epsilon_1 |\mathbf{C}-\mathbf{A}| + 4t_B\epsilon_2 |\mathbf{B}-\mathbf{A}| - 4\epsilon_3 |\mathbf{C}-\mathbf{A}|. \qquad (6)
\end{aligned}$$

Thus,

$$|\delta(\mathbf{A} + t_B(\mathbf{B}-\mathbf{A}), \mathbf{CD})| \leq \epsilon(|\mathbf{C}-\mathbf{A}| + 4|\mathbf{B}-\mathbf{A}| + 4|\mathbf{C}-\mathbf{A}|) \leq 9\epsilon(2\sqrt{2}\mathrm{M}) = 3\alpha.$$

The previous paragraph derived a bound on the distance from $\mathbf{A} + t_B(\mathbf{B}-\mathbf{A})$ to **CD**. However, $\mathbf{I}_B$ equals $(\mathbf{A} + t_B(\mathbf{B}-\mathbf{A}))_B$, which has the additional round-off error introduced by a subtraction, a multiplication, and an addition for each coordinate. We have,

$$|\mathbf{I}_B - (\mathbf{A} + t_B(\mathbf{B}-\mathbf{A}))| \leq \epsilon(|\mathbf{I}_B| + 2|t_B(\mathbf{B}-\mathbf{A})|) \leq \epsilon(\mathrm{M}\sqrt{2} + (4\sqrt{2})\mathrm{M}) < \alpha.$$

Combining this bound with the bound in the previous paragraph,

$$|\delta(\mathbf{I}_B, \mathbf{AB})| < \alpha \qquad \text{and} \qquad |\delta(\mathbf{I}_B, \mathbf{CD})| < 4\alpha.$$

As stated above, even though $\mathbf{I}_B$ lies close to each line segment **AB** and **CD**, it may not lie in the bounding boxes of these segments. In this case we simply move it to the nearest point on the boundary of $\mathrm{R}(\mathbf{AB}) \cap \mathrm{R}(\mathbf{CD})$. We claim that we move $\mathbf{I}_B$ at most 5α. To prove this claim we need the following lemma.

Lemma 4 *Let* $\mathbf{R}_1$ *and* $\mathbf{R}_2$ *be two axis-parallel rectangles such that* $\mathbf{R}_1 \cap \mathbf{R}_2 \neq \emptyset$*. Let* $\mathbf{P}$ *be any point outside* $\mathbf{R}_1$*. Moving* $\mathbf{P}$ *directly towards* $\mathbf{R}_1$ *(see Figure 13) does not increase its distance from* $\mathbf{R}_2$*.*

Proof: There are several cases to consider, but basically, if moving $\mathbf{P}$ directly towards $\mathbf{R}_1$ increases its distance from $\mathbf{R}_2$, then either the line $x = \mathbf{P}_x$ or the line $y = \mathbf{P}_y$ separates $\mathbf{R}_1$ from $\mathbf{R}_2$, contradicting the premise that their intersection is non-empty. Incidentally, this lemma is *not* true for arbitrary (non-axis-parallel) rectangles.

Now we can establish the claim made earlier. Since $\mathbf{I}_B$ is at most α from $\mathbf{AB}$ and 4α from $\mathbf{BC}$, it is at most α distant from R($\mathbf{AB}$) and 4α from R($\mathbf{BC}$). These two axis-parallel rectangles have a non-empty intersection because of the premise that $\mathbf{AB}$ and $\mathbf{CD}$ intersect. Therefore, we can move $\mathbf{I}_B$ to R($\mathbf{AB}$) without increasing its distance from R($\mathbf{BC}$). Then we can move $\mathbf{I}_B$ to R($\mathbf{CD}$) without increasing its distance from R($\mathbf{AB}$) (which is zero). Total distance moved is at most 5α. After we have altered the position of $\mathbf{I}_B$, it satisfies

$$|\delta(\mathbf{I}_B, \mathbf{AB})| < 6\alpha \qquad \text{and} \qquad |\delta(\mathbf{I}_B, \mathbf{CD})| < 9\alpha.$$

B Masc Segments

This appendix gives proofs of the theorems of Section 4.

B.1 Proof of Lemma 1

These properties listed in Lemma 1 are easy to establish. The first is a consequence of the monotonicity of f, and Figure 14 illustrates what happens when sites $\mathbf{A}$ and $\mathbf{B}$ violate the monotonicity property. The second property is a consequence of the bound on the distance from $\langle x, f(x)\rangle$ to $\mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}$.

