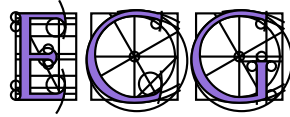


ECG
IST-2000-26473
Effective Computational Geometry for Curves and Surfaces



ECG Technical Report No. : *ECG-TR-182202-01*

*Sweeping Arrangements of Cubic Segments
Exactly and Efficiently*

Arno Eigenwillig

Elmar Schömer

Nicola Wolpert

Deliverable: 18 22 02 (item 01)
Site: MPI
Month: 18



Project funded by the European Community
under the “Information Society Technologies”
Programme (1998–2002)



Sweeping Arrangements of Cubic Segments Exactly and Efficiently*

Arno Eigenwillig
(arno@mpi-sb.mpg.de)
Max-Planck-Institut
für Informatik[†]

Elmar Schömer
(schoemer@informatik.uni-mainz.de)
Johannes-Gutenberg-
Universität Mainz[‡]

Nicola Wolpert
(nicola@mpi-sb.mpg.de)
Max-Planck-Institut
für Informatik[†]

Abstract

A method is presented to compute the planar arrangement induced by segments of algebraic curves of degree three (or less), using an improved Bentley-Ottmann sweep-line algorithm. Our method is exact (it provides the mathematically correct result), complete (it handles all possible geometric degeneracies), and efficient (the implementation can handle hundreds of segments). The range of possible input segments comprises conic arcs and cubic splines as special cases of particular practical importance.

1 Introduction

We present a method to compute the arrangement induced in the affine real plane \mathbb{R}^2 by a set of segments of algebraic curves of degree three (or less) with rational coefficients. Consider for example the three cubic segments in Figure 1. Computing arrangements is an important fundamental operation in computational geometry. In particular, it forms the central step of regularized boolean operations on polygons bounded by segments of the kind in question [16, sect. 10.8]. However, computing arrangements both exactly and efficiently in the presence of degeneracies (such as singularities or tangential intersections) is notoriously full

of difficulties. Overcoming these is an active area of research to which we contribute.

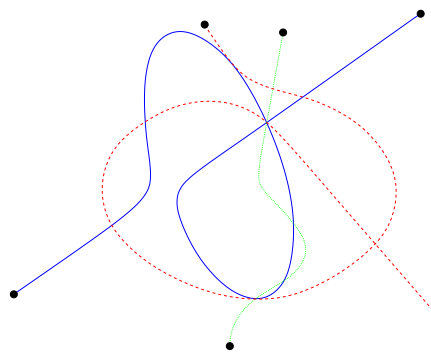


Figure 1: An arrangement of three cubic segments.

The curves we consider are defined as the sets of zeroes $(x, y) \in \mathbb{R}^2$ of bivariate polynomials $f \in \mathbb{Q}[x, y]$ of total degree $\deg(f) \leq 3$. As usual, we use the term *curve* and the symbol f both for the polynomial and the point set. The term *cubic curve*, or *cubic*, customarily denotes a curve of degree exactly 3. Whenever we talk about cubics in this text, the reader is invited to observe that our considerations cover the simpler cases of lines and conics, i.e. degrees 1 and 2, as well.

Mathematics provides a rich theory of algebraic curves. An accessible introduction to the geometry of such curves is [11] by Gibson.

A segment s of a curve f is a “smooth piece of curve between two points”, or more formally, a subset $s \subseteq f$ that consists of either a single point or its two boundary points (called its endpoints) plus an interior that is a connected C^∞ -smooth 1-manifold.

*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)

[†]Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany

[‡]Institut für Informatik, 55099 Mainz, Germany

Examples of such segments are cubic splines¹, including NURBS (as eliminating t from their rational parametrization $t \mapsto (x(t), y(t))$ yields a polynomial in x and y of degree 3), and, trivially, conic arcs and line segments. However, our method can handle all cubic curves, including those that cannot be parametrized by rational maps, so we cover much more than these examples.

Our goal is to compute the arrangement induced by a finite set of segments, that is, the decomposition of the plane into vertices, edges and faces induced by the union of the segments.

Our main result is a method that computes the arrangement exactly and efficiently in all geometric situations. Achieving these three items together is a novelty.

Exactness means that our method provides the mathematically correct result, without any change of geometry due to rounding error. Many geometric situations are sensitive to rounding error, e.g. a tangential intersection of two curves changes into two or no intersections at the slightest perturbation. To avoid this, we follow the exact computation paradigm and use exact arithmetic throughout.

