Purdue University
Purdue e-Pubs

Computer Science Technical Reports | Department of Computer Science

2003

An Inconsistency Sensitive Arrangement Algorithm for Curve Segments

Victor Milenkovic

Elisha Sacks
Purdue University, eps@cs.purdue.edu

Report Number:
03-033

Milenkovic, Victor and Sacks, Elisha, "An Inconsistency Sensitive Arrangement Algorithm for Curve Segments" (2003). *Computer Science Technical Reports.* Paper 1582.
http://docs.lib.purdue.edu/cstech/1582

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

AN INCONSISTENCY SENSITIVE ARRANGEMENT ALGORITHM FOR CURVE SEGMENTS

Victor Milenkovic
Elisha Sacks

CSD TR #03-033
November 2003

An inconsistency sensitive arrangement algorithm for curve segments

Victor Milenkovic
University of Miami

Elisha Sacks
Purdue University

Abstract

We present a robust arrangement algorithm for algebraic curves based on floating point arithmetic. The algorithm performs a line sweep, tests the consistency of each sweep update, and modifies the input to prevent inconsistent updates. The output arrangement is realizable by semi-algebraic curves that are close to the input curves. We present a new performance model for robust computational geometry in which running times and error bounds are expressed in terms of the number of input inconsistencies. An inconsistency is a combinatorial property that is derivable with a given set of numerical algorithms, but that is not realizable. The running time of the arrangement algorithm is $O((n+N)\log n + k(n+N)\log n)$ for n curves with N intersection points and with $k = O(n^3)$ inconsistencies. The distance between the realization curves and the input is $O(\epsilon + kn\epsilon)$ where ϵ is the curve intersection accuracy. The output size is always the standard $O(n+N)$. We show experimentally that k is zero for generic inputs and is tiny even for highly degenerate inputs. Hence, the algorithm running time on real-world inputs equals that of a standard sweep and the realization error equals the curve intersection error.

1 Introduction

The correctness and asymptotic complexity of computational geometry algorithms is proven in the RAM model where real arithmetic is an exact, unit-cost operation. The robustness problem is to implement these algorithms in computer arithmetic. Floating point implementations match the asymptotic complexity because rounded arithmetic has unit cost, but occasionally generate incorrect combinatorial structures. Errors occur when the algorithm branches on the sign of an expression, the expression is near zero, and floating point yields the wrong sign. Exact implementations (based on arbitrary precision arithmetic) are correct, but violate the unit-cost arithmetic assumption of the RAM model. Exact arithmetic is slower than floating point and the slowdown is exponential in the algebraic degree of the expression, as measured by tree depth.

There are two main approaches to robustness. One approach seeks to make exact arithmetic fast. Predicates are resolved in floating point whenever possible, and their degree is minimized. The challenge is to defuse the intrinsic exponential cost of exact arithmetic. The competing approach seeks to make floating point correct in the scientific computing sense, meaning that the program output is correct for an input that is close to the actual input. The challenge is to ensure correctness at an acceptable computational cost.

We extend the floating point definition of correctness by expressing algorithm running times and error bounds in terms of the number of input inconsistencies. An inconsistency is a combinatorial property that is derivable with a given set of numerical algorithms, but that is not realizable. The motivation for our definition is that best-practices numerical computing is rarely inconsistent. Inconsistency sensitive algorithms can match the asymptotic running time of RAM algorithms on real-world inputs where the number of inconsistencies is tiny. The algorithms must also satisfy polynomial bounds in the number of inconsistencies to ensure acceptable performance on any input.

We present an arrangement algorithm for algebraic curves and prove it robust in our model. The algorithm performs a line sweep, tests the consistency of each sweep update, and modifies the input to prevent inconsistent updates. The output arrangement is realizable by semi-algebraic curves that are close to the input curves. The running time is $O((n+N)\log n + k(n+N)\log n)$ for n curves with N intersection points and with $k = O(n^3)$ inconsistencies. The distance between the realization curves and the input is $O(\epsilon + kn\epsilon)$ where ϵ is the curve intersection accuracy. The output size is always the standard $O(n+N)$.

Figure 1 illustrates a potential inconsistency. Lines a, b, c form a small triangle. The horizontal order of the intersection points is $p < q < r$ and the vertical order of the segments is $a < b < c$ to the left of p. Numerical error in the line intersection algorithm can lead to five other p, q, r orders. The order $q < p < r$ is inconsistent because it implies that a, c are separated by b at their intersection. Three other orders are inconsistent analogously, while $r < q < p$ is consistent and incorrect. The example is canonical in that every inconsistency arises from three curves that intersect pairwise.

We validate the inconsistency sensitive model via extensive numerical experiments. We find that k is zero for generic inputs and is tiny even for highly degenerate inputs. Moreover, the actual cost of an inconsistency is $O(\log n)$ because the worst-case cost of $O((n+N)\log n)$ represents a long chain of very rare events. Hence, the algorithm running time on real-world inputs equals that of a standard sweep, and the realization error equals the curve intersection error.

The rest of the paper is organized as follows. Section 2 surveys prior work on robust com-

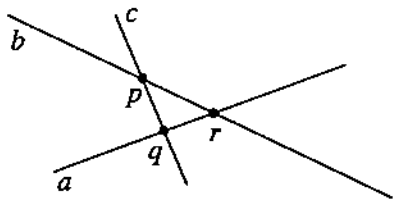


Figure 1: Potentially inconsistent line intersection.

putational geometry. Sections 3 and 4 describe the robust arrangement algorithm and prove the realizability error bound. Section 5 describes the numerical algorithms for manipulating and intersecting algebraic curves. Section 6 presents the experimental results. Section 7 discusses future work on robust algorithms for solid modeling, part layout, and mechanical design. These problems have high algebraic degree and many degeneracies, so the inconsistency sensitive approach appears more promising than the exact approach.

2 Prior work

Prior work in robust computational geometry consists of the exact method, the floating point method, and the perturbation method.

