Seventh Annual ACM Symposium on Computational Geometry (SCG), North Conway, N.H., June 10-12, 1991, pp. 334–341.

Numerical Stability of Algorithms for Line Arrangements

Steven Fortune
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Victor Milenkovic
Harvard University
Cambridge, Massachusetts 02138

Abstract

We analyze the behavior of two line arrangement algorithms, a sweepline algorithm and an incremental algorithm, in approximate arithmetic. The algorithms have running times $O(n^2 \log n)$ and $O(n^2)$ respectively. We show that each of these algorithms can be implemented to have $O(n\epsilon)$ relative error. This means that each algorithm produces an arrangement realized by a set of pseudolines so that each pseudoline differs from the corresponding line relatively by at most $O(n\epsilon)$. We also show that there is a line arrangement algorithm with $O(n^2 \log n)$ running time and $O(\epsilon)$ relative error.

1 Introduction

We analyze the behavior of line arrangement algorithms in approximate arithmetic. Approximate arithmetic is a set of arithmetic operations defined on the real numbers that make relative error ϵ; this models floating point arithmetic. The input to a line arrangement algorithm is a set of n lines specified by real number coefficients. The output is a "combinatorial arrangement", a graph that specifies the incidence structure of the set of lines. An algorithm has "relative perturbation bound $f(n, \epsilon)$" if the combinatorial arrangement has a realization by pseudolines so that the distance from the pseudoline to the corresponding line is at most $f(n, \epsilon)$. A "pseudoline" is an unbounded simple curve; of course, it may not be straight. The metric is relative: the allowed distance from a point on the pseudoline to the line increases proportionally with the distance of the point from the origin.

We show that two natural line arrangement algorithms, an incremental algorithm [1][3] and a sweepline algorithm, have relative perturbation bound $O(n\epsilon)$. For the incremental algorithm, we require that the lines be inserted in order of increasing absolute value of slope. The running times of the incremental and sweepline algorithms are $O(n^2)$ and $O(n^2 \log n)$, respectively, as in the case of exact arithmetic. To obtain the perturbation bounds, the algorithms must be implemented in a specific manner (though the details are not complicated). The primitives used are the computation of the x-coordinate of the intersection of two lines and the comparison of two such computed values. For the sweepline algorithm, the most naive implementation is adequate with two conditions: the intersection coordinate computation must be numerically stable and an intersection computed for two adjacent lines must be forced to lie on or beyond the current position of the sweepline. For the incremental algorithm, the details involve exactly how the boundary of the current face is traversed to find an exiting edge.

There are two steps to the proof of the perturbation bound. First, we show that for every pair of lines l_i, l_j, the algorithms compute the x-coordinate x_{ij} of the intersection of the lines with accuracy $O(n\epsilon)$. This means that there is some y_{ij} so that the point (x_{ij}, y_{ij}) lies within a relative distance of $O(n\epsilon)$ of both lines. Note that the computed x_{ij} may be very far from the true value if the lines are close to parallel. Second, we show that a pseudoline arrangement can be constructed from the computed coordinates so that each pseudoline does not deviate too much from the corresponding line.

The difficulty of the first part of the proof is that there is more than one way to compute x-coordinates. We require that the computed x-coordinates along a line increase in the order given by the combinatorial arrangement. This implies, for example, that for any triple of lines l_i, l_j, l_k in slope-sorted order, that either $x_{ij} \leq x_{ik} \leq x_{jk}$ or $x_{jk} \leq x_{ik} \leq x_{ij}$. However, because of approximate arithmetic, there is no a priori guarantee that these inequalities hold. The solution is, in effect, brute force: change x-coordinates so that the monotonicity condition holds. The difficulty is to estimate the accuracy of the resulting coordinates. If we are allowed to choose the order in which the coordinates are computed, then errors don't grow: we give a "dynamic programming" algorithm that computes all intersections in time $O(n^2 \log n)$ with accuracy $O(\epsilon)$. Of course, this is slower than the best exact-arithmetic algorithm, and the algorithm probably is of little practical interest. For the two algorithms presented here, the order in which intersections are computed has already been determined. However, we are still able to prove a worst-case $O(n\epsilon)$ error bound.

Lines vs. pseudolines. Measuring "relative perturbation bound" is an example of backwards error analysis: the algorithm generates the correct answer for a slightly perturbed instance of the problem. In this case, the perturbation not only moves the lines but allows them to bend. We might desire instead that the algorithm produce an arrangement obtained by moving the lines but maintaining their straightness. Unfortunately, this appears to be difficult. The obvious approach is for the algorithm at each step to test whether or not the current arrangement is realizable by straight lines. However Mnev [17] has shown that this realizability question is as hard as deciding the existential theory of the reals, which is at least NP-hard.

Exact vs. approximate arithmetic. These algorithms could also be implemented in exact arithmetic, say exact integer arithmetic. Such an implementation either requires a restriction on the bit length of input coordinates or extended precision arithmetic. For example, the primitive test for the sweepline algorithm is to compare x-coordinates of intersections of lines. Each such x-coordinate is the ratio of two 2x2 determinants in the input coordinates, so the comparison requires intermediate values with four times the bit length of the input values. Clearly, it is more convenient to implement an algorithm using approximate arithmetic. A general discussion of the issues of exact vs. approximate arithmetic appears in several places[4][9][10][16].

Double Precision Geometry. Milenkovic [14] has results similar to the results in this paper. He gives a general technique for implementing line arrangement algorithms using floating point arithmetic with precision double the input precision. He observes that each coordinate of the point of intersection of two lines can be expressed as the ratio of two 2x2 determinants. If each determinant is computed exactly, and division rounds correctly, then the computation yields the correctly rounded point of intersection. Exact comparisons on intersection points can then be used to deduce orientation of triples of lines.

