A Computational Basis for Conic Arcs and Boolean Operations on Conic Polygons*

Eric Berberich Arno Eigenwillig Michael Hemmer
Susan Hert Kurt Mehlhorn Elmar Schömer
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany

April 24, 2002

Abstract

We give an exact geometry kernel for conic arcs, algorithms for exact computation with low-degree algebraic numbers, and an algorithm for computing the arrangement of conic arcs that immediately leads to a realization of regularized boolean operations on conic polygons. A conic polygon, or polygon for short, is anything that can be obtained from linear or conic halfspaces (= the set of points where a linear or quadratic function is non-negative) by regularized boolean operations. The algorithm and its implementation are complete (it can handle all cases), exact (it gives the mathematically correct result), and efficient (it can handle inputs with several hundred primitives).

1 Introduction

We give an exact geometry kernel for conic arcs, algorithms for exact computation with low-degree algebraic numbers, and an sweep-line algorithm for computing arrangements of curved arcs that immediately leads to a realization of regularized boolean operations on conic polygons. A conic polygon, or polygon for short, is anything that can be obtained from linear or conic halfspaces (= the set of points where a linear or quadratic function is non-negative) by regularized boolean operations. A regularized boolean operation is a standard boolean operation (union, intersection, complement) followed by regularization. Regularization replaces a set by the closure of its interior and eliminates dangling low-dimensional features (Figure 1).

Our algorithm and implementation are *complete* and *exact*. They are complete in the sense that they can handle all inputs including arbitrary degeneracies. They are exact in that they always deliver the mathematically correct result. Complete and exact implementations for the linear case are available, *e.g.*, in the generalized polygon class [MN99, Section 10,8] of LEDA and in the planar map class [FHHN99] of CGAL. However, existing implementations for conic polygons are either incomplete or inexact, except perhaps for the very recent work by Wein [Wei].

There are three main parts to our work: (1) a sweep-line algorithm for computing arrangements of curved arcs, (2) predicates and functions for conic arcs, and (3) algorithms for the exact computation with low-degree algebraic numbers. For part (1), the sweep-line algorithm extends the Bentley-Ottmann sweep-line algorithm for segments [BO79]. The handling of many curves passing through the same point is considerably more involved than in the straight-line case. For (2), we give algorithms for basic predicates and functions on conics and use them to realize the functionality required in the sweep algorithm. For (3), we have integrated the representation of algebraic numbers as roots of polynomials and the representation as explicit expressions involving square roots.

*Partially supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces).

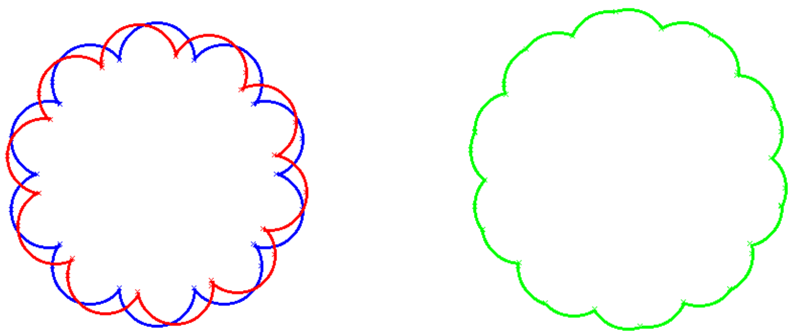

Figure 1: We compute the union of two curved polygons (left panel). The first input polygon is created from a regular 10-gon by replacing each straight edge with a half-circle with that edge as diameter. The second polygon is created from the first by rotating it around the origin by $\alpha = \pi/20$ radians. On the right is the union of these two polygons. We can compute the correct union of such rotated n-gons for any n and any α. For $n = 1000$ and $\alpha = \pi/2000$ the time required to compute the correct union with 2702 edges is less than 20 minutes on an 846 MHz Pentium III processor.

Our implementation consists of a basic layer providing polynomials, roots of polynomials, low-degree algebraic numbers, conics, and predicates and functions on conics and conic arcs and an algorithmic layer that provides arrangements of conics, the sweep-line algorithm for curves, and boolean operations on conic polygons and on polygons with circular and straight-line arcs. We have tested our implementations on inputs of various sizes and with various degeneracies; see for example Figure 2. We provide evidence of the efficiency of the approach presented here by comparing the results to the implementations in LEDA for polygons with line-segment edges. Our implementation can handle scenes with several hundred conic segments.

The rest of the paper is organized as follows. We first summarize related work (Section 2) and then review the Bentley-Ottmann sweep-line algorithm. We extend it to curved arcs and derive the required set of predicates and functions (Section 3). In Section 4 we discuss conics, functions and predicates on conics, and computation with low degree algebraic numbers. In Section 5 we give more details of our implementations and provide the results of our experiments. Section 6 offers some conclusions.

2 Related Work

The work of three communities is relevant for our work: computational geometry, solid modeling, and computer algebra.

The solid modeling community has always dealt with curved objects, and CAD systems dealing with curved objects in two and three dimensions have been available since the 60's. None of these systems is complete or exact, not even for straight-line objects. The question of complete and exact implementations has only been addressed recently. MAPC [KCMK98] provides a set of classes for manipulating algebraically defined points and curves in the plane, which includes an implementation of the naïve $O(n^2)$ algorithm for computing an arrangement of n curves in the plane. The algorithms are not complete; they handle some but not all degeneracies. Also the use of Sturm sequences to handle degenerate cases, such as tangential curves or degenerate segments, results in unnecessarily slow computations in these cases. ESOLID [KCF$^+$] performs accurate boundary evaluation of low-degree curved solids. It is explicitly stated that degeneracies are not treated.

