Proceedings of the 30th Annual Symposium on the Foundations of Computer Science (FOCS), IEEE, October 1989, pp. 500–506.

Double Precision Geometry: A General Technique for Calculating Line and Segment Intersections Using Rounded Arithmetic

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

We show for the first time how to reduce the cost of performing specific geometric constructions by using rounded arithmetic instead of exact arithmetic. Exploiting a property of floating point arithmetic called *monotonicity*, a new technique, *double precision geometry*, can replace exact arithmetic with rounded arithmetic in any efficient algorithm for computing the set of intersections of a set of lines or line segments. This technique reduces the complexity of any such line or segment arrangement algorithm by a constant factor. In addition, double precision geometry reduces by a factor of N the complexity of rendering segment arrangements on a 2^N by 2^N integer grid such that output segments have grid points as endpoints.

1 Introduction

The theory of computational geometry provides us with a large number of useful geometric algorithms. However, it is not easy to create efficient and reliable programs from these algorithms because they are either based on infinite precision arithmetic or the precision required is not explicitly stated. The standard practice among developers of geometric programs is to use finite precision floating point arithmetic as if it were infinite precision real arithmetic. Unfortunately, geometric programs are particularly sensitive to numerical errors because of they way they closely couple symbolic and numerical data. The common experience is that all floating point geometric programs fail on a small subset of their inputs and that these failures are difficult to analyze or avoid. There is clearly a practical need for a theory of *robust* geometric algorithms, algorithms which can be safely implemented with rounded finite precision arithmetic. This paper describes part of such a theory and its expected payoff, robust approximate geometric algorithms with faster running times than the best exact versions.

A theory of robust geometry requires an understanding of the precision requirements of exact algorithms: the number of bits P of arithmetic precision required for each arithmetic operation, expressed as a function of the arithmetic precision N of the input. For a line, N is the number of bits used to express each of the coefficients of the corresponding line equation. For a line segment, N is the number of bits used to express each coordinate of its endpoints. We focus here on the *line intersection* and *segment intersection* problems: find all intersections among a set of L lines or S line segments in the plane and describe the topological *arrangement* generated by the resulting set of vertices, line segments, and polygonal regions. A careful analysis shows that $P = 3N + 3$ suffices for the line intersection problem and $P = 4N + 7$ suffices for the segment problem. Thus, in order to improve on the exact method a robust method must use less than $3N + 3$ bits of precision, and we can hope to improve the running time by a factor of up to three or four.

There are a number of reasons why we should aim for a factor of three or four improvement. First, if we can reduce the constant factor for these simple problems, we might make inroads into broader classes of intersection problems in which the constant is replaced by a function of algebraic degree and the number of dimensions. Second, from the point of view of numerical analysis, if each N-bit input is an approximation to a more accurate unknown value (which is generally the case), then only the most significant N bits of any calculated value are meaningful. It is hard to accept having to calculate more meaningless bits than meaningful bits which is what using $3N$ or $4N$ bits implies. Third, in practice the factor of three or four in precision requirements forces us to simulate extended precision arithmetic in software which is many times more expensive than hardware arithmetic. The hardware arithmetic commonly available can only reliably handle inputs with 11 bits of precision, too little for most practical applications. Since efficiency is very important, developers routinely apply hardware arithmetic to inputs of 20 or more bits. It would clearly be a benefit to them and their customers if they could do this reliably.