There are two reasons not to be entirely satisfied with double precision geometry. The first is practical. Consider the situation when it is not feasible to increase the precision of floating point arithmetic (for example, the input is already given as double precision floating point). To guarantee that the 2x2 determinants are computed exactly, the input data must be rounded so that the lower half of the significant digits of each mantissa is zero. In fact this rounding must occur when every input coordinate is expressed as a floating point number with a common exponent. This is a relative perturbation of size $\sqrt{\epsilon}$. Compare this with our perturbation bound of $O(n\epsilon)$. For currently interesting values of n and ϵ (say $n = 10^6$ and $\epsilon = 10^{-16}$) $O(n\epsilon)$ is smaller. Worse, the $\sqrt{\epsilon}$ perturbation happens for every input, while the $O(n\epsilon)$ is worst-case. We believe that the $O(n\epsilon)$ bound is extremely pessimistic, as is often the case for error bounds in numerical analysis. (The $O(n\epsilon)$ error bound can be achieved for a particular input if an adversary chooses the arithmetic operations. We do not know if this example can be achieved with true floating point.) Of course, if arithmetic with double the input precision is available, then double precision geometry is a perfectly practical technique for line arrangement algorithms.

The second reason to investigate other techniques is to find methods that will generalize well to harder problems. The analog of double precision geometry in dimension d requires d-fold precision. This is certainly a savings over the $d+1$-fold precision required for exact orientation tests or the $2d$-fold precision for coordinate computations, but it is still not attractive. Similarly, double precision geometry helps, but not enough, with the growth of coordinate bit length resulting from cascading (say computing the point of intersection of lines drawn through previous intersection points) or rotation. The approach of this paper is to assume that *every* computed numeric value may have some error. The goal of algorithm analysis is to reason about an algorithm even in the presence of numeric error, and in particular to bound how large the error can grow. We speculate that this approach is the most likely to extend to harder problems.

2 Geometric definitions

2.1 Geometry and arithmetic

Approximate arithmetic is a set of arithmetic operations that make relative error at most ϵ, e.g. $\hat{-}$ satisfies for all x, y there is η with $|\eta| < \epsilon$ so that $x \hat{-} y = (x-y)(1+\eta)$. The arithmetic operations needed here are addition, subtraction, multiplication, division, and square root. Comparisons $=$ and $<$ are exact, so for example sorting a set of reals can be done exactly. Arbitrary constants can be represented (it does not matter much whether they are exact or within relative error ϵ). Floating point arithmetic is a model of approximate arithmetic, except for the possibility of underflow and overflow, which can be handled in conceptually straightforward ways.

$L = \{l_1, \ldots, l_n\}$ is a fixed set of n lines. We let e denote Euclidean distance, thus $e(p, l)$ is the distance between point p and line l and $e(l)$ is the distance of point or line l from the origin. We write $l(x)$ to denote the point on line l with x-coordinate x. The *absolute slope* of a line is the absolute value of its slope.

Set $C = \max_{l \in L} e(l)$. A natural notion of "relative distance" between a point p and a line $l \in L$ is

$$e(p, l)/\max(e(p), C).$$

Thus relative distance is proportional to Euclidean distance inside a circle of radius C and decreases outside the circle. For technical reasons we use instead the definition

$$d_L(p, l) = e(p, l)/\max(|p_x|, C),$$

thus d_L is proportional to Euclidean distance within a vertical band of width $2C$, and decreases with x-coordinate outside. The definition is extended to a set P by $d_L(P, l) = \sup_{p \in P} d_L(p, l)$. For lines l_i and l_j and real x, it is also convenient to define $d_{ij}(x)$ by

$$|d_{ij}(x)| = \min(d_L(l_i(x), l_j), d_L(l_j(x), l_i))$$

so that $d_{ij}(x)$ is positive if $l_j(x)$ is above $l_i(x)$, and negative if not. Notice that if the absolute slope of l_i is larger than the absolute slope of l_j, then $d_L(l_j(x), l_i)$ is the minimum value.

Suppose the slope of line l_j is greater than the slope of line l_i, and let $\alpha \geq 0$. A real $\hat{x}$ is an *α-approximate coordinate of $l_i \cap l_j$* if for all x, $x \leq \hat{x}$ implies $d_{ij}(x) < \alpha$ and $x \geq \hat{x}$ implies $d_{ij}(x) > -\alpha$. The true intersection coordinate x_0 is an α-approximate intersection coordinate, as is any $\hat{x}$ in the interval containing x_0 satisfying $|d_{ij}(\hat{x})| < \alpha$.

The definition is slightly subtle in the case that l_i and l_j are close to parallel. Notice then that for both positive and negative x of large absolute value, $|d_{ij}(x)|$ is small, however $|d_{ij}(0)|$ may be large. Hence to establish that $\hat{x}$ is an α-approximate intersection coordinate, it is not sufficient to establish that $|d_{ij}(\hat{x})| < \alpha$. However, it is sufficient to establish that for all x between $\hat{x}$ and the true intersection x-coordinate x_0 that $|d_{ij}(x)| < \alpha$.

2.2 Geometric primitives

The representation of a line is a triple of reals (a, b, c), denoting $\{(x, y) : ax + by - c = 0\}$. These coefficients should be

procedure Parallel(l_1,l_2)
/* $l_1 = (a_1, b_1, c_1)$, $l_2 = (a_2, b_2, c_2)$, $|b_1| \geq |b_2|$ */
$D = a_2 - a_1 \left(\frac{b_2}{b_1}\right)$
if $|D| < 4\epsilon$
  then true
  else false

procedure Intersect(l_1,l_2)
/* $l_1 = (a_1, b_1, c_1)$, $l_2 = (a_2, b_2, c_2)$,
  Assume $\neg$Parallel(l_1, l_2) and $|b_1| \geq |b_2|$ */
$D = a_2 - a_1 \left(\frac{b_2}{b_1}\right)$
$E = c_2 - c_1 \left(\frac{b_2}{b_1}\right)$
$x_{12} = E/D$

Figure 1: Procedures Parallel and Intersect

normalized so that $-1 \leq a \leq 1$, $0 \leq b \leq 1$, and $\sqrt{a^2+b^2} \approx 1$. It is possible to guarantee that $|\sqrt{a^2+b^2} - 1| < 3\epsilon$ [16]; this implies that a and b differ from the correctly normalized values $a/\sqrt{a^2+b^2}$ and $b/\sqrt{a^2+b^2}$ relatively by at most about 3ϵ. The triple (a, b, c) denotes the line through (ac, bc) tangent to the circle of radius c about the origin, up to a slight adjustment because $\sqrt{a^2+b^2} \approx 1$.