B.2 Proof of First Hidden Variable Theorem

This section gives an informal proof of Theorem 1. Let $\Delta > 0$ be less than the minimum non-zero difference between elements of the set of x-coordinates and y-coordinates of sites in the above-set A and in the below-set B. If necessary, diminish Δ so that it is smaller than both

$$\beta + \min_{\mathbf{A}\in A} \delta(\mathbf{A}, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}) \qquad \text{and} \qquad \beta - \max_{\mathbf{B}\in B} \delta(\mathbf{B}, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}).$$

Imagine a tiny test driver who must drive a course from $\mathbf{D}^{\text{orig}}$ to $\mathbf{D}^{\text{dest}}$. At each site $\mathbf{A} \in A$ and $\mathbf{B} \in B$, we place a traffic cone of radius $\Delta/3$. The driver must drive a course that keeps every A-cone to his left and every B-cone to his right. Suppose the driver takes the shortest course which satisfies these conditions. We notice that he will only turn left at A-cones and right at B-cones. Figure 15 illustrates that a course that turns left at a B-cone can be made shorter. We claim that the course, $\langle x, f(x)\rangle$, has exactly the properties we need.

We first show that the y-coordinate of the course is monotonic. Assume without loss of generality that ${\mathbf{D}^{\text{orig}}}_y < {\mathbf{D}^{\text{dest}}}_y$ ($f(x)$ should be an increasing function) and that the course has a segment that is non-increasing in y. Figure 16 depicts a string of such segments. Eventually the course must turn left to get to $\mathbf{D}^{\text{dest}}$. There is therefore a non-increasing segment starting from a right turn at some $\mathbf{B} \in B$ and ending with a left turn at some $\mathbf{A} \in A$. Thus $\mathbf{A}_x \geq \mathbf{B}_x$ and $\mathbf{A}_y \leq \mathbf{B}_y$. But this contradicts the assumption that $\mathbf{A}$ and $\mathbf{B}$ satisfy the conclusion of Lemma 1.

Now we have to show that every point of the course lies within β of $\mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}$. Let $\mathbf{T}$ be a point on the course which is the maximum distance to the left of segment $\mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}$ (such that $\delta(\mathbf{T}, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}})$ is maximal). Let us assume the contrary of the error bound, that $\delta(\mathbf{T}, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}) \geq \beta$. Because $\mathbf{D}^{\text{orig}}$ and $\mathbf{D}^{\text{dest}}$ satisfy the conclusion of the Lemma, $\mathbf{T}$ cannot equal either of these sites. Therefore $\mathbf{T}$ must be a right turn in the course, and therefore it must lie on the boundary of an B-cone as shown in Figure 17. But this implies that,

$$\delta(\mathbf{T}, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}) = \Delta/3 + \delta(\mathbf{A}, \mathbf{C}^{\text{orig}}\mathbf{C}^{\text{dest}}) < \beta,$$

where $\mathbf{B} \in B$ is the center of the cone. This contradicts our assumption. Therefore the course does not stray farther than β to the left of line segment $\mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}$. Similarly, the course does not stray farther than β to the right either.

B.3 Proof of Lemma 2

This section proves Lemma 2 of Section 4.3.2. The proof for Special Case 1 (page 13) is as follows: There are two conditions that $\mathbf{A}^{\mathrm{block}}$ must satisfy in order to permit successful splitting of $\gamma = \langle \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}, \mathbf{D}^{\mathrm{orig}}, \mathbf{D}^{\mathrm{dest}}, A, B, f\rangle$ at $\mathbf{A}^{\mathrm{block}}$. First, $\delta(\mathbf{A}^{\mathrm{block}}, \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}})$ must be bounded by 11α. Second, no site may block $\mathbf{A}^{\mathrm{block}}$ splitting γ in the way that $\mathbf{A}^{\mathrm{block}}$ prevented $\mathbf{I}$ from splitting this masc segment.

The distance bound, $|\delta(\mathbf{A}^{\mathrm{block}}, \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}})| < 11\alpha$ follows from the inequalities,

$$\begin{aligned} |\delta(\mathbf{I}, \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}})| &< 11\alpha, \\ \delta(\mathbf{A}^{\mathrm{block}}, \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}) &> -11\alpha, \\ \delta(\mathbf{I}, \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}) &\geq \delta(\mathbf{A}^{\mathrm{block}}, \mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}). \end{aligned}$$

The first is a premise of the splitting algorithm. The second is a property of masc segments (Lemma 1). The third follows from the fact that distance to $\mathbf{C}^{\mathrm{orig}}\mathbf{C}^{\mathrm{dest}}$ is a monotone function of x and y and that

$$\mathbf{I}_x \leq \mathbf{A}^{\mathrm{block}}{}_x \qquad \text{and} \qquad \mathbf{I}_y \geq \mathbf{A}^{\mathrm{block}}{}_y.$$