We describe here a technique called *double precision geometry* for performing numerical computations on lines and line segments using rounded arithmetic. This technique removes all computations with precision greater than $2N + 1$ bits ($2N + 5$ for segments) and therefore reduces by a constant factor the complexity of *any* efficient line or segment intersection algorithm based on standard exact arithmetic. This approach of removing expensive computations allows us to prove that rounded arithmetic algorithms are faster as a class than exact algorithms without knowing which is the fastest exact algorithm. Double precision geometry is also quite practical: it achieves a one-to-one mapping between arithmetic operations of the algorithm and IEEE standard 53-bit mantissa floating point operations and guarantees an output precision of at least one part in 16 million (24 bits). This is sufficient for most practical purposes. Double precision geometry allows us to avoid singular cases without the use of simulation of simplicity [2, 23]. Finally, double precision geometry can reduce by a factor of N the computational complexity of Greene and Yao's method [6] for calculating segment intersections on an integer grid (see Section 3.3). For this construction, in addition to computing the set of intersections and topology generated by a set of line segments, we wish to replace each segment by a sequence of segments joining grid points such that output segments intersect only at grid points.

1.1 Background

Double precision geometry is the first application of rounded arithmetic to geometric problems that is provably reliable and more efficient than the exact algorithm. It is part of a more general study of the trade-off between precision and running time which is summarized in the following table.

Required Arithmetic Precision	Complexity
$N+5$	$\mathrm{O}(3N2^N L)$
$N+20$	$\mathrm{O}(L^4 \log L)$
$N+\log N+6$	$\mathrm{O}(L^4)$
$N+\log N+13$	$\mathrm{O}(L^3 \log L)$
$2N+1$	$\mathrm{O}(L \log L+V)$
$3N+3$	$\mathrm{O}(L \log L+V)$

The $\mathrm{O}(L^4)$ algorithm is unpublished; otherwise see [17, 14]. The last two lines give for comparison the result of this paper and the optimal exact algorithm [3], where V is the number of vertices of intersection.

We have recently shown how to construct convex hulls of N-bit points using only $(N+6)$-bit arithmetic [15], and this also improves on the standard $(2N+2)$-bit exact algorithm. Fortune has given rounded arithmetic algorithms for constructing convex hulls and maintaining triangulations of the plane [5], but the precision required by these grows with the number of inputs.

For a discussion of the more general problem of improving the reliability of geometric computations using rounded arithmetic, see [9, 10]. Problems addressed include polygon [21, 16] and polyhedron [11, 12, 22] modeling, algebraic curves [14], and other problems in plane geometry [20]. Karasick [12] has the most practical three dimensional result to date, a polyhedral modeling system that is provably *safe*: either it generates a topologically consistent result or it indicates failure.

Karasick, Lieber and Nackman [13] have recently demonstrated an exact Delaunay triangulation algorithm with empirical cost well below the worst case. They use interval arithmetic to determine the amount of precision required to perform each specific calculation exactly. This technique can be used to improve the performance of other exact algorithms as well as the rounded algorithms presented here.

1.2 Model of Arithmetic

This section gives an abstract description of floating point arithmetic and describes a property, called *monotonicity*, which the double precision geometry technique requires. Fortunately, floating point architectures conforming to the IEEE standard [4] have this property.

A P-bit floating point arithmetic consists of a set of representable numbers,

$$\mathrm{REP} = \{a \cdot 2^e | a, e \in \mathbf{Z}, |a| < 2^P, |e| < 3P\},$$

and a rounding function,

$$\mathrm{Round} : \Re \rightarrow \mathrm{REP},$$

from the set of reals to the set of representable numbers. The value a is the *mantissa* and e is the *exponent*. In practice, the bound on the exponent is much larger, but $3P$ is all we need here. If $x \in \mathrm{REP}$, then $\mathrm{Round}(x)$ equals x. If $x \notin \mathrm{REP}$, then x lies between two representable numbers, and Round returns one of these two numbers. The rounding is performed according to some consistent rule: round down, round up, round to zero, round away from zero, round to nearest (four types). Whatever the rounding scheme, we require it to be *monotonic*,

$$\forall x, y \in \Re : x < y \rightarrow \mathrm{Round}(x) \leq \mathrm{Round}(y).$$

Arithmetic on REP is defined by means of the rounding function. For $\mathbf{op} \in \{+, -, *, /\}$:

$$(X \ \mathbf{op}\ Y)_{\mathrm{F}} = \mathrm{Round}(X \ \mathbf{op}\ Y),$$

where X and Y are in REP and the subscript F denotes the result of the computation in floating point arithmetic.