The set of lines needs to be sorted by slope; it appears naively to be sufficient to sort the a's. However, because of rounding error in normalization, it is possible that two nearly-parallel lines l_i and l_j have coefficients (a_i, b_i, c_i) and (a_j, b_j, c_j) so that $a_i > a_j$ and $b_i > b_j$. Hence the order of l_i and l_j is ambiguous. Milenkovic[16] gives a technique for exactly sorting a set of lines by slope. In particular, the sorted sequence of lines $l_1, \ldots, l_n$ satisfies $a_1 \geq \ldots \geq a_k \geq 0 \geq a_{k+1} \ldots \geq a_n$ and $b_1 \leq \ldots \leq b_k$, $b_{k+1} \geq \ldots \geq b_n$. The coefficients may have to perturbed slightly to accomplish the sort; however the perturbation is at most $O(\epsilon)$. We assume that this sort and possible perturbation have been done.

Procedure Parallel appears in Figure 1. It determines if lines l_1 and l_2 are nearly parallel. If Parallel returns "true", then the angle between l_1 and l_2 is less than 6ϵ radians. Procedure Intersect(l_1, l_2) also appears in Figure 1. It computes the x-coordinate of $l_1 \cap l_2$. Both programs assume that l_1 has smaller absolute slope than l_2.

The proof of the following lemma uses standard techniques but the details are tricky. For completeness we include it. Set $\delta = 10\epsilon$.

Lemma 2.1 *If $\neg$Parallel(l_1, l_2), then Intersect(l_1, l_2) returns a δ-approximate coordinate of $l_1 \cap l_2$.*

Proof. Let D be $a_2 - a_1\left(\frac{b_2}{b_1}\right)$, E be $c_2 - c_1\left(\frac{b_2}{b_1}\right)$, x_{12} be E/D (the x-coordinate of the intersection), $\hat{x}_{12}, \hat{D}, \hat{E}$ be computed values, c be $\max(|c_1|, |c_2|)$. Note that $\hat{D} = D + \eta_D$, and $\hat{E} = E + \eta_E c$, and $\hat{x}_{12} = (1+\eta_x)\hat{E}/\hat{D}$, where $|\eta_D|, |\eta_E| < 4\epsilon$ and $|\eta_x| < \epsilon$.

Since $|b_1| \geq |b_2|$, l_1 has smaller absolute slope than l_2, so $e(l_1(x), l_2) = Dx - E$. Thus,

$$\begin{aligned} |e(l_1(x), l_2) - (\hat{D}x - \hat{E})| &= |\eta_D x + \eta_E| \\ &\leq 4\epsilon(|x| + c). \end{aligned}$$

Also,

$$\begin{aligned} |\hat{D}\hat{x}_{12} - \hat{E}| &= |\eta_x \hat{E}| \\ &\leq \epsilon\hat{E} \\ &\leq 2\epsilon c. \end{aligned}$$

Thus $|e(l_1(\hat{x}_{12}), l_2)| \leq 4\epsilon|\hat{x}_{12}| + 6\epsilon c$, from which it follows that $|v_{12}(\hat{x}_{12})| \leq 10\epsilon = \delta$. Furthermore, assume without loss of generality that D is negative (and hence so is $\hat{D}$). For $x \geq \hat{x}_{12}$,

$$\begin{aligned} \hat{D}x - \hat{E} &\leq \hat{D}\hat{x}_{12} - \hat{E} \\ &\leq 2\epsilon c. \end{aligned}$$

Hence, $e(l_1(x), l_2) \leq 4\epsilon|x| + 6\epsilon c$, from which it follows that $v_{12}(x) \leq \delta$. Similarly, for $x \leq \hat{x}_{12}$, $v_{12}(x) \geq -\delta$. This proves that $\hat{x}_{12}$ is a δ-approximation to x_{12}. □

2.3 Pseudolines and arrangements

A *geometric pseudoline* is an unbounded x-monotonic curve (i.e. the intersection of any vertical line with the pseudoline is a point or segment). This definition is a slight specialization of the notion from [6]. Two pseudolines *overlap* if they are identical over some interval in x of positive length. Two pseudolines *cross* at x if they are strictly ordered vertically in neighborhoods to the left and right of x, and the orders are different. Note two pseudolines may share a point or vertical segment without crossing. A *geometric pseudoline arrangement* is a set of n pseudolines $p_1, \ldots, p_n$ so that no pair of pseudolines overlap and so that each pair of pseudolines cross at most once.

A *combinatorially planar graph* is a graph with the edge order around each vertex specified and having a planar embedding respecting the edge order. Easy combinatorial conditions can guarantee that a graph is a combinatorially planar graph [4] [5]. A *combinatorial line* χ in a graph is a simple path in the graph. A *combinatorial arrangement* is a combinatorially planar graph and a partition of its edges into a simple cycle κ and a set of combinatorial lines $\chi_1, \ldots, \chi_n$ so that (1) all the combinatorial lines lie to the same side of κ and have endpoints on κ and (2) each pair of combinatorial lines either do not intersect or intersect at a unique vertex at which they cross. We require that the combinatorial arrangement be *simple*, that is, distinct pairs of lines cross at distinct vertices. A combinatorial arrangement is *labeled* if each vertex is labeled with a real number (including ∞ and $-\infty$), its *x-coordinate*, so that (1) along each combinatorial line the x-coordinates are monotonically nondecreasing from $-\infty$ to ∞ and (2) all vertices labeled $-\infty$ are adjacent on κ, as are all vertices labeled ∞. In Figure 2, we see a combinatorial arrangement with the intersection of χ_1 and χ_4 labeled.

