An Exact, Complete and Efficient Implementation for Computing Planar Maps of Quadric Intersection Curves*

Eric Berberich
Max-Planck-Institut für Informatik
66123 Saarbrücken, Germany
eric@mpi-sb.mpg.de

Michael Hemmer
Johannes-Gutenberg-Universität
55099 Mainz, Germany
hemmer@uni-mainz.de

Lutz Kettner
Max-Planck-Institut für Informatik
66123 Saarbrücken, Germany
kettner@mpi-sb.mpg.de

Elmar Schömer
Johannes-Gutenberg-Universität
55099 Mainz, Germany
schoemer@uni-mainz.de

Nicola Wolpert
Max-Planck-Institut für Informatik
66123 Saarbrücken, Germany
wolpert@mpi-sb.mpg.de

ABSTRACT

We present the first exact, complete and efficient implementation that computes for a given set $P = \{p_1, \ldots, p_n\}$ of quadric surfaces the planar map induced by all intersection curves $p_1 \cap p_i$, $2 \leq i \leq n$, running on the surface of p_1. The vertices in this graph are the singular and x-extreme points of the curves as well as all intersection points of pairs of curves. Two vertices are connected by an edge if the underlying points are connected by a branch of one of the curves. Our work is based on and extends ideas developed in [20] and [9].

Our implementation is *complete* in the sense that it can handle all kind of inputs including all degenerate ones where intersection curves have singularities or pairs of curves intersect with high multiplicity. It is *exact* in that it always computes the mathematical correct result. It is *efficient* measured in running times.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*geometric algorithms*; G.1.5 [**Numerical Analysis**]: Roots of Nonlinear Equations—*methods for polynomials*; D.m [**Software**]: Miscellaneous—*robust geometric computation*

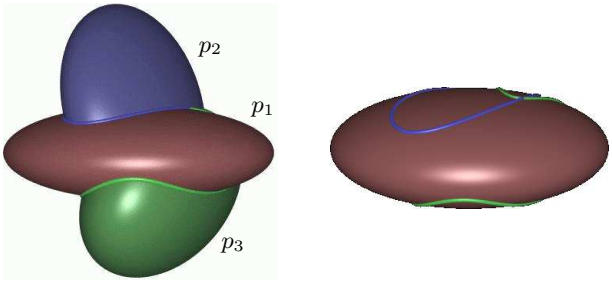
General Terms

Algorithms, Performance, Reliability

Keywords

Computational geometry, arrangements, algebraic surfaces, algebraic curves, robustness, exact geometric computation

1. INTRODUCTION



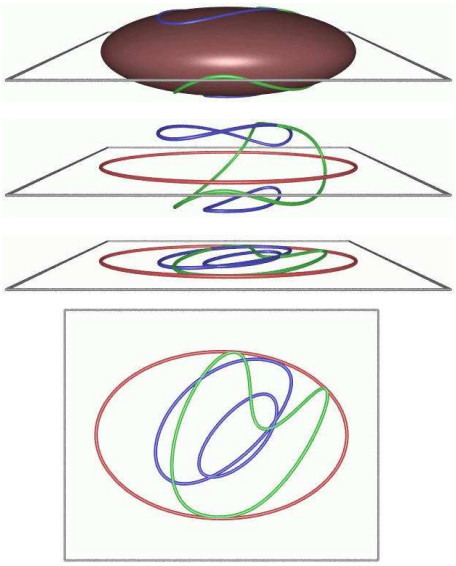
We consider implicitly defined quadric surfaces (*quadrics*) $p_1, \ldots, p_n$ in 3-dimensional space. On the surface of quadric p_1 the intersection curves $p_1 \cap p_i$, $2 \leq i \leq n$, induce a 2-dimensional arrangement. In our example, there are two intersection curves $p_1 \cap p_2$ and $p_1 \cap p_3$ running on the surface of the ellipsoid p_1. They partition the surface into maximal connected regions of dimension 2, 1 and 0. We compute the planar map induced by this arrangement. The vertices in the graph are the singular and x-extreme points of the intersection curves (*one-curve events*) as well as all intersection points of pairs of curves (*two-curve events*). Two vertices are connected by an edge if the underlying points are connected by a branch of one of the curves.

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

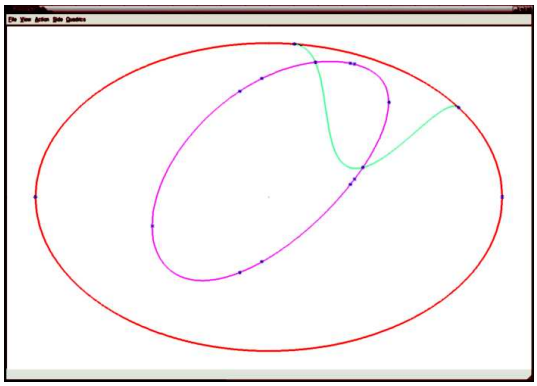
SCG'05, June 6–8, 2005, Pisa, Italy.

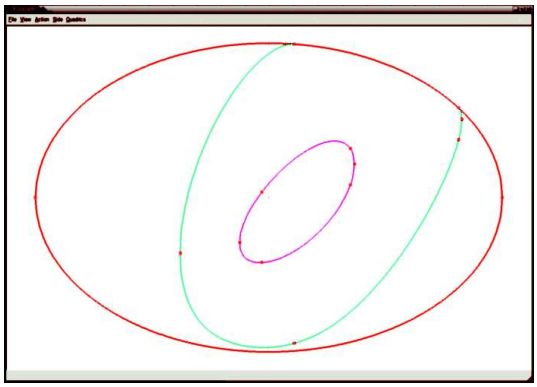
In our work we pursue three goals: We want our algorithm to be *complete* in the sense that we can deal with all kind of inputs including all degenerate ones. We want it to be *exact* in that it always computes the mathematically correct result. Finally we want it to be *efficient* measured in running times.