2 Line Arrangements

In this section we will describe a way of altering any line arrangement algorithm so that it can safely use rounded arithmetic. The only property that the algorithm must have is that it must be able to function on pseudo-linear curves, and it must be non-redundant. These properties are defined in Section 2.4, and it is shown that the standard arrangement algorithm has these properties.

2.1 Altering the Output

Before we describe how to modify an arbitrary arrangement algorithm to use rounded arithmetic, we will define the result that the modified algorithm will generate. The algorithm calculates the arrangement generated by a set of polygonal approximations to the input lines. The topology of the set of approximations will be the same as the topology of the original set of lines except that neighboring vertices may be rounded to the same point.

We are given a set of lines $\{\mathbf{l}_i : a_i x + b_i y = c_i | i = 1, 2, \ldots, L\}$. Let $\mathbf{v}_{ij} = \langle x_{ij}, y_{ij} \rangle$ be the point of intersection $\mathbf{l}_i \cap \mathbf{l}_j$. If $\mathbf{l}_i$ and $\mathbf{l}_j$ are parallel or equal, then $\mathbf{v}_{ij}$ is undefined. Define

$$\mathbf{V}_{ij} = \langle X_{ij}, Y_{ij} \rangle = \langle \mathrm{Round}(x_{ij}), \mathrm{Round}(y_{ij}) \rangle$$

to be the rounded value of $\mathbf{v}_{ij}$ under the rounding scheme used by the floating point arithmetic we have chosen.

We will generate the set $\{\mathbf{V}_{ij} | 1 \leq i < j \leq L\}$ as the set of intersection points. The topology we generate will be the same as the correct arrangement topology for the lines except that if for some i, j, k, $\mathbf{v}_{ij}$, $\mathbf{v}_{jk}$, and $\mathbf{v}_{ik}$ round to the same representable point, then the triangle $\mathbf{v}_{ij}\mathbf{v}_{jk}\mathbf{v}_{ik}$ will collapse and the three lines $\mathbf{l}_i$, $\mathbf{l}_j$, and $\mathbf{l}_k$ will be (falsely) coincident or we will report an arbitrary topological orientation for this triangle.

