International Journal of Computational Geometry & Applications


# AN APPROXIMATE ARRANGEMENT ALGORITHM FOR SEMI-ALGEBRAIC CURVES

Victor Milenkovic

*Department of Computer Science, University of Miami*
*Coral Gables, FL 33124-4245, USA*
*vjm@cs.miami.edu*

Elisha Sacks

*Computer Science Department, Purdue University*
*West Lafayette, IN 47907-2066, USA*
*eps@cs.purdue.edu*

We present an arrangement algorithm for plane curves. The inputs are (1) continuous, compact, $x$-monotone curves and (2) a module that computes approximate crossing points of these curves. There are no general position requirements. We assume that the crossing module output is $\epsilon$ accurate, but allow it to be inconsistent, meaning that three curves are in cyclic $y$ order over an $x$ interval. The curves are swept with a vertical line using the crossing module to compute and process sweep events. When the sweep detects an inconsistency, the algorithm breaks the cycle to obtain a linear order. We prove correctness in a realistic computational model of the crossing module. The number of vertices in the output is $V = 2n+N+\min(3kn, n^2/2)$ and the running time is $O(V \log n)$ for $n$ curves with $N$ crossings and $k$ inconsistencies. The output arrangement is realizable by curves that are $O(\epsilon + kn\epsilon)$ close to the input curves, except in $kn\epsilon$ neighborhoods of the curve tails. The accuracy can be guaranteed everywhere by adding tiny horizontal extensions to the segment tails, but without the running time bound. An implementation is described for semi-algebraic curves based on a numerical equation solver. Experiments show that the extensions only slightly increase the running time and have little effect on the error. On challenging data sets, the number of inconsistencies is at most $3N$, the output accuracy is close to $\epsilon$, and the running time is close to that of the standard, non-robust floating point sweep.

*Keywords*: arrangements; robust computational geometry.

## 1. Introduction

We present an arrangement algorithm for plane curves based on approximate computation of curve crossing points. The arrangement of $n$ curves with $N$ crossings can be computed in $O((n + N) \log n)$ time by sweeping. The analysis assigns unit cost to geometric operations, such as partitioning a curve into $x$-monotone segments, intersecting two curves, and sorting vertices along an axis. For semi-algebraic curves, the operations reduce to constructing algebraic numbers and computing the signs of polynomials at these numbers. The classical techniques for manipulating alge-

braic numbers incur a computational cost that grows rapidly with degree and bit complexity. The same problem arises in incremental insertion or in any other arrangement algorithm.

The mainstream approach to this problem[1] is to accelerate the geometric computations via custom algorithms, constructive root bounds, and floating point filters. This approach has led to arrangement algorithms for lines, circles, conics, and cubics. We present an alternate research direction that constructs arrangements using approximate geometric computation via numerical equation solving. The motivation is that numerical solvers are highly accurate and are orders of magnitude faster than exact algebraic computation. The scientific computing community greatly benefits from numerical computation, so it is worth exploring its applicability to computational geometry.

The first issue is that numerical solvers lack rigorous running time and error bounds, although their qualitative behavior is well understood. We take the computer science approach to this issue: define a computational model, verify it experimentally, and analyze algorithms in it. The definition appears in Sec. 3. We assume that we can compute approximations to all real intersection points of any pair of algebraic curves. The computation time is polynomial in algebraic degree. The backward error is bounded on a bounded domain. Using this model, we define a *crossing module* that computes the $x$ values where curves cross and their $y$ order between crossings. The arrangement algorithm performs all geometric computations with the crossing module.

The challenge is to reconcile the approximate nature of numerical computation with Euclidean geometry. Approximate geometric computations can violate the laws of geometry, just as floating point operations can violate the laws of algebra. In our computational model, this problem arises when the crossing module assigns three curves an inconsistent, cyclic vertical order. The canonical example (Fig. 1) is curves $a, b, c$ that form a triangle whose diameter is less than $\epsilon$. The crossing module computes the vertices $p, q, r$ with $\epsilon$ accuracy, incorrectly places $q$ to the left of $p$, and correctly places $r$ to the right of $p$. The curves are in cyclic order on $(q_x, p_x)$: $a$ is below $b$ from the $a/b$ output, $b$ is below $c$ from the $b/c$ output, and $c$ is below $a$ from the $a/c$ output.

We construct arrangements with a sweep algorithm that handles inconsistencies (Sec. 4). The inputs are (1) continuous, compact, $x$-monotone curves and (2) a crossing module. There are no general position requirements. The curves are swept with a vertical line using the crossing module to compute and process sweep events. When the sweep detects an inconsistency, it breaks the cycle to obtain a linear order. We prove correctness in the computational model of the crossing module. The number of vertices in the output is $V = 2n + N + \min(3kn, n^2/2)$ and the running time is $O(V \log n)$ for $n$ curves with $N$ crossings and $k$ inconsistencies. The output arrangement is realizable by curves that are $O(\epsilon + kn\epsilon)$ close to the input curves. As in any backward error analysis, the realization curves are proved to exist, but are not constructed. The quantities $\epsilon$ and $k$ are not part of the algorithm; they

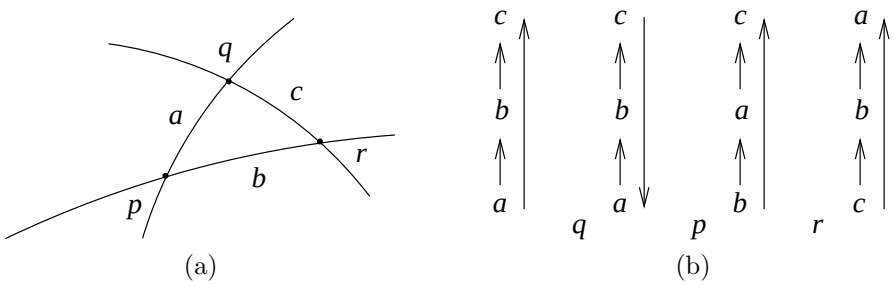


Fig. 1. Inconsistency: (a) true geometry has $p_x < q_x < r_x$; (b) computed geometry has $q_x < p_x < r_x$ with cyclic vertical order (depicted with arrows) on $(q_x, p_x)$.

are used solely in our analysis of its speed and error.

The algorithm suffers from two weaknesses that reflect the gap between our computational model and actual numerical computing. (1) The running time and the error bound depend on $k$, which is $O(n^3)$, as shown below. In practice, $k$ is constant for generic input and is $O(N)$ for degenerate input. (2) The error bound does not apply in a $kn\epsilon$ neighborhood of the curve tails. We can fix this by extending each curve to the left by a short horizontal line segment, called a *telomere,* but the telomeres can cause $O(n^2)$ extra crossings. In practice, the error bound holds at tails without telomeres, and using telomeres only slightly increases the running time.

We implement the arrangement algorithm for semi-algebraic curves (Sec. 5). The only numerical operation is computing the roots of a system of two polynomials with floating point coefficients. We obtain mean/max $\epsilon$ values of $10^{-16}/10^{-12}$ on the unit box $-1 \le x, y \le 1$ for curves of degree 1–10, including the highly degenerate ones described next. Finding the roots of two polynomials of degree $d$ takes $cd^4$ time with $c = 6$ microseconds on a 2.2GHz AMD Athlon. The running time and the error bound are independent of the number of inconsistencies, $k$, across the test data. Singular and nearly singular input curves can exhibit larger $\epsilon$ values.

We validate the software on the core operations of curve fitting and curve intersection. Any system that models with planar curves needs these operations. Exact methods are impractical because iteration generates points and curves of unbounded bit complexity: fit curves to points, intersect these curves to generate new points, fit curves to these points, and so on. The operations are challenging for approximate methods because degeneracy is common. One source of degeneracy is three curves that meet at a point by design, whereas three random curves meet with probability zero. Another source is nearly identical curves. It is common to generate a copy of a curve by fitting a second curve to a set of its points. The two curves will differ because of rounding error in the fitting algorithm. Typically, the curves are nearly identical and their crossings are degenerate. We show that our software handles these types of degeneracy. The number of inconsistencies is at most $3N$, the output accuracy is close to $\epsilon$, and the running time is close to that of the standard, non-robust floating point sweep.

## 2. Prior work

We discuss prior work on arrangement algorithms that employ exact geometry, perturbation, and numerical approximation. Halperin[2] surveys arrangements with a focus on linear objects.

### 2.1. *Exact methods*

Exact Computational Geometry employs custom geometric algorithms, constructive root bounds, and floating point filters to compute correct combinatorial structures. Yap[1] surveys the approach. The main results are as follows.

Keyser[3] computes arrangements of non-degenerate rational parametric curves with an $O(n^2)$ algorithm. Arranging 12 curves of degree at most 4 with 80 bit coefficients takes 1142 seconds on a 400MHz Pentium 2.

LEDA[4] and CGAL[5] compute arrangements of line segments via generalizations of Bentley's sweep algorithm that employ filtered rational arithmetic. Wein[6] extends the CGAL arrangement algorithm to conics. Arranging 20 random conics takes 2 seconds on a 450MHz Pentium 2. Berberich[7] extends the LEDA arrangement algorithm to conics. Arranging 60 random conics with 50 bit coefficients takes 49 seconds on a 846MHz Pentium 3. Eigenwillig[8] extends the LEDA arrangement algorithm to cubics. Arranging 60/90/120/250 random cubics with 100 bit coefficients takes 20/60/110/180 seconds on a 1.2GHz Pentium 3. Geismann[9] computes arrangements of special quartics (used to compute arrangements of 3D quadratics) with a sweep algorithm. Arranging 3 quartics with 30 bit coefficients takes 186 seconds on a Pentium 700. Wolpert[10] computes arrangements of nonsingular algebraic curves by a sweep algorithm, which is not implemented.

