

A Complete, Exact and Efficient Implementation for Computing the Edge-Adjacency Graph of an Arrangement of Quadrics

Michael Hemmer

Max Planck Institut für Informatik, Saarbrücken, Germany

Laurent Dupont

Nancy Université, LORIA, France

Sylvain Petitjean

INRIA Nancy - Grand Est, LORIA, France

Elmar Schömer

Johannes Gutenberg-Universität, Institut für Informatik, Mainz, Germany

Abstract

We present a complete, exact and efficient implementation to compute the edge-adjacency graph of an arrangement of quadrics, i.e. surfaces of algebraic degree 2. This is a major step towards the computation of the full 3D arrangement. We enhanced an implementation for an exact parameterization of the intersection curves of two quadrics, such that we can compute the exact parameter value for intersection points and from that the edge-adjacency graph of the arrangement. Our implementation is *complete* in the sense that it can handle all kinds of inputs including all degenerate ones, i.e. singularities or tangential intersection points. It is *exact* in that it always computes the mathematically correct result. It is *efficient* measured in running times, i.e. it compares favorably to the only previous implementation.

Key words: Arrangement, intersection of surfaces, quadrics, pencils of quadrics, curve parameterization.

1. Introduction

Computer Aided Design (CAD) systems dealing with curved objects have been available since the 60's. However, all these systems still suffer from approximation and rounding errors due to the use of fast but inexact floating point arithmetic. Without an exact representation of the resulting curves and without exact arithmetic it is difficult, or just impossible, to detect degenerate configurations, which are frequent in the design of geometric objects. As a consequence, all these systems are neither exact nor complete. On the other hand, computer algebra introduced very general methods, such as cylindrical algebraic decomposition presented by Collins [13]. These methods are exact and complete in principle but cause unacceptable runtimes. Our intention is to close this gap, and join the three goals exactness, completeness and efficiency.

As a first step, we aim for the computation of the three-dimensional arrangement induced by a given set of quadrics. *Quadrics* are surfaces of algebraic degree 2, that is, they are defined as the zero set of a trivariate polynomial of degree two. Quadrics cover a couple of common objects such as spheres, ellipsoids, cones, cylinders, hyperboloids, planes, and double planes. They play an important role in solid modeling and in the design of mechanical parts. The *arrangement* is the decomposition of \mathbb{R}^3 by the surfaces into cells of dimension 0 (vertices), 1 (edges), 2 (faces) and 3 (volumes) [25]. Arrangements are ubiquitous in computational geometry and can be applied to many problems, for instance, a recurring and important task in solid modeling is to perform Boolean operations on curved surfaces. However, the cardinal problem behind this task is the computation of the underlying 3D arrangement of the involved surfaces.

Towards the computation of the 3D arrangement we achieved a major milestone, namely the computation of the edge-adjacency graph, that is, we compute all vertices of the arrangement and their connectivity. As input we can deal with arbitrary rational quadrics, that is, quadrics defined by polynomials in $\mathbb{Q}[x, y, z]$. The implementation is *complete* in the sense that it can handle all cases, in particular, it can handle configurations that involve tangential or singular intersections. It is *exact* in the sense that it always computes the mathematically correct result. It is *efficient* measured in running times, that is, it compares favorably to the only previous implementation by Berberich et al. [6]. The prototype is implemented within the framework of the EXACUS [4] project.

1.1. Previous Work

Extending well-known computational geometry algorithms dealing with linear primitives to exact, complete and efficient algorithms for curved objects has received a lot of attention during the last years, in particular, the exact computation of planar arrangements. Wein [37] as well as Berberich et al. [5] extended CGAL's planar arrangement

* Project co-funded by the European Commission within FP6 (2002–2006) under contract No. IST-006413

Email addresses: hemmer@mpi-inf.mpg.de (Michael Hemmer), laurent.dupont@loria.fr (Laurent Dupont), sylvain.petitjean@inria.fr (Sylvain Petitjean), schoemer@informatik.uni-mainz.de (Elmar Schömer).

URLs: <http://www.mpi-inf.mpg.de/~hemmer/> (Michael Hemmer), <http://www.loria.fr/~dupont/> (Laurent Dupont), <http://www.loria.fr/~petitjea/> (Sylvain Petitjean), <http://www.staff.uni-mainz.de/schoemer/> (Elmar Schömer).

package [22] such that it supports planar maps of conics and conic arcs. Eigenwillig and Kerber [17] extended this framework to arbitrary algebraic curves and segments.

With respect to algorithms and systems dealing with curved objects in three dimensions, the early work of McCallum and Collins [32] basically computes the edge-adjacency graph for arbitrary sets of surfaces, though it has never been completely implemented as far as we know. The ESOLID system by J. Keyser et al. [14] provides CSG to B-rep conversion for low-degree curved solids, but the algorithms assume general position, that is, the system may fail or crash on degenerate inputs.

To the best of our knowledge, there are only two other approaches that aim for an exact and complete implementation for the arrangement of quadrics.

- The first approach, by Mourrain et al. [33], is based on a spatial sweep over the arrangement of quadric surfaces. It defines a pseudo trapezoidal decomposition in the sweep plane and studies the evolution of this decomposition during the sweep. However, we consider this as a disadvantage since the algebraic degree of the needed predicates can be very high. The approach has not been implemented so far.
- The second approach by Berberich et al. [6] is based on Wolpert [35] and Eigenwillig et al. [20]. For a given set of quadric surfaces $\mathcal{Q} = \{Q_1, \dots, Q_n\}$, it computes the planar arrangement on the surface of Q_1 that is induced by all intersection curves $Q_1 \cap Q_i$, $2 \leq i \leq n$. To do so, it projects all intersection curves onto the xy -plane and computes their planar arrangement. Thereafter, this arrangement is lifted back onto the surface of Q_1 . Since we will compare our own approach to this *projection approach*, it is again discussed in Section 8.1. As our approach, this is also implemented in the context of the EXACUS project.

Up to now, none of these approaches has finally led to a complete implementation that computes the full 3D arrangement.

Recently, Berberich et al. [7,8] provided a framework for arbitrary algebraic surfaces. The approach is also based on the computation of the projected intersection curves and a subsequent lifting of the cells of the planar arrangement. However, similar to Collins' cylindrical algebraic decomposition [13] this results in a artificially large complexity of the resulting data structure compared to the actual arrangement. Hence, this tool may only be used for a very small number of surfaces or as a basic module in a more sophisticated algorithm.