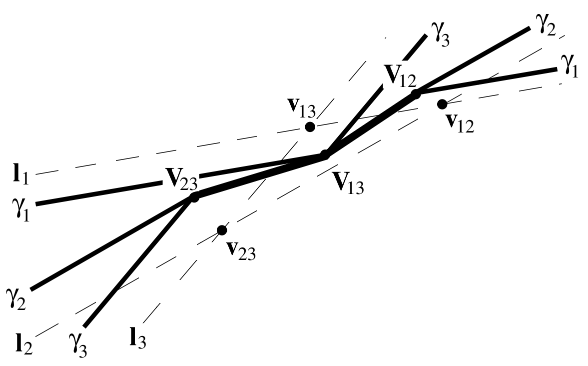

Figure 1: Paths Altered by Rounding

Imagine the vertices of the arrangement to be traffic cones and the lines to be the paths driven by test drivers on a planar test course. The course is slightly unusual in that the driver traversing line $\mathbf{l}_i$ is expect to run over the cones that correspond to vertices lying on $\mathbf{l}_i$. Rounding the vertex locations moves the cones, and the driver traversing $\mathbf{l}_i$ is forced to follow a new path γ_i in order to maintain the same topology (see Figure 1). The danger is that path γ_i may cross path γ_j at some point other than $\mathbf{V}_{ij}$, thus altering the topology. However, if each driver takes the most efficient (locally shortest) path, and if we take the limit as the turning radius of each car approaches zero, then the topology will be correct. To see why, first make the following observation: path γ_i will make a $\left\{\begin{array}{c}\text{left}\\ \text{right}\end{array}\right\}$ turn only at a vertex that lies, topologically, to the $\left\{\begin{array}{c}\text{left}\\ \text{right}\end{array}\right\}$ or on $\mathbf{l}_i$. Now suppose that line $\mathbf{l}_j$ is steeper than $\mathbf{l}_i$ and that path γ_j crosses γ_i before reaching $\mathbf{V}_{ij}$. In order to come back together again, either γ_i must turn left or γ_j must turn right. This implies the existence of a vertex west of $\mathbf{V}_{ij}$ that is left of γ_i and right of γ_j. By monotonicity (Section 1.2), this implies an unrounded vertex west of $\mathbf{v}_{ij}$, left of $\mathbf{l}_i$ and right of $\mathbf{l}_j$. Since $\mathbf{l}_j$ is steeper than $\mathbf{l}_i$, no such vertex exists.

Under the definition of the previous paragraph, the paths γ_i and γ_j may share common edges and vertices away from $\mathbf{V}_{ij}$, even though they do not actually cross. However we can apply a small perturbation to each so that they intersect only at $\mathbf{V}_{ij}$. A question that is frequently asked is whether it is possible to realize the same topology (ignoring the vertex locations) with a set of *lines* instead of polygonal curves. The answer is no because the topology may not satisfy Pappus's theorem or other theorems of projective geometry. It is an interesting open question as to whether we can alter the technique to guarantee a topology realizable by lines. Since the "language" of such topologies is equivalent to the existential theory of the reals [18], this will probably not be easy.

2.2 Altering the Arrangement Algorithm

This section shows how to modify any exact algorithm for computing line intersections. We can describe a general procedure for altering arrangement algorithms because there are only a small number of numerical operations that any algorithm will need to perform. Furthermore, only some of these require $(3N+3)$-bit arithmetic to perform exactly. These operations and their rounded versions follow. Anything not included, such as detecting and dealing with parallel lines, requires only $(2N+1)$-bit arithmetic.

Calculating vertex locations: if $\mathbf{l}_i$ and $\mathbf{l}_j$ are not parallel or equal, let $\mathbf{v}_{ij} = \langle x_{ij}, y_{ij}\rangle$ denote the exact intersection. The algorithm computes an approximation $\mathbf{V}_{ij} = \langle X_{ij}, Y_{ij}\rangle$ to $\mathbf{v}_{ij}$,

$$X_{ij} = \left(\frac{c_i b_j - c_j b_i}{a_i b_j - a_j b_i}\right)_{\mathrm{F}}, \quad Y_{ij} = \left(\frac{a_i c_j - a_j c_i}{a_i b_j - a_j b_i}\right)_{\mathrm{F}}, \qquad (1)$$

where the subscript F denotes the use of any $(2N+1)$-bit floating point arithmetic satisfying the model in Section 1.2. If space is a consideration, the vertex locations can be rounded to N bits of precision.

Comparing vertices and lines: for each vertex $\mathbf{v}_{jk}$, we need to know on which side of line $\mathbf{l}_i$ it lies. This is equivalent to calculating the sign of $a_i x_{jk} + b_i y_{jk} - c_i$ which the next section describes how to do.

Comparing vertices: an algorithm may need to sort the vertices by their x-coordinates or y-coordinates. The next section shows how to compare x_{hi} with x_{jk} or y_{hi} with y_{jk}.

Ordering vertices on a line: if $\mathbf{v}_{ij}$ and $\mathbf{v}_{ik}$ lie on $\mathbf{l}_i$, we may need to be able to determine the order in which they appear. If the the line is horizontal ($b_i = 0$) or vertical ($a_i = 0$), we can determine the order by comparing vertex coordinates. Otherwise, determining the order is equivalent to determining the sign of $a_i(y_{ij} - y_{ik})$, which also reduces to the problem of comparing coordinates since we know the sign of a_i.