Choose an embedding of the combinatorial arrangement so that all combinatorial lines are inside κ; this embedding is unique up to homeomorphism since the edge order around each vertex is specified. Direct the edges of each combinatorial line from $-\infty$ to ∞, and similarly direct the two edges of κ with endpoints $-\infty$ and ∞. There is no need to direct the other edges of κ. It is easy to check that each bounded face has boundary consisting of two directed chains plus possibly an edge of κ (otherwise there would be an alternation of labels $-\infty, \infty, -\infty, \infty$ along κ). One chain has the face to its right; it is the *upper boundary* of the face; the other

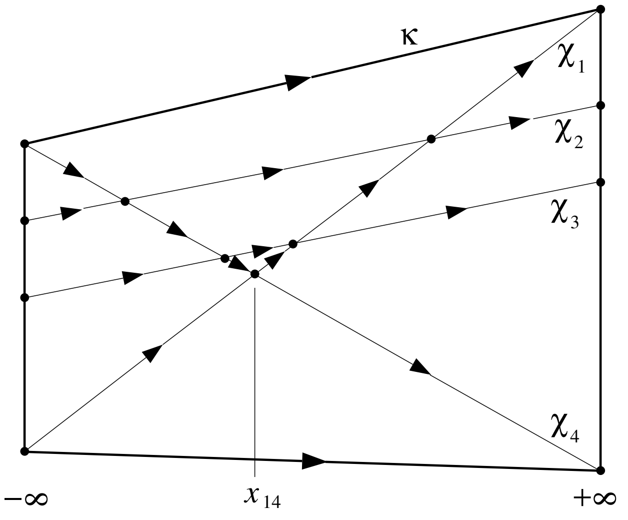

Figure 2: Combinatorial Arrangement

chain is the *lower boundary* of the face. Pick a real x distinct from any vertex label. Each combinatorial line χ_i has a unique edge ξ_i whose endpoint labels span x. Say edge ξ_i is *combinatorially directly above* edge ξ_j if ξ_i lies on the upper boundary of some face with ξ_j on the lower boundary. The *combinatorial order at* x is the induced order on combinatorial lines obtained by the transitive closure of this relation on edges. It is easy to check that this is a linear order on all combinatorial lines. Clearly the combinatorial order is the same for any x in an interval not containing a vertex label.

The line arrangement algorithms produce a combinatorial arrangement from a set of (geometric) lines. There is a one-to-one correspondence between the combinatorial lines in the arrangement and the geometric lines. Strictly speaking, "combinatorial order" is a relation on combinatorial lines. However, we will abuse notation and say that a geometric line is *combinatorially above* another geometric line at x if the combinatorial order holds on the corresponding combinatorial lines in the arrangement. Clearly, there is a distinction between this combinatorial relation and the geometric relation of one line being above another at x. The distinction should always be clear from context.

A geometric pseudoline arrangement $p_1, p_2, \ldots, p_n$ *realizes* a combinatorial arrangement with combinatorial lines $\chi_1, \chi_2, \ldots, \chi_n$ if the incidence structures are the same; if the arrangement is labeled the pseudolines must cross at the labeling x-coordinate.

3 Algorithms

We describe the implementation details of the two arrangement algorithms. The input to either algorithm is a set of n lines. The output is a labeled combinatorial arrangement. The arrangement can be represented using for example the quad-edge structure of Guibas and Stolfi [8]; we omit the routine details of maintaining the arrangement.

The sweepline algorithm. Use a vertical sweepline moving from left to right. Maintain a list of the lines in vertical order and a priority queue of the x-coordinates of intersections of adjacent lines. A single step is to move to the next intersection and swap the order of the intersecting lines in the list of lines. The only subtlety is in the computation of intersection x-coordinates: it may be that the computed intersection lies to the left of the current position of the sweepline. To guarantee monotonicity, such an intersection is reset to the current position of the sweepline and occurs immediately.

The incremental algorithm. The implementation is based on explicit computation of the x-coordinates of vertices. The arrangement is built by adding one line at a time, in order of increasing absolute slope. A new line l is inserted from left to right; without loss of generality assume that l is of positive slope. The basic step is to determine the boundary edge of the current face of the arrangement through which l exits. The current face has an upper boundary and a lower boundary; notice that the new line l has already intersected all lines of the lower boundary and none of the lines of the upper boundary. Let x_c be the x-coordinate at which l entered the current face. Line l leaves through the first edge e of the upper boundary so that the x-coordinate x_l of the left endpoint of e satisfies $x_l \leq x_c$ and so that the computed x-coordinate of the intersection of l with the line through e is at most the right endpoint of e. If no such edge is found, l exits the rightmost upper boundary edge at an x-coordinate equal to its right endpoint, and we say the intersection is *forced.* To guarantee monotonicity of x-coordinates, the x-coordinate at which l leaves the current face must be at least the maximum of x_c and the left endpoint of the exiting edge. If the x-coordinate is not this large, set it to this value.

The sweepline algorithm may overestimate an intersection x-coordinate (compared with the value returned by Intersect), but it does not underestimate x-coordinates. The incremental algorithm may both overestimate and underestimate x-coordinates.

4 Error analysis

A combinatorial arrangement has *coordinate accuracy* α if every intersection x-coordinate is an α-approximate intersection coordinate.

Theorem 4.1 *The sweepline and incremental algorithms produce combinatorial arrangements with coordinate accuracy* $O(n\epsilon)$.