We prove by contradiction that $\mathbf{A}^{\mathrm{block}}$ is not blocked from splitting γ. Suppose it is blocked by some site $\mathbf{A} \in A$,

$$\mathbf{A}^{\mathrm{block}}{}_x \leq \mathbf{A}_x \qquad \text{and} \qquad \mathbf{A}^{\mathrm{block}}{}_y \geq \mathbf{A}_y.$$

But $\mathbf{A}^{\mathrm{block}}$ blocks $\mathbf{I}$,

$$\mathbf{I}_x \leq \mathbf{A}^{\mathrm{block}}{}_x \qquad \text{and} \qquad \mathbf{I}_y \geq \mathbf{A}^{\mathrm{block}}{}_y.$$

Therefore,

$$\mathbf{I}_x \leq \mathbf{A}_x \qquad \text{and} \qquad \mathbf{I}_y \geq \mathbf{A}_y.$$

This is a contradiction, since $\mathbf{A}^{\mathrm{block}}$ with the site with the largest x and smallest y which satisfied these conditions.

Special Case 2 is proved analogously.

B.4 Proof of Second Hidden Variable Theorem

This section gives an informal proof of Theorem 2, the Second Hidden Variable Theorem. If the curves $\gamma_f(x)$ and $\gamma_g(x)$ intersect *once*, then the proof is simple. Unfortunately, we cannot guarantee *pseudo-linearity* (Section 2.3): the curves may intersect more than once. This will probably only happen very rarely and only when $\mathbf{C}_f^{\mathrm{orig}}\mathbf{C}_f^{\mathrm{dest}}$ and $\mathbf{C}_g^{\mathrm{orig}}\mathbf{C}_g^{\mathrm{dest}}$ are parallel or nearly parallel. It can only happen if f and g are both increasing or both decreasing. We will give the single-intersection proof first, and then consider the more difficult case.

Let $[x^{\mathrm{orig}}, x^{\mathrm{dest}}] = [\mathbf{D}_f^{\mathrm{orig}}{}_x, \mathbf{D}_f^{\mathrm{dest}}{}_x] \cap [\mathbf{D}_g^{\mathrm{orig}}{}_x, \mathbf{D}_g^{\mathrm{dest}}{}_x]$. Thus x^{orig} equals $\mathbf{D}_f^{\mathrm{orig}}{}_x$ or $\mathbf{D}_g^{\mathrm{orig}}{}_x$ or both, if they are equal, and x^{dest} equals $\mathbf{D}_f^{\mathrm{dest}}{}_x$ or $\mathbf{D}_g^{\mathrm{dest}}{}_x$ or both. It is clear that if the curves $\gamma_f(x)$ and $\gamma_g(x)$ intersect exactly once, then $f(x^{\mathrm{orig}}) - g(x^{\mathrm{orig}})$ and $f(x^{\mathrm{dest}}) - g(x^{\mathrm{dest}})$ have opposite sign. Thus, we can satisfy the theorem by setting $\mathbf{P}$ equal to the element of $\{\mathbf{D}_f^{\mathrm{orig}}{}_x, \mathbf{D}_g^{\mathrm{orig}}{}_x\}$ with the largest x-coordinate and by setting $\mathbf{Q}$ equal to the element of $\{\mathbf{D}_f^{\mathrm{dest}}{}_x, \mathbf{D}_g^{\mathrm{dest}}{}_x\}$ with the smallest x-coordinate.

In order to prove the harder multi-intersection case, we return to our test-car driver analogy. Suppose without loss of generality, that γ_g starts above γ_f (so that $\mathbf{D}_g^{\mathrm{orig}} \in B_f$, for example); it crosses γ_f; and then it crosses back again. Between these two crossings, either γ_g turns left or γ_f turns right; otherwise, once they cross and part the first time, they never meet again (see Figure 18). Without loss of generality, it is γ_g who turns. Therefore, γ_g makes a left turn at an A_g-cone centered at some site $\mathbf{A} \in A_g$. Since, γ_g is below γ_f at this cone, $\mathbf{A}$ is below γ_f, and thus $\mathbf{A} \in B_f$. Therefore we can satisfy the theorem by setting $\mathbf{P}$ equal to $\mathbf{D}_g^{\mathrm{orig}}$ and setting $\mathbf{Q}$ equal to $\mathbf{A}$.

References

[1] Jon L. Bentley and Thomas Ottmann. *Algorithms for Reporting and Counting Geometric Intersections.* **IEEE Transactions on Computing**, C28 (1979), pp. 643–647.

[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] Ioannis Emiris and John Canny. An Efficient Approach to Removing Geometric Degeneracies. *Proceedings of the Eighth Annual Symposium on Computational Geometry*, ACM Press, June 10-12, 1992, pp. 74–82.