Surprisingly little work in computational geometry deals with curved objects; some examples are [SV87, DSV88, DS90]. The sweep-line algorithm of Bentley-Ottmann [BO79] is known to work for x-monotone curves, at least in the absence of degeneracies. Degeneracies have only been discussed for straight-line segments. Several papers [BP97, BS99, Cha00] have looked into the question of using restricted predicates to report or compute segment intersections, the rationale being that lower-degree predicates are simpler to evaluate. All papers have to exclude at least some degenerate cases.

The exact and efficient implementation of the predicates required in our algorithms is non-trivial, since they involve algebraic numbers. One of the predicates used in our algorithms, the lexicographical comparison of vertices in an arrangement of circles and lines, has been considered by Devillers *et al.* [DFMT00]. A very efficient realization is given. The more complex predicates also needed in our algorithms are not discussed in their work and it is not clear whether the technique generalizes.

CGAL's planar map class also supports the computation of arrangements of circular arcs and line segments. Very recently, the implementation was extended to conic arcs by Wein [Wei]. The implementation is in some respects similar to ours. However, the sweep method is not yet available and the the computation of boolean operations on polygons is implemented only indirectly.

The papers [GHS01, DLLP] show how to compute arrangements of quadrics in three-space. The algorithms are complete, but (as of now) can handle only a small number of quadrics.

Root isolation of univariate polynomials, resultant computation, and exact treatment of algebraic numbers are well studied problems in computer algebra. We use the standard techniques.

3 Conic Polygons and the Sweep-line Algorithm

The Bentley-Ottmann Sweep-line Algorithm: The Bentley-Ottmann sweep-line algorithm for computing an arrangement of segments in the plane [BO79] was originally formulated for sets of segments, no three of which pass through a common point and no two of which overlap each other. A vertical line is swept across the plane and the ordered sequence of intersections between the sweep line and the segments is maintained (= Y-structure). The status of the sweep line changes when a segment starts, when a segment ends, and when two segments cross. Bentley and Ottmann observed that the algorithm can actually handle any set of x-monotone curves. Of course, when two such curves meet they may either cross or touch, which requires a minor modification to the algorithm. The events are maintained in a priority queue, referred to as X-structure.

It is was later observed, see for example [dBKOS97, MN99], that in the case of straight-line segments the algorithm can also handle arbitrary degeneracies. A number of small extensions are required. For example, when the algorithm sweeps across a point in which several segments meet, the y-order of the segments meeting at this point is reversed.

We next argue that the algorithm can also handle degenerate situations in the case of curves. The main problem is sweeping across a point where many curves meet and we restrict our attention to this problem here.

Consider a point p and assume that arcs C_1 to C_k pass through p. We assume that the curves are numbered according to their y-order just left of p. Let s_i be the multiplicity of intersection of the curves C_i and C_{i+1} at p; see [Bix98, Chapters I and IV] for a formal definition. Intuitively, the multiplicity is one if the curves meet at p and have different slopes, the multiplicity is two if the curves have identical tangent but different radii of curvature, the multiplicity is three if the curves have same tangent and identical radii of curvature but different $\ldots$. For example, the multiplicity of intersection between $y = x^i, i > 0$ and $y = 0$ at the origin is equal to i. Two curves meeting at p cross at p if the multiplicity of the intersection is odd, and they touch, but do not cross, if the multiplicity is even. The multiplicity of intersection between C_i and C_j for $i < j$ is $\min\{ s_i, \ldots, s_{j-1} \}$ because the multiplicity of intersection is the number of identical initial coefficients in the local Taylor series expansion. For distinct conics, the multiplicity of intersection at any point is at most 4. For example, the multiplicity of intersection at the origin between $y(1-x) = x^2$ and $y = x^2 + y^2$ is three.

The following algorithm determines the y-order of our curves C_1 to C_k just to the right of p in time $O(k)$. Make four passes over the sequence of curves passing through p. In the j-th pass, $4 \geq j \geq 1$, form maximal subsequences of curves, where two curves belong to the same subsequence if they are not separated by a multiplicity less than j, and reverse the order of each subsequence.

Lemma 1 *The algorithm above correctly computes the y-order of the segment passing through a common point p immediately to the right of p from the order immediately to the left of p.*

Proof: Consider two arbitrary curves C_h and C_i with $h < i$. Their y-order right of p differs from their y-order left of p iff $s = \min\{ s_h, \ldots, s_{i-1} \}$ is odd. Next observe that s is also exactly the number of times C_h and C_i belong to the same subsequence, i.e., the number of times their order is reversed. We conclude the order of C_i and C_h is reversed iff C_i and C_h cross at p. ∎

Of course, the algorithm just outlined will also work if the maximal multiplicity M of intersection is arbitrary. However, its running time will be M times k, the number of curves passing through the point. Lutz Kettner (personal communication) has shown that the permutation can always be computed in time linear in the length of the subsequence.

For our implementation of the sweep-line algorithm, we took the LEDA implementation [MN99, Section 10.7] as our starting point.