Efficiency is the natural antagonist of exactness, because exact arithmetic is expensive in general. Filtered rational arithmetic has overcome this for straight lines [16, sect. 9.7], and similarly filtered root expressions² [13] [16, sect. 4.4] solve the problem for circles. There is no efficient number type available yet that would be powerful enough to represent the coordinates of intersection points of cubics.³ Hence we resort to indirect ways of examining curves, trying to avoid the explicit representation of point coordinates whenever possible. We only need expensive unfiltered traditional methods of symbolic computation for the very special case of a cubic curve consisting of three lines intersecting in exactly three distinct

¹or rather their pieces after breaking them at singularities, if present

²Root expressions are the closure of \mathbb{Q} under field operations and taking real k -th roots.

³However, theoretical foundations for a possible approach have been laid in [5].

irrational points.

Our approach consists of supplying an improved Bentley-Ottmann sweep line algorithm with the geometric operations it needs (Section 5), based on the analysis of cubic curves (Section 3) and pairs of cubic curves (Section 4). These analyses impose certain conditions on the algebraic representation of the geometric situation (Section 6), but not on the geometry itself. The efficiency of our approach is demonstrated by an implementation and its running times (Section 7).

2 Related Work

Computing arrangements, although mostly of linear objects, is a major focus in computational geometry; see the survey articles of Halperin [12] and Agarwal and Sharir [1].

Many exact methods to handle curved objects, e.g. [17], have been formulated for the Real RAM model of computation [20], which allows constant-time operations on arbitrary algebraic numbers. This idealized model ignores the fact that exact computation with algebraic numbers is very costly.

A more realistic view was taken by Sakkalis [22] in his analysis of a real algebraic curve, but the singularities occurring in cubic curves do not necessitate the full algebraic machinery he develops.

Aspects of the crucial problem to capture behaviour at irrational points by rational arithmetic were treated by Canny [6] (Gap Theorem) and Pedersen [19] (multivariate Sturm sequences).

MAPC [14] is a library for exact computation and manipulation of algebraic curves. It includes the computation of arrangements of planar curves but does not handle all degenerate situations.

Exact, efficient, and complete algorithms for planar arrangements have been published by Wein [24] and Berberich et al. [4] for conic segments, and by Wolpert [25] (see also [10]) for special quartic curves as part of a surface intersection algorithm.

The method presented here follows the general approach of Berberich et al., including the use of a Bentley-Ottmann-type [3] sweep line algo-

rithm for segment intersection. However, conics allow root expression arithmetic for their $y(x)$ parametrization and for all event point coordinates except transversal intersections. This is not true for cubics, hence for them the basic geometric operations have to be rather different. Parts of our approach to these geometric operations stem from Wolpert’s thesis [25], recast into this new context.

3 Analyzing one curve

With the goal of sweeping in mind, we first wish to investigate the behaviour of a single cubic curve f at a given x -coordinate, meaning the number and relative position of its branches over x . This behaviour changes at only a finite number of *event points* where branches start, end, or cross, and it remains unchanged over the open intervals of x -coordinates between them, consider the left picture in Figure 2. There are two kinds of one-curve event points:

- *x -extreme points*, defined as being smooth points with an x -coordinate that is minimal or maximal among the neighbouring points on the curve. At such a point, the curve’s normal vector (f_x, f_y) is horizontal⁴ (Figure 2 i)).
- *singular points*, or *singularities*, defined as those points with vanishing normal vector. Intuitively, these are points where several branches of one curve intersect (but beware, some or all of the branches may be complex and puncture the real plane only at this point) (Figure 2 ii)).

Observe that in both cases f_y vanishes. The only other points where f_y vanishes on f are *vertical flexes* of f , i.e. non- x -extreme points with a vertical tangent (Figure 2 iii)). They are the elements of $(f \cap f_y \cap f_{yy}) \setminus f_x$. Their coordinates can be represented as root expressions. We show below how to remove them.

This leaves us with the problem of determining all points on f for which f_y vanishes, i.e. the intersection points of these curves. To do this, we

⁴The subscripts denote partial derivatives.