The exact method develops algorithms that perform exact computations on real numbers. Inputs are converted to rational numbers that are manipulated with rational arithmetic. Algebraic numbers are modeled explicitly. Geometric objects are modeled as semi-algebraic varieties and are manipulated using algebraic geometry. Floating-point filters and other techniques are used to speed up exact computation, but the worst case is exponential in expression depth. Chee Yap provides an excellent web resource for this approach (http://cs.nyu.edu/exact), which he calls "Exact Geometric Computation."

Keyser's research [16, 15, 17, 18, 19] represents the state of the art in the exact calculation of arrangements. Using the LiDIA system [4] (although he is currently switching to LEDA [23]), LAPACK [2], and PRECISE [20] (following Aberth and Schaefer's Range library [1]), Keyser's MAPC comes within an order of magnitude of the running time of the purely numerical (and non-robust) BOOLE [21] for the construction of the boundary representation of a CSG tree of low-degree sculptured solids. The algorithm does not handle degenerate inputs.

The floating point method has led to algorithms for a few problems [5, 8, 22, 6], but has not proven broadly applicable. Researchers aimed for algorithms that achieved correctness at no asymptotic cost. We believe that inconsistency sensitivity is a more realistic model that will prove broadly applicable.

Other approaches achieve robustness by perturbing the coordinates before [29, 13, 24] or after [10, 14, 11, 9, 25, 7, 12] performing a construction. These techniques are limited to line segments, circles, triangles, and spheres. Although they achieve polynomial running time, they violate "If it ain't broke, don't fix it." They perturb all ill-conditioned coordinates, not just the much less frequent inconsistent coordinates, and this results in much larger error, especially as operations are cascaded. The sole exception is shortest path rounding [25], but that only works for line segments in the plane.

3 Arrangement algorithm

The input to the arrangement algorithm is n curve segments in the (x, y) plane. A segment is a tail point, a head point, and a function $y = f(x)$. The tail x is less than the head x, except for vertical segments where they are equal and f is not defined. A trapezoidal decomposition is computed by a line sweep. A planar embedding is computed by a separate algorithm.

Roots Non-vertical segments a and b have a vertical order at every x in the intersection of their domains. The order flips solely at a/b intersection points, which are roots of $a.f(x) = b.f(x)$. Let $r_1, \ldots, r_m$ be the roots where the order flips, which include all simple roots and some multiple roots. The a/b order to the left of r_1 and the roots $r_1, \ldots, r_m$ are stored in a data structure called a root list.

Root list construction is performed numerically, as described in Section 5. The worst-case cost per root is polynomial in the system degree, while the actual cost is roughly constant. This cost is omitted from the running time bounds. The arrangement algorithm makes no assumptions about the number of roots or about their accuracy. Root lists are constructed only when they are needed by the arrangement algorithm.

Sweep We extend the standard line sweep algorithm to ensure consistency. The sweep events are segment tails, segment heads, and roots. Vertical segments are processed by the embedding algorithm. A priority queue of events is initialized with the other segment tails and heads. The events are processed in x order with ties broken by processing heads, then roots, then tails. The segments whose x range contains the current event are stored in a sweep list. The sweep list order defines the y order of the segments in the open intervals between events. The algorithm maintains the invariant that the sweep list order of adjacent segments equals their root list order on every interval.

A tail event is processed by inserting the segment, c, into the sweep list. The sweep list is represented as a balanced binary tree and the insertion is performed in $O(\log n)$ time by a standard traversal in which c is compared with tree nodes using root list order. Insertion preserves the sweep invariant: c is placed between segments a and b such that the a/c root list indicates that c is above a and the b/c root list indicates that c is below b. When c is inserted first or last, a or b is null. A head event is processed by removing the segment from the sweep list. A root event is processed by swapping the segments in the sweep list. The first time a pair of segments becomes adjacent in the sweep list, their root list is constructed and the root events after the current x are inserted into the queue.

The sweep algorithm extends the standard algorithm in three ways.

1. Root events of non-adjacent segments are ignored.

This extension prevents the algorithm from creating a non-planar combinatorial structure. The other two extensions handle the case that a pair of (freshly) adjacent segments do not satisfy the sweep invariant: their root list order disagrees with their order in the sweep list.

2. If adjacent segments are out of order and have a root that precedes the current x, they are swapped at the current x.

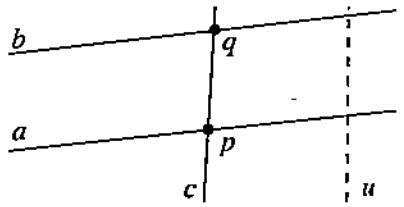


Figure 2: Potential robustness problem.

We call this extension an immediate swap. The resulting segment intersection compensates for one that was omitted from the output because it was not adjacent or because the root list was constructed too late.

3. If adjacent segments are out of order and have no prior root, the sweep algorithm is rolled back to the more recent of the two tails, the corresponding segment is inserted on the correct side of the other segment, and the sweep resumes.

An immediate swap is not used to establish the invariant in this case because it would make the number of intersections greater than the number of roots, for instance two lines could intersect twice.

Example Figure 2 shows a potential robustness problem. Line segments a and c intersect at point p, while b and c intersect at q. The coordinates p_x and q_x are almost equal because c is almost vertical. The initial sweep list order is c, a, b. The standard sweep algorithm generates an inconsistent arrangement of these segments when floating point incorrectly yields $q_x < p_x$. This event causes the algorithm to swap b with c. The swap is inconsistent because b and c are not adjacent, and yields the incorrect vertical order b, a, c. The algorithm then swaps c with a and deletes c to obtain the incorrect order b, a at u.

Our algorithm ignores the b/c root at q_x because a is between b and c. When it swaps a and c at p_x, b and c undergo an immediate swap. The correct order is obtained at u.