Mourrain[11] computes arrangements of 3D quadratics by a plane sweep algorithm, which is not implemented. Geismann[9,12] computes arrangements of 3D quadratics. Keyser[13] computes arrangements of low-degree sculpted solids without degeneracies.

Our algorithm far outperforms these exact algorithms. We have tested it on degree 10 curves, versus degree 3 for prior work. The output accuracy exceeds $10^{-9}$ even for highly degenerate inputs. It is much faster than any prior algorithm. For example, we can arrange 1000 cubics in 100 seconds, versus about 2000 seconds for the best published result (assuming that running time scales quadratically with input size and adjusting roughly for processor speed).

### 2.2. *Perturbation methods*

Halperin and Leiserowitz[14] compute arrangements of circles by a perturbation method that need not compute the correct combinatorial structure. They perturb the input so that each floating point computation is guaranteed to be correct with respect to the perturbed circles. Their method is useful when the perturbation is much less than the manufacturing accuracy, although the output may be incorrect for the input circles.

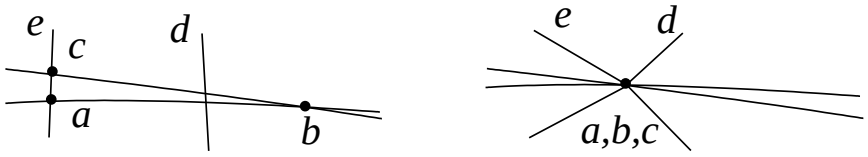


Fig. 2. Collapsing triangle *abc*: (a) actual curves; (b) realization.

### 2.3. *Numerical methods*

Fortune[15] surveys prior work on numerical methods for robustness in computational geometry that attempts to make direct use of floating point in the spirit of numerical analysis. He notes two major approaches. One approach[16] formalizes geometric rounding. Fortune points out that "the generalization to complex geometric objects is not straightforward." Triangles whose largest inscribed circle diameter is below the floating point threshold (Fig. 1) can round to points. The realization error can be large when the triangles are long and skinny (poor aspect ratio). For example, collapsing triangle *abc* in Fig. 2 forces the realizations of lines $d$ and $e$ to intersect even though the true lines are far apart.

We follow the other approach that Fortune[15] outlines: "A second floating-point approach is modeled on the error analysis of numerical methods, particularly linear algebra. The goal is to show that a suitably implemented algorithm provides an answer that is in some precise sense near the mathematically correct answer. Error analysis of geometric algorithms requires consideration of both combinatorial and numeric structure. Often it is easy to argue that an algorithm produces combinatorially valid output... at least with suitably relaxed requirements. It has turned out to be much more difficult to argue that the numerical error associated with combinatorial structure is small. Full error analysis has been carried [out] only for a few simple algorithms."

Our research distinguishes itself from prior work by expressing the running time and error in terms of the number of inconsistencies. We consider the approximate computation as a separate module and treat the number $k$ of inconsistencies that it generates as an input property. We express the running time and the error in terms of $k$. We demonstrate experimentally that $k$ is small, hence that the algorithm achieves "floating point speed with floating point accuracy." Moreover, we handle semi-algebraic curves, whereas prior work is restricted to linear objects.

## 3. Input specification

The input to the arrangement algorithm is a set $S$ of curves and a crossing module. A curve is a sub-manifold of the $x, y$ plane that is the graph of a continuous function $y = f(x) : \mathbf{I}(f) \rightarrow \Re$ with $\mathbf{I}(f)$ a compact interval. Let $\min_x(f) = \min(\mathbf{I}(f))$ and $\max_x(f) = \max(\mathbf{I}(f))$. The curve endpoints are $\text{tail}(f) = (\min_x(f), f(\min_x(f)))$

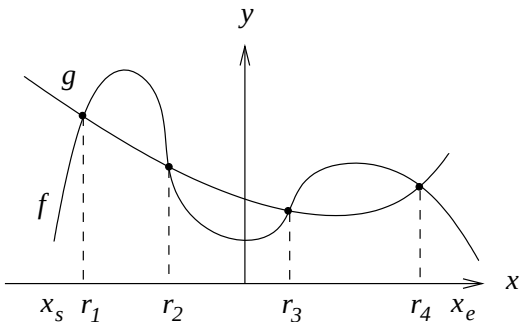


Fig. 3. Sample crossing list.

and $\text{head}(f) = (\max_x(f), f(\max_x(f)))$.

The crossing module takes curves $f, g$ and returns a *crossing list* $\langle f, g, r_1, r_2, \ldots, r_m \rangle$ where the $r_i$ approximate the roots of $f(x) = g(x)$ at which their vertical order changes (Fig. 3). Degenerate roots are discussed below. The $r_i$ are in the interior of $[x_s, x_e] = \mathbf{I}(f) \cap \mathbf{I}(g)$. The order $f, g$ indicates that $f(x) < g(x)$ for $x_s \leq x < r_1$; otherwise the module returns $\langle g, f, r_1, r_2, \ldots, r_m \rangle$. The function $\text{next}(f, g, x)$ denotes the next crossing after $x$, the minimum element of $\{r_i \,|\, r_i > x\}$, and is undefined for $x \geq r_m$. The predicate $f <_x g$ denotes that $f$ is below $g$ at $x$ according to the crossing list. It is true for $x \in [x_s, r_1) \cup [r_2, r_3) \cup [r_4, r_5) \cup \cdots$ and is false elsewhere in $[x_s, x_e]$. This definition implies $f <_x g \equiv \neg(g <_x f)$.

Curve endpoints and crossings are floating point numbers. The only operations that the sweep algorithm performs on them are $x < y$ and $x = y$, which are exact in floating point. We assume that $f <_x g$ and $\text{next}(f, g, x)$ are evaluated in constant time. The actual cost as a function of algebraic degree is analyzed in Sec. 5.2.

### 3.1. *Inconsistency*

The crossing module is inconsistent for curves $f, g, h$ at $x$ when they are in cyclic vertical order: $f <_x g$, $g <_x h$, and $h <_x f$. The inconsistent $x$ lies in the interval $[r, s)$ that is bounded by the closest crossings/endpoints $r \leq x$ and $s > x$ in the three lists. The curves $f, g, h$ and the interval $[r, s)$ comprise an inconsistency. Let $k$ denote the number of inconsistencies among all triples of input curves for all $x$ values. One could compute $k$ by invoking the crossing module on all pairs of curves and examining the results for each triple. Thus, $k = \text{O}(cn^3)$ for curves with $c$ crossings, where $c = \text{O}(d^2)$ for curves of algebraic degree $d$. Although four or more curves also can be inconsistent, we show below that the number of triples governs the performance of the arrangement algorithm.

### 3.2. *Degeneracy*

Crossing lists cannot represent degenerate intersections where two curves touch without crossing or are identical over an $x$ interval. These cases are handled as follows. Touching points are omitted. If $f, g$ are identical over $[x_s, x_e]$, their crossing list is $\langle f, g\rangle$. If they are identical over one or more subintervals, each interval is merged with an adjacent interval. The crossing list accuracy is unaffected by these modifications because arbitrarily small perturbations of the curves make the modified lists correct. Degenerate and nearly degenerate crossings are the main cause of inconsistent crossing lists. The sweep algorithm detects and corrects them efficiently, accurately, and without symbolic perturbation.

### 3.3. *Accuracy*

We assume that each $f, g$ crossing, $p$, in the unit box $-1 \le x, y \le 1$ is connected to an approximate crossing, $q = (r_i, q_y)$, by a path that stays within $\epsilon$ of both curves. The approximate crossing is $\epsilon$ close to both curves, but it can be far from the true crossing. In numerical analysis terms, we bound the backward error, not the forward error. The crossing list has the following properties.

**Lemma 1.** *Let $f <_x g$: (1) $f(x) \le g(x)$ or $\exists y$ for which $\mathrm{dist}((x, y), f) \le \epsilon$ and $\mathrm{dist}((x, y), g) \le \epsilon$; (2) if $\forall i\, |x - r_i| > \epsilon$, $f(x) \le g(x)$ or $\mathrm{dist}((x, f(x)), g) \le 2\epsilon$ and $\mathrm{dist}((x, g(x)), f) \le 2\epsilon$.*

**Proof.** The crossing list order is correct outside the intervals between true and approximate crossings, so $f <_x g$ implies $f(x) \le g(x)$ there. It remains to consider the interval $[p_x, q_x]$; the interval $[q_x, p_x]$ is similar. As a point, $q'$, traverses an $\epsilon$ path from $q$ to $p$, $q'_x$ covers $[p_x, q_x]$ (Fig. 4). For $x = q'_x$, set $y = q'_y$ to prove (1). The projections of $q'$ onto $f$ and $g$, $a'$ and $b'$, cover $[p_x, \min(a_x, b_x)]$ where $a$ and $b$ are the points of $f$ and $g$ closest to $q$. Since $|a'b'| \le |a'q'| + |q'b'|$, $\mathrm{dist}(a', g) \le 2\epsilon$ and $\mathrm{dist}(b', f) \le 2\epsilon$. This proves (2) because $|qa| \le \epsilon$ implies $a_x \ge q_x - \epsilon$ and likewise for $b$. If $f$ is steep, $q_x - a_x$ can be arbitrarily close to $\epsilon$, as shown. □

We give an informal mathematical estimate of the $\epsilon$ that is reasonable to expect from a numerical solver. Curves $f$ and $g$ are subsets of the zero sets of bivariate polynomials $F(x, y), G(x, y)$ that have a common zero at $p$ in the unit box. Every solver generates a sequence, $q_i$, for which $F(q_i)$ and $G(q_i)$ converge to zero. We assume that floating point evaluation of the sequence converges to a $q$ for which $|F(q)| < \delta$ and $|G(q)| < \delta$ where $\delta$ is the polynomial evaluation accuracy. We expect $\delta$ to be on the order of the floating point rounding unit (about $10^{-16}$ for ANSI double float), assuming the polynomials have bounded degree and coefficients. As $|F(q)|$ converges to zero, the distance from $q$ to the curve $F$ converges to $|F|/||\nabla F||$, evaluated at $q$, where $\nabla F = (F_x, F_y)$ is the gradient and where subscripts denote partial derivatives. Thus, the distances from $q$ to $F$ and $G$ are bounded by $\delta/||\nabla F||$ and $\delta/||\nabla G||$. The larger of these two quantities is $\epsilon$.