1.2. Outline

Given a set \mathcal{Q} of quadric surfaces we are interested in the computation of the edge-adjacency graph of the arrangement induced by \mathcal{Q} . Hence, we are first of all interested in the intersection curves of two quadrics and the intersection points that are induced by the intersection with a third quadric.

Generically the intersection of two quadrics is a smooth quartic, a curve of algebraic degree 4. However, in some cases the intersection can be a nodal quartic, cuspidal quartic, or decompose into *algebraic components* of lower degree, that is, cubics, conics or lines. It may even consist of isolated points as we are

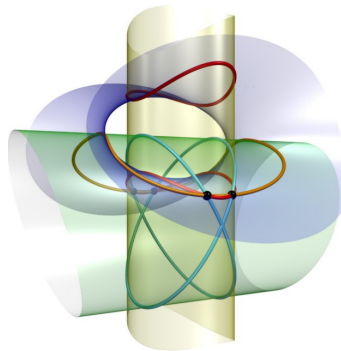


Figure 1. Quadric Arrangement.

interested in the real part of the intersection. In any case, the algebraic degrees of all components sum up to 4. An exhaustive classification can be found in Dupont et al. [16].

Figure 1 illustrates an arrangement of two cylinders and a hyperboloid of one sheet. The four common points are depicted in black. The intersections with the hyperboloid are general quartic curves, one of them consisting of two connected components. The intersection of the two cylinders decomposes into two conics.

The core of [16] is a complete algorithm providing an exact parameterization of all algebraic components that appear in the intersection of two quadrics. A brief discussion of the algorithm and the relevant details of the parameterization are provided in Section 2. Given this exact parameterization, the idea of our approach is to represent the vertices of the arrangement by their exact parameter values with respect to the algebraic components they lie on. This has the advantage that we can easily determine the adjacency of points by sorting them along the curves. However, there are some stumbling blocks as well:

- In some cases it is not possible to avoid the construction of the same algebraic component twice. Moreover, the algorithm by Dupont et al. cannot guarantee a unique parameterization of algebraic components. This entails two problems. First, we have to identify equal algebraic components before we start to intersect them with the other quadrics. Second, before we delete a redundant parameterization of an algebraic component, we have to ‘rescue’ points that are already defined with respect to that parameterization.¹ We examine this problem in Section 3.
- In all but the smooth quartic case the parameterization is given in terms of rational functions. In these cases the intersection with a third quadric is straightforward. The parameterization of a smooth quartic is not rational as it involves a square root of a polynomial. Hence this case requires a separate and more sophisticated consideration. This is discussed in Section 4.
- For each intersection point we obtain several representations, one representation for each algebraic component the point lies on. Since each intersection point should be represented by one object only, it is essential to identify and join the different representations of the same point. We found a very efficient method to match the intersection points that result from the intersection of three quadrics all at once. This is presented in Section 5.
- Though the polynomial degree of the parameterizations is minimal, working with such parameterizations imposes to deal with an increase in complexity at the level of their coefficients. This refers to both the bit size as well as the algebraic complexity of the coefficients. Therefore, the approach demands an elaborate application of filtering techniques throughout the algorithm. Now, the approach compares favorably to other existing methods, as is documented in Section 8.

Draft: Overall Algorithm.

We will now give a brief version of the overall algorithm, a more detailed version is given in Section 6.

Given a set \mathcal{Q} of quadric surfaces, defined by rational coefficients of any size, our algorithm computes the edge-adjacency graph $\mathcal{G}(\mathcal{Q})$ of the arrangement $\mathcal{A}(\mathcal{Q})$, that is, it computes all vertices and their connectivity along the edges of $\mathcal{A}(\mathcal{Q})$.

For a given set \mathcal{Q} of rational quadrics do:

¹ Note that these points are just points that are induced by the intersection of two quadrics, for instance, the intersection of two different algebraic components of the intersection of two quadrics.

0. Remove duplicates from \mathcal{Q} and ensure that all quadrics are coprime. If quadrics are not coprime, replace them by their common factor and the according remainders, that is, rational planes. Thus, the set \mathcal{Q} may also contain rational planes.²
1. Construct all lower dimensional features induced by one quadric, e.g. the singular point of a cone.
2. For each pair of quadrics, compute all algebraic components of their intersection using the approach by Dupont et al. [16]. Ensure for each component that it is new, otherwise unify it with the existing one.
3. For each triple of quadrics compute all the representations of their intersection points. Use the algorithm from Section 5 to match these representations. In general, this results in three representations for each intersection point.
4. For each component, sort all points on that component. Note that in this step we may detect further equalities indicating a common intersection point of four or more quadrics. Output the edge-adjacency graph.

Since the vertices are sorted along their algebraic components and since every vertex knows all the algebraic components it lies on, it is easy to compute the full edge-adjacency graph connecting all vertices of the arrangement with their neighbors. The next step is to explore the local neighborhood of each vertex. See Section 9 on further work for more details.

2. Intersecting Two Quadrics

In this section, we recall definitions and the main results on the exact parameterization of the intersection of quadrics as presented in Dupont et al. [16].

2.1. Definitions and Notations

We work in real projective space $\mathbb{P}^3 = \mathbb{P}^3(\mathbb{R})$, the set of quadruplets

$$\mathbf{X} = (x_0, x_1, x_2, x_3) \neq (0, 0, 0, 0),$$

with the equivalence relation $(x_0, x_1, x_2, x_3) \sim (\lambda x_0, \lambda x_1, \lambda x_2, \lambda x_3)$ for all $\lambda \neq 0$. A quadric surface Q_S is defined in \mathbb{P}^3 by an implicit equation of degree 2:

$$\sum_{0 \leq i < j \leq 3} \alpha_{ij} x_i x_j = 0, \alpha_{ij} \in \mathbb{Q}.$$

Since this equation can also be expressed as $\mathbf{X}^T S \mathbf{X} = 0$, with S a 4×4 real symmetric matrix, we associate the quadric Q_S to this matrix S .

Classification by Inertia.

Since S is symmetric, it is clear that all of its eigenvalues are real. Let σ^+ and σ^- be the numbers of positive and negative eigenvalues of S , respectively. The *rank* of Q_S (and S) is the sum of σ^+ and σ^- . We define the *inertia* of Q_S (and S) as the pair

$$(\max(\sigma^+, \sigma^-), \min(\sigma^+, \sigma^-)).$$

² Throughout the rest of this paper we won't mention this special case since it is always possible to temporarily replace a rational plane by a double plane, which is a quadric.