Conic Polygons: Regularized boolean operations on straight-line or conic polygons can be built on top of the sweep-line algorithm for segment intersection, see [MN99, Section 10.8]. The corresponding data structure in LEDA is called generalized polygons. We reused it with only one small change. When curved edges are used, a polygonal chain with only two edges is possible. We also made the implementation more general by parameterizing it with the type of the polygon used to represent the boundaries (Section 5).

Required Predicates and Functions: We are now ready to summarize the predicates and functions that must be defined for the points and segments used in the sweep-line algorithm and in the computation of generalized polygons.

compare_xy(p, q) – compares points p and q lexicographically. This predicate is used to maintain the order of the events in the X-structure.

seg.y_order(p) – determines if a point p in the x range of segment *seg* is vertically above, below, or on *seg*. This predicate is used to insert starting segments into the Y-structure.

seg.compare_right_of_common_point(seg2, p) – compares *seg* and *seg*2 to determine their y-ordering just right of their common point p. This predicate is also used to insert starting segments into the Y-structure. The order of segments starting at the same point is determined by this predicate.

seg.common_point_multiplicity(seg2, p) – returns the multiplicity of intersection of *seg* and *seg*2 at their common point p. This predicate is used to handle curves passing through a common point as described above.

seg.has_on(p) – determines if p lies on *seg*. This predicate is not needed in the sweep line algorithm, but in the algorithm for boolean operations. It is used to determine the containment of one boundary inside another.

intersect(seg1, seg2, result) – determines if *seg*1 and *seg*2 intersect or not and, if so, inserts their intersection points in lexicographical order into *result*

4 Conics and Computations with Conics

We discuss predicates and constructions on conics and their algorithmic realization in this section.

Every conic is defined as the zero-set in $\mathbb{R}^2$ of a quadratic implicit equation P in the variables (x, y) i.e.,

$$\alpha_1 x^2 + \alpha_2 y^2 + 2\alpha_3 xy + 2\alpha_4 x + 2\alpha_5 y + \alpha_6 = 0$$

with $\neg(\alpha_1 = \alpha_2 = \alpha_3 = 0)$. A conic is either empty, a single point, a pair of (intersecting, parallel, or identical) lines, or an ellipse, a parabola, or a hyperbola. We restrict attention to the non-degenerate conics in this paper as lines bring nothing new algorithmically. For every value x there are at most two values of y satisfying the equation. We can obtain an x-monotone parameterization by rewriting the equation as a quadratic equation for y and solving for y. We obtain

$$Q = a(x)y^2 + b(x)y + c(x) = 0,$$

where

$$a(x) = \alpha_2, \qquad b(x) = 2\alpha_3 x + 2\alpha_5, \qquad c(x) = \alpha_1 x^2 + 2\alpha_4 x + \alpha_6.$$

Solving for y yields

$$y = \begin{cases} \frac{-b(x)\pm\sqrt{b(x)^2-4a(x)c(x)}}{2a(x)} & \text{if } \alpha_2 \neq 0 \\ \frac{-c(x)}{b(x)} & \text{if } \alpha_2 = 0 \text{ and } b(x) \neq 0 \end{cases}$$

Degenerate conics are easily recognized. For example, if $\alpha_2 \neq 0$ and $h(x) = b(x)^2 - 4a(x)c(x)$ has a double root, the conic degenerates to a pair of intersecting lines. The same holds true if $\alpha_2 = 0$, and $b(x)$ and $c(x)$ have a common factor. One can also easily read off the type of the conic. For example, if h has two distinct roots and its leading coefficient is negative, we have an ellipse.

A *conic arc* is an x-monotone curve that corresponds to one choice of sign in the equation above. A conic decomposes into either one or two conic arcs. In the latter case we refer to the two arcs as the lower and the upper arc. For a conic C we use C_0 and (if it exists) C_1 to denote the arcs of C. We view the arcs as functions of x, i.e., $C_1(x)$ is the y-value of the upper arc of C at x.

Intersection of Two Conics: Consider two conics Q_1 and Q_2. Then there is a polynomial R in x of degree at most four such that the x-coordinates of the intersections of Q_1 and Q_2 are roots of R. R is called the *resultant* of Q_1 and Q_2, see [CLO97] for a discussion of resultants. A root x of R does not necessarily correspond to an intersection of Q_1 and Q_2 in $\mathbb{R}^2$; the corresponding y-values may have non-zero imaginary parts. We can also derive the resultant by elementary means. Write

$$Q_1 = a_1y^2 + b_1y + c_1 = 0 \qquad Q_2 = a_2y^2 + b_2y + c_2 = 0$$

and solve the first equation for y. Plug the result into the second equation and remove roots by squaring. For example, if $a_1 \neq 0$ and $a_2 \neq 0$, we obtain $y = (-b_1 \pm \sqrt{b_1^2 - 4a_1c_1})/(2a_1)$ from the first equation. Plugging this into Q_2 yields

$$\frac{2a_2b_1^2 - 4a_2a_1c_1 - 2b_2a_1b_1 + 4c_2a_1^2 \pm (-2a_2b_1 + 2b_2a_1)\sqrt{b_1^2 - 4a_1c_1}}{4a_1^2} = 0$$

and squaring finally gives

$$R = \frac{a_2b_1^2c_2 + a_2^2c_1^2 - a_2c_1b_2b_1 - 2a_2a_1c_1c_2 - b_2a_1b_1c_2 + c_2^2a_1^2 + b_2^2a_1c_1}{a_1^2} = 0$$

Low-Degree Algebraic Numbers: The x-coordinates (and similarly the y-coordinates) of intersections points are roots of polynomials of degree at most four. We call an algebraic number a *one-root-number* if it is of the form $\alpha + \beta\sqrt{\gamma}$ with $\alpha, \beta, \gamma \in \mathbb{Q}$. The following well-known lemma is useful.