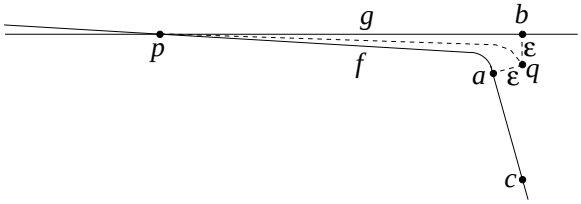


Fig. 4. Worst case for approximate intersection $q$: $q_x - p_x$ can be large, and the distance from $c = (q_x, f(q_x))$ to $g$ can also be large.

## 4. Arrangement algorithm

The arrangement algorithm is a vertical line sweep that uses the crossing module to compute and process sweep events. When the module output is consistent, our algorithm is equivalent to the Bentley-Ottman algorithm. Inconsistencies can create two novel situations: (1) a swap event is encountered for two curves that are not adjacent in the sweep list and (2) two curves become adjacent in the sweep and their sweep order contradicts their crossing list order. The former is handled by discarding the impossible swap. The latter is handled by swapping the curves to bring them into crossing list order.

The algorithm employs two data structures.

(1) a list $L$ of curves, called the *sweep list,* that represents the order of the curves from lowest to highest along a vertical sweep line. $L$ is implemented as a red-black binary tree whose in-order traversal is the list order. The successor and predecessor of $f$ in $L$ are denoted $\text{succ}(f)$ and $\text{pred}(f)$.
(2) a priority queue $P$ of events: $\text{insert}(f, x)$, $\text{remove}(f, x)$, $\text{swap}(f, g, x)$, and $\text{check}(f, g, x)$. Events are dequeued in increasing $x$ order. Ties are broken arbitrarily, except that removes come before other events and inserts come after other events. $P$ is implemented as a heap.

For each $f \in S$, the algorithm initializes $P$ with $\text{insert}(f, \min_x(f))$ and $\text{remove}(f, \max_x(f))$. It then repeatedly dequeues and processes events from $P$.

- $\text{insert}(f, x)$: Insert $f$ into $L$: if $f <_x g$ at node $g$, go left else go right. If $f$ is not first in $L$, enqueue $\text{check}(\text{pred}(f), f, x)$ into $P$. If $f$ is not last in $L$, enqueue $\text{check}(f, \text{succ}(f), x)$.
- $\text{remove}(f, x)$: Remove $f$ from $L$. If $f$ is neither first nor last in $L$, enqueue $\text{check}(\text{pred}(f), \text{succ}(f), x)$.
- $\text{swap}(f, g, x)$: If $g \neq \text{succ}(f)$, discard the event. Otherwise, swap $f$ and $g$ in $L$. Enqueue $\text{check}(g, f, x)$. If $f$ is not last, enqueue $\text{check}(f, \text{succ}(f), x)$. If $g$ is not first, enqueue $\text{check}(\text{pred}(g), g, x)$.
- $\text{check}(f, g, x)$: If $g \neq \text{succ}(f)$ or $\max_x(f) \leq x$ or $\max_x(g) \leq x$, discard the event. If $g <_x f$, enqueue $\text{swap}(f, g, x)$. Otherwise, if the next crossing $r = \text{next}(f, g, x)$ is defined, enqueue $\text{swap}(f, g, r)$.

Event processing presupposes that curves can be located in $L$. Location is performed by assigning every curve a pointer to its tree node, and every node a pointer to its parent. The evolving sweep list is converted to an arrangement structure using standard techniques. When $P$ becomes empty, the sweep ends and the arrangement is complete. Vertical line segments can be added to the output in linear time; we omit the details.

### 4.1. *Running time*

The sweep defines an *output crossing list* for each pair of curves. Let $L(r)$ denote the state of $L$ immediately after the algorithm finishes processing every event in $P$ with $x \leq r$. Let $f <'_r g$ denote that $f$ precedes $g$ in $L(r)$. The $f, g$ output crossing list is $\langle f, g, r'_1, r'_2, \ldots, r'_{m'} \rangle$ where the $r'_i$ are the $x$ values where $f$ and $g$ swap in $L$. The $r'_i$ are identical to the $r_i$ in the absence of inconsistency, but differ when swaps are discarded or are added at non-crossings by check events. Due to inconsistency, it is even possible that $g <_{x_s} f$ yet $f <'_{x_s} g$ at $x_s = \max(\min_x(f), \min_x(g))$: the input crossing list is $\langle g, f, r_1, r_2, \ldots, r_m \rangle$ and the output crossing list is $\langle f, g, r'_1, r'_2, \ldots, r'_{m'} \rangle$. This can happen when $g$ is inserted in the sweep on the incorrect side of $f$ (from the crossing list perspective) because an intervening segment prevents $f, g$ from being compared during the insertion. We have $f <'_x g$ for $x \in [x_s, r'_1) \cup [r'_2, r'_3) \cup [r'_4, r'_5) \cup \cdots$ and $g <'_x f$ elsewhere in $[x_s, x_e]$.

We show that the output crossing lists are consistent and have $C = N + \min(3kn, n^2/2)$ crossings for $n$ curves with $N$ crossings and $k$ inconsistencies. This implies that the arrangement has $V = 2n + C$ vertices and that the running time is $\mathrm{O}(V \log n)$.

**Lemma 2.** *The output crossing lists are consistent.*

**Proof.** If $f <'_x g$ and $g <'_x h$, $f$ precedes $g$ precedes $h$ in $L(x)$, so $f$ precedes $h$ and $f <'_x h$. □

**Lemma 3.** *Immediately after* $\mathrm{insert}(f, x)$ *is processed,* $\mathrm{pred}(f) <_x f$ *and* $f <_x \mathrm{succ}(f)$.

**Proof.** After insertion and before balancing, $f$ is a leaf. Its successor is the nearest ancestor $g$ whose left subtree contains $f$. The insertion went left at $g$, so $f <_x g$. Balancing the tree does not change the successor. Predecessors are analogous. □

**Lemma 4.** *If* $g = \mathrm{succ}(f)$ *in* $L(x)$, *then* $f <_x g$.

**Proof.** Let $[a, b)$ be a maximal interval on which $g = \mathrm{succ}(f)$ in $L(x)$. Some event at $a$ establishes $g = \mathrm{succ}(f)$ and enqueues $\mathrm{check}(f, g, a)$. There can be many establishing and disestablishing events at $a$; only the end result matters. The fact that $g = \mathrm{succ}(f)$ in $L(a)$ implies that $f <_a g$. Otherwise, the check would enqueue

swap$(f, g, a)$ and $g = \text{succ}(f)$ in $L(a)$ would be false. The value of $f <_x g$ next changes at $r = \text{next}(f, g, a)$. If $r < b$ were true, swap$(f, g, r)$ would be executed, which contradicts the maximality of $[a, b)$. Hence, $r \geq b$ and $f <_x g$ on $[a, b)$. □

This lemma shows that adjacent curves in $L(x)$ are in crossing list order. We generalize this *local consistency* property to sublists of $L(x)$. The list $H = h_1, \ldots, h_p$ with $p \geq 2$ is locally consistent when $h_i <_x h_{i+1}$ for $i < p$. It is *minimal* when removing any of $h_2, \ldots h_{p-1}$ yields an inconsistent sublist, which implies that $h_{i+2} <_x h_i$ for $i < p - 1$. Although non-adjacent curves in $L(x)$ need not be in crossing list order, they can be linked by a minimal locally consistent list of length at most $k_x + 2$ with $k_x \leq k$ the number of inconsistencies at $x$.