[4] Farouki, R. T. *The Characterization of Parametric Surface Sections.* **Computer Vision, Graphics and Image Processing**, vol. 33 (1986), pp. 209–236.

[5] Steven Fortune. Stable Maintenance of Point-Set Triangulations in Two Dimensions. In *30th Annual Symposium on the Foundations of Computer Science*, IEEE, October 1989.

[6] S. Fortune and V. J. Milenkovic. Numerical Stability of Algorithms for Line Arrangements. *Seventh Annual ACM Symposium on Computational Geometry*, North Conway, N.H., June 10-12, 1991, pp. 334–341.

[7] S. Fortune and C. Van Wyk, Efficient Exact Arithmetic for Computational Geometry. To appear in *Ninth Annual ACM Symposium on Computational Geometry*, 1993.

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

[9] John E. Hopcroft and Peter J. Kahn. *A Paradigm for Robust Geometric Algorithms.* TR 89-1044, Department of Computer Science, Cornell University, Ithaca, New York, October 1989.

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

[11] M. Karasick and D. Lieber and L. R. Nackman. Efficient Delaunay Triangulation using Rational Arithmetic. *ACM Transactions on Graphics*, 10:71–91, January 1991.

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

[13] 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.

[14] Victor Milenkovic. Calculating approximate curve arrangements using rounded arithmetic. In *Proceedings of the Symposium on Computational Geometry*, ACM, 1989.

[15] Victor J. Milenkovic. Robust geometric computations for vision and robotics. In *Proceedings of the DARPA IUS Workshop*, pages 764–773, DARPA, May 1989.

[16] Victor Milenkovic. Double Precision Geometry: A General Technique for Calculating Line and Segment Intersections Using Rounded Arithmetic. In *30th Annual Symposium on the Foundations of Computer Science*, IEEE, October 1989.

[17] Victor Milenkovic and Zhenyu Li. Constructing Strongly Convex Hulls Using Exact or Rounded Arithmetic. *Algorithmica*, 8:345–364, 1992.

[18] N. E. Mnev. The universality theorems on the classification problem of configuration varieties and convex polytopes varieties. In *Topology and Geometry: Rohlin Seminar, Lecture Notes in Mathematics, No. 1346* (O. Ya. Viro, ed.), Springer-Verlag, New York, 1988.

[19] 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.

[20] 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.

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

[22] Sugihara, K. *An Approach to Error-Free Solid Modeling.* In: **IMA Summer Program on Robotics**. Institute for Mathematics and Applications, University of Minnesota, 1987.

[23] Kokichi Sugihara and Masao Iri. Construction of the Voronoi Diagram for over 10^5 Generators in Single-Precision Arithmetic. Presented at *First Canadian Conference on Computational Geometry*, McGill University, August 1989.