Lemma 2 *A degree-four polynomial p either has four simple roots or all roots of p are one-root-numbers. The two cases are easily distinguished and the one-root-expressions can be determined in the latter case.*

Proof: Let $r = \gcd(p, p')$ be the greatest common divisor of p and its derivative. If $\deg r = 0$, p has only simple roots.

If the degree of r is one, p has two simple and one double root. Let $\xi \in \mathbb{Q}$ be the root of r. Then $p = q(x - \xi)^2$ and hence $q = p/(x - \xi)^2$ is a rational polynomial. Thus all roots of p are one-root-numbers.

If the degree of r is two, p either has two double roots or a simple root and a triple root. In either case the roots of p are precisely the roots of $q = p/r$ and hence are one-root-expressions.

If the degree of r is three, p has a four-fold root ξ. Let $p = x^4 + a_3x^3 + \dots$. Then $a_3 = -4\xi$. ∎

We represent algebraic numbers x in one of two ways. Either as one-root-expressions or as triples (P, l, r) where P is a polynomial with only simple roots, l and r are rational numbers, P has exactly one real root in the open interval (l, r) and $P(l) \neq 0 \neq P(r)$. Such an interval is called an *isolating interval* for the root. In our implementation, l and r have the additional property that their denominators are powers of two. We determine isolating intervals by means of Uspensky's algorithm [CA76, RZ01]. Isolating intervals are easily *refined* by considering the point $m = (l + r)/2$. If $P(m) = 0$, we have a one-root-expression for x. Otherwise, we replace the isolating interval by either (l,m) or (m,r) depending on the sign of $P(m)$.

One-root-numbers are represented as objects of the number type leda-real, cf. [BFM+01] or [MN99, Section 4.4]. Integers are leda-reals, and if x and y are leda-reals, so are $x \pm y$, $x * y$, x/y, and $\sqrt[k]{x}$ for arbitrary integer k. Leda-reals have exact comparison operators $\leq$, $<$ and $=$. In particular, if x is a leda-real and P is a polynomial with integer coefficients, we can determine the sign of $P(x)$.

We next describe how to compare two algebraic numbers x and y. If both of them are given as leda-real, we use the comparison operator of leda-real. If $x = (P, l, r)$ and y is a leda-real, we proceed as follows: If $y \leq l$ or $y \geq r$, the outcome of the comparison is clear. So assume $l < y < r$. If $P(y) = 0$, $x = y$. So assume that $P(y) \neq 0$. Then(!!!) $x \neq y$. We now refine the isolating interval for x as described in the preceding paragraph until $y \notin (l,r)$.

In order to compare two algebraic numbers $x = (P, l_x, r_x)$ and $y = (Q, l_y, r_y)$ we have to work slightly harder. If the isolating intervals are disjoint, we are done. Otherwise, let $I = (l,r)$ be the intersection of the isolating intervals. We have $x = y$ iff P and Q have a common root in I. We first refine the isolating intervals of x and y using the endpoints of I. Then it is either the case that both intervals are I or the intervals are disjoint. If they are disjoint, then we are done. Otherwise, we know that P and Q both have exactly one simple root in I. These roots are equal if $g = gcd(P,Q)$ has a root z in I, in which case z must be a simple root. Thus the degree of g can be used to decide quickly about equality or inequality in certain cases. For example, if $deg(g) = 0$, we know that x and y are not equal, and similarly, if $deg(g) = 4$, we know they are equal. Otherwise we use the fact that g has only simple roots. So $x = y = z$ iff $sign(g(l)) \neq sign(g(r))$. Furthermore, depending on the degree of g, a shorter representation for x and y may be obtained. Namely:

- if $deg(g) = 1$, then if x and y are equal then they are rational;
- if $deg(g) = 2$, we can represent them as one-root-numbers in any case;
- if $deg(g) = 3$, then if x and y are not equal, they are rational;

The *critical values* of a conic are the roots of $h(x)$ if $\alpha_2 \neq 0$ and is the root of $b(x)$ if $\alpha_2 = 0$. At the critical values the conic either has a vertical tangent or a pole. Critical values are one-root-numbers.

Intersection of Two Conics, Continued: Let R be the resultant of conics C and D and let x be a root of R. We show how to decide whether arcs C_i and D_j intersect at x.

If x is given as a leda-real, we simply compare $C_i(x)$ and $D_j(x)$ using leda-reals. So assume that x is not available as a leda-real. Then, by Lemma 2, x is a simple root of R and hence arcs C_i and D_j cross at x if they intersect at all. Let (R, l, r) be the representation of x. We refine the representation of x until the isolating interval contains no critical value of either C or D. Then C_i and D_j are defined on the entire interval $[l, r]$ and they intersect at most once in $[l, r]$. Thus C_i and D_j intersect at x iff the signs of $C_i(r) - D_j(r)$ and $C_i(l) - D_j(l)$ differ. We compute the signs using leda-reals.