**Lemma 5.** *If $f <'_x g$, there exists a minimal locally consistent list from $f$ to $g$ of length at most $k_x + 2$.*

**Proof.** The list $h_1 = f, h_2 = \text{succ}(h_1), \ldots, h_p = g$ is locally consistent by Lemma 4. If $h_i <_x h_{i+2}$ for some $i < p - 1$, delete $h_{i+1}$ to obtain a locally consistent list of length $p - 1$. Repeat this process as long as possible to obtain a minimal list of length $l$. Each of the $l - 2$ triples of consecutive list elements is inconsistent at $x$, so $l - 2 \leq k_x$ and $l \leq k_x + 2$. □

**Lemma 6.** *If $f <'_x g$ and $g <_x f$, then $f, h_2, h_3$ are inconsistent at $x$ for some $h_2, h_3 \in S$.*

**Proof.** Form the minimal locally consistent list from $h_1 = f$ to $h_l = g$. We have $l > 2$ because $g <_x f$, so $h_1 = f, h_2, h_3$ are inconsistent at $x$. □

**Lemma 7.** *The algorithm executes at most $C = N + \min(3kn, n^2/2)$ swap events.*

**Proof.** Let the crossing list for $f, g \in S$ be $\langle f, g, r_1, \ldots, r_m \rangle$, so $f <_x g$ is constant on the $m + 1$ intervals $(-\infty, r_1), [r_1, r_2), [r_2, r_3), \ldots, [r_{m-1}, r_m), [r_m, \infty)$. The only time swap$(f, g, x)$ is executed is when $g = \text{succ}(f)$ and $g <_x f$. This makes $g = \text{succ}(f)$ false, so no further swaps are enqueued in the current interval. Therefore, at most one swap is executed in each of the $m + 1$ intervals.

If a swap is executed at $a < r_1$, it is swap$(g, f, a)$, since $f <_a g$, and $g$ precedes $f$ in $L$ before the swap. Suppose $f$ is inserted later than $g$ and let $b = \min_x(f)$. If $b = a$, Lemma 3 implies that $f$ cannot be inserted as succ$(g)$. Inserts are processed after deletes and swaps, so the intervening curves persist in $L(a)$ and $f, g$ cannot swap at $a$. We conclude that $b < a$, $g <'_b f$, and $f$ belongs to an inconsistency according to Lemma 6. Charge the extra crossing to this inconsistency. Each curve can have $n - 1$ crossing lists, hence $n - 1$ extra crossings, and each inconsistency involves 3 curves. The $k$ inconsistencies are charged at most $3k(n-1)$ times. There can be at most one extra crossing for each of the $n(n - 1)/2$ pairs of curves. □

**Theorem 1.** *The arrangement has at most $V = 2n + C$ vertices and the sweep has running time $O(V \log n)$.*

**Proof.** Swaps generate at most $C$ vertices by Lemma 7, insertions and deletions generate $2n$ vertices, and checks generate no vertices. This proves the vertex bound of $V$. Each insert, remove, and executed swap enqueues up to three checks. Each check enqueues at most one swap. Therefore, the total number of events is a constant times the number of insert, remove, and executed swaps, which is bounded by $V$. An event is processed by updating $L$ and $P$ in $O(\log n)$ time. □

### 4.2. *Realizability*

The sweep algorithm constructs a combinatorial arrangement with one vertex per insert, remove, and executed swap. It determines the vertex $x$ coordinates, but not their $y$ coordinates. We prove that this combinatorial structure is realized by curves that are close to the input. The proof consists of three steps: (1) define offset curves that realize the sweep output; (2) show that the realization is $\delta$ accurate when the output crossing lists are $\delta$ *realizable;* and (3) prove $\delta$ realizability.

**Lemma 8.** *For every $x \in \mathbf{I}(f)$, $\mathrm{dist}((x, y), f)$ is zero at $y = f(x)$ and increases monotonically as $y$ moves away from $f(x)$.*

**Proof.** The distance is zero at $y = f(x)$ because $(x, y)$ is on $f$. Given $p$ and $q$ with $p_x = q_x$ and $p_y > q_y > f(p_x)$, we show that $\mathrm{dist}(q, f) < \mathrm{dist}(p, f)$. Let $p'$ be a point of $f$ nearest to $p$. If $p'_y \le q_y$, then $\mathrm{dist}(q, f) \le |qp'| < |pp'| = \mathrm{dist}(p, f)$. If $p'_y > q_y$, we know $f(p_x) < q_y$ and $f(p'_x) = p'_y > q_y$. By the intermediate value theorem, there exists $x \in (p_x, p'_x)$ such that $f(x) = q_y$. Let $q' = (x, f(x))$. Hence, $\mathrm{dist}(q, f) \le |qq'| = |q_x - q'_x| < |q_x - p'_x| = |p_x - p'_x| \le |pp'| = \mathrm{dist}(p, f)$. □

Lemma 8 proves the existence of curves $y = f_{+\delta}(x)$ and $y = f_{-\delta}(x)$ at distance $\delta > 0$ above and below $f$ for $x \in \mathbf{I}(f)$. Using them, we realize each $f \in S$ with

$$\hat{f}(x) = \max_{e <'_x f} e_{-\delta}(x).$$

The condition $e <'_x f$ is shorthand for $e \in S$, $x \in \mathbf{I}(e)$, and $e <'_x f$. We define $f <'_x f$ for $x \in \mathbf{I}(f)$, so $f_{-\delta}(x)$ is included in the maximum. The function $\hat{f}(x)$ is possibly discontinuous when $\{e \mid e <'_x f\}$ changes, which can only happen at $f$ crossings. We define $\hat{f}$ to be the continuous curve that results from filling in the discontinuities with vertical line segments.

Figure 5 shows the realization of line segments $f$ and $g$ near crossing $r$. The segments $f_{-\delta}$ and $g_{-\delta}$ are parallel to $f$ and $g$. The realization curve $\hat{f}$ (shown with bold dashes) consists of four line segments. In segment 1, $\hat{f}(x) = f_{-\delta}(x)$ because $f <'_x g$. In segments 3 and 4, $\hat{f}(x) = \max\{f_{-\delta}(x), g_{-\delta}(x)\}$ because $g <'_x f$. The crossing $r$ causes the discontinuity in $\hat{f}$ that is bridged by segment 2. The realization curve $\hat{g}$ is $g_{-\delta}$.

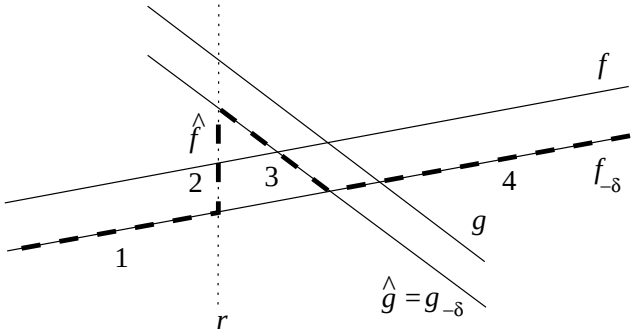


Fig. 5. Realization of curves $f$ and $g$ near crossing $r$.

**Lemma 9.** *$f <'_x g$ implies $\hat{f}(x) \leq \hat{g}(x)$.*

**Proof.** By Lemma 2, $e <'_x f$ and $f <'_x g$ imply $e <'_x g$. Therefore $\hat{f}(x)$ is the maximum of a subset of the elements that define $\hat{g}(x)$ and $\hat{f}(x) \leq \hat{g}(x)$. □

Lemma 9 shows that vertical order of $\hat{f}$ and $\hat{g}$ equals the output crossing list order. The domain of $\hat{f}$ equals that of $f$. Thus, the $\hat{f}$ realize the output crossing lists. The vertical segments are a consequence of the discontinuity in $f <'_x g$ at output crossings. They can be eliminated by a local perturbation to obtain realization *functions* that are equal at crossings.

**Definition 1.** The $f, g$ output crossing list is $\delta$ **realizable** at $x$ if $f <'_x g$ implies $f_{-\delta}(x) \leq g_{+\delta}(x)$.

**Lemma 10.** *if all the output crossing lists are $\delta$ realizable at $x$, then $f_{-\delta}(x) \leq \hat{f}(x) \leq f_{+\delta}(x)$.*

**Proof.** Since $f_{-\delta}(x)$ is one of the elements in the definition of $\hat{f}$, $f_{-\delta}(x) \leq \hat{f}(x)$. For each $e$ with $e <'_x f$, $e_{-\delta}(x) \leq f_{+\delta}(x)$ by $\delta$ realizability, so $\hat{f}(x) \leq f_{+\delta}(x)$. □

It remains to prove that each $f, g$ output crossing list is $\delta$ realizable, except near $x_s$. The geometric intuition behind the proof is as follows. Suppose $f$ is below $g$ at $x$ in the output order ($f <'_x g$). If $f$ is below $g$ according to the crossing module ($f <_x g$), then by Lemma 1 (1) either $f$ is truly below $g$ ($f(x) \leq g(x)$) or the curves are $\epsilon$ close to a point $(x, y)$. In the latter case, a downward offset of $f$ is below an upward offset of $g$ ($f_{-\epsilon}(x) \leq g_{+\epsilon}(x)$). If there is no such $x$, we form a minimal locally consistent list (MLCL) from $f$ to $g$. Using Lemma 1 (2), we find a chain of points on these segments such that each consecutive pair is correctly ordered in $y$ or within $2\epsilon$. This chain implies a bound on the $f, g$ realization error.

The first step in the proof is to define a sequence of $x$ candidates. Let $x = x_0$ be a value such that $f <'_{x_0} g$ and select an MLCL from $f$ to $g$ at $x_0$. Let $x_1$ be

the minimal value such that this list is an LCL for $x \in [x_1, x_0)$. If $f$ and $g$ do not swap or start at $x_1$, select an MLCL at $x = x_1^-$, the largest floating point value less than $x_1$. Let $x_2$ be the minimal value such that this list is an LCL for $x \in [x_2, x_1)$. Similarly, construct $x_3, x_4, \ldots$. Let $l_j+1$ denote the length of the LCL in the interval $[x_{j+1}, x_j)$.