Running time The running time of the standard sweep is $O((n + N) \log n)$ where N is the number of roots in the root lists. The costs of the extensions is as follows: 1) An ignored root costs zero; 2) an immediate swap costs $O(\log n)$; and 3) a rollback costs $O((n+N) \log n)$. The algorithm running time is $O((n + N) \log n + k(n + N) \log n)$ with k the number of extension instances. We believe that the rollback cost could be reduced to $O(k \log n)$ via dependency directed backtracking. There is no practical value in pursuing this reduction because rollbacks never occur and would only cost $O(\log n)$ in practice even with the current algorithm. Note: rollbacks ensure the output *size* is always the standard $O(n + N)$.

Define an inconsistency as a triple of segments whose root list order is cyclic over an open interval: a is below b in the a/b list, b is below c in the b/c list, and c is below a in the a/c list. There are $O(n^3)$ segment triples and each root list has $O(d^2)$ intervals with d the curve degree, so the number of inconsistencies is $O(n^3 d^6)$. The next paragraph proves that every instance of the three extensions implies an inconsistency, so $k = O(n^3 d^6)$.

The proof is by cases. In an a/b immediate swap or rollback (extension 2 or 3), the prior event must cause a and b so "see" each other. Hence, there was some segment c between a and b in

the sweep list, and the prior event was its intersection with a or its intersection with b or its head. Prior to this event, a is below c and c is below b according to the a/c root list and the c/b root list. Subsequent to this event, a and b see each other, and b is below a according to the a/b root list. Since this swap or rollback does *not* correspond to a calculated root of a/b, b is below a according to their root list prior to c exiting or ending. Thus, the triple a, c, b is cyclic on the interval ending at x. In a non-adjacent a/b root event (extension 1), the sweep list has a sublist $c_1 = a, c_2, \ldots, c_m = b$ with c_i below c_{i+1} according to the c_i/c_{i+1} root list for $1 \leq i \leq m-1$. No c_i can end at the swap x because head events are processed before root events. Hence, the a/b $(= c_1/c_m)$ root list order flips at x and the other orders persist, which implies that $c_1, c_2, \ldots, c_m$ are cyclic in the interval starting at x. This cyclic chain contains a cyclic triple by induction of m. If $m = 3$, it is trivially true. For $m > 3$, if $c_m = b$ is above c_2 according to the c_2/c_m root list, then $c_1/c_2/c_m$ is a cyclic triple. Otherwise, $c_2, c_3, \ldots, c_m$ is cyclic and contains a cyclic triple by induction.

Embedding The sweep algorithm outputs the segment intersection points and the sweep order on the intervals between events. With a little extra bookkeeping, this output can be converted into a trapezoidal decomposition, a cell decomposition, or a structure that can determine the vertical ordering of any pair of segments along any vertical line in $O(\log n)$ time. The extensions are well-known and straightforward. The resulting combinatorial structures can contain degenerate cells due to simultaneous events. The embedding algorithm eliminates these cells and generates a generic arrangement.

The embedding algorithm generates vertices for the segment tails, heads, and intersection points, splits each segment into edges at its intersection points with other segments, and computes the cyclic order of incident edges at each vertex. The challenge is to assign y values to intersection points in a consistent manner. The computed y value is fine in the generic case where the event x's are distinct. Simultaneous events lead to special cases and to potential inconsistencies. If a's tail is (x, y_a) and b's tail is (x, y_b), then the numerical order of y_a, y_b must agree with the a/b sweep list order, and likewise for heads. If a and b start or end at the same point, it is inconsistent for c to intervene in the sweep list. Vertical segments are another source of simultaneous events and present further complications. We have developed an inconsistency sensitive algorithm, but a detailed description calls for a separate paper.

4 Error analysis

Since the sweep algorithm is numerical, its combinatorial outputs may be incorrect for the given inputs even if there are no inconsistencies. This section shows that they *are* correct for a perturbed input. Numerical bounds on the perturbation are proven for various structures. This is standard backwards error analysis applied to the arrangement setting. In particular, for a measure ϵ of the accuracy of the root list calculation, the trapezoidal decomposition can be realized with accuracy $2\epsilon + 2kn\epsilon$ with mild restrictions on the input, and the cell decomposition can always be realized with this accuracy.

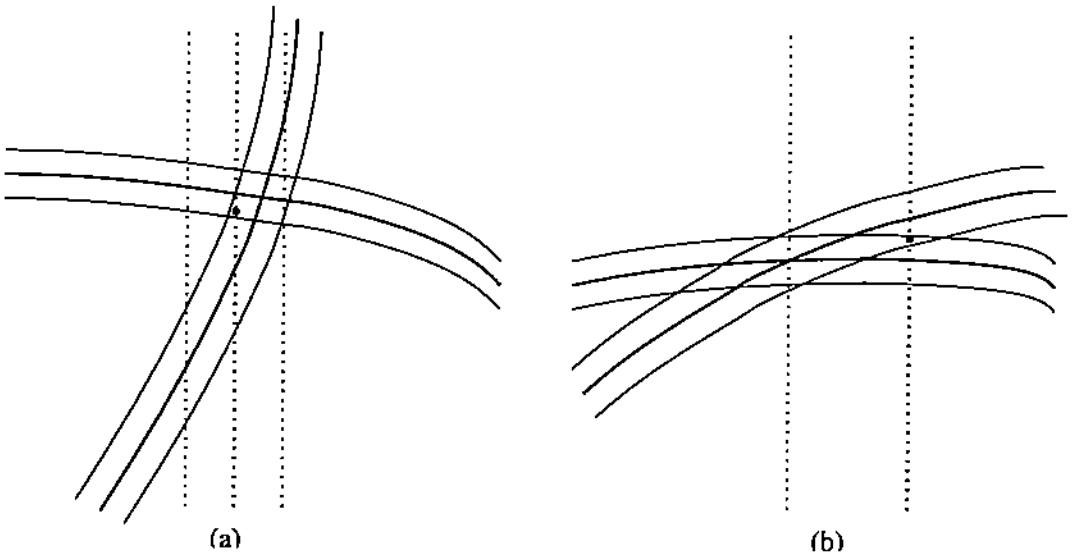


Figure 3: Curve intersection types: (a) transverse; (b) tangential.