2.3 Replacing Arithmetic with Comparisons

This section shows how to compute the signs of certain *relevant quantities*: $x_{hi} - x_{jk}$, $y_{hi} - y_{jk}$, and $a_i x_{jk} + b_i y_{jk} - c_i$. Using exact arithmetic we must use $(3N+3)$-bit arithmetic[1] to compute these signs correctly. However, we can calculate these signs robustly using only arithmetic comparisons and no addition or multiplication.

First we observe that X_{ij}, which we have already computed using $(2N+1)$-bit arithmetic, is the rounded value of x_{ij}. By monotonicity (Section 1.2), if $X_{ij} \neq X_{jk}$ then $X_{ij} - X_{jk}$ and $x_{ij} - x_{jk}$ have the same sign. So long as the vertices $\mathbf{v}_{ij}$, $\mathbf{v}_{jk}$ and $\mathbf{v}_{ik}$ do not round to the same point, we will know the sign of at least one relevant quantity involving the indices i, j and k. Second we observe that the three expressions,

$$\begin{array}{c}(x_{ij} - x_{ik})(b_i)(a_i b_j - a_j b_i)(a_i b_k - a_k b_i),\\ -(y_{ij} - y_{ik})(a_i)(a_i b_j - a_j b_i)(a_i b_k - a_k b_i),\\ -(a_k x_{ij} + b_k y_{ij} - c_k)(a_i b_j - a_j b_i),\end{array}$$

and the six similar expressions obtained by cycling the indices i, j and k, all have the same sign. Third, we observe that have already computed the exact value of expressions like $a_i b_j - a_j b_i$ when we computed the vertices. Thus as long as we can compute the sign of one relevant quantity involving i, j and k, we can compute the signs of all the others using only comparisons and branches. In the case that $\mathbf{V}_{ij}$, $\mathbf{V}_{jk}$ and $\mathbf{V}_{ik}$ are equal, we choose an arbitrary sign for one relevant quantity and derive the other eight from it.

Relevant quantities involving *four* indices, such as $x_{hi} - x_{jk}$, are not linked to other relevant quantities in the way that three-index quantities are. If $X_{hi} \neq X_{jk}$, we can compute the sign correctly. Otherwise we make an arbitrary choice.

[1] Please note: $3N+3$ is not a proven lower bound, but in any case we are eliminating all of these computations.

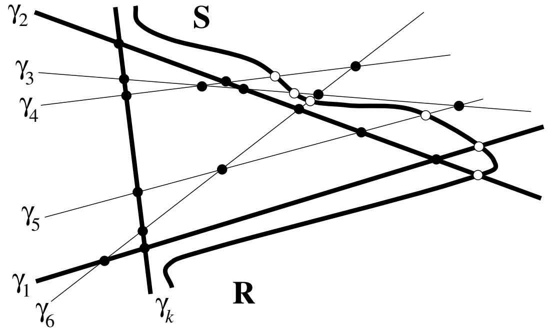

Figure 2: Proving Non-Redundancy in the Edelsbrunner *et al* Algorithm

2.4 Correctness

It remains to be shown that the arbitrary decisions we make in the previous section have no effect on the shapes of the pseudo-linear curves defined in Section 2.1 and have no effect on the correctness of any intersection algorithm modified according to the rules in Section 2.2.

The first fact is easy to show. If $\mathbf{V}_{ij} = \mathbf{V}_{jk} = \mathbf{V}_{ik}$ then the path γ_k passes through all three points. It makes no difference whether we decide $\mathbf{v}_{ij}$ lies to the left or to the right of $\mathbf{l}_k$.