**Lemma 11.** *$l_j \leq k + 1 - j$.*

**Proof.** By construction, $f$ and $g$ do not start or swap at $x_i$, $i = 1, 2, \ldots, j$. Since $x_i$ is the leftmost value at which the MLCL at $x_{i-1}^-$ remains an LCL, some other segment in the LCL must start at $x_i$ or some pair of consecutive segments in the LCL must have a crossing at $x_i$. Therefore, $x_i$ is at or to the left of the beginning of one of the inconsistencies in the MLCL. (The left endpoint of this inconsistency might be to the right of $x_i$ because the LCL is not necessarily minimal at $x_i$.) Since each $x_i^-$ is to the left of a different inconsistency, the number of inconsistencies in the interval $[x_{j+1}, x_j)$ is at most $k - j$. The result follows from Lemma 5. □

We say that an interval $[x_{j+1}, x_j)$ is long if $x_j - x_{j+1} \geq 2(l_j - 1)\epsilon$. Our strategy for analyzing the realizability of $f$ and $g$ at $x_0$ depends on whether there is a long interval. We use Lemma 13 to handle the case where there are no long intervals and use Lemma 14 to handle long intervals.

**Lemma 12.** *Let $p = (x, y_1)$ and $q = (x, y_2)$ with $y_1 \leq y_2$. If $\mathrm{dist}(p, f) \leq \delta$ and $\mathrm{dist}(q, g) \leq \delta$, then $f_{-\delta}(x) \leq g_{+\delta}(x)$.*

**Proof.** By Lemma 8, $f_{-\delta}(x) \leq p_y$ and $q_y \leq g_{+\delta}(x)$. □

**Lemma 13.** *If $f <_x g$, then $f$ and $g$ are $|x_0 - x| + \epsilon$ realizable at $x_0$.*

**Proof.** By Lemma 1, either $f(x) \leq g(x)$ or $\exists y$ with $\mathrm{dist}((x, y), f) \leq \epsilon$ and $\mathrm{dist}((x, y), g) \leq \epsilon$. If $f(x) \leq g(x)$, set $p = (x_0, f(x))$ and $q = (x_0, g(x))$. Since $(x, f(x))$ lies on $f$, $\mathrm{dist}(p, f) \leq |x_0 - x|$. If $f(x) > g(x)$, set $p = q = (x_0, y)$, for which $\delta(p, f) \leq |x_0 - x| + \delta((x, y), f) \leq |x_0 - x| + \epsilon$. Similarly, $\mathrm{dist}(q, g) \leq |x_0 - x| + \epsilon$ in both cases. By Lemma 12, $f$ and $g$ are $|x_0 - x| + \epsilon$ realizable (Def. 1) at $x_0$. □

**Lemma 14.** *If $[x_{j+1}, x_j)$ is long, $f$ and $g$ are $x_0 - x_j + (2l_j - 1)\epsilon$ realizable at $x_0$.*

**Proof.** Set $x = x_j - (l_j - 1)\epsilon$. Since $x_j - x_{j-1} \geq 2(l_j - 1)\epsilon$ by definition of long, $x_j - x = (l_j - 1)\epsilon$ and $x - x_{j-1} \geq (l_j - 1)\epsilon$.

Let $h_0, \ldots, h_{l_j}$ be the MLCL from $f = h_0$ to $g = h_{l_j}$ selected at $x_j^-$. We will place points $p^0, \ldots, p^{l_j}$ on $h_0, \ldots, h_{l_j}$ such that either $p_x^{i-1} = p_x^i$ and $p_y^{i-1} \leq p_y^i$ or such that $|p^{i-1}p^i| \leq 2\epsilon$ for $i = 1, \ldots, l_j$. In either case, $|p_x^{i-1} - p_x^i| \leq 2\epsilon$.

If $l_j$ is even, for $i = l_j/2$, set $p^i = (x, h_i(x))$. By Lemma 1(2), either $h_i(p_x^i) \leq h_{i+1}(p_x^i)$ or $\mathrm{dist}(p^i, h_{i+1}) \leq 2\epsilon$. If the former, set $p^{i+1} = (p_x^i, h_{i+1}(p_x^i))$. If the latter,

set $p^{i+1}$ to be the point on $h^{i+1}$ closest to $p^i$. Repeat for $i = l_j/2+1, \ldots, l_j - 1$. Similarly, for $i = l_j/2$ again, set $p^{i-1}$ equal $(p^i_x, h_{i-1}(p^i_x))$ or the point of $h_{i-1}$ closest to $p^i$, and repeat for $i = l_j/2 - 1, \ldots, 1$.

If $l_j$ is odd, again set $i = l_j/2$, even though this is half-integral. If $h_{i-\frac{1}{2}}(x) \leq h_{i+\frac{1}{2}}(x)$, set $p^i = (x, (h_{i-\frac{1}{2}}(x) + h_{i+\frac{1}{2}}(x))/2)$, $p^{i-\frac{1}{2}} = (x, h_{i-\frac{1}{2}}(x))$, and $p^{i+\frac{1}{2}} = (x, h_{i+\frac{1}{2}}(x))$. (Since $i = l_j/2$ is half-integral, $i-\frac{1}{2}$ and $i+\frac{1}{2}$ are integers.) Otherwise, set $p^i = (x, y)$ from Lemma 1(1) and set $p^{i-\frac{1}{2}}$ and $p^{i+\frac{1}{2}}$ to the points of $h_{i-\frac{1}{2}}$ and $h_{i+\frac{1}{2}}$ closest to $p^i$. By the lemma, either $p^i_x = p^{i+\frac{1}{2}}_x$ and $p^i_y \leq p^{i+\frac{1}{2}}_y$ or $|p^i p^{i+\frac{1}{2}}| \leq \epsilon = 2(\frac{1}{2})\epsilon$. In either case, $|p^i_x - p^{i+\frac{1}{2}}_x| \leq 2(\frac{1}{2})\epsilon$. Similarly for $i - \frac{1}{2}$. For $i = (l_j+1)/2 + 1, \ldots, l_j$ and for $i = (l_j - 1)/2 - 1, \ldots, 0$, set $p^i$ using Lemma 1(2) as for even $l_j$. Both even and odd $l_j$ create $p^{l_j/2}$, but for odd $l_j$, it does not lie on any $h_i$ because $l_j/2$ is half-integral.

This process fails if we apply Lemma 1(2) too near to an approximate root of some pair $h_{i-1}, h_i$, $i = 1, \ldots, l_j$. However, the farthest points from $x$ to which we apply the lemma are $p^1$ and $p^{l_j-1}$. Telescoping the bounds on $|p^{i-1}_x - p^i_x|$, we obtain $|p^1_x - p^{l_j/2}_x| \leq 2\epsilon(l_j/2 - 1) = (l_j - 2)\epsilon$ and similarly, $|p^{l_j-1}_x - p^{l_j/2}_x| \leq 2\epsilon(l_j/2 - 1) = (l_j-2)\epsilon$. Since $x$ is $(l_j-1)\epsilon$ distant from $x_{j-1}$ and $x_j$ and since the MLCL is an LCL in that interval, $p^1_x$ and $p^{l_j-1}_x$ are $\epsilon$ distant from a crossing of $p^0, p^1$ and $p^{l_j-1}, p^{l_j}$, and the lemma applies.

Now that $p^0, p^1, \ldots, p^{l_j}$ are defined, define points $p$ and $q$ as follows. If $p^0_y \leq p^{l_j/2}_y$, set $p = (x_0, p^0_y)$, else set $p = (x_0, p^{l_j/2}_y)$. If $p^{l_j}_y \geq p^{l_j/2}_y$, set $q = (x_0, p^{l_j}_y)$, else set $q = (x_0, p^{l_j/2}_y)$. Clearly, $p_y \leq p^{l_j/2}_y$ and $q_y \geq p^{l_j/2}_y$ and hence $p_y \leq q_y$. To finish the proof, we just have to show $\text{dist}(p, f) \leq x_0 - x_j + (2l_j - 1)\epsilon$ and $\text{dist}(q, g) \leq x_0 - x_j + (2l_j - 1)\epsilon$ and apply Lemma 12.