This definition differs slightly from the usual definition of the inertia in the literature, which in general denotes the inertia of S as the pair (σ^+, σ^-) . However, the definition chosen here effectively reflects the fact that the quadratic forms Q_S and $-Q_S$ represent one and the same quadric.

Theorem 1 (Sylvester's Inertia Law). *The inertia of a quadric is invariant under real projective transformations.*

This theorem is named after the English mathematician J. J. Sylvester. A proof of this fundamental result of matrix theory can be found in [10], [24] or [28]. It essentially states that the inertia is invariant under change of basis. Thus, we identify the projective type of a quadric by the inertia of Q_S .

- **rank 4:** A quadric of inertia $(4, 0)$ is an empty quadric, that is, empty of real points. The only quadrics with a negative determinant are those of inertia $(3, 1)$. All quadrics of inertia $(2, 2)$ are doubly ruled quadrics, that is, they can be swept out by two 1-dimensional families of lines in space. This is a very important property, since this can be used to provide a very convenient parameterization, see also Section 2. Note that all quadrics with positive determinant are either ruled or empty.
- **rank 3:** A quadric of rank 3 is called a cone. The cone is said to be real if its inertia is $(2, 1)$. If the inertia is $(3, 0)$ it is an imaginary cone, with the singular point being its only real solution.
- **rank 2:** A quadric of rank 2 is a pair of planes. The pair of planes is real if the inertia is $(1, 1)$. If the inertia is $(2, 0)$ the quadric consists of two imaginary planes intersecting in a real rational line.
- **rank 1:** A quadric of inertia $(1, 0)$ is called a double plane and is necessarily real.

Pencil.

For two quadrics Q_S and Q_T , an essential notion to compute a parameterization of their intersection is the pencil $\mathcal{P}(Q_S, Q_T) = \{\lambda Q_S + \mu Q_T \mid (\lambda, \mu) \in \mathbb{P}^1(\mathbb{R})\}$. The intersection curve of two quadrics can be identified by the corresponding pencil and vice versa, that is, the intersection can be computed from any pair of distinct quadrics out of $\mathcal{P}(Q_S, Q_T)$.

A fundamental invariant of a pair of quadrics is the univariate polynomial $\det(\lambda Q_S + \mu Q_T)$, called the characteristic polynomial.

2.2. Overview of the Intersection of Two Quadrics

We give here a sketch of computation of a parameterization of the intersection of two quadrics as presented in [16]. Given a pencil \mathcal{P} the principal idea is to compute a parameterization of some quadric $Q_S \in \mathcal{P}$ and plug it into the equation of another quadric $Q_T \in \mathcal{P}$. This leads to an equation in two variables, which is solved with respect to one variable in terms of the other. Thereafter, within the parameterization of Q_S this variable is substituted by its solution(s), which results in the parameterization of the intersection curve. However, the parameterization of an arbitrary $Q_S \in \mathcal{P}$ is in general too complicated with the consequence that the bivariate equation cannot be solved explicitly.

The key result of [16] is that it is always possible to find a good quadric in \mathcal{P} such that the parameterization is simple. In particular, it is always possible to derive an explicit parameterization of the intersection curve.

Generic Case: Smooth Quartic.

Since the intersection of a smooth quartic with another quadric will be the most involved, see also Section 4, we now take a closer look at the parameterization of the smooth quartic.

In this case Dupont et al. pick a quadric Q_R of inertia $(2, 2)$ such that there is a linear transformation, with coefficients in \mathbb{Q} , sending Q_R into the canonical form $x_0^2 + x_1^2 - x_2^2 - \delta x_3^2 = 0$, with positive $\delta \in \mathbb{Q}$. In this frame Q_R can be parameterized by

$$\left(ut + vs, us - vt, ut - vs, \frac{us + vt}{\sqrt{\delta}} \right), \quad (1)$$

where the parameters $\xi = (u, v)$ and $\tau = (s, t)$ are both in \mathbb{P}^1 . Sending this back into the original frame we obtain the parameterization

$$\mathbf{X}_R(\xi, \tau) \in [\mathbb{Q}(\sqrt{\delta})[\xi, \tau]]^4, \text{ with } (\xi, \tau) \in \mathbb{P}^1 \times \mathbb{P}^1$$

of Q_R , where $[\mathbb{Q}(\sqrt{\delta})[\xi, \tau]]^4$ is the vector space of dimension 4 of bivariate polynomials with coefficients in an algebraic extension of degree 2. As Equation (1) shows, \mathbf{X}_R has the nice property that every coordinate is **bilinear** in (u, v) and (s, t) . Now \mathbf{X}_R is plugged into the implicit equation of another quadric $Q_S \in \mathcal{P}$ to obtain the **biquadratic** implicit equation $f := \mathbf{X}_R^T S \mathbf{X}_R \in \mathbb{Q}(\sqrt{\delta})[\xi, \tau]$ of the intersection curve within the parameter space of \mathbf{X}_R :

$$f(\xi, \tau) = a_2(\xi)s^2 + a_1(\xi)st + a_0(\xi)t^2, \quad (a_i)_{i=0,1,2} \in \mathbb{Q}(\sqrt{\delta})[\xi].$$

Since f is quadratic in (s, t) (but also in (u, v) because the $a_i(\xi)$ are quadratic), solving for τ yields:

$$\tau_\varepsilon(\xi) = (s_\varepsilon(\xi), t_\varepsilon(\xi)) = \left(-a_1(\xi) + \varepsilon\sqrt{\Delta(\xi)}, 2a_2(\xi) \right), \quad (2)$$

where $\varepsilon = \pm 1$ and $\Delta(\xi) = a_1^2(\xi) - 4a_0(\xi)a_2(\xi) \in \mathbb{Q}(\sqrt{\delta})[\xi]$ is of degree 4. Using this within the parameterization \mathbf{X}_R of Q_R , the final parameterization of the intersection curve of Q_S and Q_T is defined as:

$$\mathbf{X}_\varepsilon(\xi) = \mathbf{X}_R(\xi, \tau_\varepsilon(\xi)) \in [\mathbb{Q}(\sqrt{\delta})[\xi, \sqrt{\Delta}]]^4.$$

Therefore, the parameterization consists of two arcs, a positive arc $\mathbf{X}_{\varepsilon=1}$ and a negative arc $\mathbf{X}_{\varepsilon=-1}$, each defined within the domain $\mathbb{D} = \{\xi \in \mathbb{P}^1 \mid \Delta(\xi) \geq 0\}$.