As far as the second fact is concerned, the curves implied by Section 2.1 are pseudo-linear, not linear, but the task of distinguishing one type from the other is NP-hard, and thus it is unlikely that any efficient line or segment arrangement algorithm will not also handle pseudo-lines. There is also the issue of *redundancy*. An algorithm is *redundant* if it asks the low level routines to make a decision whose outcome could have been deduced from the decisions already made. However if the robust version of an algorithm makes an unnecessary arbitrary decision, then the exact version makes an unnecessary $(3N+3)$-bit or $(4N+7)$-bit computation, and therefore a redundant algorithm is inefficient. If an algorithm is non-redundant, then it cannot make contradictory arbitrary decisions, and therefore is has a correct double precision version. We prove below that Edelsbrunner, O'Rourke, and Seidel's line arrangement algorithm [3] is non-redundant. Chazelle and Edelsbrunner's segment intersection algorithm [1] and Mulmuley's algorithm [19] are somewhat complex to analyze, but they are almost certainly non-redundant also. Any algorithm more efficient than these will surely be non-redundant also.

Suppose we are running the Edelsbrunner *et al* algorithm and we are adding line $\mathbf{l}_k$ (actually γ_k) to the arrangement generated by $\mathbf{l}_1, \mathbf{l}_2, \ldots, \mathbf{l}_{k-1}$. We have reached region $\mathbf{R}$ and we make an arbitrary decision with respect to vertex $\mathbf{v}_{12}$. Assume an redundant decision is possible: $\mathbf{v}_{12}$ should be on the right, not on the left. The straight path in Figure 2 is the correct construction:[2] γ_k reaches region $\mathbf{S}$ after passing through γ_1 and γ_2 and a number of pseudo-lines γ_i ($i = 3, 4, 5, \ldots$) between them. However, the triangle $\mathbf{V}_{12}\mathbf{V}_{1k}\mathbf{V}_{2k}$ is degenerate and therefore all the triangles $\mathbf{V}_{i2}\mathbf{V}_{ik}\mathbf{V}_{2k}$ are also. Hence we could have reached $\mathbf{S}$ by passing to the right of $\mathbf{v}_{12}$ and all the intervening $\mathbf{v}_{i2}$ as depicted by the curved path. Furthermore, once we reach $\mathbf{S}$, we have crossed the same set of lines and can continue the construction as if we passed to the left of $\mathbf{v}_{12}$. This contradicts our assumption that a redundant decision is possible.

[2] This is a topological, not geometric, diagram. Most of the vertices are actually located at the same geometric point.

2.5 Complexity

No matter what algorithm we use, we must compute the vertex locations to at least N bits of precision. Numerical analysis shows that $(2N+1)$-bit arithmetic suffices for this, and there is no straightforward way to use less in all cases. There is no other arithmetic in the robust algorithm, only comparisons and logic. Therefore, the robust algorithm has to be cheaper than the exact version.

3 Segment Arrangements

This section will describe a general technique for altering any algorithm that locates the intersections of a set of line segments using standard exact arithmetic. As in the case of line arrangements, the algorithm must be able to handle pseudo-linear curves, and it must be non-redundant. From now on, we will assume that the rounding scheme we are using is round-to-nearest.

3.1 Altering the Problem

We will first describe how to alter the set of segments to reduce the precision required to implement the intersection algorithm. Let us assume we are working with a set of line segments $\{\mathbf{A}_i\mathbf{B}_i | i = 1, 2, \ldots, S\}$ whose endpoints lie on an N-bit integer grid:

$$|\mathbf{A}_{i,x}|, |\mathbf{A}_{i,y}|, |\mathbf{B}_{i,x}|, |\mathbf{B}_{i,y}| < 2^N.$$

First, we can calculate the line $\mathbf{L}_i : A_i x + B_i y = C_i$ which passes through $\mathbf{A}_i$ and $\mathbf{B}_i$,

$$\begin{array}{c} A_i = (\mathbf{A}_{i,y} - \mathbf{B}_{i,y}), \qquad B_i = -(\mathbf{A}_{i,x} - \mathbf{B}_{i,x}), \\ C_i = (\mathbf{A}_{i,y}\mathbf{B}_{i,x} - \mathbf{A}_{i,x}\mathbf{B}_{i,y}). \end{array} \tag{2}$$