For this section assume that the lines are numbered in order of increasing slope, i.e. $i < j$ iff l_i has smaller slope than l_j. Hence the lines are ordered 1 to n from top to bottom at $-\infty$ in x. Set $y_{ij}(x)$ to be the distance from $l_i(x)$ to $l_j(x)$ with the same sign convention as $d_{ij}(x)$, i.e., $y_{ij}(x)$ is positive if $l_j(x)$ is above $l_i(x)$. Define the *normalized slope*, m, of a line of slope s as $m = \sqrt{1+s^2}$; thus if a point is distance d from a line of normalized slope m, the vertical distance from the point to the line is md. Set m_i to be the normalized slope of line l_i. Notice that $y_{ij}(x) = d_{ij}(x)\max(m_i, m_j)\max(|x|, C)$. For lines l_i, l_j, let x_{ij} be the true intersection coordinate and $\hat{x}_{ij}$ the computed coordinate.

We first bound how much x-coordinates can be underestimated. It is clear that if the x-coordinates are computed by the sweepline algorithm, then $i < j$ and $x \geq \hat{x}_{ij}$ implies $d_{ij}(x) < \delta$.

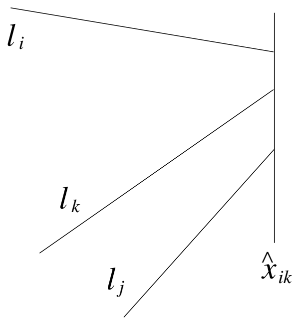

Figure 3: Actual Positions of Lines l_i, l_j and l_k

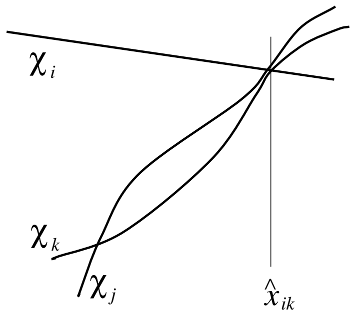

Figure 4: Combinatorial Structure Calculated for Lines l_i, l_j and l_k

Lemma 4.2 *Suppose the coordinates are computed by the incremental algorithm. Let $i < j$. If $x \geq \hat{x}_{ij}$, then $d_{ij}(x) < n\delta$. If l_i and l_j are combinatorially adjacent at x and $x > \hat{x}_{ij}$, then $d_{ij}(x) < \delta$.*

Proof. Say the *level* of the unforced intersection of lines l_i and l_j, denoted $\lambda(i, j)$, is 1. If the intersection of l_i and l_j is forced, then up to interchanging i and j, line l_j was forced to leave a face whose rightmost upper and lower boundary edges are edges along l_i and some line l_k. Define $\lambda(i, j) = \lambda(i, k)+1$. Suppose l_i and l_j are the r_ith and r_jth lines to be inserted; then it is easy to check that $\lambda(i, j) \leq \max(r_i, r_j)$.

We claim that if $\lambda(i, j) \leq r$ and $x \geq \hat{x}_{ij}$, then $d_{ij}(x) < r\delta$ (where i, j satisfy $i < j$) and if $x > \hat{x}_{ij}$ and l_i and l_j are combinatorially adjacent at x, then $d_{ij}(x) < \delta$. This establishes the lemma. The proof is by induction on the order in which lines are inserted, and for a particular line in order of intersections along the line. The basis case is immediate. So suppose l_j is the next line to be inserted, the next intersection is with line l_i, and $\lambda(i, j) = s$. If $s = 1$ then both parts of the claim are immediate, so assume $s > 1$. Then for some line l_k, l_j was forced to leave the face with rightmost upper and lower boundary edges along lines l_i and l_k, and $\lambda(i, k) = s - 1$. Notice l_j has steeper absolute slope than l_i and l_k; without loss of generality we have l_i, l_k, l_j in order of increasing slope. Figure 3 shows what the actual positions of l_i, l_j and l_k may be. Figure 4 shows what the combinatorial structure resulting from the forced intersection. Clearly $\hat{x}_{ij} = \hat{x}_{ik}$. By the induction hypothesis, for $x \geq \hat{x}_{ik}$ we have $d_{ik}(x) < (s - 1)\delta$. Now l_j must have intersected l_k previously; if $\hat{x}_{kj} < \hat{x}_{ik}$ then again by the induction hypothesis, using l_j, l_k combinatorially adjacent, $d_{kj}(x) < \delta$. If $\hat{x}_{kj} = \hat{x}_{ik}$, then it is easy to see that the intersection of l_j and l_k must have been unforced, and again $d_{kj}(x) < \delta$. We have $y_{ij}(x) = y_{ik}(x)+y_{kj}(x)$. Since $m_j \geq m_i$ and $m_j \geq m_k$, we have

$$\begin{aligned} y_{ik}(x) &= d_{ik}(x)\max(m_i, m_k)\max(|x|, C), \\ &\leq d_{ik}(x)m_j \max(|x|, C), \\ y_{kj}(x) &= d_{kj}(x)\max(m_k, m_j)\max(|x|, C), \\ &\leq d_{kj}(x)m_j \max(|x|, C). \end{aligned}$$

Hence we get $y_{ij}(x) < m_j \max(|x|, C)s\delta$, so $d_{ij}(x) < s\delta$. For the second part of the claim, note that for $x > \hat{x}_{ik}$, lines l_j and l_i can never become combinatorially adjacent, since both have crossed k and k currently lies between them. $\square$