4.1 Background Assumptions

The error analysis for the arrangement algorithm depends on the accuracy of the numerical root list calculation. We use a model in which each curve is surrounded by an error band of width ϵ. An absolute bound is used for simplicity. We restrict the arrangement to a bounding box and obtain ϵ of a few rounding units (10^{-16} for IEEE double precision) times the box size.

The form of the error bands depends on the curve intersection angles, which are measured by the condition numbers of their equations at the roots. The root is within $c\epsilon$ of the true intersection point with c the condition, which implies that the y order of the curves is correct outside a $c\epsilon$ neighborhood of the roots. Figure 3 shows the two qualitative cases that arise. In the transverse case, the condition is $O(1)$ and the y order is correct outside an $O(\epsilon)$ neighborhood of the roots. In the tangential case, the condition is unbounded and the y order is incorrect on a wide interval, but every point on each curve is $O(\epsilon)$ distant from a point on the other curve.

The two cases motivate the definition of ϵ-accuracy for the root list of two segments. We stipulate a minimum distance of 2ϵ between roots. If the numerical solver generates closer roots, they are treated as a double root. Suppose segment a is supposed to be above segment b in root list interval (r_{i-1}, r_i). If $x \in (r_{i-1} + \epsilon, r_i - \epsilon)$, then either $a.f(x) > b.f(x)$ or the point $(x, a.f(x))$ lies within ϵ of segment b and the point $(x, b.f(x))$ lies within ϵ of segment a. In other words, each segment is at most ϵ to the wrong side of the other segment outside ϵ neighborhoods of their roots.

4.2 Inconsistency-Free Case

The remainder of the error analysis seeks to determine the realizability of the arrangement in terms of the accuracy ϵ of the root lists. This subsection considers the case in which all the root lists are consistent. Lemma 4.1 indicates how to realize individual root lists, and Theorem 4.2 extends this realization to the entire arrangement.

Figure 4 illustrates the realizations of the two cases in the previous figure. In the transverse case, the value of the curve ϵ away from the calculated x is extended horizontally into the interval

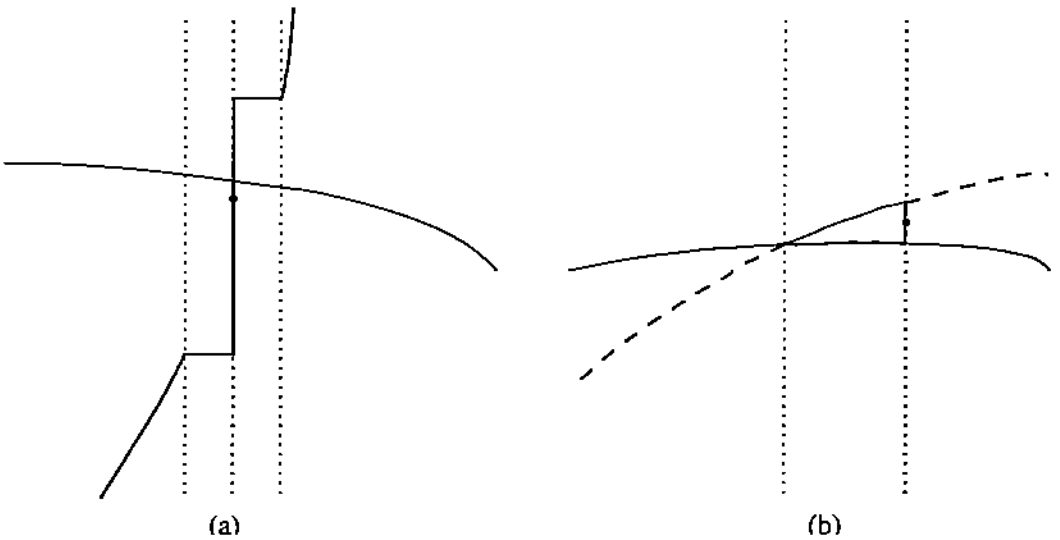


Figure 4: Realization curves: (a) transverse; (b) tangential.

within ϵ of the root. The gap is then filled with a vertical segment. Only the "steep" segment is realized in this manner in the figure. Lemma 4.1 does not distinguish between the intersecting segments and so the other segment would be similarly realized. In the tangential case, the identities of the two segments are "swapped" inside the x interval between correct root and the calculated root. Again the resulting gap is patched with a vertical segment. Since the lemma does not distinguish between shallow and steep, each segment is also horizontal within ϵ of the calculated root. This is not depicted.

Lemma 4.1 *Let a and b be segments and $r_1, ..., r_m$ be their root list. Let x be a value at which the root list implies that a is above b. If the root list is ϵ-accurate, then there exists values $a.\tilde{f}(x)$ and $b.\tilde{f}(x)$ such that $a.\tilde{f}(x) \geq b.\tilde{f}(x)$ and $(x, a.\tilde{f}(x))$ lies within 2ϵ of a and $(x, b.\tilde{f}(x))$ lies within 2ϵ of b.*

Proof. If $a.f(x) \geq b.f(x)$, then we can just set $a.\tilde{f}(x) = a.f(x)$ and $b.\tilde{f}(x) = b.f(x)$. Suppose $a.f(x) < b.f(x)$. If $x \in (r_{i-1} + \epsilon, r_i - \epsilon)$, then set $a.\tilde{f}(x) = b.f(x)$ and $b.\tilde{f}(x) = a.f(x)$ (swap them). Otherwise, there exists x' equal to either $r_i - \epsilon$ or $r_i + \epsilon$ for some root r_i such that $|x - x'| \leq \epsilon$. In that case, we set $a.\tilde{f}(x) = a.\tilde{f}(x')$ and $b.\tilde{f}(x) = b.\tilde{f}(x')$. In the worst case, $(x', a.\tilde{f}(x'))$ lies within ϵ of b and $(x, a.\tilde{f}(x)) = (x, a.\tilde{f}(x'))$ lies within ϵ of $(x', a.\tilde{f}(x'))$, yielding a maximum error of 2ϵ. □

The function $a.\tilde{f}(x)$ is piecewise algebraic with at most one discontinuity per root. Filling in the gaps with vertical line segments results in a continuous curve that approximates a to within 2ϵ. Similarly for b. Hence, the root list is the correct set of crossings for an input perturbed by at most 2ϵ. We call this property 2ϵ-realizable.