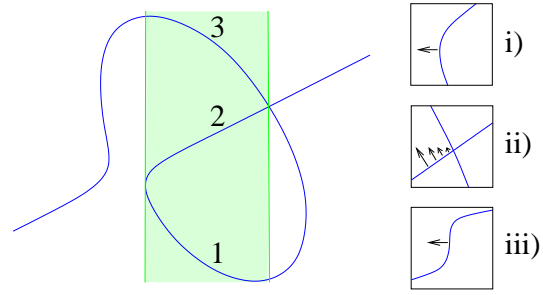


Figure 2: The number and relative position of branches does not change between two event points.

eliminate the variable y between the two equations $f = 0, f_y = 0$ by computing their *resultant*, a univariate polynomial $\text{res}(f, f_y, y) \in \mathbb{Q}[x]$ whose zeroes are precisely the x -coordinates of the intersection points [7, ch. 3]. For degree reasons, a cubic curve f can never have two covertical intersection points with f_y , hence the zeroes of the resultant are in one-to-one correspondence with the intersections, and the multiplicities of the zeroes are precisely the multiplicities of intersection. The multiplicity of intersection is 1 for x -extreme points and ≥ 2 for singularities.

We factor the resultant by multiplicities, i.e. $\text{res}(f, f_y, y) = \prod_{i=1}^k r_i^i$ such that the $r_i \in \mathbb{Q}[x]$ are square-free coprime polynomials [9, ch. 8]. The real zeroes of each r_i can be isolated in disjoint intervals using Uspensky’s algorithm [2, ch. 7] [21] or Sturm sequences [2, ch. 7] [26].

This leads to a representation of an x -coordinate by a square-free polynomial vanishing at this coordinate, and an isolating interval. We can test such numbers for equality (by inspecting their isolating intervals and the gcd of the defining polynomials) and compare two distinct such numbers (by refining their isolating intervals to disjointness), see [4]. Using this, we sort all zeroes of $\text{res}(f, f_y, y)$, yielding the decomposition of the x -axis into open intervals between the events.

Having determined an x -coordinate, we need to extend it back to the event point by determining which branches of f are involved. For sin-

gularities, we resort to exact arithmetic.⁵ Except for the case of a cubic curve consisting of three lines intersecting in exactly three distinct irrational points, rational or root expression arithmetic suffices to represent singularities.

For x -extreme points, we begin by refining their isolating interval to exclude event points of f_y . Then we substitute the interval boundaries for the variable x , solve f and f_y for y by root isolation, and sort the points over each of the two boundaries w.r.t. y -coordinates.⁶ The branches of f_y do not change their relative position between the interval boundaries. They separate the branches of f , consider Figure 3. Thus we can tell which two branches begin or end at the x -extreme point.

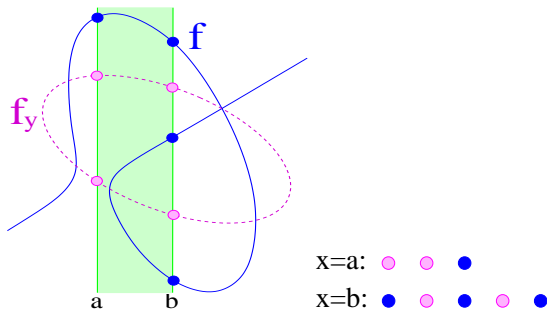


Figure 3: Determine x -extreme points by sequence comparison.

This idea of analyzing an event at an implicitly given x -coordinate by inspecting the change in behaviour of certain curves between its left and right side is central to our method, and we will use it repeatedly in the next section.

A point, having been found like this, is represented by its x -coordinate x (as above), a supporting curve f , and a *branch number* $i \in \mathbb{N}$, meaning that it is the i -th point of f over x (counted in ascending y order, without multiplicities).

We also need to classify singularities in order to trace input segments through them. The textbook

⁵Note that once we know x in some explicit form, the y coordinates of both the singularity and the point on a third branch (if present) can be determined by substituting x into f and factoring the resulting univariate polynomial by multiplicities.

⁶using the same real algebraic number type we used before for the projected x -coordinates

classification of complex projective cubic curves [11] together with the fact that a unique singularity must be rational indicates that irrational singularities are always transversal intersections of two branches. Rational singularities can be analyzed by translating them to the origin and inspecting the quadratic part $ay^2 + bxy + cx^2$ of the translated polynomial.