Determining the planar map of intersection curves on a surface is the crucial operation with respect to exactness when dealing with quadric surfaces. This is independent of the actual computation (e.g. boolean operations) we are interested in. The coordinates of the one- and two-curve events in general are algebraic numbers. In degenerate situations using floating point approximations of these numbers can lead to completely wrong results because of approximation errors, rather than just slightly inaccurate outputs. As soon as we can locate and connect event points correctly on every surface and are able to identify identical event points on different surfaces, the main problem with respect to exactness is solved.



Our work is based on cylindrical algebraic decomposition [6]. We project all intersection curves $p_1 \cap p_i$, $2 \leq i \leq n$, and additionally the silhouette $p_1 \cap \frac{\partial p_1}{\partial z}$ of p_1 into the (x, y)-plane. This projection step applied to our example proceeds like shown in the above picture. We obtain a planar arrangement of algebraic curves of degree at most 4. In [20] the geometry of the resulting curves and of pairs of curves is thoroughly discussed. An improved Bentley-Ottmann sweep-line method for cubic curves based on the analysis of curves and pairs of curves is presented in [9]. We enhance these previous results by providing efficient schemes for the curve analysis needed to apply the Bentley-Ottmann sweep-line method [2] to our projected arrangements.
At the projection step we loose the spatial information: branches on the upper and lower part of p_1 are projected on top of each other. We present a method to regain this information and to split the arrangement into one for the upper and one for the lower part of p_1. Both arrangements together completely describe the arrangement of intersection curves on the surface of p_1. For combining information on different surfaces we shortly sketch how to identify identical event points.





We present a full implementation with extensive benchmarking of our algorithm. Our implementation achieves all three goals: it is complete, exact and efficient. Above there are screen shots made by our program. We compute the planar arrangements on the upper and on the lower part of the ellipsoid p_1, see the respective upper and lower picture.

2. PREVIOUS WORK

Quadric surfaces are of great importance because they are the simplest of all curved surfaces and they are widely used in solid modeling for the design of mechanical parts. Concerning efficiency the algorithms in CAD systems profit from floating point arithmetic. But just this makes them very sensitive to approximation and rounding errors. None of these systems are exact nor complete. Some efforts towards exact and efficient implementations have been made in the libraries MAPC [13] and ESOLID [7] which deal with algebraic points and curves and with low-degree surfaces, respectively. Both libraries are not complete.

Computer algebra methods, based on exact arithmetic, guarantee the correctness of the results. But one has to be careful in choosing only those techniques that perform well in the problem-specific context and that yield acceptable running times when compared to the floating-point approach. Collins [6] introduced the cylindrical algebraic decomposition. In principle this method is implementable and our algorithm is based on it. But after the projection steps one is left with roots of univariate polynomials. The main question with respect to efficiency is how to perform the backwards construction based on these algebraic numbers.

For a long time the focus in computational geometry was mainly on computing arrangements of linear objects. The ones dealing with curved objects all neglected the problem of exact computation in the way that they were based on an

idealized real arithmetic provided by the real RAM model of computation [19]. This model of unit cost per operation is not in accordance with real computers. Recently the exact computation of arrangements of non-linear objects has come into the focus of research. Wein [21] extended the CGAL implementation of planar maps to conic arcs. Berberich et al. [5] made a similar approach for conic arcs based on the improved LEDA [17] implementation of the Bentley-Ottmann sweep-line algorithm [2]. Eigenwillig et al. [9] extended the sweep-line approach to cubic curves. A generalization of Jacobi curves (used below for locating tangential intersections) is described by Wolpert [23]. Finally there are efforts to extend CGAL with a kernel for curved objects [10].

For computing with quadric surfaces Levin [15], [16] developed a method to parameterize the spatial intersection curves. The resulting formulas were not suited for further symbolic processing. Dupont et al. [8] succeeded in finding parameterizations which overcome this difficulty. Recently Lazard et al. [14] provided a complete implementation of this approach. The disadvantage, however, is that it cannot be generalized to higher degree surfaces, due to the lack of manageable parameterizations for the occurring intersection curves. A second method dealing with arrangement of quadrics by Mourrain et al. [18] reduces the static 3-dimensional problem to a dynamic 2-dimensional one.

As mentioned before, our work is based on a third approach introduced by Geismann et al. [12] and Schömer and Wolpert [20] (for an extended version consider [22]) that works by projection. We extend and improve the previous ideas for computing small disjoint boxes around the event points to a full analysis of the topology of curves and curve pairs.

3. OUR RESULTS

We briefly summarize the results we present in this paper:

- We have the first complete, exact, and efficient implementation that computes for an arbitrary set $P = \{p_1, \ldots, p_n\}$ of quadric surfaces the planar map induced by all intersection curves $p_1 \cap p_i$, $2 \leq i \leq n$, running on the surface of p_1.
- There is no restriction to the input. Every polynomial $p \in \mathbb{Q}[x, y, z]$, $\deg(p) \leq 2$, defines a valid quadric.
- A projection reduces the 3-dimensional problem to the one of computing a planar arrangement of algebraic curves. The geometry of the curves is discussed in [20]. Based on this we derive efficient realizations for the analysis of curves and pairs of curves needed by the Bentley-Ottmann sweep. Until now the sweep line algorithm was only realized for cubic curves [9]. Our curves can be of degree 4.
- All event points in the planar arrangements (including singular points and intersection points of high multiplicity) are computed correctly and connected in the right way.
- We present a method to regain the spatial information we loose during the projection. The planar arrangement is split into two parts: one consisting of all branches running on the upper part of p_1, the other one consisting of all those running on the lower part. The two planar maps of these arrangements completely describe the planar map on the surface of p_1.
- Our approach allows to identify identical event points on different surfaces. This is important if one is interested in applying further operations (for example boolean operations) to the quadrics.
- Our implementation is done in C++ as part of the Exacus library [11]. It uses the Bentley-Ottmann sweep for segments.
- We present benchmarking results.