It might seem that if there is a stack of segments, the error might build up. As the following theorem indicates, the error does not build up if there are no inconsistencies. This follows from an application of Helly's theorem for set of intervals (on a vertical line): if every pair of intervals shares a common point, then they all share a common point.

Theorem 4.2 *If there are no inconsistencies, then the arrangement is 2ϵ-realizable.*

Proof. Fix a value of x. The output of the arrangement algorithm implies a particular vertical ordering of the segments at x. For each segment s in the list, define above(s) to be the set of points (x, y) such that either $y \geq s.f(x)$ or (x, y) lies within 2ϵ of s. Similarly define below(s). Define $s.\tilde{f}(x)$ as the the y-coordinate of the lowest point in the intersection of the following sets: below(s), above(s), above(b) for each segment b below s in the list, and below(a) for each segment a above s in the list. It remains to be shown that 1) this intersection is never empty and 2) if a is adjacent to and above b, then $a.\tilde{f}(x) \geq b.\tilde{f}(x)$.

1) Since there are no inconsistencies, if a is above b in the list at x, then this order agrees with their root list at x, and therefore below(a) $\cap$ above(b) is not empty. The rest follows from Helly's theorem.

2) The intersection that defines $b.\tilde{f}(x)$ differs from the intersection that defines $a.\tilde{f}(x)$ in that it above(a) is replaced by below(b). This cannot increase the minimum y. □

4.3 Arrangements with Inconsistencies

If there are inconsistencies, then at some x, the arrangement algorithm could put segment a below segment b in the sweep list even though the root list for a and b indicates that a is above b at x. Even so, there will be a "stack of evidence" $c_1 = a, c_2, c_3, \ldots, c_m = b$ such that c_i appears below c_{i+1} in the sweep list, $1 \leq i \leq m-1$, and the root list for c_i/c_{i+1} also indicates that c_i is below c_{i+1} at x. By the invariant of the algorithm, adjacent segments in the sweep list have an order that agrees with their root list order, and therefore we can just set $c_1, c_2, \ldots, c_m$ equal to the sublist of the sweep list from a to b. However, "stack of evidence" also allows c's that are not adjacent in the sweep list. As the following lemma indicates, *any* stack of evidence that persists for longer than $2n\epsilon$ allows us to realize a and b with accuracy $(n-1)\epsilon$.

Lemma 4.3 *Suppose there is a stack of evidence $c_1 = a, c_2, \ldots, c_m = b$ that is valid along an interval $x \in (x^-, x^+)$ of length $x^+ - x^- > 2n\epsilon$. For $x \in (x^- + n\epsilon, x^+ - n\epsilon)$, either $a.f(x) < b.f(x)$ or $(x, a.f(x))$ lies within $(n-1)\epsilon$ of b and $(x, b.f(x))$ lies within $(n-1)\epsilon$ of a.*

Proof. It suffices to just look at the worst case: $m = n$ and each consecutive pair of curves c_i, c_{i+1} is ϵ to the wrong side of each other. The point $(x, a.f(x))$ lies within ϵ of a point on c_2, which lies within ϵ of a point on c_3, ..., which lies within ϵ of a point on c_{n-1}, which lies within ϵ of a point on b. This chain of points is at most $(n-1)\epsilon$ away from where it starts, and therefore it still lies within the interval $(x^- - \epsilon, x^+ + \epsilon)$, and so the definition of root list accuracy applies. The length of the chain is bounded by $(n-1)\epsilon$. □

Lemma 4.3 implies that sufficiently long intervals of "constant evidence" are "o.k." even even if they are inconsistent. We would like to show next that there cannot be "too many" short intervals in a row.

Lemma 4.4 *The interval(s) in x over which a pair of segments a and b have inconsistent order can be covered by at most k intervals with a constant stack of evidence.*

Proof. We will say that a stack of evidence is "minimal" at x if for $1 \leq i \leq m-2$ the root list of c_i and c_{i+2} does *not* say that c_i is below c_{i+2} at x. We can make any stack minimal by repeatedly throwing away c_{i+1} if this condition is not met. For a minimal stack, $c_i/c_{i+1}/c_{i+2}$ have a standard

triple inconsistency at x. As we move right, a minimal stack becomes non-minimal or invalid only when one of these triples becomes valid. At this point, we must pick a different minimal stack. This can happen only as many times as there are triples of inconsistency which is k. □

So now we know that long inconsistent intervals are "o.k." and that there can be at most k short inconsistent intervals with constant evidence in a row. The following theorem puts these two facts together to prove an error bound for the arrangement. The corollary proves a bound for the cell decomposition.

Theorem 4.5 *If every pair of segments that overlap in x overlap by more than $2kn\epsilon$, then the arrangement is $2\epsilon + 2kn\epsilon$-realizable.*

Proof. Lemma 4.1 showed that if a pair of curves is ϵ-accurate at least ϵ away from a root, then they are 2ϵ-realizable. Lemma 4.3 showed that curves are $n\epsilon$-accurate (actually $(n-1)\epsilon$-accurate) for x at least $n\epsilon$ away from the endpoints of a long inconsistent interval. A similar proof to Lemma 1 shows that the pair is therefore $2n\epsilon$-realizable. The sticking point is short inconsistent intervals. Lemma 4.4 showed that the evidence of inconsistency cannot change more than k times. Even if the intervals of constant evidence are short ($< 2n\epsilon$), their total length cannot exceed $2kn\epsilon$.