Each step in the sequence $p^0, \ldots, p^{l_j/2}, \ldots, p^{l_j}$ is either straight up in $y$ or less that $2\epsilon$ in distance. (If $l_j$ is odd, and $l_j/2$ is half-integral, the distance bound for $p^{(l_j-1)/2}, p^{l_j/2}$ and $p^{l_j/2}, p^{(l_j+1)/2}$ is $\epsilon$ since the index increases by $\frac{1}{2}$.) So if $p^0_y \leq p^{l_j/2}_y$, $\text{dist}(p, f) \leq (x_0 - x_j) + (x_j - x) + |x - p^0_x| + \text{dist}(p^0, f) \leq (x_0 - x_j) + (l_j - 1)\epsilon + 2\epsilon(l_j/2 - 0) + 0 = x_0 - x_j + (2l_j - 1)\epsilon$. The third term comes from telescoping the bounds on $|p^{i-1}_x - p^i_x|$ and the fourth term from the fact that $f = h_0$. If $p^0_y > p^{l_j/2}_y$, then $|p^0 p^{l_j/2}| \leq 2\epsilon(l_j/2 - 0) = l_j\epsilon$ since the ultimate change in $y$ is negative and so the steps up in $y$ do not add to the error. Hence, $\text{dist}(p^{l_j/2}, f) \leq |p^{l_j/2} p^0| \leq l_j\epsilon$, since $p^0$ lies on $f$. Finally, $\text{dist}(p, f) \leq (x_0 - x_j) + (x_j - x) + \text{dist}(p^{l_j/2}, f) \leq (x_0 - x_j) + (l_j - 1)\epsilon + l_j\epsilon = x_0 - x_j + (2l_j - 1)\epsilon$. The bound on $\text{dist}(q, g)$ is obtained similarly. □

**Theorem 2.** *If $x_0 \geq \max(\min_x(f), \min_x(g)) + K\epsilon$, then $f$ and $g$ are $(K+1)\epsilon$ realizable at $x_0$ for $K = \min(k(k+1), 2\min(k, n-2)\min(k, n+N))$.*

**Proof.** Consider the sequence $x_0 > x_1 > \cdots > x_m$ such that $m \leq k$: the sequence might be longer, but we stop at $m = k$. If the sequence has no long intervals, then

$x_0 - x_m < 2(l_0-1)\epsilon + 2(l_1-1)\epsilon + \cdots + 2(l_{k-1}-1)\epsilon \leq 2\epsilon(k+(k-1)+\cdots+1) = k(k+1)\epsilon$ by Lemma 11. On the other hand, an MLCL can have at most $n$ elements and hence $l_j - 1 \leq \min(k, n-2)$. Each $x_j^-$ is to the left of the left end of an inconsistency, which is either a segment left endpoint or crossing, and there are $\min(k, n+N)$ of those. Since $x_j - x_{j+1} < 2(l_j - 1)$, $x_0 - x_m < 2\min(k, n-2)\min(k, n+N)$. Hence, $x_0 - x_m < K\epsilon$, and since $x_0 \geq \max(\min_x(f), \min_x(g)) + K\epsilon$, neither segment starts at $x = x_m$ and $f <'_{x_m^-} g$ is defined. If $f <'_{x_m^-} g$ is true, $m = k$ (otherwise we would not have stopped at $m$) and $l_m = 1$ by Lemma 11. This means that $f, g$ are adjacent on $x \in [x_{m+1}, x_m)$, so $f <_{x_m} g$ by local consistency. If $g <'_{x_m^-} f$, the arrangement algorithm must have swapped $f$ and $g$ at $x = x_m$, so $f <_{x_m} g$ as a post condition. By Lemma 13, $f$ and $g$ are $|x_0 - x_m| + \epsilon \leq K\epsilon + \epsilon = (K+1)\epsilon$ realizable at $x_0$.

If the sequence has a long interval, let $[x_{j+1}, x_j)$ be the long interval nearest to $x_0$ (smallest $j$). All intervals to the right of $x_j$ must be short and therefore $|x_0 - x_j| \leq 2(l_0 - 1)\epsilon + 2(l_1 - 1)\epsilon + \cdots + 2(l_{j-1} - 1)\epsilon$. By Lemma 14, $f$ and $g$ are $|x_0 - x_j| + (2l_j - 1)\epsilon \leq 2(l_0-1)\epsilon + 2(l_1-1)\epsilon + \cdots + 2(l_{j-1}-1)\epsilon + 2(l_j-1)\epsilon$ realizable at $x_0$. Since $j < m$, $j \leq k-1$ and the argument of the previous paragraph bounds this sum by $K\epsilon$. □

Trimming $K\epsilon$ off the left end of each segment $f$ means restricting its domain to $[\min_x(f) + K\epsilon, \max_x(f)]$. Let $f^t$ denote the trimmed segment. The following corollary underlies our practical solution to the lack of an error bound near curve tails. We add a short horizontal "telomere" line segment to the tail of each segment, calculate the arrangement, and then trim off the telomeres. In cell biology, a telomere at the end of a strand of DNA loses a few base pairs every time the cell divides. The telomere does not encode any genes: it merely acts to protect the genes from loss of information. Analogously, our telomere segments protect the input segments from insertion error in the arrangement algorithm.

**Corollary 1.** *Let each segment, $f$, be horizontal for $x \in [\min_x(f), \min_x(f) + K\epsilon]$. Trim $K\epsilon$ off every segment. For all $x$ such that $f^t <'_x g^t$, $f^t$ and $g^t$ are $(K+1)\epsilon$ realizable at $x$.*

**Proof.** Given a horizontal segment $ab$ ($a_x < b_x$ and $a_y = b_y$) and a point $p$ such that $p_x \geq b_x$, $\text{dist}(p, ab) = |pb|$. Therefore, if $f$ is constant for $x \in [\min_x(f), \min_x(f) + K\epsilon]$ and if $p_x \geq \min_x(f) + K\epsilon$, then the point $(x', f(x'))$ of $f$ closest to $p$ must have $x' \geq \min_x(f) + K\epsilon$. Therefore, $\text{dist}(p, f) = \text{dist}(p, f^t)$.

By definition of trimming, $f^t <'_x g^t$ implies $x \geq \max(\min_x(f), \min_x(g)) + K\epsilon$ and hence $f$ and $g$ are $(K+1)\epsilon$ realizable at $x$: there exists $p$ and $q$ such that $p_x = q_x = x$, $p_y \leq q_y$, $\text{dist}(p, f) \leq (K+1)\epsilon$, and $\text{dist}(q, g) \leq (K+1)\epsilon$. By the previous paragraph, $\text{dist}(p, f) = \text{dist}(p, f^t)$ and $\text{dist}(q, g) = \text{dist}(q, g^t)$, and hence $f^t$ and $g^t$ are $(K+1)\epsilon$ realizable at $x$. □

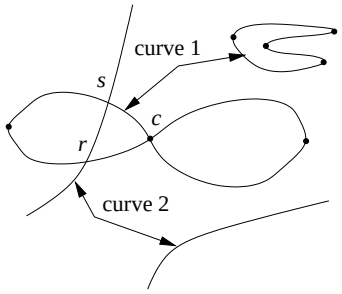


Fig. 6. Algebraic curves.

## 5. Implementation

This section describes our implementation of the arrangement algorithm for semi-algebraic curves and our validation on generic and degenerate inputs.

### 5.1. *Semi-algebraic curves*

Semi-algebraic curves are defined in terms of algebraic curves. An algebraic curve is the zero set of a polynomial $F(x, y)$. A curve point is regular when $\nabla F$ is nonzero and is singular otherwise. The regular points partition into 1D manifolds, called branches, that are topological circles or lines. Two or more branches meet at a singular point. We specify an input curve as a compact, $x$-monotone segment of a branch. Every compact semi-algebraic curve can be expressed as a finite disjoint union of such curves.

Figure 6 shows two algebraic curves that cross at $r$ and $s$. Curve 1 consists of three branches: a topological circle and two topological lines (the left/right loops of a horizontal figure 8) that meet at singular point $c$. Curve 2 consists of two unbounded branches. The dots mark the singular and critical points, which delimit the allowable input curves.

### 5.2. *Crossing module*

The crossing module decomposes each input algebraic curve into an arrangement of $x$-monotone branch segments. The vertices are the singular points and the turning points. The singular points are computed by solving $F_x = F_y = 0$ and returning the roots where $F = 0$. The turning points are computed by solving $F = F_y = 0$. The number of edges that are incident on a vertex is determined by the signs of polynomials in $F_x, F_y, F_{xx}, F_{xy}, F_{yy}$. A standard sweep computes the branch structure from this local information. The local analysis assumes that the turning points and the singular points are non-degenerate. A degenerate point can have arbitrarily many incident edges. Computing its local branch structure is a hard problem for which numerical methods are ill suited.

Given a segment $f$ of $F$ and an $x_0 \in [\min_x(f), \max_x(f)]$, the crossing module computes $f(x_0)$ by solving $F(x_0, y) = 0$ for $y$ then using the branch structure to choose the correct root in $y$. The crossing module constructs the crossing list for $f, g$ on $F, G$ by solving $F = G = 0$, assigning the crossings to the monotone segments of each algebraic curve, and sampling midway between crossings to determine the vertical order. We store crossing lists to avoid recomputing them.

Correctness of these three algorithms is straightforward in exact arithmetic. We make no robustness claims about our floating point implementations. The sole numerical operation is computing the roots of a system of one or two polynomials with floating point coefficients.

Root finding is a core numerical analysis task with several practical solutions. Although good univariate solvers are available,[17] provable robustness is an open problem.[18] One multivariate root finding strategy[19] is to construct a homotopy from the input system to a start system that is easily solvable. Each input system root $(x_r, y_r, 1)$ is linked to a start root $(x_s, y_s, 0)$ by a branch in $(x, y, t)$ space. The branches are traced from the start roots to the input roots, typically by a predictor/corrector integrator. Another approach[20,21] is to construct matrices whose eigenvectors are the system roots then to compute the eigenvectors with a linear algebra package. We adopt this method because it has proved accurate and fast.

#### 5.2.1. *Speed*

We measured the running time for two curves of degree $d$. Theory ensures a polynomial bound. Experiments on random and degenerate inputs yields $cd^4$ time with $c = 6$ microseconds on a 2.2GHz AMD Athlon. Half the time goes to matrix setup in C and the rest to eigenvalue computation with LAPACK.[22] A further factor of 2–4 speedup may be possible by using BLAS in matrix setup and by optimizing the LAPACK implementation.