4. GEOMETRY OF CURVES AND CURVE PAIRS

We shortly summarize the notation and results achieved in [20]. We assume that the reader is familiar with basic algebraic geometry. Let $P = \{p_1, \ldots, p_n\}$ be the set of input-quadrics, i.e., algebraic surfaces of degree 2. If necessary we use gcd-computations to make quadrics squarefree and pairs of them coprime. This does not change the arrangement but just the way it is represented. We assume throughout the whole paper that every quadric $p = q_0(x, y)z^2 + q_1(x, y)z + q_2(x, y)$ is *z-regular* meaning $q_0(x, y) \not\equiv 0$. This is no loss of generality because a violation can be easily checked by inspecting the leading coefficients of the quadrics in P. A random shear $(x, y, z) \to (x + s \cdot z, y + r \cdot z, z)$ for all $p_i \in P$ establishes regularity with high probability. Then from the point of view of the (x, y)-plane a quadric p has three different parts: the *lower part* | *silhouette* | *upper part* $\{(a, b, c) \in \mathbb{R}^3 \mid p(a, b, z) = q_0(a, b)(z - c)(z - \tilde{c}),\ c\ \{< \mid = \mid >\}\ \tilde{c}\}$.

We compute the resultants $\mathrm{res}(p_1, p_i, z) \in \mathbb{Q}[x, y]$ for all $2 \leq i \leq n$ and also $\mathrm{res}(p_1, \frac{\partial p_1}{\partial z}, z)$ realizing the projection of all intersection curves and of the silhouette of p_1 into the (x, y)-plane. We obtain a planar arrangement of algebraic curves where we call $\mathrm{res}(p_1, p_i, z)$ *cutcurve* and $\mathrm{res}(p_1, \frac{\partial p_1}{\partial z}, z)$ *silhouettecurve*. Let F be this set of curves. By construction exactly one curve in F is a silhouettecurve and of degree at most 2, the others are cutcurves of degree at most 4. We assume without loss of generality that all curves in F are squarefree, y-regular (the leading coefficient with respect to y does not vanish – a random shear of the input quadrics realizes this with high probability) and pairs of them are coprime. We call two curves *non-covertical* if no two distinct intersection points share a common x-coordinate.

Let us first consider a single curve $f \in F$. We always denote $f_x := \frac{\partial f}{\partial x}$. A point $p \in \mathbb{R}^2$ is a *one-curve event* of f if it is *x-extreme* ($f(p) = f_y(p) = 0$, $f_x(p) \neq 0 \neq f_{yy}(p)$) or *singular* ($f(p) = f_y(p) = f_x(p) = 0$). We call p a *vertical flex* if $f(p) = f_y(p) = f_{yy}(p) = 0$, $f_x(p) \neq 0$.

LEMMA 1. *The one-curve events and vertical flexes of f are equal to the intersection points of f and f_y.*

The x-coordinates of intersection points of f and f_y are exactly the roots of the polynomial $R = \mathrm{res}(f, f_y, y)$. We partially factorize R with respect to the multiplicities of its roots: $R = R_1 \cdot R_{\geq 2}$ with $R_1, R_{\geq 2} \in \mathbb{Q}[x]$. R_1 contains all simple roots of R, $R_{\geq 2}$ all multiple ones.

LEMMA 2. *Let f and f_y be non-covertical. Then the real roots of R_1 and $R_{\geq 2}$ are exactly the x-coordinates of extreme points and of singular points and vertical flexes of f, respectively.*

The real roots of a rational quadratic polynomial we call *one-root numbers*. A one-root number α can be expressed in the form $\alpha = a + b\sqrt{c}$ with $a, b, c \in \mathbb{Q}$. If $a, b, c \in \mathbb{Q}(\sqrt{\rho})$, $\rho \in \mathbb{Q}$, we call α a *two-root number*. Computing with one-root and two-root numbers is much easier than with arbitrary algebraic numbers because one has an explicit representation of these numbers and one can compare two of them for example using repeated squaring or separation bounds [17, ch. 4.4]. All singular points of a cutcurve f can be expressed as one- or two-root numbers:

THEOREM 3. *Let f be a cutcurve which originates from the intersection curve of the two quadrics p_1 and p_i. Only if f consists of four lines, the x-coordinates of its singular points are two-root numbers. For all other cutcurves they are one-root numbers. If f has no vertical flexes and no covertical one-curve events, we can factorize $R_{\geq 2}$ into quadratic factors over $\mathbb{Q}(\sqrt{\rho})[x]$ or $\mathbb{Q}[x]$, respectively.*

Next we consider non-covertical pairs of curves $f, g \in F$. There are two cases: either f is the silhouettecurve and g is a cutcurve, or both f and g are cutcurves. Let us consider the first case. We compute the resultant $R = \mathrm{res}(f, g, y)$. The real roots of R are exactly the x-coordinates of intersection points. With respect to the sweep-line algorithm we call these points *two-curve events*. We partially factorize R according to multiplicities: $R = R_1 \cdot R_2 \cdot R_{\geq 3}$, $R_i \in \mathbb{Q}[x]$, such that R_1 contains all simple roots, R_2 contains all double roots and $R_{\geq 3}$ contains all roots of multiplicity ≥ 3.

THEOREM 4. *The real roots of R_1 are the x-coordinates of transversal intersection points of f and g. The real roots of R_2 and $R_{\geq 3}$ are x-coordinates of tangential intersection points. At every intersection point with x-coordinate in R_2 the* Jacobi-curve *$J := f_x g_y - f_y g_x$ cuts both f and g transversally. Every root of $R_{\geq 3}$ can be computed as a one-root number.*

Now let both f and g be non-covertical cutcurves. They are the result of intersecting p_1 with two other quadrics p_i and p_j, respectively.