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

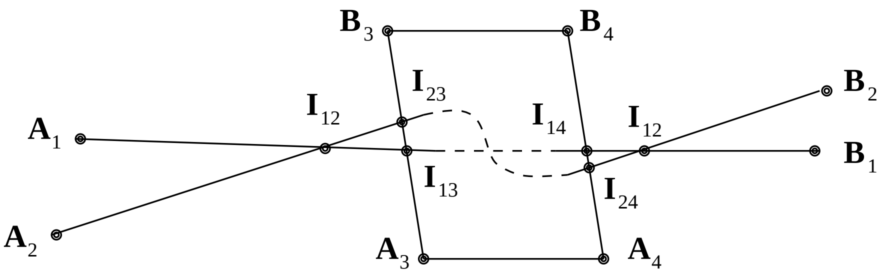

Figure 1: Non-Planar and Non-Pseudo-Linear Structure

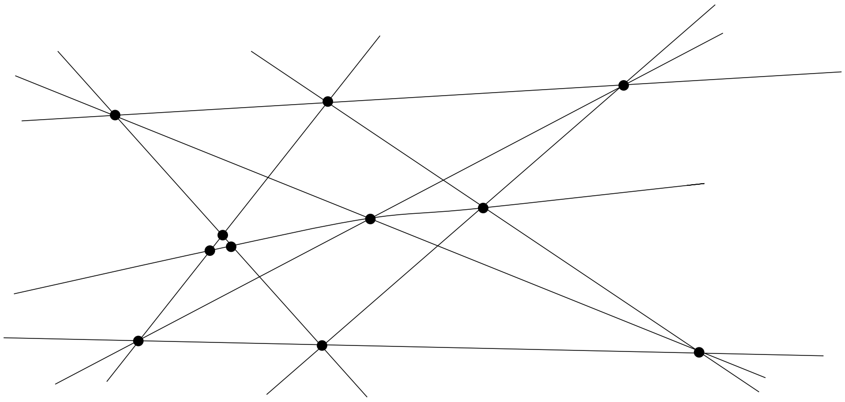

Figure 2: Violation of Pappus Theorem

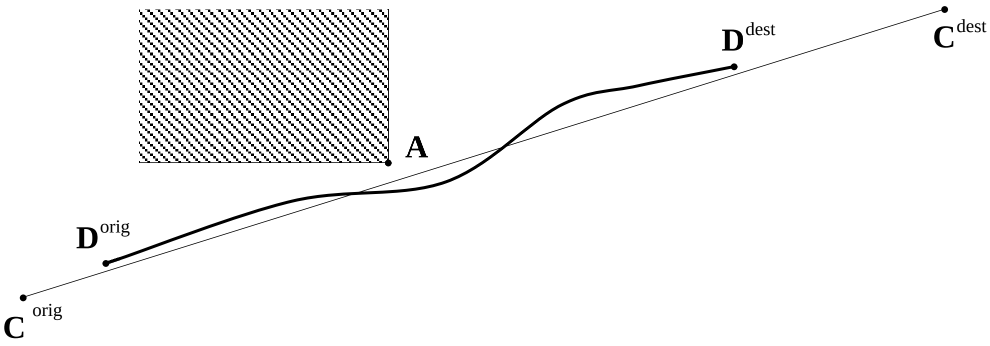

Figure 3: Properties of a MASC Segment

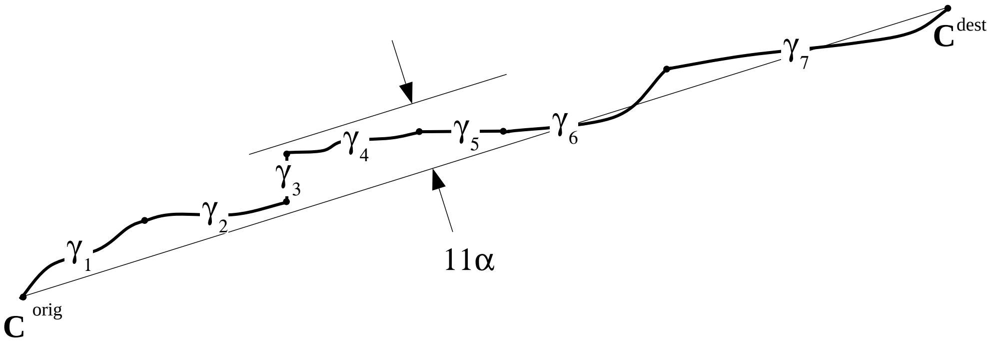

Figure 4: Result of Intersection Algorithm