To perform the analysis of one curve, we demand the following general position properties:

- The coefficient of $y^{\deg(f)}$ in f must be nonzero (*leading coefficient condition*). This is needed for the smooth working of the resultant formalism. It is sufficient for the absence of vertical asymptotes⁷ (where branches could disappear to infinity instead of ending at an event point).
- To simplify the algorithm, we demand the absence of vertical tangents at flexes and singularities.

This does not restrict the geometric situations we can handle, only the choice of a y -axis. We can detect all violations of these properties and remove them by shearing the scene (see Section 6).

We also require the curve's defining polynomial to be square-free. This is no restriction on its set of zeroes. Assuming the leading coefficient condition, the square-free part of f is $f / \gcd(f, f_y)$.

Having performed the analysis of one curve, we know the number of points on it over any given x . We know where x -extreme points and singularities lie, and we can trace segments through them.

4 Analyzing two curves

We now turn to the behaviour of a pair $\{f, g\}$ of curves. Again, we have open intervals of constant behaviour, and a finite number of event points in between, being one-curve events or intersections.

Between two adjacent events, the relative position of branches of f and g can be determined easily by substituting a rational x into f and g , solving for y by root isolation, and sorting the results. Thus, given an arbitrary x from such an interval,

⁷Over the complex numbers, it is also necessary.

we know how to map i to k such that the i -th point on f over x , counted in ascending y -direction, is the k -th point of $f \cup g$ over x . This we call *slicing* a pair of curves. It allows us to compare segments and points in y -direction, and it will form the basis of implementing the predicates needed by the sweep algorithm (Section 5). We also need to slice at event points.

Using the analysis of a single curve, it is easy to extend slicing to x -coordinates over which just a one-curve event happens. The rest of this section describes how to slice at the remaining x -coordinates; those over which an intersection occurs.

As before, we use a resultant $\text{res}(f, g, y)$ to project the intersection points onto their x -coordinates. We introduce an additional requirement on the choice of coordinates, namely that no two intersections of f and g are covertical, yielding a one-to-one correspondence of intersections and roots of $\text{res}(f, g, y)$. Then, as above, the multiplicity of a root is the multiplicity of the corresponding intersection.⁸

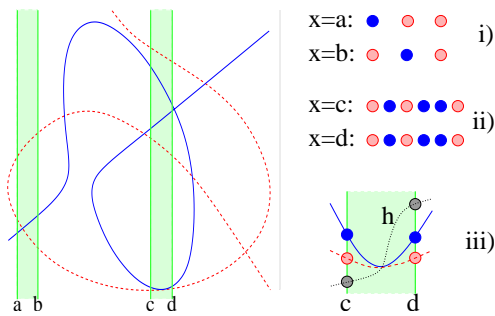


Figure 4: Determine intersection points of two curves.

Again, we need to find out the branches intersecting at a given zero x of $\text{res}(f, g, y)$. First consider the case of an intersection not covertical to any singularity. If the intersection multiplicity is odd, the intersecting branches change their position. Hence by inspecting the branches over the intervals left and right of the intersection, we see exactly one transposition, indicating the

⁸For non-singular points, an intersection multiplicity 1, 2, 3, ... means that the curves agree in this common point, this point and their slope, or this point and their slope and their curvature, etc.

branches involved (Figure 4 i)). If the intersection multiplicity is 2, we use an auxiliary curve of degree 4, the Jacobi curve $h = f_x g_y - f_y g_x$ (see [25]) that cuts transversally through the intersection, yielding an observable transposition of branches (Figure 4 ii) and iii)). For even multiplicities > 2 , we employ exact arithmetic with root expressions or rationals. This is possible because $\deg(\text{res}(f, g, y)) = \deg(f) \cdot \deg(g) \leq 9$, such that factoring by multiplicities produces a factor of degree at most 2 for multiplicity 4 or of degree at most 1 for multiplicities 6 and 8.

This way of looking left and right does not handle cases involving x -extreme points, because the branches starting or ending in an x -extreme point do not extend into both intervals around the point. Thus, if an intersection occurs covertical to or at an x -extreme point, we shear the scene.

Now consider the case that f has a singularity at an x -coordinate but g does not. If x is rational, we can substitute, solve, and sort. Otherwise, the singularity must be a transversal intersection of two branches, and the intersecting branch of the other curve cannot be tangential to both, hence again we have an observable transposition, consider for example Figure 5.