If we move left from a short interval of constant evidence, we must eventually come to either 1) an intersection of a and b (according to the arrangement), 2) a consistent interval for a and b, 3) a long inconsistent interval for a and b, or 4) a tail of a or b. In case 1, a and b are 2ϵ-realizable immediately to the right of the intersection and therefore $2\epsilon + 2kn\epsilon$-realizable in the short interval because it is as most $2kn\epsilon$ distant in x. Case 2 is similar. Case 3 implies an additional $2n\epsilon$ error, but since this interval is among the k the short inconsistent interval is $2kn\epsilon$-realizable. Case 4 requires us to look forward instead. If we encounter a consistent interval or a long inconsistent interval, the reasoning is the same. It is not o.k. to encounter an intersection of a and b because the algorithm may have postponed this intersection too much and spoiled any bound on realizability to the left of the intersection. (Keep in mind that the algorithm never forces an intersection if two segments do not "want to" intersect, and that means the the realization is bounded on the right. On the other hand, the algorithm can prevent an intersection even if the segments "want to" intersect in the case that the segments cannot "see each other".) However, the algorithm is designed to roll back if a and b were inconsistent prior to the first intersection, and therefore this cannot happen. Finally, because each pair of segments overlap by more than $2kn\epsilon$ in x, we cannot encounter the head of either segment.

Taken together, these cases imply a $2\epsilon + 2kn\epsilon$ realizability bound on pairs of segments. Applying Helly's theorem as in the proof of Theorem 4.2 implies that the entire arrangement is realizable with this error bound. □

Corollary 4.6 *The cell decomposition of the arrangement is realizable by perturbed segments which lie within $2\epsilon + 2kn\epsilon$ of all points of the input segment except perhaps points within $2kn\epsilon$ in x of the endpoints.*

Proof. Two segments with a tiny overlap in x can make the trapezoidal decomposition unrealizable because the trapezoidal decomposition relates them in y order even if they do not bound the same cell of the arrangement. However, tiny local deformations of at most $2kn\epsilon$ will eliminate the tiny overlaps in x without changing the cell decomposition. This deformation will prevent the realizations of the segments from covering the entire input segments. However, they will cover each input segment contracted by at most $2kn\epsilon$ in x from each endpoint. □

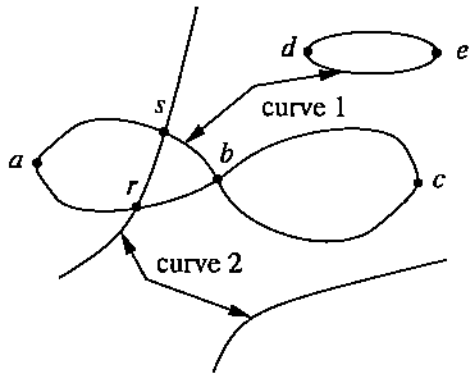


Figure 5: Algebraic curves.

5 Algebraic curves

The arrangement algorithm has been validated on planar algebraic curves without degenerate singularities. This class of curves is ample for shape modeling. We have developed numerical software that partitions an algebraic curve into x-monotone segments and that computes segment root lists.

Algebraic curves An algebraic curve is the zero set of a polynomial $f(x, y)$. A point $p = (x, y)$ on the curve is regular/singular when ∇f is nonzero/zero. The regular points partition into 1D manifolds, called branches, that are topological circles or lines. A topological line begins and ends where x diverges to infinity or where x converges to a finite value, called a vertical asymptote, and y diverges to infinity. A vertical asymptote is a zero of the leading coefficient of f when written as a polynomial in y over $R[x]$. A branch partitions into x-monotone segments. Two segments meet at a point where $f_y = 0$, called a turning point. The segments lie locally on the left/right side of the tangent when $f_x f_{yy}$ is positive/negative. The turning point is degenerate when $f_{yy} = 0$.

Figure 5 shows two algebraic curves. Curve 1 consists of two branches. The lower branch has turning points a and c, singular point b, and four x-monotone segments. The upper branch has turning points d and e, which generate two segments. Curve 2 consists of two branches that begin/end at a vertical asymptote. The curves intersect at r and s.

A singular point is degenerate when its Hessian matrix has a zero eigenvalue, is isolated when both eigenvalues have the same sign, and is simple otherwise. Two branches intersect transversely at a simple singular point. More complicated intersections can occur at degenerate singular points.

Segment construction We partition each input algebraic curve into segments that start and end at turning points, singular points, and vertical asymptotes. Degenerate points are not handled. Vertical lines are handled via preprocessing. Isolated singular points are ignored for simplicity, since they are useless for solid modeling, but would be easy to handle.

The turning points and singular points are computed by solving $f, f_y = 0$; a root is a singular point when $f_x = 0$ and is a turning point otherwise. The vertical asymptotes are computed by solving the leading coefficient equation. Root finding is performed by the homotopy method [28]. The method finds all the roots with high probability given a randomized starting point. The rare failures are corrected with restarts. The algorithm converges quadratically at simple roots and the

root accuracy is bounded by the condition times the unit roundoff error of about 10^{-16}. At multiple roots, convergence is linear and the accuracy is smaller, but these do not occur in our calculations.

The turning and singular points partition the x axis into intervals on which there are a fixed number of segments. The segment vertical order is computed by a line sweep. The events are the interval endpoints. Two branches start at a right turning point, two end at a left turning point, one ends and one starts at a vertical asymptote of odd y degree, two start or end at a vertical asymptote of even y degree, and two end and two start at a singular point.

Sweep updates use root finding to find the relevant branches. The y values of the sweep branches at $x = x_0$ are computed by solving $f(x_0, y)$, sorting the roots, and assigning the sorted roots to the branches, which are stored in y order. If (x_0, y_0) is a turning or singular point, y_0 is a double root of $f(x_0, y)$, hence is ill-conditioned and hard to compute. We factor $y - y_0$ out of $f(x_0, y)$ and solve the reduced polynomial for the other roots. At a start point p, the new branches are inserted between the branches whose y values bracket p_y. At an end or singular point, the two branches whose y values are closest to p_y are updated. At a vertical asymptote, there are one or two branches in the sweep and they are updated.