Breakdown of Parameterization.

Though $\tau_\varepsilon(\xi)$ is a valid parameterization for most values of ξ there is a problem for roots of $a_2(\xi)$. Let ξ_0 be a real root of $a_2(\xi)$ and consider $\tau_\varepsilon(\xi_0)$ as it is given in Equation (2). Then:

$$\begin{aligned} \tau_\varepsilon(\xi_0) &= (-a_1(\xi_0) + \varepsilon\sqrt{\Delta(\xi_0)}, 2a_2(\xi_0)), \\ &= (-a_1(\xi_0) + \varepsilon\sqrt{a_1(\xi_0)^2}, 0), \\ &= (-a_1(\xi_0) + \varepsilon|a_1(\xi_0)|, 0), \\ &= |a_1(\xi_0)|(\varepsilon - \text{sign}(a_1(\xi_0)), 0). \end{aligned}$$

The parameterization becomes invalid at ξ_0 for $\varepsilon = \text{sign}(a_1(\xi_0))$.³ Though this is a removable discontinuity of $\tau_\varepsilon(\xi)$ it is a source of numerical instability. Therefore, we always consider the alternative parameterization of τ as well, namely

$$\tilde{\tau}_\varepsilon(\xi) = (\tilde{s}_\varepsilon, \tilde{t}_\varepsilon) = \left(2a_0(\xi), -a_1(\xi) - \varepsilon\sqrt{\Delta(\xi)} \right).$$

Note that $\tau_\varepsilon(\xi)$ and $\tilde{\tau}_\varepsilon(\xi)$ are identical up to their removable discontinuities, which are at the real roots of $a_0(\xi)$ and $a_2(\xi)$, respectively.

Of course it is essential that at least one parameterization remains valid, that is, we have to guarantee that $a_0(\xi)$ and $a_2(\xi)$ have no common root. In general $a_0(\xi)$ and $a_2(\xi)$ have no common root and nothing needs to be done. However, in the rare case where $a_0(\xi)$ and $a_2(\xi)$ have a common root ξ_0 , we can rely on the fact that $a_1(\xi_0) \neq 0$. Otherwise, this would imply a vertical line at ξ_0 , which contradicts the fact that f represents a smooth quartic. Hence, it is easy to find a new frame for τ such that $a_0(\xi)$ and $a_2(\xi)$ have no common root.

Singular Case: Rational Parameterizations.

In case $\det(\lambda Q_S + \mu Q_T)$ is not square-free, additional invariants (type – real or complex – and multiplicity of the roots, projective type of the associated quadrics, ...) are computed to give a complete case distinction along the different algebraic components the intersection curve might consist of. In all singular cases Dupont et al. [16] give a rational parameterization of each algebraic component. However, the parameterizations may be defined in an algebraic extension, an overview is given in Table 1.

Notice here that two different algebraic components of the same curve can have some intersection points. In this case these points are reported and matched to be used in the sequel of the algorithm.

component	maximal algebraic degree of extension	degree for the entire intersection
smooth quartic	2	2
nodal quartic	2	2
cuspidal quartic	1	1
cubic	1	1
conic	4 (2 × 2)	4
line	4	24
point	4 (2 × 2)	4

Table 1. Maximal algebraic extension for the different possible algebraic components.

³ This effect is even independent of ε if ξ_0 is a common root of $a_1(\xi)$ and $a_2(\xi)$.

3. Unifying Equal Algebraic Components

All intersection points are represented by the parameter values with respect to the algebraic components they lie on. Hence, it is very important to guarantee a unique parameterization of each algebraic component that appears in the data structure. Unfortunately, it is not possible to avoid unnecessary constructions of algebraic components, since a component may be defined by several pencils. For instance, a conic may be contained in the intersection of three linearly independent quadrics as it is illustrated in the figure to the right. The red conic is part of the intersection of the hyperboloid of one sheet with the horizontal cylinder as well as of the intersection of the two cylinders.

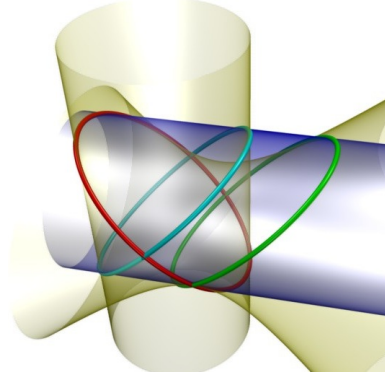


Figure 2. A common conic.

Moreover, the algorithm by Dupont et al. [16,30] cannot guarantee a unique parameterization of algebraic components by construction. This has the consequence that before we can start to intersect algebraic components with other quadrics, we have to ensure that within the data structure the same algebraic component is not represented more than once. First of all we avoid a lot of unnecessary construction using a cache. Given two quadrics Q_S and Q_T , we construct a unique representation of the pencil defined by Q_S and Q_T . This pencil is used as the key for the cache. Moreover, the cache stores the result of the intersection of the quadrics Q_S and Q_T . Note that due to the fact that a quartic is unambiguously defined by its pencil and vice versa the cache avoids reconstructions for all quartics, i.e. smooth quartics, nodal quartics and cuspidal quartics.

However, for the other algebraic component types the cache is not sufficient in order to avoid unnecessary constructions. For these cases, we proceed as follows. For a new parameterization X'_C of C we check whether there exists an old parameterization X_C of C . If this is the case, we replace the new parameterization X'_C by the old parameterization X_C . But before we can do this, we have to ‘rescue’ the points defined on X'_C and redefine them with respect to X_C . Therefore, the overall algorithm has been designed such that this is only needed in a very early stage, namely right after the construction of an algebraic component. This has the advantage that the only points defined on X'_C are the intersection points with other components in the pencil of X'_C .

As the details of the unification step are rather technical and involve a case-by-case analysis for each type of component (cubics, conics and lines), we postpone them to Appendix A.