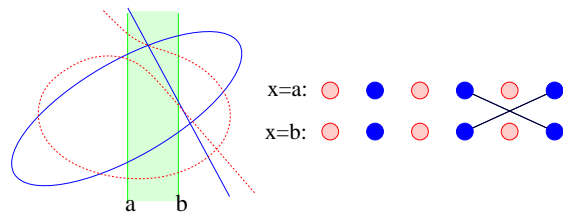


Figure 5: At an irrational singular point we have an observable transposition.

To check the absence of two covertical intersections at x in the above cases, observe that substituting x into g leads to a square-free polynomial, i.e. $g(x, \cdot)$ has only simple zeroes. Hence $d := \deg(\text{gcd}(f(x, \cdot), g(x, \cdot)))$ is the number of (complex) intersection points over x . We demand it to be 1.⁹ But substituting x is not an effective op-

⁹That also removes the problem of a complex intersection $(x_0, y_0 + iy_1)$ causing a real zero x_0 of the resultant, because it would be covertical to its conjugate $(x_0, y_0 - iy_1)$.

eration, since in general we have no explicit representation for x .

Recall that $\text{res}(f, g, y)(x) = 0$ implies $d \geq 1$, as there is at least one intersection point. Extending the notion of a resultant, one can define the *first subresultant* $\text{sres}_1(f, g, y)$ of f and g such that $\text{res}(f, g, y)(x) = \text{sres}_1(f, g, y)(x) = 0$ implies $d \geq 2$ [23, 6.10] [15]. Hence our check reduces to testing whether $\text{sres}_1(f, g, y)(x) = 0$. We can do this even with our implicit representation of x -coordinates. If $\text{sres}_1(f, g, y)(x) = 0$, we need to shear the scene.

It remains to treat the case of an intersection covertical to singularities on both curves. This can be dealt with using the explicit representations from the analyses of the individual curves.

In summary, we impose the following additional general position assumptions:

- No two events (intersections and one-curve events) may be covertical. This is even stricter than above, but simplifies the algorithm, as it ensures a one-to-one correspondence of event points and the union of zeroes of $\text{res}(f, f_y, y)$, $\text{res}(g, g_y, y)$ and $\text{res}(f, g, y)$.
- An intersection point must not be x -extreme on either curve.
- If we use a Jacobi curve, it must not have a one-curve event covertical to the intersection we are analyzing, as that might disturb finding its transversal intersection.

Again, these requirements only restrict the choice of a y -axis, not the geometry of the scene. We can check them and enforce them by shearing.

Using the resultant formalism requires that f and g do not have a common component, i.e. a non-constant polynomial dividing both. If there are common components, one can remove them by computing $h = \text{gcd}(f, g) \in \mathbb{Q}[x, y]$ and replacing f, g by $f/h, g/h, h$. By our condition of square-freeness, these are now coprime. This does not restrict the segments we want to define on the curves: a piece of curve that “changes component” is not C^∞ -smooth.

5 Sweeping an Arrangement

We show how to perform a Bentley-Ottmann-like sweep of segments of cubic curves, based on the analysis of one and two curves.

As input, we accept a set of cubic *input segments*, each defined by a supporting curve, a rational start point and tangent vector (indicating in which direction the segment leaves the point), and a rational target point.¹⁰ The tangent vector is necessary to resolve certain ambiguities, e.g. for a segment lying on a circle-shaped part of the curve or starting in a singularity.

As output, our method computes a planar map labelled with points (including auxiliary points like x -extreme points) and input segments, representing the arrangement.

In a preprocessing step, input segments are broken into sweepable segments (*segments* for short) such that each segment s has these properties:

- s is x -monotone and has no singularities in its interior, i.e. $\text{interior}(s) \cap f_y = \{\}$.
- All points in the interior of s have the same branch number.

A segment is represented by its endpoints, its supporting curve, and its respective branch numbers in the interior and at the endpoints.

A point is represented as in the preceding section, viz. by an x -coordinate, a supporting curve and a branch number.

We use an improved version of the Bentley-Ottmann sweep for segment intersection [3] derived from the LEDA implementation of line segment intersection [16, sect. 10.7] which is free of general position assumptions. In particular, segments can overlap, and an arbitrary number of segments can start, end, and intersect in a single event point.

¹⁰The requirement of rational coordinates is intended to allow an efficient application of a shear as a preprocessing step. It naturally holds for segments constructed in the typical fashion of interpolating a conic through five rational points or defining a spline by a rational control polygon. The actual sweeping algorithm imposes no requirement of this kind.