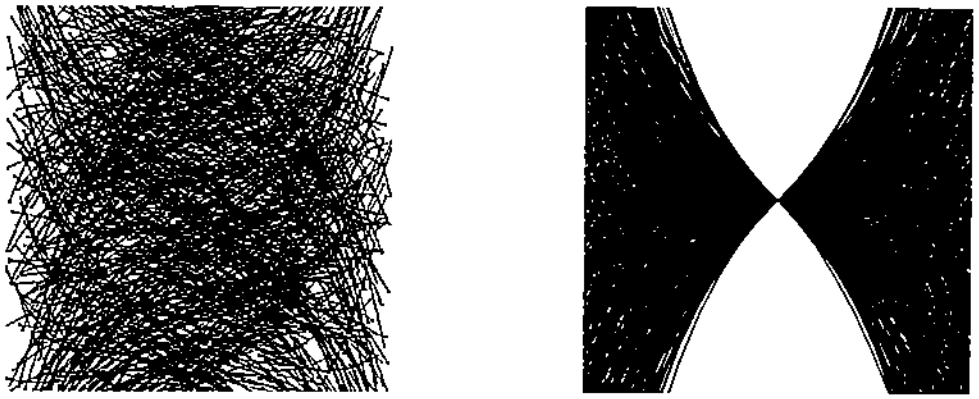
Root lists We compute a root list for every pair of branches on curves f and g. We solve $f, g = 0$, assign each root to a pairs of f/g branches, then process every pair of branches. Branches a and b interact over the intersection $[x_1, x_2]$ of their x ranges, assuming this interval is not empty. The roots of a/b split $[x_1, x_2]$ into intervals. We compute the vertical order of the branches on an interval by comparing their y values at the midpoint.

6 Experimental results

The inconsistency sensitive approach works best when the number of inconsistencies is small (although it is always polynomial in the input size, whereas the exact method is exponential). We estimated the number of inconsistencies on several types of arrangement problems. We generated 1000 random curves of degree d, constructed their arrangement, and repeated the experiment 50 times. The algorithm never found an inconsistency for $d = 1, 2, 3, 4, 5, 6, 10, 15$. Nor were there any inconsistencies for 1000 random horizontal and vertical line segments, which are common in VLSI, mechanical design, and other applications. Figure 6a shows an arrangement of 1000 quartics on the unit square: the 47511 intersection points are marked with green circles and the 2000 endpoints are marked with red circles.

We generated 200 random line segments and perturbed the segment endpoints by a random number in $[-\delta, \delta]$ to obtain 200 pairs of nearly identical segments. We checked all 10.5 million triples of segments for inconsistencies and obtained 0.06% inconsistent for $\delta = 0.001$ and none for $\delta = 0.1, 0.0001, 10^{-6}, 10^{-10}$. We generated 200 near-identical line segments by perturbing the endpoints of a single segment, generated all triples, and obtained no inconsistencies.

We generated 200 random polynomials $y = f(x)$ of degree d then modified their constant terms to make them go through the point $p = (1, 2)$, as shown in Figure 6b. The polynomial $f(x)$ was replaced with $g(x) = f(x) - f(1)$, so that $g(1) = 0$ except for rounding error. We counted the inconsistencies among all 1.3 million triples of polynomials and obtained none for $d = 1$, 5% for $d = 2$, 39% for $d = 3$, 42% for $d = 4$, 48% for $d = 5$, 54% for $d = 6$, 58% for $d = 10$, and 60% for $d = 15$. The roots were computed by Laguerre's method. We repeated the experiments with



(a) (b)

Figure 6: Arrangement of 1000 quartics: (a) with random coefficients; (b) modified to pass through $(1, 2)$.

random implicit polynomials and obtained 60% inconsistent triples for degree 2 and higher. The roots were computed by the homotopy method. The maximum width of an inconsistent interval was 10^{-10} over all the experiments. The running time per root was roughly constant.

We perturbed the constant terms of the polynomials by a random number in $[-\delta, \delta]$ and obtained 0.003% inconsistencies instead of 48% for $d = 5$ and $\delta = 10^{-8}$, 0.1% instead of 66% for $d = 15$ and $\delta = 10^{-8}$, and 3.5% for $d = 15$ and $\delta = 10^{-10}$.

We conclude that inconsistencies are vanishingly rare in generic input and in many structured, hence degenerate, inputs. The only case where we found many inconsistencies is among triples of curves that almost meet at a point. The curves form a tiny triangle with 4 inconsistent vertex orders and 2 consistent orders, as shown in Figure 1. As d increases, the floating point resolution of the triangle decreases until the vertex order becomes essentially random at $d = 15$, that is 60% inconsistent versus 66% for a random choice of 4 out of 6 orders.

Although degenerate, small triangles occur in some applications. For example, consider the layout problem of cutting a maximum number of clothing parts from a strip of fabric. Every part will touch two other parts (or the strip boundary) in an optimal configuration, which implies that three contact curves intersect in every three-part configuration space. In mechanical design, redundancy and symmetry can generate intersecting triples of contact curves. Even so, these degeneracies and the inconsistencies they cause will be confined to small regions, and it is hard to conceive of a practical input for which inconsistencies will even double the running time of the entire arrangement construction.

7 Discussion

We have presented a robust arrangement algorithm for algebraic curves based on floating point arithmetic. Its performance is analyzed in terms of the number of combinatorial inconsistencies that occur due to numerical computation. The running time is $O((n+N)\log n + k(n+N)\log n)$, which is $O(n^3(n+N)\log n)$ because $k = O(n^3)$. The output size is always the standard $O(n+N)$. We have presented extensive experimental evidence that k is tiny in practice, hence that the actual running time matches a standard floating point sweep. Curve intersection is performed by

solving a system of two polynomial equations via the homotopy method. The output arrangement is realizable by semi-algebraic curves that are close to the input curves. The distance between the realization curves and the input is $O(\epsilon + kn\epsilon)$ where ϵ is the curve intersection accuracy.