THEOREM 5. *We can compute two polynomials $R, \tilde{R} \in \mathbb{Q}[x]$, $\deg(R) \leq 8$, $\deg(\tilde{R}) \leq 8$, such that the real roots of $R \cdot \tilde{R}$ are exactly the x-coordinates of intersection points of f and g and the real roots of R are exactly the x-coordinates of common intersection points of p_1, p_i and p_j. Both polynomials can be factorized according to multiplicities $R = R_1 \cdot R_2 \cdot R_{\geq 3}$, $\tilde{R} = \tilde{R}_1 \cdot \tilde{R}_2 \cdot \tilde{R}_{\geq 3}$ and we can apply Theorem 4 to both factorizations.*

5. ANALYSIS OF CURVES AND OF CURVE PAIRS

In this section we show how we realize the analysis of one and two curves. As presented in [9] this immediately leads to the predicates needed by a modified Bentley-Ottmann sweep [2] in the revised formulation from LEDA [17, ch. 10.7] and [5]. Let $f, g \in F$ be curves in our arrangement. We take a *y-per-x view* and consider how the arcs of f and g evolve along the vertical line $x = x_0$ when we move x_0 along the x-axis. From the Implicit Function Theorem it follows that between intersection points of f and f_y the arcs of f evolve smoothly.

Let us start with the analysis of one curve $f \in F$. We compute the resultant $R = \mathrm{res}(f, f_y, y)$ and call its real roots *f-critical points*. Let $\mathcal{C} := \{x_i \mid 1 \leq i \leq m\}$ be the set of all f-critical points. For every f-critical point we determine a rational interval representation. In case we know the x-coordinate as a one- or two-root number due to Theorem 3, we additionally represent it explicitly in this form. The interval representations allow us to compare and order the f-critical points, wlog. $x_1 < x_2 < \cdots < x_m$. Let $\mathcal{J} := \{(x_i, x_{i+1}) =: I_i \mid 1 \leq i < m\} \cup \{(-\infty, x_1) =: I_0, (x_m, \infty) =: I_m\}$ be the set of *intervals between f-critical points*. Using the rational endpoints of the isolating intervals it is easy to find a rational number $r_i \in I_i$ for every $I_i \in \mathcal{J}$.

DEFINITION 6. *We call a curve f* generic *if it has no vertical flexes and no covertical intersections with f_y. Let f be generic.*

1. *We define a function $A_f : \mathcal{J} \to \mathbb{N}$. For an interval $I_i \in \mathcal{J}$ it computes the number of arcs of f over I_i.*
2. *We define a second function $B_f : \mathcal{C} \to \mathbb{N}^4$. For an f-critical point $x_i \in \mathcal{C}$ it outputs (n_b, n_l, n_r, n_a) with n_b the number of arcs below the one-curve event at $x = x_i$, n_l and n_r the number of arcs of f that go from the left into and to right out of the event point, and n_a the number of arcs of f above the event point.*

LEMMA 7. *The functions A_f and B_f are well-defined and the outcome of the computations for every f-critical point and every interval between f-critical points completely determines the topology of f.*

PROOF. If f is generic, all intersections of f and f_y are one-curve events (Lemma 1) and cause different f-critical points. Between two f-critical points the number of branches of f does not change. □

The functions A_f and B_f are defined only for a generic f. We next show how to test whether a curve $f \in F$ is generic. If this fails due to an unlucky choice of the coordinate system, we randomly shear the set of input quadrics and start the computation right from the beginning. Afterwards we show how to implement the functions B_f and A_f for the f-critical points and the intervals between.

THEOREM 8. *The silhouettecurve is always generic. For a cutcurve we can test whether it is generic.*

PROOF. Let f be the silhouettecurve. It is of degree at most 2. For any x_0 the polynomials $f(x_0, y)$ and $f_y(x_0, y)$ have at most one common root.

Next let f be a cutcurve. We first use the results behind Theorem 3 and try to split $R_{\geq 2}$ into quadratic polynomials. If this fails, f is not generic. Otherwise we know every root x_i of $R_{\geq 2}$ as a one- or two-root number. We still have to make sure that no x_i is the x-coordinate of a vertical flex or of two covertical one-curve events. To test the latter we compute whether $f(x_i, y)$ and $f_y(x_i, y)$ have exactly one common root y_0. In this case also y_0 is a one- or two-root number. Finally we test $f_{yy}(x_0, y_0) \neq 0$ to be sure that the point (x_0, y_0) is an event point and not a vertical flex. □

LEMMA 9. *Let $f \in F$ be generic, I_i an interval between f-critical points and $r_i \in I_i$ the precomputed rational number in I_i. Then $A_f(I_i) = \mid \{\beta \in \mathbb{R} \mid f(r_i, \beta) = 0\} \mid$.*

LEMMA 10. *Let f be the silhouettecurve, x_i an f-critical point, and I_{i-1} and I_i the two neighbored intervals to the left and to the right of x_i, respectively. Then $B_f(x_i) = (0, A_f(I_{i-1}), A_f(I_i), 0)$.*

THEOREM 11. *For a generic cutcurve f and an f-critical point x_i we can compute $B_f(x_i)$.*

PROOF. If x_i is a multiple root of R, it is the x-coordinate of a singular point and we know it explicitly as a one- or two-root number. We know that $f(x_i, y)$ has exactly one multiple root y_i. We count the numbers n_b and n_a of simple roots below and above y_i. Then $B_f(x_i) = (n_b, A_f(I_{i-1}) - n_b - n_a, A_f(I_i) - n_b - n_a, n_a)$.