#### 5.2.2. *Accuracy*

We estimated $\epsilon$ accuracy on 10,000 pairs of algebraic curves of degree $d$ with random coefficients in $[-1, 1]$. We sampled the crossing lists on the unit box with uniform spacing of 0.01 in $x$. We bound the perpendicular error, which determines $\epsilon$, by the vertical error. For $f <_x g$, the vertical error at $x$ is $\min(0, f(x) - g(x))$. The mean/max vertical errors over 10,000 pairs are $10^{-16}/10^{-12}$ for degree 1–10. The arrangement algorithm validation provides similar $\epsilon$ estimates.

We expect the same accuracy for any input, except near singularities. Accuracy drops at isolated singularities. Intervals of singularity cannot be handled by any known numerical solver. Two such curves are identical or share a component. Floating point computation converts these into approximate properties. Nearly identical curves yield accurate crossing lists no matter what crossings the solver computes. Shared components defeat our program. Nearly identical curves arise from standard

curve fitting operations, but shared components do not.

#### 5.2.3. *Inconsistencies*

The crossing module generates no inconsistencies on 100,000 triples of algebraic curves of degree 1–10 with random coefficients in $[-1, 1]$. We replaced each curve $F(x, y)$ with $F(x, y) - F(0.1, 0.2)$, so that all the curves meet at $(0.1, 0.2)$ except for rounding error. Independently of degree, 6% of the curve triples are inconsistent on an interval of average length $10^{-15}$ near $(0.1, 0.2)$. We added a random constant in $[0, 10^{-14}]$ to each polynomial and obtained no inconsistencies. The arrangement algorithm validation yields at most $3N$ inconsistencies for highly degenerate arrangements with $N$ crossings.

### 5.3. *Arrangement algorithm validation*

The arrangement algorithm validation has several goals. The algorithm handles inputs with many localized degeneracies: many vertices incident on many algebraic curves. It handles *evil twins:* multiple versions of the same algebraic curve with slightly different coefficients. The accuracy is within a factor of two of the limit imposed by the root solver. The number of calls to the crossing module, which take over 90% of the running time, is within a few percent of the minimum for any sweep arrangement algorithm.

#### 5.3.1. *Generating Arrangements*

We validate our algorithm on inputs with many degeneracies and near degeneracies, which are the hardest cases for any algorithm. To create such an input, generate random sets of points in the unit box and fit polynomials to them until there are 10 segments. Calculate the arrangement of these segments. Select random sets of vertices from this arrangement and fit polynomials to them until there are 90 more segments. Calculate the arrangement of the $10 + 90 = 100$ segments. Select random sets of vertices from these arrangements and fit polynomials to them until there are 900 more segments. Calculate the arrangement of the $10+90+900 = 1000$ segments. This is the *good* arrangement. To generate an *evil* arrangement, generate only 400 instead of 900 segments in this manner. Generate the remaining 500 segments as evil twins of the 100 segments in the second arrangement. To do so, select at random one of the polynomials of the first 100 segments. From the second arrangement, select a random set of vertices that lie on one of the segments of that polynomial. Fit an evil twin polynomial to those vertices. In exact arithmetic, the evil twin is identical to the original polynomial. However, there is rounding error in the calculation of the vertices and in the fitting to these vertices. Therefore, the evil twin is nearly identical to the original. Add segments from evil twins until there are 500 evil twin segments for a total of $10 + 90 + 400 + 500 = 1000$ segments.

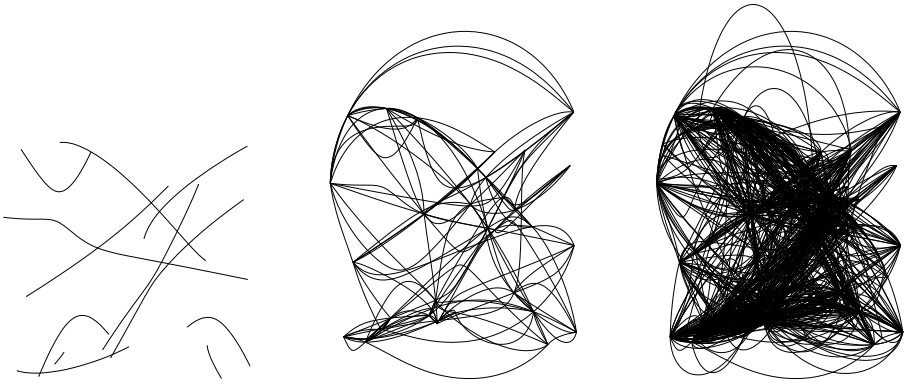

Fig. 7. Input arrangements of degree 3.

Fitting is performed as follows. A polynomial $F(x, y)$ of degree $d$ is fitted to $D-1$ points, since it has $D = (d+1)(d+2)/2$ coefficients of which $D-1$ are independent. The constant term is set to 1 and the other $D-1$ coefficients are computed with a standard linear solver. To avoid stressing the solver, we reject inputs where the minimum distance between two points is less than 0.01. We subdivide the algebraic curve $F(x, y) = 0$ into monotone segments then generate the maximal subsegments that end at fit points or at turning points. We include segments that start at turning points to obtain a realization error estimate at segment tails with very large slopes, a situation where the error bound does not apply and we expect relatively large errors. If we detect a singularity, we reject $F$. Our algorithm handles singularities, but we validate on degeneracies only.

Figure 7 shows test arrangements with 10, 100, and 1000 segments of degree 3. The following is the histogram of incidences of algebraic curves on vertices from 1 to 50: 30 91044 935 455 262 140 95 58 55 45 50 34 24 25 24 20 17 15 11 12 10 7 9 9 5 5 7 1 3 3 0 6 1 3 1 0 1 0 5 0 1 0 1 2 0 0 1 0 0 1. For example, 30 vertices are incident on one algebraic curve. These include turning points, which are incident on two segments but only one curve. Most of the 91044 vertices incident on two curves are newly generated intersections. The evil twin arrangement (not shown) is even more degenerate: 80 41595 6029 2595 1257 668 425 290 192 150 108 110 68 44 44 29 28 28 24 17 25 22 11 17 20 13 11 16 11 11 8 7 4 4 6 5 4 2 0 4 4 2 0 2 0 1 0 1 1 0 0 0 0 1.

5.3.2. *Results*

We ran ten experiments on arrangements of 1000 curves of algebraic degree 3–10. Tables 1 and 2 show the results for good and evil arrangements.

We estimate the realization error just to the left of each vertex. The rationale is that the error should be maximal just before incorrectly ordered segments swap, which occurs solely at sweep events. We examine the segments incident on the

Table 1. Statistics for good arrangements. Columns divide into input, output, and telomere sections. Input: $d$ is the algebraic degree, $N_{\text{med}}$ and $N_{\max}$ are the median and maximum number of crossing module crossings, $k_{\text{med}}$ and $k_{\max}$ are the median and maximum number of inconsistencies, and $e_{\text{med}}$ and $e_{\min}$ are the median and minimum crossing list accuracy in bits. Output: $t$ is average running time in seconds, $t_a$ is the percentage of $t$ spent outside the crossing module, $c_k$ is the maximum number of output crossings due to inconsistencies, and $e_r$ is the maximum ratio of realization error to crossing list accuracy. Telomere data in percent: $\Delta t$ is the maximum increase in $t$, $\Delta c_k$ is the minimum decrease in $c_k$, and $\Delta_- e_r$ and $\Delta_+ e_r$ are the maximum decrease and increase in $e_r$. Experiments were run on 2.2GHz AMD Athlon(tm) 64-bit processor with a 1M cache running Linux 2.6 and LAPACK 3.0.

| $d$ | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|
| $N_{\text{med}}$ | 104862 | 99787 | 91135 | 77704 | 63483 | 48498 | 39779 | 33971 |
| $N_{\max}$ | 109218 | 110336 | 109687 | 97785 | 85436 | 60866 | 53025 | 35123 |
| $k_{\text{med}}$ | 16354 | 37681 | 86897 | 46197 | 33346 | 18490 | 9998 | 5453 |
| $k_{\max}$ | 24700 | 74159 | 135842 | 70454 | 79474 | 19410 | 22515 | 7585 |
| $e_{\text{med}}$ | 42 | 41 | 40 | 39 | 38 | 39 | 38 | 36 |
| $e_{\min}$ | 39 | 33 | 39 | 38 | 34 | 37 | 16 | 11 |
| $t$ | 47 | 65 | 80 | 91 | 106 | 121 | 138 | 157 |
| $t_a$ | 1.2 | 0.9 | 0.8 | 0.5 | 0.4 | 0.2 | 0.2 | 0.1 |
| $c_k$ | 718 | 904 | 1706 | 702 | 726 | 403 | 343 | 175 |
| $e_r$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1.3 |
| $\Delta t$ | +12 | +12 | +12 | +11 | +9 | +9 | +8 | +5 |
| $\Delta c_k$ | −97 | −98 | −98 | −98 | −97 | −96 | −94 | −89 |
| $\Delta_- e_r$ | −58 | −12 | −0 | −5 | −19 | −0 | −6 | −0 |
| $\Delta_+ e_r$ | +9 | +27 | +0 | +0 | +6 | +0 | +0 | +73 |

vertex and the two segments directly above and below them in the sweep list. We estimate the error of each segment relative to every segment in the sweep list. For $f, g$ with $f <'_x g$, the estimate is the minimum $\delta$ for which $f_{-\delta}(x) = g_{+\delta}(x)$, which is zero when $f(x) \le g(x)$ and is positive otherwise. The maximum over all pairs and vertices is our estimate of the $\delta$ realizability of the output crossing lists. It is at most $1.3\epsilon$ for the good arrangements and at most $1.5\epsilon$ for the evil ones. The medians are of the ten maximum values.