4. Intersecting With a Third Quadric

In this section we discuss the most important step of our algorithm, namely the intersection of an algebraic component $C \subseteq Q_S \cap Q_T$ with another quadric Q_U . If $C \subset Q_U$ this should be detected and otherwise we wish to compute the exact parameter values of all real intersection points in $C \cap Q_U$ with respect to the parameterization $X_C(\xi)$ of C .

In those cases in which the parameterization is given in terms of rational functions the intersection with another quadric is straightforward. These cases are discussed all at

Algorithm 1 Let $\mathcal{C} \subseteq Q_S \cap Q_T$ be a singular quartic, cubic, conic or line and $\mathbf{X}_{\mathcal{C}}(\xi)$ its rational parameterization. Given another quadric Q_U , if $\mathcal{C} \subset Q_U$ report \mathcal{C} . Otherwise, compute the exact parameter values of all real intersection points $\mathcal{C} \cap Q_U$ with respect to $\mathbf{X}_{\mathcal{C}}(\xi)$.

- (1) $h(\xi) := \mathbf{X}_{\mathcal{C}}^T(\xi)U\mathbf{X}_{\mathcal{C}}(\xi)$
 - (2) **if** $\{ h(\xi) \equiv 0 \}$ report \mathcal{C} and return. **end if**
 - (3) $\mathcal{F} := \text{square_free_fac}(h)$ // of the form $\{(fac_0, m_0), \dots, (fac_k, m_k)\}$
 - (4) **for all** $\{(fac_i, m_i) \in \mathcal{F}\}$ **do**
 - (5) compute and report all roots of fac_i
 - (6) **end for**
-

once in Section 4.1. The remaining case, the smooth quartic, is a bit more involved due to the fact that the parameterization involves a square root of a polynomial. This case is discussed separately in Section 4.2.

Note that we do not compute the multiplicity associated to each intersection point, since it is not needed by the overall algorithm and causes some overhead in the computation. However, since the multiplicity may be of independent interest, we indicate in each case how to obtain the multiplicity as well. It will turn out that in most cases we gain the multiplicity for free.

4.1. Rational Parameterizations

For any algebraic component but the smooth quartic, we are in the comfortable situation that the parameterization is given in terms of rational functions. Moreover, the algorithm by Dupont et al. guarantees that the degree of the involved polynomials is minimal with respect to the parameterized algebraic component. Hence, the intersection with a third quadric is straightforward (cf. Algorithm 1). The parameterization $\mathbf{X}_{\mathcal{C}}(\xi)$ is defined for $\xi \in \mathbb{P}^1(\mathbb{R})$.

Given a rational parameterization $\mathbf{X}_{\mathcal{C}}(\xi)$ of an algebraic component \mathcal{C} and another Q_U we just plug $\mathbf{X}_{\mathcal{C}}(\xi)$ into the implicit equation of Q_U . We obtain a univariate polynomial

$$h(\xi) = \mathbf{X}_{\mathcal{C}}(\xi)^T U \mathbf{X}_{\mathcal{C}}(\xi).$$

If $\mathcal{C} \subset Q_U$ this is the zero polynomial. Otherwise, the degree of $h(\xi)$ is 8, 6, 4 and 2 in case of singular quartics, cubics, conics and lines, respectively. This is minimal, since the degree of $\mathbf{X}_{\mathcal{C}}(\xi)$ is minimal as well. Hence, each real root ξ_i of $h(\xi)$ and its multiplicity m_i exactly corresponds to one intersection point in $\mathcal{C} \cap Q_U$. The only minor exception is the nodal quartic with a non-isolated singular point. In this case the parameterization passes the point twice and there are two parameter values for this point. However, this is not a problem since the multiplicity is computed separately as well.

4.2. Smooth Quartic

In case of a smooth quartic $\mathcal{C}_{S \cap T} = Q_S \cap Q_T$ the situation is a bit complicated since the parameterization involves the square root of a polynomial. Therefore, the parameterization splits into two arcs, $\mathbf{X}_+(\xi)$ and $\mathbf{X}_-(\xi)$. This has the consequence that a point on $\mathcal{C}_{S \cap T}$ has to be identified by its corresponding value for ξ but also by the arc it lies on.

Recall that the parameterization of \mathcal{C} was obtained within the parameter space of a ruled quadric $Q_R \in \text{pencil}(Q_S, Q_T)$, w.l.o.g. $Q_R \neq Q_S$. Let $\mathbf{X}_R(\xi, \tau)$ be the parameterization of Q_R . In this parameter space $Q_S \cap Q_T$ is defined by the zero set of the biquadratic polynomial

$$f(\xi, \tau) = \mathbf{X}_R(\xi, \tau)^T S \mathbf{X}_R(\xi, \tau) = a_2(\xi)s^2 + a_1(\xi)st + a_0(\xi)t^2, \quad (3)$$

where $(a_i)_{i=0,1,2} \in \mathbb{Q}(\sqrt{\delta})[\xi]$ are of degree 2. Hence, the parameterization is given by

$$\tau_\varepsilon(\xi) = (s_\varepsilon, t_\varepsilon) = \left(-a_1(\xi) + \varepsilon\sqrt{\Delta(\xi)}, 2a_2(\xi) \right), \quad (4)$$

where $\varepsilon = \pm 1$ and $\Delta(\xi) = a_1(\xi)^2 - 4a_0(\xi)a_2(\xi) \in \mathbb{Q}(\sqrt{\delta})[\xi]$, see also Section 2. Subsequently, we will compute the intersection of $\mathcal{C} \cap Q_U$ within the parameter space of Q_R using the fact that

$$Q_S \cap Q_T \cap Q_U = Q_S \cap Q_R \cap Q_U = (Q_R \cap Q_S) \cap (Q_R \cap Q_U).$$

In the parameter space of Q_R the intersection $Q_R \cap Q_U$ is given by the zero set of

$$g(\xi, \tau) = \mathbf{X}_R(\xi, \tau)^T U \mathbf{X}_R(\xi, \tau) = b_2(\xi)s^2 + b_1(\xi)st + b_0(\xi)t^2, \quad (5)$$

where $(b_i)_{i=0,1,2} \in \mathbb{Q}(\sqrt{\delta})[\xi]$. Since we want to compute the parameter values of $\mathcal{C}_{S \cap T} \cap Q_U$, we are first of all interested in the ξ -coordinates of the common solutions of f and g . Hence, we use a classical resultant approach to eliminate τ . The projective Sylvester resultant of f and g with respect to τ is given by:

$$\text{res}(\xi) := \text{resultant}(f, g, \tau) = \begin{vmatrix} a_2 & a_1 & a_0 & 0 \\ 0 & a_2 & a_1 & a_0 \\ b_2 & b_1 & b_0 & 0 \\ 0 & b_2 & b_1 & b_0 \end{vmatrix} = s_{02}(\xi)^2 - s_{01}(\xi)s_{12}(\xi), \quad (6)$$

where $s_{ij} = a_i b_j - a_j b_i \in \mathbb{Q}(\sqrt{\delta})[\xi]$.