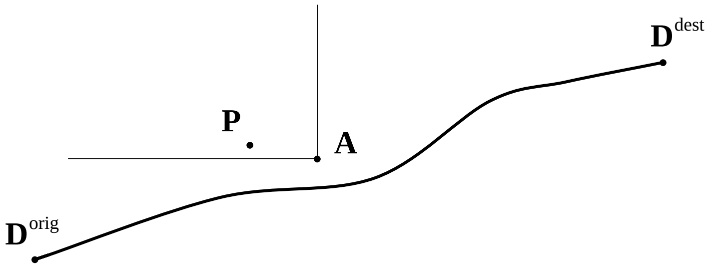

Figure 5: Rule 1: Forced to Put **P** in Above-Set

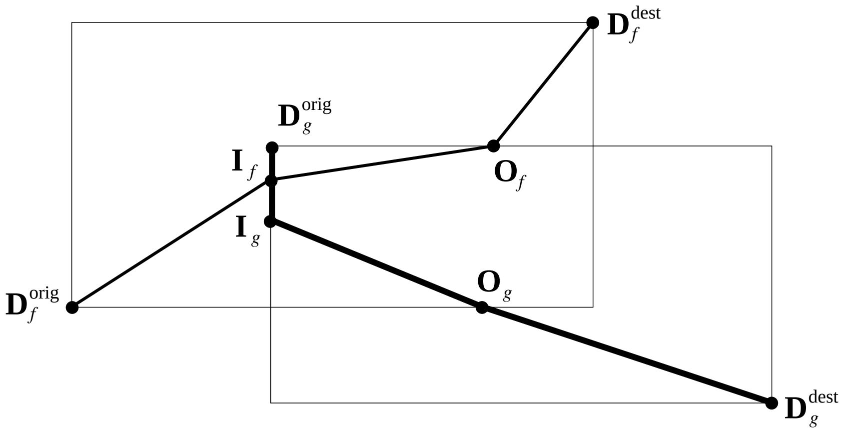

Figure 6: Special Case: Intersection Lies on Boundary

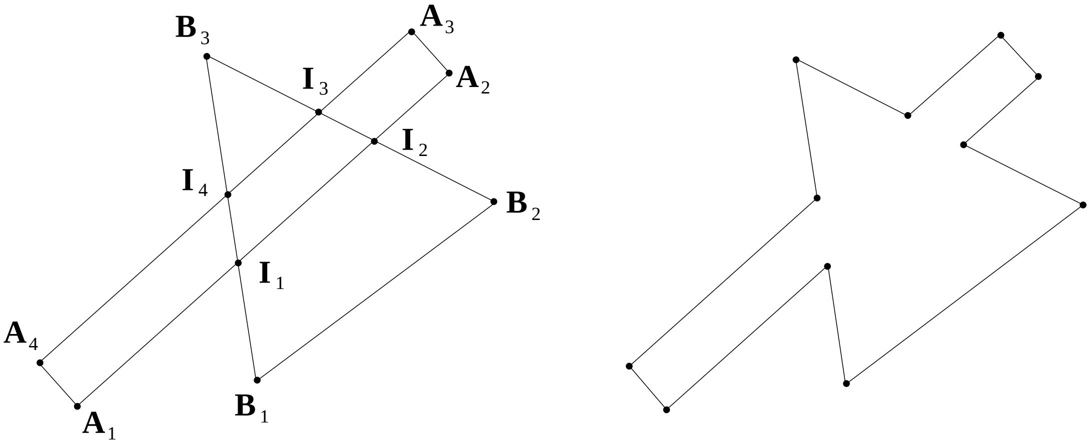

Figure 7: Union of Rectangular and Triangular Planar Regions

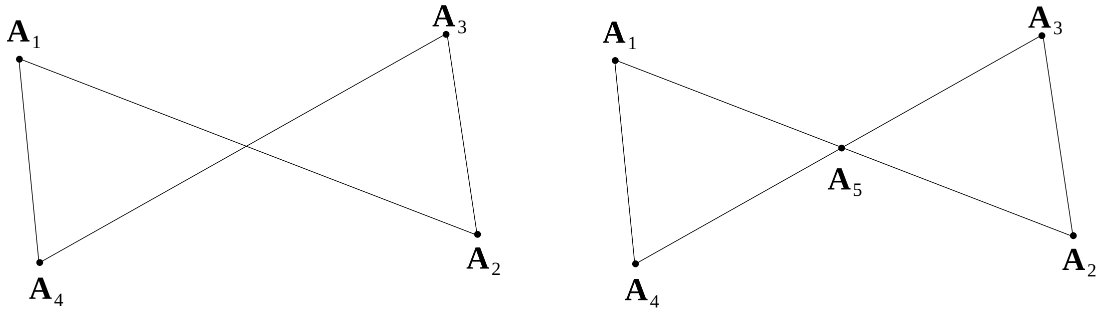

Figure 8: Invalid and Valid Polygons

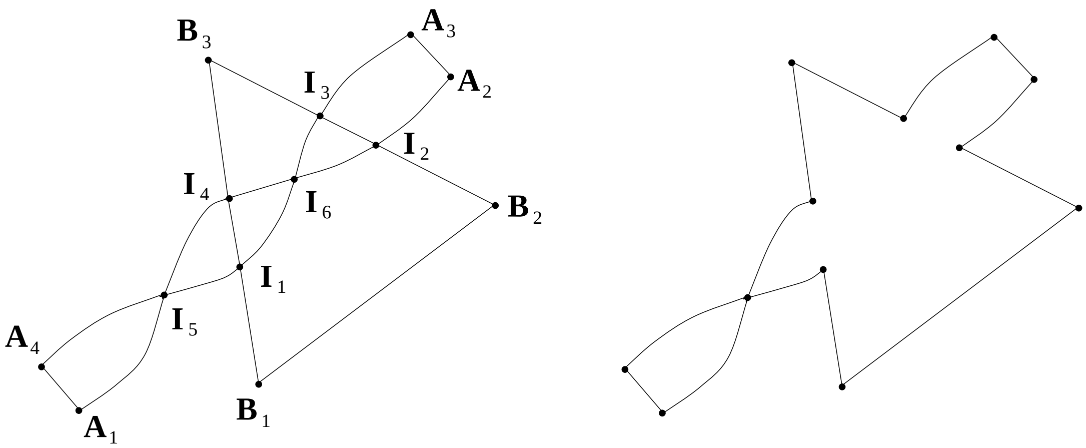