Recall how sweeping works: A vertical line is swept over the set of segments. Left of it, the planar map has already been constructed. The segments currently intersecting this sweep line are stored in a sequence called the Y-structure. All remaining endpoints and the remaining intersection points of at least those segments that are adjacent in the Y-structure are stored in a queue called the X-structure, which is sorted in (x, y) lexicographic order. Until the X-structure has been emptied, the following steps are performed:

- Extract next event from the X-structure and advance the sweep line to it. Find segments involved in the event by locating the event in the Y-structure, exploiting its order.
Necessary predicate: Is a point p above, on, or below a given segment s ? Decide this using the analysis of the two curves supporting p and s , by comparing their branches at the point's x -coordinate.
- Remove ending segments.
- Reorder intersecting segments according to multiplicity of intersection, see [4].
Necessary predicate: intersection multiplicity. This can be read off immediately from the multiplicity of the corresponding root of $\text{res}(f, g, y)$.
Necessary predicate: do segments overlap? This is easy to decide as it can only happen if the two supporting curves are identical¹¹.
- Add starting segments to the Y-structure, obeying its ordering.
Necessary predicate: comparison of segments right of a common point. Decide this by analyzing the two supporting curves and comparing the supporting branches over the interval right of the intersection.
- Add intersections of newly adjacent segments to X-structure¹², obeying its ordering.
Necessary operation: compute intersection points. Do so by performing the analysis of

¹¹... up to a constant factor. Recall that we require them to be coprime.

¹²This is the crucial point of the sweep line algorithm: Only neighbouring segments have their intersections computed, avoiding the naive $O(N^2)$ analysis of all pairs.

two curves in the relevant x -range and selecting the intersections on the appropriate branches.

Necessary predicate: (x, y) lexicographic comparison of points. If comparing x does not break the tie, analyze the two curves supporting the points and compare the supporting branches over the common x coordinate.

Observe how the use of predicates by the sweep line algorithm automatically reduces all geometric analyses to at most two curves at a time.¹³

Running the algorithm and evaluating the predicates may necessitate to shear the whole scene. Once the algorithm terminates, the edges of the planar map are labelled with the original segments. This is no problem. However, the vertices are labelled with an implicit representation of points which is meaningless after shearing back because the notion of an i -th point on a curve at a given x is disturbed by shearing. Hence there is a post-processing step in which the coordinates of the implicitly represented points are computed and sheared back numerically within an arbitrarily low error bound specified by the caller. Since this happens only afterwards, the topology of the arrangement is always correct, even if the caller requests a low numeric precision (e.g. for displaying results at a low resolution with several vertices falling onto the same pixel).

6 Choosing a Projection Direction

We imposed certain position conditions that do not restrict the geometry of the arrangement but its algebraic representation, in particular the choice of the y -axis. (The other limitations are the squarefreeness and coprimality assumptions, which can be dealt with by preprocessing or by restarting the algorithm if a violation is detected.)

There is a finite number of lines which are independent of the choice of coordinates to which the y -axis must not be parallel:

- real and complex asymptotes (for the leading coefficient condition)

¹³Above, we have discussed the general cases of two distinct supporting curves. It is clear how to handle the special cases of two supporting curves that happen to be identical.

- tangents at flexes and singularities
- lines connecting event points with real or complex intersection points in any pair of curves we consider
- tangents at intersection points of any pair of curves we consider
- lines connecting an intersection of multiplicity 2 and a one-curve event on the corresponding Jacobi curve¹⁴ (including complex projective events)

Almost every direction of the y -axis will do. We exploit this by randomly shearing our input segments, starting the algorithm, and hoping for the best.

A *shear* is an invertible linear map $S_r: (x, y) \mapsto (x + ry, y)$ for a fixed $r \in \mathbb{Q}$. It leaves the x -axis fixed and tilts the y -axis. Shearing a point p means replacing it by $S_r(p)$. Shearing a curve f means replacing f by $f \circ S_r^{-1}$, because $f(p) = 0 \Leftrightarrow (f \circ S_r^{-1})(S_r(p)) = 0$.

If a violation of the conditions is detected, we start over (with a new shearing parameter r chosen at random from a set whose size grows exponentially with the number of unlucky earlier choices). Eventually (and very quickly in practice, as “most” forbidden lines are irrational) a choice is lucky and we succeed to compute the arrangement.