Proposition 1. Let f , g and res be defined as in Equations (3), (5) and (6) respectively. $Q_R \cap Q_S = Q_R \cap Q_U$ iff $\text{res} \equiv 0$.

Proof. First of all it is clear that $Q_R \cap Q_S = Q_R \cap Q_U$ implies $\text{res} \equiv 0$. Now, note that f and g are both biquadratic and that f is irreducible since it is representing a smooth quadric. Moreover, $\text{res} \equiv 0$ implies that f and g have a common factor of positive degree. Hence, f and g are equal up to a constant factor. \square

In general, i.e. $\mathcal{C} \not\subset Q_U$, the resultant res is a polynomial of degree 8. By Bezout's Theorem the roots of res are the ξ -coordinates of the intersection points of f and g , multiplicities counted. It remains to discard the complex intersection points and to determine the correct arc for the real intersection points. This is embodied in Theorem 2.

Theorem 2. Let f , τ_ε , g and $\text{res} \neq 0$ be defined as in Equations (3), (4), (5) and (6) respectively. And let ξ_0 denote a real root (if any) of res . Moreover, let $\tau_\varepsilon(\xi)$ be a valid parameterization for ξ_0 , that is, ξ_0 is not a root of $a_2(\xi)$.

There are 3 cases:

- (1) $\Delta(\xi_0) < 0$: ξ_0 corresponds to two complex intersection points.
- (2) $\Delta(\xi_0) = 0$: ξ_0 corresponds to a real endpoint of both arcs.

(3) $\Delta(\xi_0) > 0$:

- If $s_{12}(\xi_0) \neq 0$, then ξ_0 corresponds to one real intersection point on arc $\mathbf{X}_\varepsilon(\xi)$, with $\varepsilon = \text{sign}(s_{12}) \text{sign}(a_1 s_{12} - 2a_2 s_{02})|_{\xi_0}$.
- If $s_{12}(\xi_0) = 0$, then ξ_0 corresponds to two real points, one on each arc.

Proof. $f|_{\xi=\xi_0}(\tau)$ is a quadratic polynomial in τ and $\Delta(\xi_0)$ is its discriminant. Since this proves the first two statements consider the third. Due to the fact that $\Delta(\xi_0)$ is positive there are two possible solutions, namely $(\xi_0, \tau_{\varepsilon=+1}(\xi_0))$ and $(\xi_0, \tau_{\varepsilon=-1}(\xi_0))$. First of all, it is clear that at least one of these two possibilities is a valid solution. Now observe that $g(\xi, \tau_\varepsilon(\xi))|_{\xi_0}$ is independent of ε iff $s_{12}(\xi_0) = 0$, since $g(\xi, \tau_\varepsilon(\xi))$ can be written as:

$$\begin{aligned} g(\xi, \tau_\varepsilon(\xi)) &= b_2 s_\varepsilon^2 + b_1 s_\varepsilon t_\varepsilon + b_0 t_\varepsilon^2, \\ &= b_2(-a_1 + \varepsilon\sqrt{\Delta})^2 + b_1(2a_2)(-a_1 + \varepsilon\sqrt{\Delta}) + b_0(2a_2)^2, \\ &= 2a_1(a_1 b_2 - a_2 b_1) - 4a_2(a_0 b_2 - a_2 b_0) - 2(a_1 b_2 - a_2 b_1)\varepsilon\sqrt{\Delta}, \\ &= 2[a_1 s_{12} - 2a_2 s_{02} - \varepsilon s_{12}\sqrt{\Delta}]. \end{aligned}$$

From this it follows that both possible solutions are valid if $s_{12}(\xi_0) = 0$. Otherwise, $g(\xi, \tau_\varepsilon(\xi)) = 0$ implies $a_1 s_{12} - 2a_2 s_{02} \neq 0$ and we have to choose ε such that the expressions $a_1 s_{12} - 2a_2 s_{02}$ and s_{12} have the same sign. \square

Remark: In case $\tau_\varepsilon(\xi)$ is not a valid parameterization for ξ_0 a symmetric consideration for $\tilde{\tau}_\varepsilon(\xi) = (\tilde{s}_\varepsilon, \tilde{t}_\varepsilon)$ leads to:

$$\begin{aligned} g(\xi, \tilde{\tau}_\varepsilon(\xi)) &= b_2 \tilde{s}_\varepsilon^2 + b_1 \tilde{s}_\varepsilon \tilde{t}_\varepsilon + b_0 \tilde{t}_\varepsilon^2, \\ &= b_2(2a_0)^2 + b_1(2a_0)(-a_1 - \varepsilon\sqrt{\Delta}) + b_0(-a_1 - \varepsilon\sqrt{\Delta})^2, \\ &= 2[2a_0 s_{02} - a_1 s_{01} - \varepsilon s_{01}\sqrt{\Delta}]. \end{aligned}$$

Hence, in this case there are two real points if $s_{01}(\xi_0) = 0$. Otherwise, ε is chosen such that $2a_0 s_{02} - a_1 s_{01}$ and s_{01} have the same sign. Note that at least one parameterization remains valid, see also Section 2.

Remark: We indicate in Appendix B how to compute the multiplicity of each intersection point.

Algorithm.

Algorithm 2 intersects a smooth quartic \mathcal{C} with a third quadric Q_U . In general all operations are performed over an algebraic extension of degree 2. This in particular complicates an efficient implementation of the square free factorization (line 9), the root isolation (line 11) and the ‘is-root-of’ tests (lines 15, 19, 21). The algorithm is designed such that all sign computations (lines 17, 22, 25) are known to have results different from zero. Hence, we can use multiprecision floating point interval arithmetic (MPFI) in order to compute the signs. We start with a low precision and double the precision of the floating point arithmetic until an unambiguous sign is computed. All required tools are provided by the library NUMERIX, which is part of the EXACUS project, see also Section 7.