Next assume that x_i is a simple root of R with isolating interval $[a, b]$. Then x_i is the x-coordinate of an x-extreme point. It is $|A_f(I_{i-1}) - A_f(I_i)| = 2$ and we assume wlog. $A_f(I_i) - A_f(I_{i-1}) = 2$ (for a right-extreme point just exchange the left and right side). In general x_i is not expressible as a one- or two-root number. Instead of computing directly on the line $x = x_i$ we do all the computations along the line $x = a$. If $A_f(I_{i-1}) = 0$, then $B_f(x_i) = (0, 0, 2, 0)$. Otherwise refine the interval $[a, b]$ until it contains no one-curve event of f_y. Compute interval representations for the real roots of $f(a, y)$ and $f_y(a, y)$ and sort them in ascending order. Let $\gamma_1 < \cdots < \gamma_k$ be the real roots of $f_y(a, y)$. By case distinction one can show $n_b = |\{\beta \mid f(a, \beta) = 0 \wedge \beta < \gamma_{\lceil k/2 \rceil}\}|$ and $n_a = |\{\beta \mid f(a, \beta) = 0 \wedge \beta > \gamma_{\lceil k/2 \rceil}\}|$ resulting in $B_f(x_i) = (n_b, 0, 2, n_a)$. □

Next we consider the analysis of two curves $f, g \in F$. Again we want to compute the order of the curves of f and g along every vertical line $x = x_0$. This order does only change at the f- and g-critical points and at the x-coordinates of the two-curve events. The latter are the real roots of $\mathrm{res}(f, g, y)$. We call all these points (f, g)-*critical*. Let $\mathcal{D}$ be the set of (f, g)-critical points and $\mathcal{K}$ be the set of open intervals in-between. We can again sort them: $K_0, x_1, K_1, x_1, \ldots, K_{m-1}, x_m, K_m$ with $x_i \in \mathcal{D}, K_i \in \mathcal{K}$. We compute a rational number $s_i \in K_i$ for all $0 \le i \le m$. Let in the following $R_f := \mathrm{res}(f, f_y, y)$, $R_g := \mathrm{res}(g, g_y, y)$, and $R_{f,g} := \mathrm{res}(f, g, y)$.

DEFINITION 12. *We call a pair of curves $\{f, g\}$* generic *if f and g are generic, there are no distinct covertical one- or two-curve events and no extreme point of f or g coincides with a two-curve event. Let $\{f, g\}$ be generic.*

1. *We define a function $C_{f,g} : \mathcal{K} \to \cup_{0 \le j \le 8}\{f, g\}^j$. For an interval $K_i \in \mathcal{K}$ it computes the sequence of arcs of f and g over K_i.*
2. *We define a function $D_{f,g} : \mathcal{D} \to \cup_{0 \le j \le 7}\{f, g, \star\}^j$. For an (f, g)-critical point $x_i \in \mathcal{D}$ it outputs the order of the arcs of f and g along the line $x = x_i$. This means more mathematically the sequence of polynomials associated with the ordered sequence of real roots of $f(x_i, y)$ and $g(x_i, y)$ (without multiplicities). The entry equals $\star$ if $f(x_i, y)$ and $g(x_i, y)$ share a common root.*

LEMMA 13. *The functions $C_{f,g}$ and $D_{f,g}$ are well-defined. Exactly one entry in the outcome of $D_{f,g}$ is of the form $\star$. The outcome of the computations of A_f, B_f, $C_{f,g}$, and $D_{f,g}$ completely determines the topology of the arrangement of f and g.*

The functions $C_{f,g}$ and $D_{f,g}$ are defined only for a generic pair $\{f, g\}$. If they are not generic this is again only caused by a bad choice of the coordinate system and can be removed by a shear.

THEOREM 14. *For every pair of curves $f, g \in F$ we can test whether it is generic.*

PROOF.

- We already know how to test coverticality of one-curve events of f (of g).
- If R_f and R_g have a common root this should only be caused by two singularities having the same coordinates. Both are explicitly known and therefore easily comparable.
- If R_f and $R_{f,g}$ have a common root this should only be due to an intersection of g at a singular point (x_0, y_0) of f. We explicitly test $g(x_0, y_0) = 0$. Analogously for R_g and $R_{f,g}$.
- No two different two-curve events should be covertical. This can be tested using the first subresultant (sometimes also called first principal subresultant coefficient) of f and g. □

LEMMA 15. *Let $f, g \in F$ be generic, K_i an interval between (f, g)-critical points and $s_i \in K_i$ the precomputed rational number in K_i. Then $C_{f,g}(K_i)$ can be computed by sorting the real roots of $f(s_i, y)$ and $g(s_i, y)$.*

THEOREM 16. *For two generic curves $f, g \in F$ and an (f, g)-critical point x_i we can compute $D_{f,g}(x_i)$.*

PROOF. For this proof we define a function

$$E_{u,v} : \cup_{0 \le j \le 8}\{f, g\}^j \to \cup_{0 \le j \le 7}\{f, g, \star\}^j.$$

For a sequence $(h_1, \ldots, h_j)$ of f's and g's it replaces the entries $h_u \ldots h_v$ by one entry $\star$.

If $R_f(x_i) = 0$ and $R_g(x_i) \neq 0 \neq R_{f,g}(x_i)$, we just have a one-curve event of f at $x = x_0$. It is obvious that the results of $B_f(x_i)$ and of $C_{f,g}(K_{i-1})$ can be combined to compute $D_{f,g}(x_i)$.