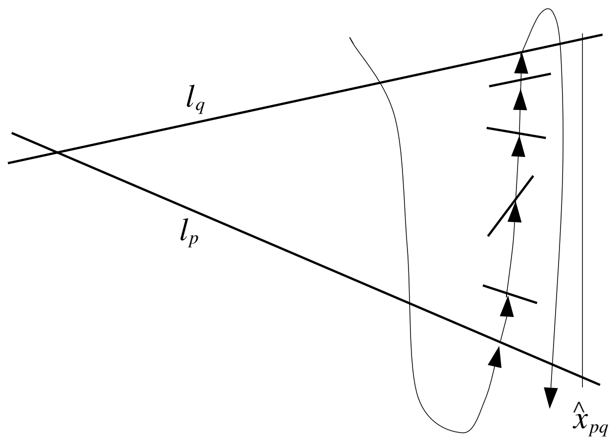

Figure 5: Combinatorial Structure on Lines from l_p and l_q

Proof. (of theorem 4.1) Pick particular lines l_p, l_q with $p < q$ and set $m = \max(m_p, m_q)$. We show that for all $x < \hat{x}_{pq}$, $y_{pq}(x) < 8nm\delta \max(|x|, C)$; from this it follows that $\hat{x}_{pq}$ is an $8n\delta$-approximate intersection coordinate. For simplicity, we drop the "$\max(|x|, C)$" in subsequent distance calculations; in all cases the distance being measured is along a vertical segment, so there is no ambiguity about the value of x.

Suppose $x \neq \hat{x}_{ij}$, any i, j, so that combinatorial order is defined at x. Consider lines in combinatorial order from l_p to l_q at x. In Figure 5, the arrows indicate combinatorial order of the lines from highest to lowest at some value of x. As we see, the combinatorial order may not agree with the geometric order. For consecutive lines l_i and l_j, we have $y_{ij}(x) < 0$ if the combinatorial and geometric orders agree, i.e., $l_j(x)$ is below $l_i(x)$, and $y_{ij}(x) > 0$ if not. Also we have

$$y_{pq}(x) = \sum y_{ij}(x),$$

where the sum is taken over all adjacent pairs of lines l_i, l_j in the combinatorial order from l_p to l_q. If $y_{ij}(x) > 0$, then there are two possibilities: either l_i and l_j crossed prematurely, i.e., $j < i$ and $x < x_{ij}$, or l_i and l_j should have crossed but did not, i.e. $i < j$ and $x > x_{ij}$. In either case $y_{ij} < \max(m_i, m_j)\delta$, in the first case by the lemma and in the second case because no intersection was discovered for adjacent lines.

Say a line l_i is *gentle* if $m_i \leq 2m$ and is *steep* otherwise. Suppose all lines in the combinatorial order from l_p to l_q

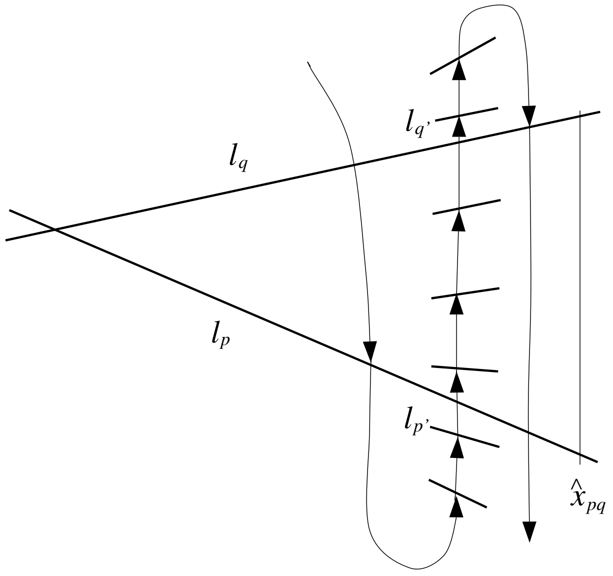

Figure 6: Combinatorial Chain from $l_{p'}$ to $l_{q'}$

are gentle. Then for consecutive l_i, l_j, $y_{ij}(x) < 2m\delta$, and $y_{pq}(x) < 2m\delta k$, where k is the total number of gentle lines.

Suppose the coordinates were computed by the incremental algorithm. Then the proof is finished, because only gentle lines are present when $\hat{x}_{pq}$ is computed.

Otherwise, suppose that the coordinates were computed by the sweepline algorithm. Suppose that l_r is steep and l_r lies combinatorially between l_p and l_q. One of $r < p < q$ or $p < q < r$ holds; suppose the latter. The algorithm must have decided that l_r intersects l_q at some x-coordinate $\hat{x}_{qr}$. Define the *dead time* for line l_r to be the interval $(\hat{x}_{qr}, x'_{qr})$, where x'_{qr} is defined so that $l_r(x'_{qr})$ is δm_r vertically above $l_q(x'_{qr})$.

The dead time for l_r has two properties. The first is that at any x' after the dead time, the algorithm must have decided that l_r intersects any combinatorially adjacent gentle line l_g so that $l_g(x')$ is at or below $l_q(x')$. This is simply because of size of the vertical distance between l_g and l_q.

The second property is that the net change in y_{pq} during l_r's dead time is at most $8\delta m$. To see this, note that the net change in the vertical distance between l_r and l_q during l_r's dead time is at most $2\delta m_r$, since $l_r(\hat{x}_{qr})$ can be at most δm_r below $l_q(\hat{x}_{qr})$. It follows easily from $m_r \geq 2m \geq 2m_q$ that the absolute slope of l_r is at least twice the absolute slope of l_q, hence the net change in the y-coordinate of l_r during l_r's dead time is at most $4\delta m_r$. Finally, this easily implies that the net change in the y-coordinate of l_q during l_r's dead time is $4\delta m_q \leq 4\delta m$. Similarly the net change in the y-coordinate of l_p is $4\delta m$, so the net change in y_{pq} is $8\delta m$.

If l_r is steep and $r < p < q$ holds, then the definition of dead time is similar. At any x' after the dead time, the algorithm must have decided that l_r intersects any combinatorially adjacent gentle line l_g so that $l_g(x')$ is at or above $l_p(x')$. Similarly, the net change in y_{pq} during the dead time is at most $8\delta m$.