Conic Points, Conic Segments, and Comparisons: We specify conic points by an x-coordinate (= an algebraic number) and a conic arc C_i. The point has coordinates $(x, C_i(x))$. If x is a leda-real, we can compute the y-coordinate as a leda-real. A conic segment is the part of a conic arc between two conic points.

The x-compare of two conic points is tantamount to the comparison of two algebraic numbers, which was discussed above. We turn to the xy-compare. Let (x_1, C_i) and (x_2, D_j) be two conic points. If $x_1 \neq x_2$, we are done. If $x_1 = x_2$ and we know a one-root-expression for x_1, we simply compare $C_i(x_1)$ and $D_j(x_2)$. So assume otherwise. We compute the resultant R of C and D and compare x_1 to the roots of R. Three cases arise: either x_1 is distinct from all roots of R, or we know a one-root-expression for x_1, or x_1 is equal to a root of R and is a simple root of R. In the latter two cases, we proceed as described in the paragraph on intersection of two conics. In the first case, we refine the isolating interval of x_1 until is contains no critical values of C and D and then use the fact that the y-order of the two arcs at x_1 is the same as the y-order at the right end of the isolating interval of x_1. The strategy just described also resolves the *has_on* predicate as well as *y_order*.

Multiplicity of Intersection: Let $p = (x, y)$ be an intersection point of arcs C_i and D_j. Then x is a zero of the resultant of C and D. Let m be the multiplicity of x as a zero of the resultant. If $m = 1$, the multiplicity of the intersection is 1. So assume that $m > 1$. Then we know a one-root-expression for x. We have to deal with two difficulties: the conics may have two arcs and hence there might be intersections between C_{1-i} and D_{1-j} at x and secondly there might be intersections with non-real y-value. We argue first that the latter case cannot arise since the y-values would have non-zero imaginary parts and hence would come in conjugate pairs. Thus the conic would have three points on the complex line through x parallel to the y-axis and hence the conics would be degenerate. However, we assumed our conics to be non-degenerate.

If one of the conics has only one arc, the multiplicity of the intersection is m. So assume $m > 1$ and that both conics have two arcs. Let d be the (currently unknown) multiplicity of the intersection at p and let e be the multiplicity of the intersection of C_{1-i} with D_{1-j} at x. Then $d + e = m$. We test $e \geq 1$ by testing the equality $C_{1-i}(x) = D_{1-j}(x)$ using leda-reals. If $e = 0$, we are done. So assume $e \geq 1$ and let q be the corresponding intersection point. We next test $d \geq 2$, by checking whether the normal vectors $(C_x(p), C_y(p))$ and $(D_x(p), D_y(p))$ are parallel. Here C_x and C_y are the partial derivatives of C. The check is again a computation using leda-reals since $p = (x, C_i(x))$. If $(d < 2)$, $d = 1$ and we are done. So assume $d \geq 2$. If $m = 3$, $d = 2$ and we are done. Assume otherwise, i.e., $m = 4$. We check whether $e \geq 2$. If $e = 1$, $d = m - 1 = 3$. If $e \geq 2$, $d = e = 2$.

Order Immediately to the Right of a Common Point: Consider arcs C_i and D_j containing $p = (x, y)$ and extending to the right. We show how to determine their y-order immediately to the right of p. Assume that we know a leda-real u such that $x < u$, arcs C_i and D_j are defined on $[x, u]$ and do not intersect in the interval $(x, u]$. Then the y-order just right of x is the same as the y-order at u; the latter can be computed using leda-reals. How can we obtain u? If x is a simple root of the resultant of C and D, we refine the isolating interval of x until it contains no critical values of C and D and take u as the right endpoint of the isolating interval. If x is a multiple root of the resultant, we have one-root-expressions for all roots and we simply take a rational point to the right of x and within the x-range of the two arcs. This can be, for example, the midpoint of x and the next larger root of the resultant if there is one.

5 Implementation and Empirical Results

We describe our implementations and report about experiments. All our implementations are in C++. They use components of LEDA, CGAL, and the standard template library.

Algebraic Numbers and Conics: We have implemented four classes: *alg_number*, *conic*, *arrangement*2, and *X_mono_conic_segment*. The first realizes algebraic numbers as discussed in Section 4. The main ingredients are polynomials, gcd of polynomials, resultant computations, Uspensky's algorithm for root isolation, and leda-reals. The main functionality is exact comparison between algebraic numbers. The class *conic* realizes conics and the predicates and constructions discussed in Section 4. The class *arrangement*2 uses the functionality of the two other classes to construct the arrangement of two conics. The class *X_mono_conic_segment* realizes an implementation of a segment type as required by the sweep-line algorithm and generalized polygon class.

The Naive Algorithm for Computing Arrangements of Conics: This algorithms takes a set of conics and computes the subdivision of the plane defined by them. It operates in two phases. It first computes the arrangement of any pair of conics and then merges the pairwise arrangements into a single arrangement. We give more details on the second step below. We have implemented the naive algorithm for two reasons: to have a test-bed for classes *alg_number*, *conic*, and *arrangement*2, and to have a reference implementation for the sweep-line algorithm.

The first phase of the naive algorithm produces a list of all intersections (x, C_i, D_j), where x is an algebraic number and C_i and D_j are conic arcs. We generate two conic points (x, C_i) and (x, D_j) for each intersection and sort them using *compare_xy*. Then it is easy to remove duplicates, and the vertices of the arrangement A are known. We create additional vertices for the points of vertical tangency and also create a dictionary that maps conic points to the vertices of A. Then we sort our conic points a second time. For the second sorting, the conic arc is the main key and the second key

	processing time per pair of conics	input precision
general position	30 ms	50 bits
degenerate position	48 ms	50 bits
perturbed degenerate position	48 ms	100 bits

Table 1: Average computation time required by the naive algorithm per pair of conics.