Inconsistency sensitive analysis is a new computational geometry paradigm. We plan to apply this paradigm to many problems beyond arrangements. One goal is to construct and manipulate the configuration spaces of planar parts. Configuration spaces are the key to novel algorithms for part layout, mechanical design, and path planning. Another goal is solid modeling with explicit and implicit surfaces. In both cases, the computational geometry task is to arrange surface patches of high degree.

We also plan to develop iterative algorithms that cascade geometric computations, meaning that the output of each iteration is the input to the next iteration. Many non-geometric numerical algorithms use cascading, for example Newton's method. We believe that geometric algorithms would also use cascading extensively if there were an effective way to implement it. For example, Milenkovic uses cascaded numerical geometric operations in part layout [27, 24, 26]. However, one can construct any algebraic expression by cascading two simple geometric constructions: (1) join two points to form a line and (2) intersect two lines [3, 25]. This suggests that exact geometric cascading is as hard as exact scientific computing, which is untenable. Inconsistency sensitive algorithms could make cascading practical by replacing this exponential factor with a small constant.

References

[1] Oliver Aberth and Mark J. Schaefer. Precise computation using range arithmetic via C++. *ACM Transactions on Mathematical Software*, 18(4):481–491, 1992.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen. *LAPACK User's Guide, Release 1.0.* SIAM, Philadelphia, 1992.

[3] Behnke, Bachmann, Fladt, and Kunle. *Fundamentals of Mathematics, Volume II: Geometry.* MIT Press, Cambridge, MA, 1974.

[4] L. Bielhl, J Buchmann, and T. Papanikolaou. LiDIA: A library for computational number theory. Technical Report SFB 124-C1, Fachbereich Informatik, Universität des Säarlandes, Saarbrucken, West Germany, 1995.

[5] S. Fortune. Stable maintenance of point-set triangulation in two dimensions. In *30th Annual Symposium on the Foundations of Computer Science*, pages 494–499. IEEE, October 1989.

[6] S. Fortune. Numerical stability of algorithms for 2d delaunay triangulations. *International Journal of Computational Geometry and Applications*, 5:193–213, 1995.

[7] S. Fortune. Vertex-rounding a three-dimensional polyhedral subdivision. *Discrete and Computational Geometry*, 22:593–618, 1999.

[8] S. Fortune and V. J. Milenkovic. Numerical stability of algorithms for line arrangements. In *Proceedings of the Seventh Symposium on Computational Geometry*, pages 334–341. ACM, 1991.

[9] M.T. Goodrich, L.J. Guibas, J. Hershberger, and P.J. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proceedings of the 13th Symposium on Computational Geometry*. ACM, 1997.

[10] D. H. Greene. Integer line segment intersection. unpublished manuscript.

[11] Leonidas Guibas and David Marimont. Rounding arrangements dynamically. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 190–199, 1995.

[12] D. Halperin and E. Packer. Iterated snap rounding. *Computational Geometry: Theory and Applications*, 23(2):209–222, 2002.

[13] D. Halperin and C.R. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Computational Geometry: Theory and Applications*, 10(4):273–288, 1998.

[14] J. D. Hobby. Practical segment intersection with finite precision output. submitted for publication.

[15] John Keyser. *Exact Boundary Evaluation for Curved Solids*. Ph.D. thesis, University of North Carolina, Chapel Hill, NC, 2000.

[16] John Keyser, Tim Culver, Mark Foskey, Shankar Krishnan, and Dinesh Manocha. Esolid: A system for exact boundary evaluation. In *Proceedings of Seventh ACM Symposium on Solid Modeling and Applications (ACM Solid Modeling '02)*, pages 23–34. ACM, 2002.

[17] John Keyser, Tim Culver, Dinesh Manocha, and Shankar Krishnan. Efficient and exact manipulation of algebraic points and curves. *Computer Aided Design*, 32:649–662, 2000.

[18] John Keyser, Shankar Krishnan, and Dinesh Manocha. Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic: I – representations. *Computer Aided Geometric Design*, 16(9):841–859, 1999.

[19] John Keyser, Shankar Krishnan, and Dinesh Manocha. Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic: II – computation. *Computer Aided Geometric Design*, 16(9):861–882, 1999.

[20] Shankar Krishnan, Mark Foskey, Tim Culver, John Keyser, and Dinesh Manocha. PRECISE: Efficient multiprecision evaluation of algebraic roots and predicates. Technical Report TR00-008, Department of Computer Science, University of North Carolina,, Chapel Hill, 2000.

[21] Shankar Krishnan, Dinesh Manocha, M. Gopi, Tim Culver, and John Keyser. BOOLE: A boundary evaluation system for boolean combinations of sculptured solids. *International Journal of Computational Geometry and Applications*, 11(1):105–144, 2001.

[22] Z. Li and V. J. Milenkovic. Constructing strongly convex hulls using exact or rounded arithmetic. *Algorithmica*, 9:345–364, 1992.

[23] K. Mehlhorn and S. Näher. Leda, a library of efficient data types and algorithms. Technical Report A 04/89, Fachbereich Informatik, Universität des Säarlandes, Säarbrucken, West Germany, 1989.

[24] V. J. Milenkovic. Rotational polygon containment and minimum enclosure using only robust 2d constructions. *Computational Geometry: Theory and Applications*, 13:3–19, 1999.

[25] Victor J. Milenkovic. Shortest path geometric rounding. *Algorithmica*, 27(1):57–86, 2000.

[26] Victor J. Milenkovic. Densest translational lattice packing of non-convex polygons. *Computational Geometry: Theory and Applications*, 22:205–222, 2002.

[27] V.J. Milenkovic and K. Daniels. Translational polygon containment and minimal enclosure using mathematical programming. *International Transactions in Operational Research*, 6:525–554, 1999.

[28] Alexander P. Morgan. *Solving Polynomial Systems Using Continuation for Scientific and Engineering Problems*. Prentice-Hall, Englewood Cliffs, NJ, 1987.

[29] S. Raab. Controlled perturbation for arrangements of polyhedral surfaces. In *Proceedings of the 15th Symposium on Computational Geometry*, pages 163–172. ACM, 1999.