To finish the proof, we need to show that $y_{pq}(x) < 8n\delta m$. First suppose x is not a dead time for any steep line. Consider the lines is combinatorially order from l_p to l_q. Let $l_{q'}$ be the first line so that $l_{q'}(x)$ is at or above $l_q(x)$, and let $l_{p'}$ be the line before $l_{q'}$ as close as possible to $l_{q'}$ so that $l_{p'}(x)$ is at or below $l_p(x)$. The chain of lines from $l_{p'}$ to $l_{q'}$ is illustrated in Figure 6. We claim that every line from $l_{p'}$ to $l_{q'}$ inclusive is gentle; clearly any line in between is gentle. It cannot be that $l_{p'}$ and $l_{q'}$ are both steep with no lines in between, because the vertical distance between them is sufficient to force an intersection. It cannot be that $l_{p'}$ is steep with a gentle successor, because again the vertical distance is sufficient to force an intersection. Similarly, $l_{q'}$ cannot be steep. By the previous analysis, $y_{pq}(x) \leq y_{p'q'}(x) \leq 2k\delta m$, where k is the number of gentle lines.

Now suppose x is a dead time. Choose $x' < x$ as large as possible so that x' is not dead. Then y_{pq} can increase by at most $8\delta m$ per steep line from $y_{pq}(x')$, so $y_{pq}(x) < 8n\delta m$. $\square$

5 Realizability

Theorem 5.1 *Let $L = \{l_1, \ldots, l_n\}$ be a set of lines and A be a combinatorial arrangement with coordinate accuracy α. Then there is a set of pseudolines $P = \{p_1, \ldots, p_n\}$ realizing A so that $d_L(p_i, l_i) \leq \alpha$, for $i = 1, \ldots, n$.*

Proof. We first give a slightly simplified construction of the pseudoline arrangement, allowing overlap. Assume that the lines are indexed in order of increasing absolute slope; the pseudolines are defined in the same order. Set $p_1 = l_1$. For p_k, consider the open intervals of the real line obtained by deleting the x-coordinates $\hat{x}_{ij}$, $1 \leq i, j \leq k$. Choose such an interval I. Let l_i, l_j be the first lines combinatorially below and above l_k in interval I in the combinatorial arrangement restricted to the first k lines. In interval I, choose p_k as close as possible to l_k so it stays on or above p_i and on or below p_j. If there is no line l_i below l_k, then there is no lower restriction on p_k, and similarly if there is no line l_j above l_k. At each x-coordinate $\hat{x}_{ij}$, p_k consists of a vertical segment connecting the pieces of p_k to either side of $\hat{x}_{ij}$.

We claim $d_L(p_k, l_k) < \alpha$. First, pick x not equal to $\hat{x}_{ij}$ for any $1 \leq i, j \leq k$. We show $d_L(p_k(x), l_k) < \alpha$. We say p_k is *forced above* l_k at x, *naturally placed* at x, or *forced below* l_k at x as $p_k(x)$ is above, equal to, or below $l_k(x)$. Suppose $p_k(x)$ is forced above l_k at x. Then for some $k' < k$, $p_k(x) = p_{k'}(x)$, where l_k is combinatorially above $l_{k'}$ at x. Either $l_{k'}$ is naturally placed at x, or it is forced above or below some $l_{k''}$, and so forth. Consider the sequence of "forced above" and "forced below". The first case is that every successive pseudoline is forced above until some pseudoline p_j is naturally placed at x. Then l_k is combinatorially above l_j at x but $l_k(x)$ is below $p_k(x) = p_j(x) = l_j(x)$. Since $\hat{x}_{jk}$ is an α-approximate intersection coordinate and l_k has steeper absolute slope than l_j, $d_L(p_k(x), l_k) = d_L(l_j(x), l_k) < \alpha$. The second case is that every successive pseudoline is forced above until some pseudoline p_j is forced below. Notice then that $l_j(x)$ is above $p_k(x) = p_j(x)$ is above $l_k(x)$. Hence $d_L(p_k(x), l_k) < d_L(l_j(x), l_k)$, and the rest of the proof is as before. Finally, suppose $x = \hat{x}_{ij}$, some $1 \leq i, j \leq k$. Then $p_k(x)$ consists of a vertical segment. However, d_L is convex along a vertical line. Hence its maximum is attained at one of the endpoints of the vertical segment, for which the previous argument applies.

As described, the constructed pseudolines may overlap. However, it is clear that when p_k is constructed in interval I, we can choose it instead to be slightly above p_i and slightly below p_j while still maintaining the bounds on distance from l_k. □

A pseudoline is *monotonic* with respect to a line if its projection on the line is one-one. It is possible to show that the combinatorial arrangement has a monotonic pseudoline realization, that is, so that each pseudoline is monotonic with respect to its corresponding line. To achieve this it is necessary to slightly increase the allowed deviation of the pseudoline from the line. It is also possible to explicitly compute a set of y-coordinates for the intersection vertices. This can be done so that there is a monotonic realization of the arrangement by pseudolines passing through the computed coordinates. The details of these two claims will appear in the full paper.

6 A more accurate algorithm

It is possible to obtain a line arrangement algorithm with $O(n^2 \log n)$ running time and $O(\epsilon)$ relative error. The algorithm is probably not of practical interest but is included for completeness.

The algorithm represents a line as a pair of reals (a, b), denoting $\{(x, y) : y = ax + b\}$. Vertical lines can be easily handled as a special case, though the details are omitted here. The set of lines $L = \{l_1, \ldots, l_n\}$ should be indexed in order of decreasing a, that is, in order of increasing slope. The x coordinate of the intersection of lines l_i and l_j is

$$x_{ij} = \frac{b_i - b_j}{a_j - a_i}.$$

The intersection coordinate $\hat{x}_{ij}$ computed using this formula is a *strong approximation* to x_{ij}, that is, $\hat{x}_{ij}$ satisfies $\hat{x}_{ij} = x_{ij}\eta$, where $(1-\epsilon)^3 < \eta < (1+\epsilon)^3$.