We then find the integer K so that,

$$2^{N-K} \leq \max(A_i, B_i) < 2^{N+1-K}.$$

Set $a_i = A_i 2^K$ and $b_i = B_i 2^K$. Set c_i equal to $C_i 2^K$ rounded to the nearest multiple of 2^{N-1}. The resulting line $\mathbf{l}_i : a_i x + b_i y = c_i$ is a close approximation to $\mathbf{L}_i$.

For each segment $\mathbf{A}_i\mathbf{B}_i$ we define, but do not compute, a new segment $\mathbf{a}_i\mathbf{b}_i$ as follows. If $|a_i| \leq |b_i|$ we define $\mathbf{a}_i$ and $\mathbf{b}_i$ to be the intersections of $\mathbf{l}_i$ with the lines $x = \mathbf{A}_{i,x}$ and $x = \mathbf{B}_{i,x}$. If $|a_i| > |b_i|$ we define $\mathbf{a}_i$ and $\mathbf{b}_i$ to be the intersections of $\mathbf{l}_i$ with the lines $y = \mathbf{A}_{i,y}$ and $y = \mathbf{B}_{i,y}$. It is easily shown the $\mathbf{A}_i$ and $\mathbf{B}_i$ are the nearest integer grid points to $\mathbf{a}_i$ and $\mathbf{b}_i$ (Figure 3).

3.2 Altering the Algorithm

Now we have defined every point of interest, including the endpoints of the segments, as the intersection of two lines. Therefore we can use the techniques of Section 2.3 to answer any topological query. To calculate $\mathbf{V}_{ij}$, the approximate intersection of $\mathbf{a}_i\mathbf{b}_i$ and $\mathbf{a}_j\mathbf{b}_j$, we apply Equation 1 to $\mathbf{l}_i$ and $\mathbf{l}_j$ and then round the result to the nearest grid point. It can be shown that using precision $P = 2N+5$ assures that this calculation will yield the nearest grid point to the actual intersection $\mathbf{v}_{ij} = \langle x_{ij}, y_{ij} \rangle$ of $\mathbf{a}_i\mathbf{b}_i$ and $\mathbf{a}_j\mathbf{b}_j$.

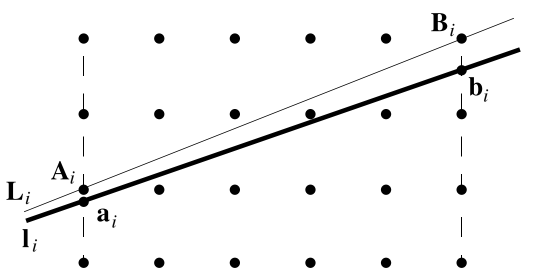

Figure 3: Replacing $\mathbf{A}_i\mathbf{B}_i$ with $\mathbf{a}_i\mathbf{b}_i$

3.3 Correctness

In the case of the line arrangement algorithm we replaced each line $\mathbf{l}_i$ with a polygonal curve γ_i which passed through all vertices $\mathbf{V}_{jk}$ adjacent to $\mathbf{l}_i$. In the case of the line segment algorithm, the curve γ_i starts at $\mathbf{A}_i$ and ends at $\mathbf{B}_i$ and passes only through vertices $\mathbf{V}_{jk}$ or $\mathbf{A}_j$ or $\mathbf{B}_j$ adjacent to segment $\mathbf{a}_i\mathbf{b}_i$. In all other respects, the definition of correctness is the same.