Next consider the case $R_{f,g}(x_i) = 0$ but $R_f(x_i) \neq 0 \neq R_g(x_i)$. If the multiplicity of x_i as a root of $R_{f,g}$ is odd, the sequences $C_{f,g}(K_{i-1})$ and $C_{f,g}(K_i)$ differ exactly by one flip of f and g at positions u and $u + 1$. We have $D_{f,g}(x_i) = E_{u,u+1}(C_{f,g}(K_{i-1}))$. If the multiplicity of x_i is 2, we use Theorem 4 and additionally consider the Jacobi curve J. As before we compute the indices n_f and n_g of the branches of f and of g that intersects J transversally, respectively (we omit some technical details about precomputations for J). Then $D_{f,g}(x_i) = E_{n_f+n_g-1,n_f+n_g}(C_{f,g}(K_{i-1}))$. In all other cases, due to Theorems 4 and 5, we explicitly know x_i as a one-root number. We compute $D_{f,g}(x_i)$ as the sequence of roots of $f(x_i, y)$ and $g(x_i, y)$.

It remains to discuss the situations $R_f(x_i) = R_{f,g}(x_i) = 0$. If additionally $R_g(x_i) = 0$, we know that the singular point of f at $x = x_i$ and the singular point of g at $x = x_i$ are the ones that are involved in the intersection. Combining the results of $B_f(x_i)$, $B_g(x_i)$ and of $C_{f,g}(K_{i-1})$ leads to the desired output. Now assume $R_g(x_i) \neq 0$. We have a singular point of f at $x = x_i$ so its y-coordinate y_i is explicitly known as a one- or two-root number. Let y_i be the k-th root of $g(x_i, y)$ and let $B_f(x_i) = (n_b, n_l, n_r, n_a)$. Then $D_{f,g}(x_i) = E_{n_b+k,n_b+n_l+k}(C_{f,g}(K_{i-1}))$. □

6. REGAINING THE SPATIAL INFORMATION

As shown in [9] the analysis of one curve can be used to split each curve $f \in F$ at its one-curve events into maximally connected x-monotone segments and to represent them. The predicates for the modified Bentley-Ottmann sweep work on the x-monotone segments and use the analysis of curve pairs. So we are able to compute the planar map of the arrangement of curves in F, but this is not precisely the result we are interested in. Instead we want two planar maps: one for the upper part including the silhouette and one for the lower part including the silhouette of the underlying quadric p_1. It remains to split the curves in F into open x-monotone segments such that each segment belongs to the upper or to the lower part or to both parts (for more details consider [3]). This gives us two sets of segments on which we then run the sweep algorithm separately.

DEFINITION 17. *We call a maximally connected open part of the silhouettecurve* sweepable *if it does not contain a one-curve event in its interior. For a cutcurve we additionally require that it does not contain an intersection with the silhouettecurve.*

THEOREM 18. *All curves $f \in F$ can be cut into sweepable segments. The interior of each sweepable segment of a cutcurve belongs completely to the lower or to the upper part of p_1 or to both parts and the assignment can be computed.*

PROOF. Splitting the curves into sweepable segments is easy due to our results in the previous section. Also the second statement is easy to see: A connected segment of a spatial intersection curve running on both parts of p_1 intersects in its interior the silhouette and so do its projection and the silhouettecurve. We focus on the assignment. Let s be a sweepable segment of a cutcurve f which results from projecting the spatial intersection curve of p_1 and p_i. If s is of length zero, that means $s = (x_0, y_0)$ is an isolated point, we know its coordinates explicitly as one- or two-root numbers. We compute the real roots $z_1 \leq z_2$ of $p_1(x_0, y_0, z)$. If $z_1 = z_2$, the point matches to a point on the silhouette. Otherwise, if $p_i(x_0, y_0, z_1) = 0$ the point lies on the lower part of p_1, if $p_i(x_0, y_0, z_2) = 0$ the point lies on its upper part.

Next consider the case that s has positive length. If we explicitly know a point (x_0, y_0) with one-root coordinates on s we proceed as described before. Otherwise let x_0 be a rational number in the open x-range of s and let i be the arc-number of s with respect to f. Compute an interval representation $[y_{\text{low}}, y_{\text{high}}]$ of the i-th root of $f(x_0, y)$. Refine this interval until it contains no intersection with the silhouettecurve. We shoot two rays $r_1 := f(x_0, y_{\text{low}}, z)$, $r_2 := f(x_0, y_{\text{high}}, z) \in \mathbb{Q}[z]$ in z-direction and compute the sequence of intersections of p_1 and p_i along r_1 and along r_2. Comparing these sequences determines whether the lower or upper part is involved in a flip and therefore whether s belongs to the lower or upper part. □

Now we can compute the planar maps for the upper and the lower part of p_1. A vertex v is uniquely represented via $v = (x_0, f, k, p_1, j)$. Here x_0 is the x-coordinate, f is the underlying planar curve, k is the arc number indicating that the y-coordinate is the k-th real root of $f(x_0, y)$, p_1 is the underlying quadric and finally $j = 1, 2$ determines the z-coordinate, namely whether v is a vertex on the lower ($j = 1$) or upper ($j = 2$) part of p_1. For an application, for example for performing boolean operations on quadrics, it can be that we have to compute the planar maps on different surfaces and to test different vertices for equality. This poses no problem. Let $v = (x_0, f, k, p, j)$ and $\tilde{v} = (\tilde{x}_0, \tilde{f}, \tilde{k}, \tilde{p}, \tilde{j})$ be two vertices. We first compare x and $\tilde{x}$. If they are equal, applying the analysis of two curves from Section 5 to f and $\tilde{f}$ decides whether also the y-coordinates are equal. To compare the z-coordinates we consider the cutcurve $g = \text{res}(p, \tilde{p}, z)$, compute the sweepable segment of g on which the (x, y)-coordinate of the vertices lie on (if it exists), assign the segment to the respective parts of p and $\tilde{p}$ and test whether the parts equal the ones defined by j and $\tilde{j}$. Remark that in an actual implementation one would widely use caching to avoid computing cutcurves and the analysis of them several times.