Figure 9: Union Algorithm in Pathological Case

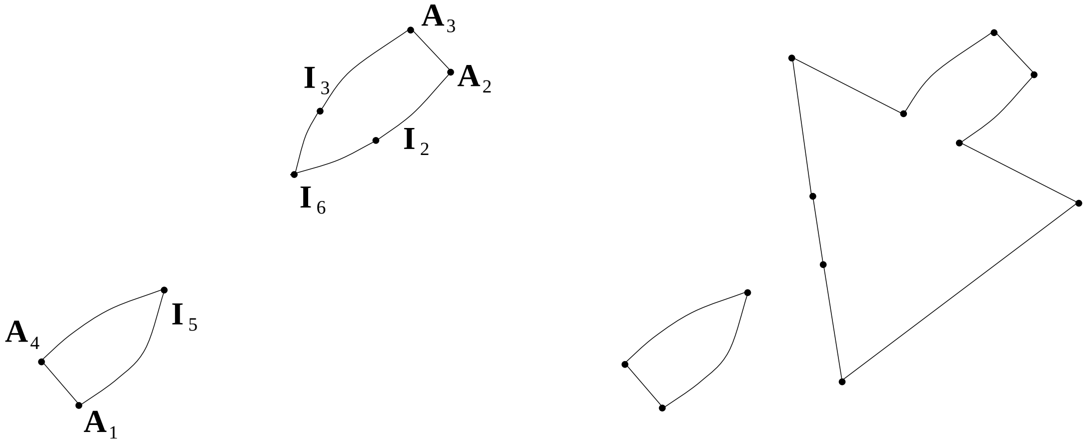

Figure 10: Physical Cleaning of Polygon before Union

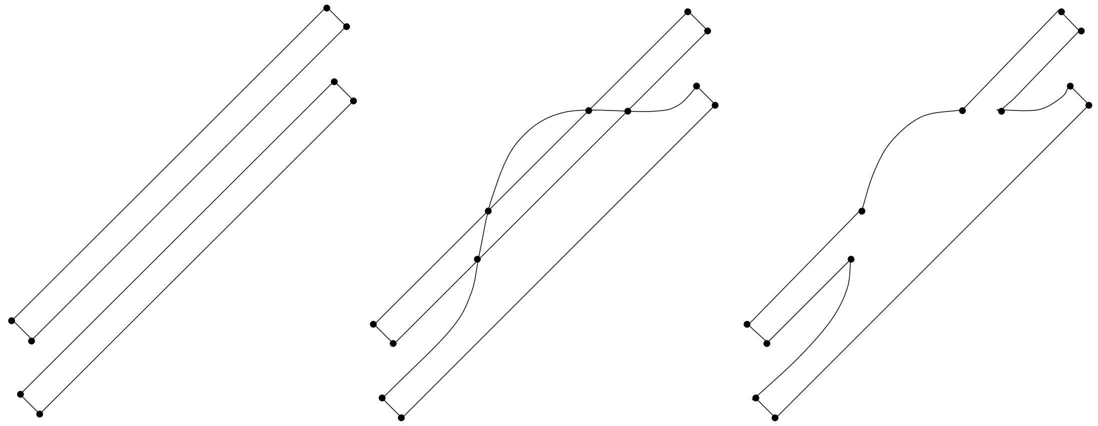

Figure 11: Cleaning Algorithm Applied to Doubly Misplaced Curve

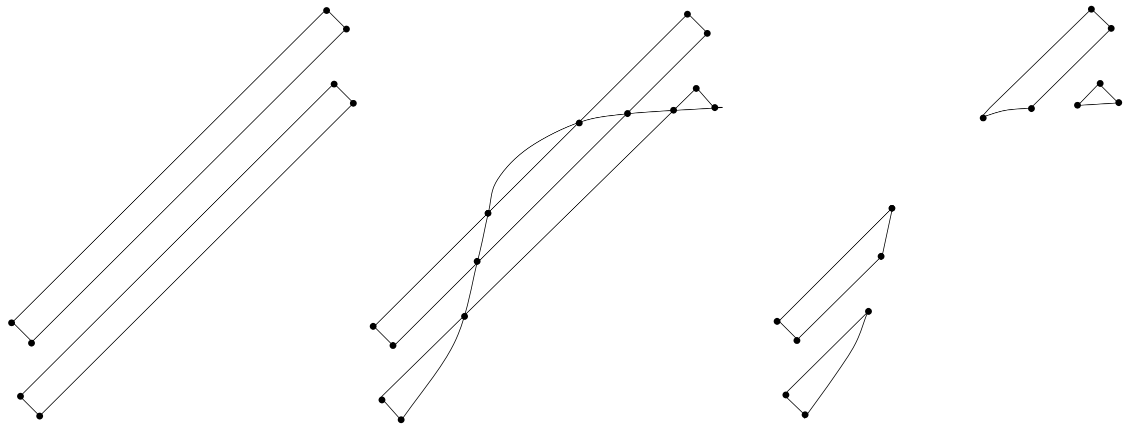

Figure 12: Cleaning Algorithm Applied to Triply Misplaced Curve

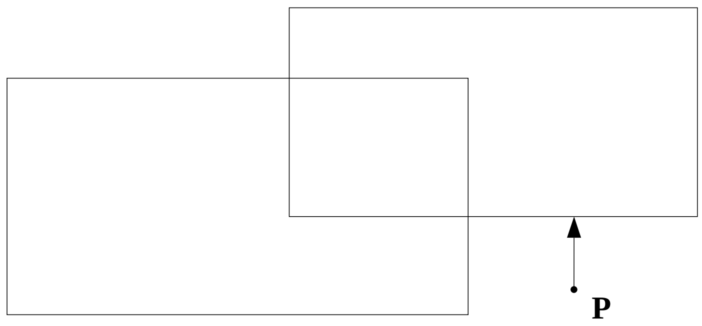

Figure 13: Moving a Point Towards a Rectangle