We prefer to start with a small set of shearing parameters with limited bit length (instead of making an a priori estimate of a sufficiently large set) because the running time increases with the bit length of the curves’ coefficients and hence with the bit size of the shearing parameter.

Note that the position conditions come from the analysis of curves, not from the sweep. For example, covertical intersection points are no problem as long as they do not involve the same two curves. As far as the conditions relate to pairs of curves, they only affect pairs that are subjected to an analysis. We do not consider all pairs of all curves in the general case; only those that interact geometrically and those needed while searching the Y-structure. Also, the enforcement

¹⁴which is invariant under a change of coordinates

of the conditions can be limited to the range of x -coordinates of interest.

7 Implementation and Timings

The method described in this text is being implemented in C++ as part of the EXACUS project [8]. The modular structure of the implementation is similar to this text: The analysis of one curve, of two curves, the implementation of points and segments along with their operations, and finally the actual sweep code build hierarchically on top of each other. The analyses of curves happen only to the extent requested by modules calling them, and their results are cached to avoid repetitions of costly operations.

The basic number types used currently are those of LEDA (including LEDA “reals” for root expressions), but the code is templated with a traits class to allow future changes. The case of three irrational singularities is handled classically with explicit arithmetic in $\mathbb{Q}[x]/(p)$, see [26, 6.1.3]. Resultant and gcd operations are implemented using subresultant remainder sequences [9]; root isolation is performed with Uspensky’s algorithm [2, ch. 7] [21].

Given the space limitations of an extended abstract and the continuing evolution of the implementation, we restrict our benchmarks to the core operation: sweeping a set of segments once the coordinates have been chosen suitably.

For benchmarking, we generate cubics $f = \sum_{j=0}^3 \sum_{i=0}^{3-j} a_{ij} x^i y^j$ by interpolating through 9 points $\{(x_k, y_k)\}_{k=1}^9$ chosen randomly on a 256×256 grid, yielding 9 linear conditions on the 10 unknowns a_{ij} . One can exchange some of these conditions for linear conditions generated by prescribing slope, curvature, etc. at the remaining points. Solving for the 10 unknowns a_{ij} and scaling them to be coprime integers yields polynomials with coefficients of roughly 100 bits length each. We break these into sweepable segments, clipping at $x = \pm 10000$.

On a 1.2GHz Pentium III, the following measurements have been made (all numbers averaged):

- 10 independent random curves yield about 80 sweepable segments. Computing their arrangement takes about 6.1 seconds and results in a planar map whose size is about 370 vertices and 1260 edges. This is roughly 20 milliseconds per event point.
- For 30 independent random curves, the figures are 230 segments in 70 seconds, resulting in 3000 vertices and 11000 edges.
- 10 curves interpolated through random subsets of 18 interpolation points such that slope and with probability 0.5 also curvature are shared yield about 100 sweepable segments. In 24 seconds an arrangement of about 350 vertices and 1130 edges is computed. Many of the intersections occur independent of common interpolation points and hence are simple. The larger running time per vertex is due to the analysis of Jacobi curves necessitated by the intersections of multiplicity 2.

This benchmark is limited to rational interpolation points (x_k, y_k) , but as our implementation does not factor aggressively, this does not affect running times, and most simple intersections occur independent of interpolation points at irrational coordinates.

8 Conclusions

We have demonstrated that it is practically feasible to compute an arrangement of segments of cubic curves with rational coefficients by a method that can handle all possible geometric degeneracies. This also solves the essential difficulty in performing boolean operations on polygons bounded by cubic segments exactly and efficiently in all cases [16, sect. 10.8].

Our approach to implementing the basic geometric operations is not tied to a sweep-line algorithm. Alternative algorithms for arrangement computation, e.g. Randomized Incremental Construction (see [18]), could be based on similar geometric operations.