If we examine Greene and Yao's definition of a *finite-resolution* segment arrangement [6], we find that the curves $\gamma_1, \ldots, \gamma_C$ satisfy exactly the conditions they set forth. Furthermore each γ_i has a factor of N fewer points than the curve generated by their algorithm. All we need is a way to generate γ_i explicitly. Briefly, we use Greene and Yao's technique to find all $\mathbf{V}_{jk}$ adjacent to $\mathbf{l}_i$ ($\mathbf{l}_i$ intersects the unit square centered at $\mathbf{V}_{jk}$). Let $\mathbf{V}_{gi}$ and $\mathbf{V}_{hi}$ be consecutive vertices on $\mathbf{l}_i$. We create a simple polygon starting at $\mathbf{V}_{hi}$, passing through vertices left of and adjacent to $\mathbf{l}_i$ until we reach $\mathbf{V}_{gi}$, and then coming back through vertices adjacent and to the right. We then apply Guibas *et al*'s shortest path algorithm [8] to construct the shortest polygonal curve inside this polygon and joining $\mathbf{V}_{gi}$ to $\mathbf{V}_{hi}$. Connecting these paths together we generate γ_i. This algorithm is linear in the size of the output and uses only $(2N+5)$-bit arithmetic.

3.4 Complexity

The discussion in Section 2.5 is equally applicable here. However, evaluating the sign of $a_j\mathbf{a}_{i,x} + b_j\mathbf{b}_{i,x} - c_j$ may involve computing the intersection of $\mathbf{l}_j$ and the line $x = \mathbf{A}_{i,x}$, which we did not have to do in the case of line arrangements. Calculating the extra intersection requires one multiplication, one subtraction, and one division. The original algorithm solves the same problem by evaluating the sign of $A_j\mathbf{A}_{i,x} + B_j\mathbf{B}_{i,x} - C_j$ which involves two multiplications, one addition and one subtraction. Unless division is more expensive than multiplication, the altered version is slightly cheaper.

We achieve an additional saving since we replaced segment $\mathbf{A}_i\mathbf{B}_i$ with line $\mathbf{l}_i$. Without this replacement, it takes $3N+3$-bit arithmetic to compute $\mathbf{V}_{ij}$ and $4N+7$ bits to compute the relevant quantities (Section 2.3). With the replacement, we use $2N+5$ bits to compute $\mathbf{V}_{ij}$ and eliminate the latter computations altogether.

4 Concluding Remarks

Double precision geometry might better be called *once-rounded geometry*, for it is the fact that there is only one arithmetic rounding per vertex coordinate which makes this technique work. An obvious direction in which to generalize is to examine other geometric computations which can be made more efficient without sacrificing reliability by allowing one rounding per output value. We will also continue working on robust algorithms which allow multiple roundings since these are obviously more generally applicable.

References

[1] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. In *29th Annual Symposium on the Foundations of Computer Science*, IEEE, October 1988.

[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] Herbert Edelsbrunner, Joseph O'Rourke, and Raimund Seidel. Constructing arrangements of lines and hyperplanes with applications. In *24th Symposium on the Foundations of Computer Science*, pages 83–91, IEEE, 1983.

[4] Richard Fateman. High-level language implications of the proposed ieee floating-point standard. *ACM Transactions on Programming Languages and Systems*, 4(2):239–257, 1982.

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

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

[7] Branko Grunbaum. *Arrangement and Spreads. Conference Board of the Mathematical Sciences Regional Conference Series in Mathematics, No. 10*, American Mathematical Society, 1972.

[8] Leo Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear time algorithms for visibility and shortest path problems inside simple polygons. In *Proceedings of the Symposium on Computational Geometry*, ACM, 1986.

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

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

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

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

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

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

[15] Victor Milenkovic and Zhenyu Li. Constructing Strongly Convex Hulls Using Exact or Rounded Arithmetic. Submission to ACM-SIAM Symposium on Discrete Algorithms, 1989.

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

[17] 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.

[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] K. Mulmuley. A Fast Planar Partition Algorithm, I. In *29th Annual Symposium on the Foundations of Computer Science*, IEEE, October 1988.

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

[21] 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.

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

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