7. IMPLEMENTATION & BENCHMARKS

We have implemented all presented objects and algorithms within the EXACUS project at the Max-Planck-Institut für Informatik in Saarbrücken, Germany. EXACUS contains Libraries for Efficient and Exact Algorithms for Curves and Surfaces that are available under an open-source license, see [4] for the details of its content, design, and implementation. The libraries follow the generic programming paradigm [1] and cover distinct parts of curved computational geometry, for example mathematical foundations and arrangement of conics [5].

Our work contributes to the QUADRIX library with 15.500 lines of commented code, whereof about 2500 lines are for the representation of quadrics. About one third is needed for projected curves and another third for projected curve pairs and their interface to the generic algebraic curves and surfaces (GAPS) in EXACUS. The remainder contributes mostly to the graphical output of the projection and the shearing of the coordinate system.

The generic implementation allows to exchange important pieces of implementation, such as number types and the algorithm from SWEEPX or from CGAL used to compute the arrangement.

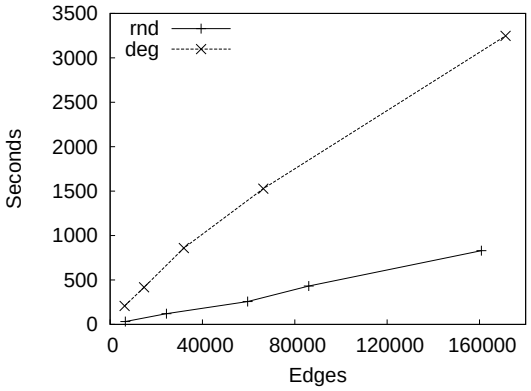
We want to prove that our projection method is practical, i.e., efficient, for computing arrangements. We have not analyzed the worst case in the bit-complexity model (and it is clear in the real RAM model), since we argue that our algorithms are adaptive in the bit-complexity and a worst-case analysis would not be representative, see [9] for details of this argument. Instead we show the efficiency of our approach with the following experiments.

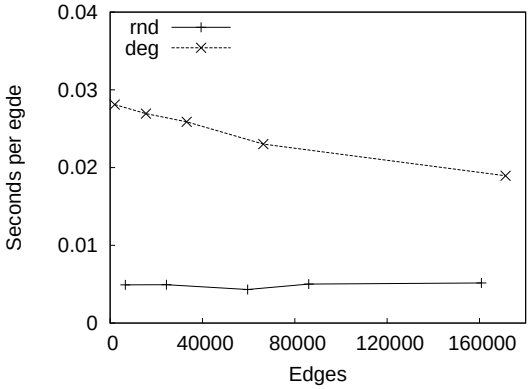
We always compute the two planar maps on the first quadric for different sets of quadrics based on the SWEEPX sweep implementation and LEDA 4.4.1 number types. The experiments were measured on a Pentium(R) M processor 1.7 GHz with 1024 KB cache under Linux and the GNU C++ compiler v3.3.3 with optimizations (`-O2`) and disabled assertions (`-DNDEBUG`).

In our experiments we use interpolated quadrics. We need nine constraints to define a quadric uniquely. In the generic case, these are nine rational interpolation points with a coordinate range of $[-99, 99]$. In the degenerate case, we prescribe values for partial derivatives and higher-order derivatives, usually at one of the interpolation points, to enforce tangential or singular intersections.

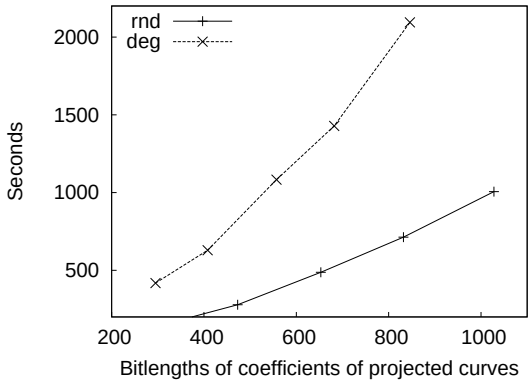
The first test sets contain random generic (rnd) and degenerate (deg) arrangements of quadrics whose coefficients need 89 bits and whose projected intersection curves need coefficients of 329 bits. The degenerate sets contain different kinds of degeneracies with decreasing probabilities (tangential intersections are more often than singular intersections). Arrangement computation is sensitive to the output size, therefore we report the runtime per computed edge: We need about 7 milliseconds per edge in the generic cases and about 25 milliseconds in the degenerate cases. Additionally, we observe that the average runtime per edge decreases with increasing number of edges, which is due to higher cost for analyzing the linear number of one-curve events compared to the quadratic number of two-curve events.

kind	nodes	edges	bits	time
rnd_25	1972	6574	79 (267)	32.3
rnd_50	6656	24410	77 (273)	120.3
rnd_75	16136	59552	80 (276)	257.3
rnd_100	22674	86056	79 (275)	432.0
rnd_150	42270	160810	78 (255)	830.0
deg_25	1969	6266	89 (308)	176.2
deg_50	4382	14710	83 (295)	417.9
deg_75	9043	31948	86 (286)	858.9
deg_100	17757	66350	89 (289)	1527.3
deg_150	45205	171314	88 (329)	3247.2





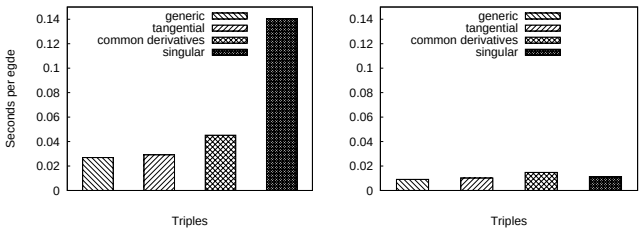
To analyze *coefficient growth*, we picked the instances with 50 quadrics, scaled each interpolation point p by a factor $s \in \{10, 100, 1\,000, 10\,000\}$, and perturbed it to $sp + \varepsilon$ with an offset vector ε chosen randomly from $\{-5, \ldots, 5\}^3$. This preserves degeneracies and most of the time the combinatorial structure of the arrangement. We can see that while the bit length grows linearly the runtime increase quadratically.