Suppose $i < j < k$, so lines l_i, l_j, l_k are in slope-sorted order. Then the true intersection x-coordinates satisfy $x_{jk} \leq x_{ik} \leq x_{ij}$ or $x_{ij} \leq x_{ik} \leq x_{jk}$. The directly-computed x-coordinates may not satisfy either order. Let $\{a, b\}$ denote the interval $[\min(a, b), \max(a, b)]$ and $\mathrm{snap}(y, I)$ be y if y lies in the interval I, and the closer endpoint of the interval if not. Then surely $\hat{x}'_{ik} = \mathrm{snap}(\hat{x}_{ik}, \{\hat{x}_{ij}, \hat{x}_{jk}\})$ satisfies $\hat{x}_{jk} \leq \hat{x}'_{ik} \leq \hat{x}_{ij}$ or $\hat{x}_{ij} \leq \hat{x}'_{ik} \leq \hat{x}_{jk}$. However, $\hat{x}'_{ik}$ is a strong approximation to x_{ik}. This is easily proved directly; a proof appears as Lemma 3.4(3d) of [4]. (The statement in [4] is given in the dual form involving the slopes of segments connecting points.)

Consider the following dynamic programming algorithm. Each variable x_{ij} should be initialized with the directly-computed x-coordinate of $l_i \cap l_j$.

for $k = 2, \ldots, n$
  for $i = k-1, \ldots, 1$ **by** -1
    $x_{ik} = \mathrm{snap}(x_{ik}, \bigcap_{i<j<k}\{x_{ij}, x_{jk}\})$

Note that if $i = k - 1$, the intersection is over an empty set of intervals; in this case the snap should not change the value of x_{ik}. Two properties follow easily: for $i < k$, x_{ik} is a strong approximation of the true x-coordinate of $l_i \cap l_k$, and for $i < j < k$, either $x_{jk} \leq x_{ik} \leq x_{ij}$ or $x_{ij} \leq x_{ik} \leq x_{jk}$. It is easy to construct a labeled combinatorial arrangement from the $\{x_{ij}\}$ in time $O(n^2)$. A naive implementation of this algorithm has running time $O(n^3)$, since there is an implicit loop to compute the interval intersection.

Theorem 6.1 *There is a line arrangement algorithm that runs in time $O(n^2 \log n)$ and computes a strong approximation to each intersection x-coordinate.*

Proof. For each line maintain a balanced binary tree of the intersection x-coordinates in order along the line. The computation of x-coordinates is done in the same order as the dynamic programming algorithm. The computation of x_{ik} requires the computation of the intersection $\bigcap_{i<j<k}\{x_{ij}, x_{jk}\}$; this can be done in time $O(\log n)$ as follows. The left endpoint of this interval is the maximum value of x_{ij} subject to $x_{ij} < x_{jk}$; this can be found by binary search in the tree for line i. Similarly the right endpoint is the minimum value of $x_{ij'}$ subject to $x_{kj'} < x_{ij'}$. □

References

[1] B. Chazelle, L.J. Guibas, D.T. Lee. The power of geometric duality, *24th Annual Symposium on the Foundations of Computer Science*, pages 217–225, IEEE, 1983.

[2] H. Edelsbrunner and L. Guibas. Topologically Sweeping and Arrangement *Proceedings of the Symposium on Computational Geometry*, pages 389–403, ACM, June 1986.

[3] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing Arrangements of Lines and Hyperplanes with Applications, *24th Annual Symposium on the Foundations of Computer Science*, pages 83–91, IEEE, 1983.

[4] Steven Fortune. Stable Maintenance of Point-Set Triangulation in Two Dimensions, manuscript, 1990, ATT Bell Laboratories. An abbreviated version appeared in *30th Annual Symposium on the Foundations of Computer Science*, IEEE, October 1989.

[5] J. L. Gross, T. W. Tucker. *Topological Graph Theory*, John Wiley and Sons, 1987.

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

[7] Leonidas Guibas, David Salesin, and Jorge Stolfi. Epsilon Geometry: Building Robust Algorithms from Imprecise Computations, *Proceedings of the Symposium on Computational Geometry*, pages 208–217, ACM, June 1989.

[8] L. Guibas, J. Stolfi. Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams, *ACM Transactions on Graphics*, 4:74–123, 1985.

[9] C. Hoffmann. The Problems of Accuracy and Robustness in Geometric Computation, *Computer*, Vol. 22, No. 3, March 1989, pp. 31-42.

[10] C. Hoffmann, J. Hopcroft, M. Karasick. Towards Implementing Robust Geometric Computations, *Proceedings of the Fourth Annual Symposium on Computational Geometry*, 1988, pp. 106-117.

[11] J. Hopcroft, P. Kahn. A Paradigm for Robust Geometric Algorithms, Cornell University Technical Report TR 89-1044, October 1989.

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

[13] Z. Li, V. Milenkovic. Constructing Strongly Convex Hulls using Exact or Rounded Arithmetic, *Proceedings of the Sixth Symposium on Computational Geometry*, pages 235–243, ACM, June 1990.

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

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

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

[17] N.E. Mnev. The Universality theorems on the classification problem of configuration varieties and convex polytopes varieties, O.Y. Viro, ed., it Topology and Geometry - Rohlin Seminar, LN Mathematics, 1346, pp. 527-544, Springer-Verlag, 1989.

[18] K. Sugihara, M. Iri, Geometric Algorithms in Finite-Precision Arithmetic, Research Memorandum RMI 88-10, University of Tokyo, September, 1988.

[19] K. Sugihara, M. Iri, Construction of the Voronoi Diagram for One Million Generators in Single Precision Arithmetic, First Canadian Conference on Computational Geometry, Montreal, Canada, 1989.