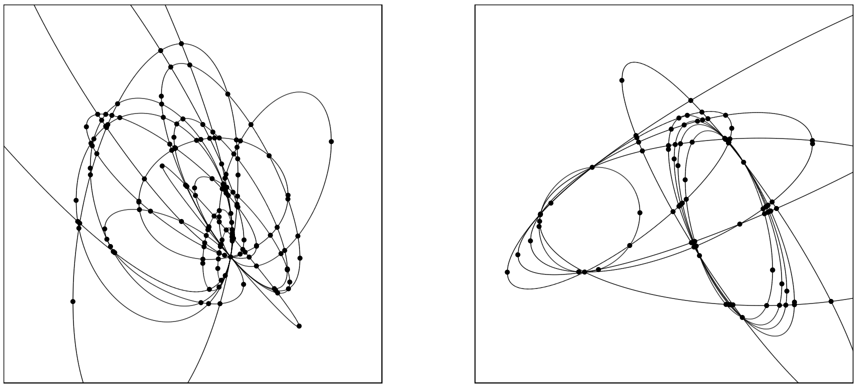

Figure 2: On the left a set of 15 ellipses all of which intersect at one point in the lower right quadrant of the picture with varying multiplicities. On the right is a set of 10 ellipses, also in highly degenerate positions.

is the x-coordinate. We obtain the sorted list of intersection points on each arc. Using the map from conic points to vertices of A, we create the edges of A. Finally, we determine the cyclic order of the edges incident to each vertex using the predicate *compare_right_of_common_point*. The final result is a planar map.

We checked the correctness of the implementation in two ways. For some examples, we manually inspected the arrangement (number of nodes and their position in the plane, number of edges and their cyclic order around vertices). For a large number of examples, we tested whether the arrangement produced by the algorithm is indeed a combinatorially embedded plane graph (leda-function *is_plane_map*). This test is not sufficient for the correctness of the implementation, but is a good heuristic test.

We ran the algorithm on three kinds of examples: (1) conics in general position, (2) conics with carefully designed degeneracies, and (3) conics with perturbed degeneracies (*i.e.*, almost degenerate points). Figure 2 shows two examples of the second kind. Both feature high-degree intersection points with various multiplicities of intersections between the curves at these points. The example on the left also contains intersection points with equal x-coordinate. Table 1 shows that, as one would expect, running time increases when degeneracies are present and also when the input precision is increased to represent nearly degenerate cases. It should be noted, however, that the increase is not unreasonable. With the naive algorithm, we are able to compute, for example an arrangement of 200 conics with approximately 55000 intersection points in around 28 minutes.

A Generic Implementation of the Sweep Algorithm: Our implementation of the algorithms, classes and predicates described in Sections 3 and 4 has been done in C++. It is based on the CGAL geometry kernel and the original implementations in LEDA of the sweep-line algorithm and the generalized polygons. To achieve an implementation that can easily accommodate different segment types, point types and predicate implementations, we have followed the generic programming paradigm [MS89, MS94] and used the concept of geometric traits classes introduced with CGAL [FGK+00]. The term “traits class” was originally introduced by Myers [Mye95]; we use it here to refer to a class that aggregates (geometric) types and operations. By supplying different traits classes, the same algorithm can be applied to different kinds of objects or using different predicate implementations.

Consider, for example, the function that implements the sweep-line algorithm. When dealing with curves of degree at most 2, the combinatorial part of the algorithm is exactly the same no matter what kind of segments are used or how they are represented. The predicates that are used by the algorithm are also exactly the same no matter what segment representation is used; only their realizations for different curve segment and curve point types changes. Thus to achieve a generic implementation we need only specify precisely which predicates and functions are required by the implementation (as was done in Section 3) and provide an easy means to communicate to the function our chosen implementations of the predicates, functions, and geometric objects. This is where the traits class comes into play.

To see more clearly how this works, consider the interface for the sweep-line function:

```
template <class Iterator, class Traits>
sweep_curves(Iterator first, Iterator beyond, Graph& G, Traits traits)
```

There are two types, `Iterator` and `Traits`, that have been specified as so-called template parameters. Such parameters are used to provide the flexibility inherent in generic programming as they allow one to supply any concrete types that satisfy certain requirements for the template parameters; one is not bound to a particular type. The requirements of an `Iterator` essentially make it behave like a pointer. For this function, the two iterators `first` and `beyond` provide access to the input curve segments. For readers not familiar with the concept of iterators, you can think of a simple pointer into a linked list, with `first` pointing to the first item in this list and `beyond` pointing to a sentinel value past the end of the list (or having the value `NULL`). Thus, given these two values, and a means of incrementing the pointer to point to the next item in the list, one can easily move through all items in the list. By using iterators (a feature of the Standard Template Library [Aus98]), we free ourselves from the need to store our segments in a particular type of container.

The requirements of the template parameter `Traits` encapsulate the geometric machinery required by the sweep-line algorithm. In particular, for this function, the requirements are that this class provides a point type and a segment type that support the predicates described in Section 3 as well as a means to access the point comparison predicate and intersection functions. Our implementation supplies several instances of such a traits class, to enable us, for example, to sweep line segments alone, circular arcs alone, a combined curve segment type that can be either of these, or a general conic segment.