To measure the handling of degeneracies, we constructed four benchmarks, each consisting of twenty data sets containing three quadrics, let us call them triples in the following, that have the following properties (unconstrained degrees of freedom are fixed with additional interpolation points that are distinct for the triple).

We made significant progress in the running times during the last months due to changes in the representation of root numbers. To illustrate this we present two runs of our benchmark instances. The data of the old run in December 2004 are presented in the left plot and the running times of the newest implementation in March 2005 are shown in the right plot.

In the first benchmark, triples are generic in the sense that they share eight interpolation points. In the second benchmark, triples share four interpolation points with equal tangent conditions at those interpolation points. The runtime graph illustrates the additional cost incurred by the Jacobi curve (Theorem 4) in this case. In the third benchmark, triples share two interpolation points and share also the first three derivatives at these interpolation points. Such intersection points with high multiplicities force us to use exact and expensive arithmetic. In the fourth benchmark, two quadrics share a common spatial singular point through which the third quadric passes. The expensive costs associated to the analyzes of singular points vanished with the new implementation. But exact arithmetic for two-curve analysis can still be seen to be more costly than handling generic cases.



We use extensive caching: Every curve/curve pair exists only once and all algebraic data is stored after its first computation. This has two impacts: 1) The first planar map of a quadric is always more expensive than the second (150 random quadrics: 822 vs. 51 seconds). 2) The first quadric intersection map is more costly than later ones, in particular the last quadric intersection map obtains all information about cutcurves from the cache.

8. FUTURE WORK

In the future we want to develop floating point filters, especially to improve the running time needed for the analyzes of one-curve and two-curve event points. Another goal is to provide a system that can compute the full three-dimensional arrangement of quadrics based on the planar maps on the quadric surfaces. This gives us the possibility to convert a Constructive Solid Geometry input description to a Boundary-Representation and finally to perform complete, exact and efficient boolean operations on solids bounded by quadratic surfaces.

9. REFERENCES

[1] M. H. Austern. *Generic Programming and the STL: Using and Extending the C++ Standard Template Library*. Addison-Wesley, 1998.

[2] J. L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979.

[3] E. Berberich. *Exact Arrangements of Quadric Intersection Curves*. Universität des Saarlandes, Saarbrücken, 2004. Master Thesis.

[4] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, L. Kettner, K. Mehlhorn, J. Reichel, S. Schmitt, E. Schömer, D. Weber, and N. Wolpert. EXACUS—efficient and exact algorithms for curves and surfaces. Technical Report ECG-TR-361200-02, 2004.

[5] 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.

[6] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proc. 2nd GI Conf. on Automata Theory and Formal Languages*, volume 6, pages 134–183. Lecture Notes in Computer Science, Springer, Berlin, 1975.

[7] T. Culver, J. Keyser, M. Foskey, , S. Krishnan, and D. Manocha. Esolid - a system for exact boundary evaluation. *Computer-Aided Design (Special Issue on Solid Modeling)*, 36, 2003.

[8] L. Dupont, D. Lazard, S. Lazard, and S. Petitjean. A new algorithm for the robust intersection of two general quadrics. In *Proc. 19th Annu. ACM Sympos. Comput. Geom.*, pages 246–255, 2003.

[9] A. Eigenwillig, L. Kettner, E. Schömer, and N. Wolpert. Complete, exact, and efficient computations with cubic curves. In *Proc. 20th Annu. ACM Symp. Comput. Geom.*, pages 409–418, 2004.

[10] I. Emiris, A. Kakargias, S. Pion, M. Teillaud, and E. Tsigaridas. Towards an open curved kernel. In *Proc. 20th Annu. ACM Symp. Comput. Geom.*, pages 438–446, 2004.

[11] Exacus - Libraries for Efficent and Exact Algorithms for Curves and Surfaces. http://www.mpi-sb.mpg.de/projects/EXACUS/.

[12] 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.

[13] 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.

[14] S. Lazard, L. M. Penaranda, and S. Petitjean. Intersecting quadrics: An efficient and exact implementation. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 419–428, 2004.

[15] J. Levin. A parametric algorithm for drawing pictures of solid objects composed of quadric surfaces. *Commun. ACM*, 19(10):555–563, Oct. 1976.

[16] J. Levin. Mathematical models for determining the intersections of quadric surfaces. *Comput. Graph. Image Process.*, 11:73–87, 1979.

[17] K. Mehlhorn and S. Näher. *LEDA – A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[18] B. Mourrain, J.-P. Técourt, and M. Teillaud. Sweeping an arrangement of quadrics in 3d. In *Proc. 19th European Workshop on Computational Geometry*, pages 31–34, 2003.

[19] F. P. Preparata and M. I. Shamos. *Computational geometry and introduction*. Springer-Verlag, New York, 1985.

[20] E. Schömer and N. Wolpert. An exact and efficient approach for computing a cell in an arrangement of quadrics. CGTA (Special Issue on Robust Geometric Algorithms and their Implementations), submitted, 2004.

[21] R. Wein. High level filtering for arrangements of conic arcs. In *ESA 2002, Lecture Notes in Computer Science*, pages 884–895, 2002.

[22] N. Wolpert. *An Exact and Efficient Approach for Computing a Cell in an Arrangement of Quadrics*. Universität des Saarlandes, 2002. Ph.D. Thesis.

[23] N. Wolpert. Jacobi curves: Computing the exact topology of arrangements of non-singular algebraic curves. In *ESA 2003, Lecture Notes in Computer Science*, pages 532–543, 2003.