Algorithm 2 Given the parameterization $\mathbf{X}_C(\xi)$ of a smooth quartic $\mathcal{C} = Q_S \cap Q_T$ and another quadric Q_U . If $\mathcal{C} \subset Q_U$ report \mathcal{C} . Otherwise, compute the exact parameter values of all real intersection points $\mathcal{C} \cap Q_U$ with respect to $\mathbf{X}_C(\xi)$.

```

// All coefficients are defined in  $\mathbb{Z}$  or  $\mathbb{Z}[\sqrt{\delta}]$ ,  $\delta \in \mathbb{Z}$ 
(1)  $f(\xi, \tau) := \mathbf{X}_R^T(\xi, \tau)S\mathbf{X}_R(\xi, \tau) = a_2(\xi)s^2 + a_1(\xi)st + a_0(\xi)t^2$ 
(2)  $g(\xi, \tau) := \mathbf{X}_R^T(\xi, \tau)U\mathbf{X}_R(\xi, \tau) = b_2(\xi)s^2 + b_1(\xi)st + b_0(\xi)t^2$ 
(3)  $\Delta(\xi) := a_1(\xi)a_1(\xi) - 4a_0(\xi)a_2(\xi)$ 
(4)  $s_{01}(\xi) := a_0(\xi)b_1(\xi) - a_1(\xi)b_0(\xi)$ 
(5)  $s_{02}(\xi) := a_0(\xi)b_2(\xi) - a_2(\xi)b_0(\xi)$ 
(6)  $s_{12}(\xi) := a_1(\xi)b_2(\xi) - a_2(\xi)b_1(\xi)$ 
(7)  $res(\xi) := s_{02}(\xi)s_{02}(\xi) - s_{01}(\xi)s_{12}(\xi)$ 

(8) if {  $res \equiv 0$  } report  $\mathcal{C}$  and return. end if

(9)  $\mathcal{F} := square\_free\_fac(res)$  // of the form  $\{(fac_0, m_0), \dots, (fac_k, m_k)\}$ 
(10) for all  $\{(fac_i, m_i) \in \mathcal{F}\}$  do
(11)     isolate all roots of  $fac_i$  // e.g. the Descartes Method
(12)     store all these roots together with multiplicity  $m_i$  in  $\mathcal{R}$ .
(13) end for

(14) for all  $\{(\xi_i, m_i) \in \mathcal{R}\}$  do
(15)     if {  $\xi_0$  is root of  $\Delta(\xi)$  }
(16)         Report  $(\xi_i, 0)$ .
(17)     else if {  $m_i \equiv 0 \pmod{2}$  and  $\text{sign}(\Delta(\xi_0)) < 0$  } // use MPFI
(18)         Reject  $\xi_i$ . // complex intersection
(19)     else if {  $m_i \geq 2$  and  $\xi_0$  is root of  $s_{01}$  and  $s_{12}$  }
(20)         Report  $(\xi_i, +1)$  and  $(\xi_i, -1)$ .
(21)     else if {  $\xi_0$  is not a root of  $a_2$  }
(22)          $\varepsilon := \text{sign}(s_{12}) \text{sign}(a_1s_{12} - 2a_2s_{02})|_{\xi_0}$  // use MPFI
(23)         Report  $(\xi_i, \varepsilon)$ .
(24)     else //  $\xi_0$  is a root of  $a_2$  but not of  $a_0$ 
(25)          $\varepsilon := \text{sign}(s_{01}) \text{sign}(2a_0s_{02} - a_1s_{01})|_{\xi_0}$  // use MPFI
(26)         Report  $(\xi_i, \varepsilon)$ .
(27)     end if
(28) end for

```

5. Comparing Points

We will now focus on how to compare intersection points, as they have been defined in Section 4. This problem actually consists of two parts.

First, in general an intersection point $p \in Q_S \cap Q_T \cap Q_U$ has three representations, one for each of its algebraic components in $Q_S \cap Q_T$, $Q_S \cap Q_U$ and $Q_T \cap Q_U$, respectively. As explained in Section 5.1 it would be very expensive to compare just two representations of the point p . The solution is to consider and match the representations of all points in $Q_S \cap Q_T \cap Q_U$ simultaneously. This is the reason why phase 3 of the main algorithm considers triples of quadrics.

Second, we have to compare points which are defined on the same algebraic component, that is, we need to sort points along a common algebraic component. This is discussed in Section 5.2. Obviously, the sorting is required to initialize the edge-adjacency graph, but it also has an important side effect. It detects further equalities among representations of the same intersection point that originate from different triples of quadrics.

5.1. Matching of Intersection Points

A disadvantage of representing intersection points by their parameter value is that for each point we have to keep one representation for each component it lies on. Moreover, it must be possible to identify representations of the same intersection point, which is the focus of this section.

First of all it should be clear that given just two representations on different components, it is very hard to decide whether they represent the same intersection point or not, that is, it is hard to detect equality. This is due to the fact that we are forced to compare the points by their coordinates in \mathbb{P}^3 , since the representations are not given in the same parameter space. Now observe that the degree of the involved algebraic expressions can be very large. In the worst case, that is, each point is defined on another smooth quartic, each coordinate involves, among others, the evaluation of a square root of a polynomial of degree 4 evaluated at a parameter value defined as the real root of a polynomial of degree 8 and all this over an algebraic extension of degree 2. Thus, to make a long story short, it is at least very inefficient to compare two representations via their exact coordinates in \mathbb{P}^3 .

On the other hand it is very easy to detect inequality of two points using interval arithmetic based on multiprecision floats (MPFI). The procedure is as follows: start with a low precision and compute the coordinates of each point using MPFI. By the inclusion property of MPFI this results in two bounding boxes, one for each point. If the boxes still overlap, double the precision and start again. In most cases the two points are far apart and the process terminates after a few iterations. Note that the process does not terminate if the points are equal. However, in case of inequality this is fast and leads to the following idea.

Assume given a representation of a point p and a sequence of representations seq . If we know that p has exactly one corresponding representation in seq we can find this representation very fast using MPFI arithmetic as discussed above. We just increase the precision until we can exclude all but one representation in seq , which must be the one we are looking for. This idea is encoded in Algorithm 3.

Note that Algorithm 3 is very efficient since it is using MPFI arithmetic in an adaptive way. We consider it as one of the main sources of efficiency in our approach. In fact the overall algorithm, see Section 1.2, is designed such that it is possible to apply Algorithm 3.