We use traits classes throughout our implementation to maximize its flexibility and applicability in different settings. In particular, we use a traits class:

- as a parameter of the *sweep_curves* function to specify the types of the segments and points and the predicates used there;
- as a parameter of the class *Gen_polygon_2* to specify, in addition to the segment and point types used in the boundary representations, the polygon type that employs these types to represent a boundary.

Furthermore, we have point representations parameterized by the kernels in which the predicates operating on these points are defined and segment representations parameterized by the type of point representing the endpoints. Such a generic implementation made light work of producing the empirical results presented below that compare different segment types, different predicate implementations and different underlying kernels.

Sweeping Circular Arcs and Straight Line Segments: When only circular arcs and line segments are used, the implementation of the predicates and functions become easier since the coordinates of all intersection points are one-root-numbers. This means in particular that when the circles and lines supporting the segments are specified through rational points, much of the computation can be carried out using rational numbers. Only when it is time to compute the coordinates of the points using the *sqrt* function will the leda-real number type be used. This is generally a big win since it implies that the real number type representations remain relatively simple and thus comparisons and computations done involving these numbers are faster than otherwise. Furthermore, we are able to exploit the fact that the intersection point between any two rational circles can be described via the intersection of a rational line and a rational circle [DFMT00] in our implementation of the *compare_xy* predicate for points that lie on circular arcs or line segments. That is, before comparing the actual coordinates of the points p and q, we first determine if points p and q were constructed in the same way. If so, we

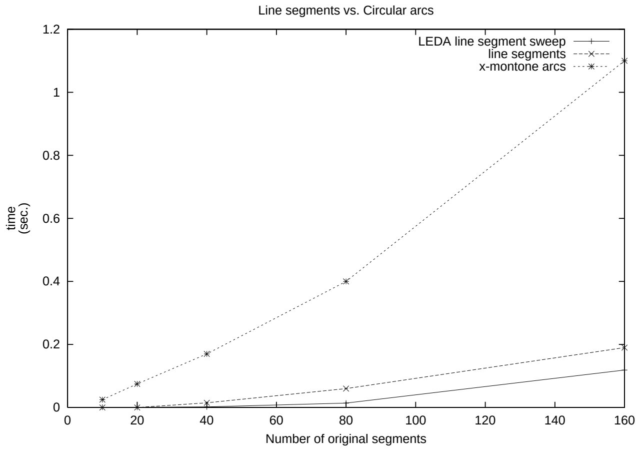

Figure 3: A comparison of running times for the optimized version of the LEDA line segment sweep algorithm, our generic implementation of the sweep-line algorithm using line segments and the same implementation using circular arcs. Running times were recorded on an 846 MHz Pentium III processor.

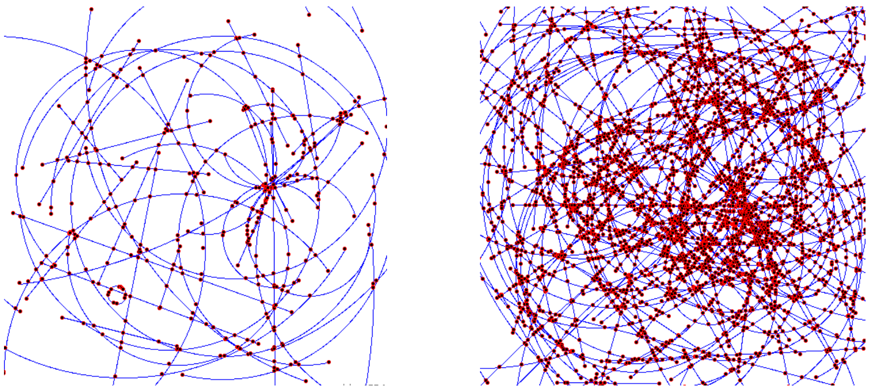

Figure 4: On the left the planar graph that results from the sweep of a set of 50 circular arcs and line segments with many degeneracies. The graph was produced in less than 1 second on an 846 MHz Pentium III processor. On the right is an example with 241 circular arcs and line segments, which required approximately 2 seconds to compute.

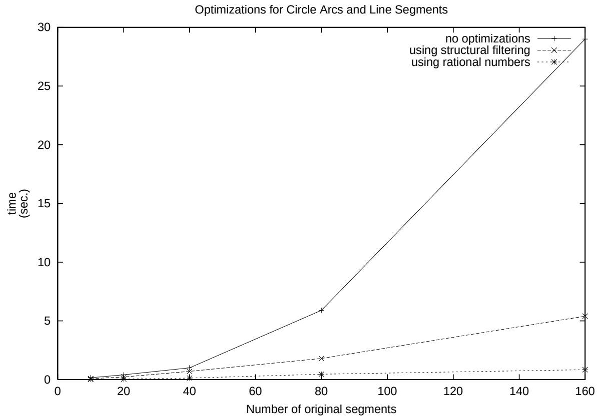

Figure 5: A comparison of running times using various optimizations for circular arcs and line segments.

know their coordinates are equal and we are done. Otherwise, we compare the coordinate values as usual. We call this technique *structural filtering*. Notice that, in contrast to usual numeric filters used in the exact computation paradigm, which filter out the easy cases that can be dealt with quickly by imprecise number types (*i.e.*, when the points' coordinates are vastly different from each other), this filtering technique works by filtering out the most difficult case (when the points' coordinates are identical) and thus assures that the cases left are usually relatively easy.