The method we presented makes only very limited use of the idea of floating-point filtering: The evaluation of root expressions is done automatically in a filtered fashion. The representation of

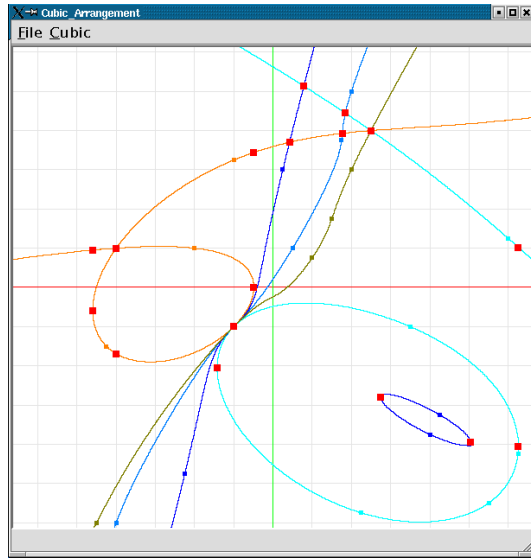


Figure 6: Arrangement computed by our program. The small dots are the interpolation points of the curves. The boxes mark the calculated event points.

real algebraic numbers by a defining polynomial and an isolating interval that is refined only as much as needed also yields a form of delayed adaptive-precision computation, but still with exact number types. Better filtering techniques are a likely source of significant speed-ups.

Parts of our method generalize to algebraic curves of arbitrary degree, in particular the derivation of geometric predicates from slices (see sect. 4) of two curves, and many aspects of analyzing curves in the absence of singularities. The major critical dependencies on degree ≤ 3 that remain are:

- Use of efficient explicit arithmetic to handle singularities and vertical flexes. This requires low degrees of the algebraic numbers involved.
- Use of the complete classification of complex projective cubics to trace input segments through singularities.

9 Acknowledgements

The authors would like to thank Susan Hert, Kurt Mehlhorn, Michael Hemmer, and Eric Berberich for useful comments and for contributions to the prototype implementation.

References

- [1] P. K. Agarwal and M. Sharir. Arrangements and their applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
- [2] A. G. Akritas. *Elements of Computer Algebra With Applications*. Wiley, New York, 1989.
- [3] J. L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979.
- [4] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, K. Mehlhorn, and E. Schömer. A computational basis for conic arcs and boolean operations on conic polygons. In *ESA 2002, Lecture Notes in Computer Science*, pages 174–186, 2002.
- [5] 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.
- [6] J. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1987.
- [7] D. Cox, J. Little, and D. O’Shea. *Ideals, Varieties, and Algorithms*. Springer, New York, 1997.
- [8] The EXACUS project. www.mpi-sb.mpg.de/projects/EXACUS/.
- [9] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer, 1992.
- [10] N. Geismann, M. Hemmer, and E. Schömer. Computing a 3-dimensional cell in an arrangement of quadrics: Exactly and actually! In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 264–271, 2001.
- [11] C. G. Gibson. *Elementary Geometry of Algebraic Curves*. Cambridge University Press, 1998.
- [12] D. Halperin. Arrangements. In J. E. Goodman and J. O’Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 21, pages 389–412. CRC Press LLC, Boca Raton, FL, 1997.
- [13] V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. *The CORE Library Project*, 1.2 edition, 1999. <http://www.cs.nyu.edu/exact/core/>.
- [14] J. Keyser, T. Culver, D. Manocha, and S. Krishnan. MAPC: A library for efficient and exact manipulation of algebraic points and curves. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 360–369, 1999.
- [15] T. Lücking. Subresultants. Master’s thesis, University of Paderborn, Germany, 2000.
- [16] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 2000.
- [17] K. Mulmuley. A fast planar partition algorithm, II. *J. ACM*, 38:74–103, 1991.
- [18] K. Mulmuley. *Computational Geometry*. Prentice Hall, Englewood Cliffs, NJ, 1994.
- [19] P. Pedersen. Multivariate sturm theory. In *Proceedings of Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 318–323, 1991.
- [20] F. P. Preparata and M. I. Shamos. *Computational geometry and introduction*. Springer-Verlag, New York, 1985.
- [21] F. Rouillier and P. Zimmermann. Efficient isolation of polynomial real roots. Technical Report 4113, INRIA, 2001.
- [22] T. Sakkalis. The topological configuration of a real algebraic curve. *Bulletin of the Australian Mathematical Society*, 43:37–50, 1991.
- [23] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.
- [24] R. Wein. On the planar intersection of natural quadrics. In *ESA 2002, Lecture Notes in Computer Science*, pages 884–895, 2002.
- [25] N. Wolpert. *An Exact and Efficient Approach for Computing a Cell in an Arrangement of Quadrics*. Universität des Saarlandes, Saarbrücken, 2002. Ph.D. Thesis.
- [26] C. K. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, New York, Oxford, 2000.