We estimate the crossing list accuracy as the maximum $\delta$ over the adjacent pairs. These pairs are in crossing list order by local consistency, so $\delta$ is the minimum offset that realizes the crossing list order. This is a conservative estimate of $\epsilon$ because it does not exclude $\epsilon$ intervals around crossings. The median accuracy slowly drops from 42 bits to 34 bits as the degree rises from 3 to 10. The good arrangements had bad accuracy on one degree 9 and one degree 10 test. The evil arrangements had bad accuracy one one degree 7 and one degree 10 test. In each case, one root is approximated poorly due to an ill conditioned eigenvalue problem.

Crossing time dominates total running time. At most 1.3% of the time is spent outside the crossing module, and this percentage drops to 0.1% for degree 10. The

Table 2. Statistics for evil arrangements.

| $d$ | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|
| $N_{\text{med}}$ | 90359 | 101455 | 80612 | 81953 | 67704 | 49626 | 46178 | 34054 |
| $N_{\max}$ | 96873 | 112373 | 116589 | 95050 | 77767 | 56596 | 59651 | 35072 |
| $k_{\text{med}}$ | 60773 | 93111 | 111716 | 72183 | 55838 | 29102 | 21413 | 12205 |
| $k_{\max}$ | 72504 | 136861 | 245111 | 91522 | 95639 | 36098 | 29056 | 16388 |
| $e_{\text{med}}$ | 43 | 42 | 40 | 40 | 39 | 39 | 37 | 34 |
| $e_{\min}$ | 41 | 38 | 32 | 37 | 14 | 33 | 30 | 11 |
| $t$ | 47 | 75 | 88 | 102 | 120 | 137 | 163 | 174 |
| $t_a$ | 1.3 | 1.1 | 1 | 0.5 | 0.4 | 0.2 | 0.1 | 0.1 |
| $c_k$ | 1617 | 1490 | 3156 | 1159 | 1027 | 755 | 525 | 438 |
| $e_r$ | 1.5 | 1 | 1 | 1 | 1 | 1.2 | 1.3 | 1.4 |
| $\Delta t$ | +14 | +13 | +11 | +10 | +8 | +7 | +6 | +4 |
| $\Delta c_k$ | −98 | −98 | −97 | −96 | −98 | −98 | −95 | −91 |
| $\Delta_- e_r$ | −7 | −13 | −0 | −0 | −0 | −0 | −4 | −0 |
| $\Delta_+ e_r$ | +16 | +0 | +0 | +9 | +58 | +12 | +0 | +187 |

number of crossings declines, presumably because higher degree curves leave the bounding box faster. The number of inconsistencies, $k$, is bounded by $2N$ for good arrangements and $3N$ for evil. The inconsistencies increase the number of crossings by at most 2% for good arrangements and by at most 4% for evil arrangements.

The experiments were run with a telomere length of $2^{12}$ rounding units. We chose this length to make it roughly comparable to the maximum $\epsilon$ we had seen. Telomeres increased the running time at most 14% and usually much less. They decreased the number of extra crossings by 90–98%. They had little effect on accuracy, as measured by $e_r$, which is close to the optimal value of 1 in most cases and is at most 3.

## 6. Conclusions

We have presented a robust arrangement algorithm for plane curves based on an approximate crossing module. Its performance is analyzed in terms of the number of combinatorial inconsistencies, $k$, that occur due to the approximation error, $\epsilon$. The running time and output size match those of the standard sweep algorithm with exact, unit-cost algebraic computation, plus a $kn \log n$ term with $n$ the input size. The output accuracy is $\epsilon + kn\epsilon$. We have presented experimental evidence that inconsistencies increase the output size by at most 4% even on highly degenerate inputs. Hence, the performance matches a standard sweep with floating point computation.

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 the curve degree increases, the floating

point resolution of the triangle decreases until the vertex order becomes essentially random. 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 cross in every three-part configuration space. In mechanical design, redundancy and symmetry can generate crossing triples of contact curves. Even so, the inconsistencies are confined to small regions. We conjecture that if the $k$ inconsistencies are pairwise $\epsilon$ separated, where $\epsilon$ is the crossing module accuracy, then the running time is linear in $k$ and the output accuracy is $\epsilon$.

Inconsistency sensitive analysis is a new computational geometry paradigm that we plan to explore further. Our next goal is to construct and manipulate the configuration spaces of rigid planar parts, which are key to algorithmic 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.[23,24,25] 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.[26,27] This suggests that exact geometric cascading is as hard as exact scientific computing, which is untenable. The shape modeling results suggest that our approach can make cascading practical by replacing this exponential factor with a small constant.

## Acknowledgments

Research supported by NSF grants IIS-0082339, CCF-0306214, and CCF-0304955. A preliminary version of this paper appears in the 2006 Symposium on Computational geometry.

## References

1. Chee Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of discrete and computational geometry*. CRC Press, Boca Raton, FL, second edition, 2004.
2. Dan Halperin. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of discrete and computational geometry*. CRC Press, Boca Raton, FL, second edition, 2004.
3. 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.

4. K. Melhorn and S. Näher. *The LEDA platform for combinatorial and geometric computing*. Cambridge University Press, 1999.
5. Eyal Flato, Dan Halperin, Iddo Hanniel, Oren Nechushtan, and Eti Ezra. The design and implementation of planar maps in CGAL. *The ACM Journal of Experimental Algorithmics*, 5(13), 2000.
6. Ron Wein. High-level filtering for arrangements of conic arcs. In Rolf H. Möhring and Rajeev Raman, editors, *Algorithms - Esa 2002, 10th Annual European Symposium*, volume 2461 of *Lecture Notes in Computer Science*, pages 884–895, Rome, Italy, 2002. Springer.
7. Eric Berberich, Arno Eigenwillig, Michael Hemmer, Susan Hert, Kurt Mehlhorn, and Elmar Schömer. A computational basis for conic arcs and boolean operations on conic polygons. In Rolf H. Möhring and Rajeev Raman, editors, *Algorithms - Esa 2002, 10th Annual European Symposium*, volume 2461 of *Lecture Notes in Computer Science*, pages 174–186, Rome, Italy, 2002. Springer.
8. Arno Eigenwillig, Lutz Kettner, Elmar Schömer, and Nicola Wolpert. Complete, exact, and efficient computations with cubic curves. In *Proceedings of the 20th ACM Symposium on Computational Geometry*, 2004.
9. Nicola Geismann, Michael Hemmer, and Elmar Schömer. Computing a 3-dimensional cell in an arrangement of quadrics: exactly and actually! In *Proceedings of the 17th ACM Symposium on Computational Geometry*, pages 264–273, 2001.
10. Nicola Wolpert. Jacobi curves: computing the exact topology of arrangements of non-singular algebraic curves. In *Proceedings of the 11th ACM Symposium on Algorithms*, pages 532–543, 2003.
11. Bernard Mourrain, Jean-Pierre Técourt, and Monique Teillaud. Sweeping an arrangement of quadrics in 3d. In *Proceeding of the 19th European Workshop on Computational Geometry*, pages 31–34, 2003.
12. Elmar Schömer and Nicola Wolpert. An exact and efficient approach for computing a cell in an arrangement of quadrics. *Computational Geometry: Theory and Application*, 33:65–97, 2006.
13. John Keyser, Tim Culver, Mark Foskey, Shankar Krishnan, and Dinesh Manocha. ESOLID—a system for exact boundary evaluation. *Computer-Aided Design*, 36(2):175–193, 2004.
14. Dan Halperin and Eran Leiserowitz. Controlled perturbation for arrangements of circles. In *Proceedings of the 19th ACM Symposium on Computational Geometry*, pages 264–273, San Diego, 2002.
15. S. Fortune. Robustness issues in geometric algorithms. In M. Lin and D. Manocha, editors, *Applied Computational Geometry*, volume 1148 of *Lecture Notes in Computer Science*, pages 9–14, New York, 1996. Springer.
16. David Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms*. Springer, 1996.
17. William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, England, 1990.
18. Steven Fortune. An iterated eigenvalue algorithm for approximating roots of univariate polynomials. *Journal of Symbolic Computation*, 33(5):627–646, May 2002.
19. Alexander P. Morgan. *Solving Polynomial Systems Using Continuation for Scientific and Engineering Problems*. Prentice-Hall, Englewood Cliffs, NJ, 1987.
20. H. Stetter and W. Auzinger. An elimination algorithm for the computation of all zeros of a system of multivariate polynomial equations. In *Conference in Numerical Analysis*, volume 86 of *International Series on Numerical Mathematics*, pages 11–30. Birkhauser, 1988.

21. L. Buśe, H. Khalil, and B. Mourrain. Resultant-based method for plane curves intersection problems. In *Proceedings of the Conference on Computer Algebra in Scientific Computing, Volume 3718 of LNCS*, pages 75–92. Springer, 2005.
22. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.
23. V.J. Milenkovic and K. Daniels. Translational polygon containment and minimal enclosure using mathematical programming. *International Transactions in Operational Research*, 6:525–554, 1999.
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. Densest translational lattice packing of non-convex polygons. *Computational Geometry: Theory and Applications*, 22:205–222, 2002.
26. Behnke, Bachmann, Fladt, and Kunle. *Fundamentals of Mathematics, Volume II: Geometry*. MIT Press, Cambridge, MA, 1974.
27. Victor J. Milenkovic. Shortest path geometric rounding. *Algorithmica*, 27(1):57–86, 2000.