The graph in figure 5 illustrates the advantages of using these optimizations.

Sweeping Conic Segments: When constructing arrangement of general conic segments such as the ones shown in Figure 2, running times are naturally higher since the number types and predicates involved are more complicated. Examples similar to the ones shown in Figure 2 with 30 and 60 conic segments require, respecitvely, 19 seconds and 49 seconds using our current implementation. Table 1 indicates that, as one would expect, running time degrades when degeneracies are introduced. However, with the application of appropriate filtering techniques in our predicate implementations we are confident that the running times for conic segments will come in line with those for circular arcs.

6 Conclusions

We described an exact kernel for conic arcs, algorithms for dealing with low-degree algebraic numbers, a sweep-line algorithm for curved segments, and algorithms for boolean operations on conic polygons. Our algorithms and their implementations are complete, exact, and efficient. We feel that it was crucial for our work that we had all three goals in mind right from the beginning of our work.

References

[Aus98] Matthew H. Austern. *Generic Programming and the STL*. Addison-Wesley, 1998.

[BFM+01] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. In *ESA 2001*, Lecture Notes in Computer Science, pages 254–265, 2001. `http://www.mpi-sb.mpg.de/\~{}mehlhorn/ftp/ImprovedSepBounds.ps.gz`.

[Bix98] R. Bix. *Conics and Cubics: A Concrete Introduction to Algebraic Curves*. Springer Verlag, 1998.

[BO79] J.L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transaction on Computers C 28*, pages 643–647, 1979.

[BP97] Jean-Daniel Boissonnat and Franco P. Preparata. Robust plane sweep for intersecting segments. Research Report 3270, INRIA, Sophia Antipolis, September 1997.

[BS99] J.-D. Boissonnat and J. Snoeyink. Efficient algorithms for line and curve segment intersection using restricted predicates. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 370–379, 1999.

[CA76] G. E. Collins and A.-G. Akritas. Polynomial real root isolation using Descartes' rule of sign. In *SYMSAC*, pages 272–275, Portland, OR, 1976.

[Cha00] Timothy M. Chan. Reporting curve segment intersection using restricted predicates. *Computational Geometry*, 16(4):245–256, 2000.

[CLO97] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer-Verlag New York, Inc., 2nd edition, 1997.

[dBKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 1997.

[DFMT00] Olivier Devillers, Alexandra Fronville, Bernard Mourrain, and Monique Teillaud. Exact predicates for circle arcs arrangements. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, 2000.

[DLLP] L. Dupont, D. Lazard, S. Lazard, and S. Petitjean. A new algorithm for the robuts intersection of two general quadrics. submitted to Solid Modelling 2002.

[DS90] D. P. Dobkin and D. L. Souvaine. Computational geometry in a curved world. *Algorithmica*, 5:421–457, 1990.

[DSV88] D. P. Dobkin, D. L. Souvaine, and C. J. Van Wyk. Decomposition and intersection of simple splinegons. *Algorithmica*, 3:473–486, 1988.

[FGK+00] Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra, and Sven Schönherr. On the design of CGAL, the computational geometry algorithms library. *Software – Practice and Experience*, 30:1167–1202, 2000.

[FHHN99] E. Flato, D. Halperin, I. Hanniel, and O. Nechushtan. The design and implementation of planar maps in CGAL. In *Proceedings of the 3rd Workshop on Algorithm Engineering*, volume 1668 of *Lecture Notes in Computer Science*, pages 154–168. Springer, 1999.

[GHS01] N. Geismann, M. Hemmer, and E. Schömer. Computing a 3-dimensional cell in an arrangement of quadrics: Exactly and actually. In *ACM Conference on Computational Geometry*, 2001.

[KCF+] J. Keyser, T. Culver, M. Foskey, S. Krishnan, and D. Manocha. Esolid - a system for exact boundary evaluation. submitted to Solid Modelling 2002.

[KCMK98] J. Keyser, T. Culver, D. Manocha, and S. Krishnan. MAPC: A library for efficient and exact manipulation of algebraic points and curves. Technical Report TR98-038, University of N. Carolina, Chapel Hill, 1998.

[MN99] K. Mehlhorn and S. Näher. *The LEDA Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999. 1018 pages.

[MS89] David R. Musser and Alexander A. Stepanov. Generic programming. In *1st Intl. Joint Conf. of ISSAC-88 and AAEC-6*, pages 13–25. Springer LNCS 358, 1989.

[MS94] David R. Musser and Alexander A. Stepanov. Algorithm-oriented generic libraries. *Software – Practice and Experience*, 24(7):623–642, July 1994.

[Mye95] Nathan C. Myers. Traits: A new and useful template technique. *C++ Report*, June 1995. `http://www.cantrip.org/traits.html`.

[RZ01] F. Rouillier and P. Zimmermann. Efficient isolation of polynomial real roots. Technical Report 4113, INRIA, 2001.

[SV87] A. A. Schäffer and C. J. Van Wyk. Convex hulls of piecewise-smooth Jordan curves. *J. Algorithms*, 8:66–94, 1987.

[Wei] Ron Wein. High-level filtering for arrangements of conic arcs. submitted to ESA 2002.