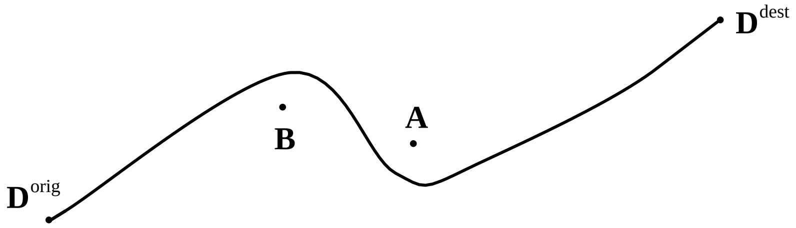

Figure 14: Violation of Monotonicity Property

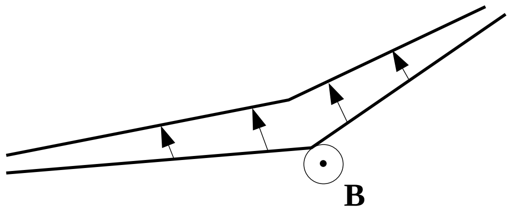

Figure 15: Non-Shortest Course

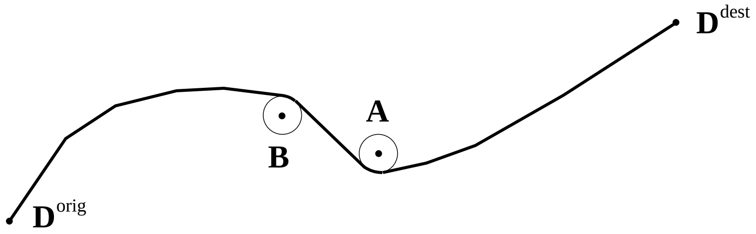

Figure 16: Proof of Monotonicity

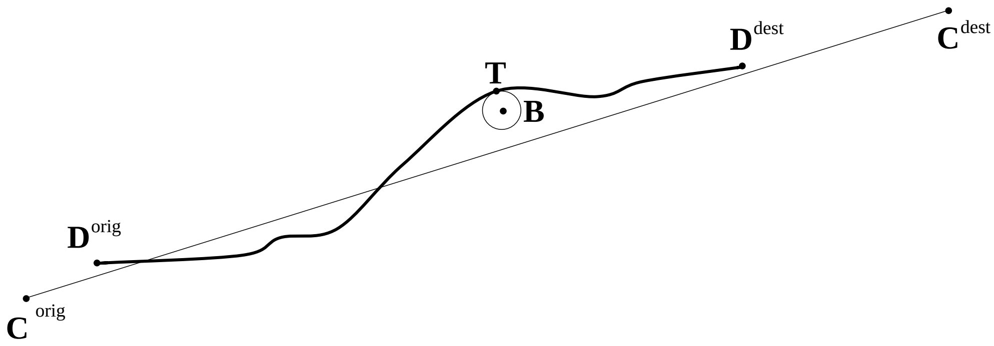

Figure 17: Proof of Accuracy

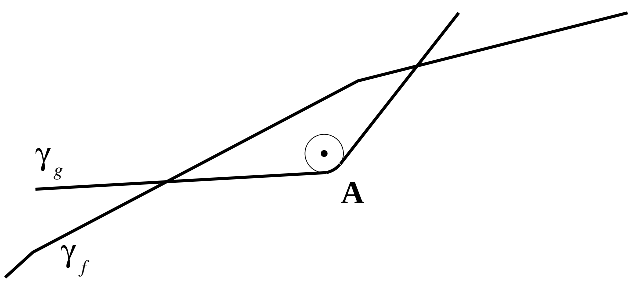

Figure 18: Proof of Second Hidden Variable Theorem