5.2. Sorting Points on Algebraic Components

The representation of the intersection points by their parameter value has the fundamental advantage that it is very easy to sort them along a common component.

In principal we just have to compare real roots of univariate polynomials, which is provided by the library NUMERIX. In almost all cases the domain of the parameterization is \mathbb{P}^1 . This induces a cyclic order along the component. Hence, the domain is broken up at $(1, 0) \in \mathbb{P}^1$ yielding a total order along the component, i.e. the one induced by \mathbb{R}^1 . The only two cases that need a bit more attention are the nodal quartic and the smooth quartic.

Algorithm 3 Given the sequences $seq_1 = \{p_1, \dots, p_m\}$ and $seq_2 = \{q_1, \dots, q_n\}$ representing the point sets \mathcal{Q}_1 and \mathcal{Q}_2 respectively, with $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$ and each set containing only distinct points, compute the injective map φ_{12} matching the representations in seq_1 with their counterparts in seq_2 .

```

// start with the empty map
 $\varphi_{12} := \{\}$ 
// number of used bits in all MPFI computations
mpfi_precision := 2
while  $\{seq_1 \neq \emptyset\}$  do
  for all points in  $seq_1$  and  $seq_2$  do
    precompute a bounding box using MPFI
  end for
  if a  $BBOX(p_i)$  intersects only one  $BBOX(q_j)$  then
    add  $(i, j)$  to  $\varphi_{12}$ 
    remove  $p_i$  from  $seq_1$ 
    remove  $q_j$  from  $seq_2$ 
  end if
  // double the precision for MPFI computations
  mpfi_precision :=  $2 \cdot mpfi\_precision$ 
end while
Report  $\varphi_{12}$  and return.

```

Sorting on Nodal Quartic.

As in the other cases the parameterization $X_{\mathcal{C}}$ of a nodal quartic \mathcal{C} is given in terms of rational functions and the domain of definition is \mathbb{P}^1 . However, due to the singular point the situation becomes a bit more involved. There are two cases:

- (1) The singular point is an isolated point: In this case the parameterization does not reach the singular point at all. Therefore, the point is represented by a separate value and excluded from the normal sorting process.
- (2) The singular point is not isolated: Since there are two arcs passing through the singular point there are two parameter values⁴ representing this point. Removing one of these values from the domain makes it isomorphic to \mathbb{R}^1 and hence capable for sorting. We just have to keep track of this artifact in the final data structure.

Sorting on Smooth Quartic.

First of all recall that the parameterization consists of two arcs, a positive arc $X_{\varepsilon=1}$ and a negative arc $X_{\varepsilon=-1}$. Each arc is defined within the domain

$$\mathbb{D} = \{\xi \in \mathbb{P}^1 \mid \Delta(\xi) \geq 0\},$$

where $\Delta(\xi)$ is a univariate square-free polynomial of degree 4. Hence, the number of possible real roots of $\Delta(\xi)$ is 0, 2 or 4. This induces 3 different situations in parameter space, according to the possible real roots of Δ , as illustrated in Figure 3. In the sequel we discuss the sorting according to the number of real roots of Δ :

0 In this case $X_{\varepsilon=1}(\xi)$ and $X_{\varepsilon=-1}(\xi)$ never touch. There are two connected components defined on \mathbb{P}^1 , each formed by one of the arcs. The points on the different arcs are treated separately.

⁴ Defined by a known quadratic polynomial used in the parameterization, see Dupont et al. [16].

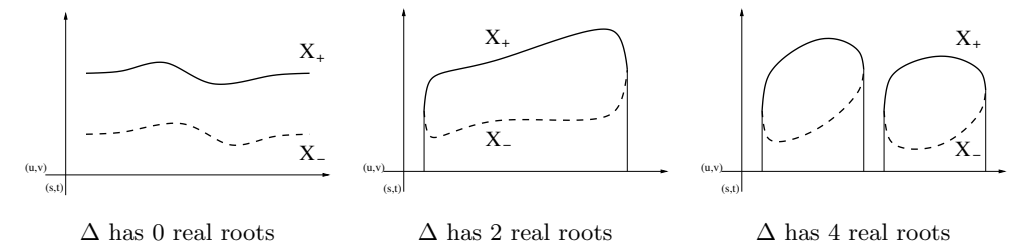


Figure 3. Different situations for smooth quartics as illustrated in parameter space.

- 2 In this case the arcs touch at the two roots of Δ and form one connected component. We define a cyclic order on the component by reversing the order on the negative arc. The cyclic order is broken up at the first root of Δ .
- 4 In this case the domain is broken up into two intervals. The two arcs are connected at the endpoints, i.e. the roots of Δ . Hence, there are two connected components treated separately, the sorting is performed as in the case of one connected component.

6. Details of Overall Algorithm

Given a set \mathcal{Q} of quadric surfaces, defined by rational coefficients of any size, our algorithm computes the edge-adjacency graph $\mathcal{G}(\mathcal{Q})$ of the arrangement $\mathcal{A}(\mathcal{Q})$, that is, it computes all vertices and their connectivity along the edges of $\mathcal{A}(\mathcal{Q})$.

6.1. Data Structure

The data structure representing the edge-adjacency graph $\mathcal{G}(\mathcal{Q})$ is meant to be a preliminary stage towards the arrangement $\mathcal{A}(\mathcal{Q})$. We plan to store the arrangement in a variant of a structure used to represent *Nef polyhedra*⁵ as presented in [23]. This structure is a vertex-oriented structure, that is, the information stored within the vertices is in principal sufficient to represent the arrangement. In particular, each vertex stores its local neighborhood in a so called *sphere map*. The data structure representing $\mathcal{G}(\mathcal{Q})$ is organized in the same spirit. Beside the connectivity information to the other vertices, each vertex already stores all information needed to determine its local neighborhood.

For each vertex v we store:

- All quadrics the vertex v lies on.
- For each algebraic component \mathcal{C} the vertex v lies on:
 - the parameterization $X_{\mathcal{C}}$ of \mathcal{C}
 - the parameter value with respect to $X_{\mathcal{C}}$
 - the next vertex with respect to the sorting on \mathcal{C}
 - the previous vertex with respect to the sorting on \mathcal{C}
- An explicit representation of the coordinates in \mathbb{P}^3 if available.

For each algebraic component \mathcal{C} we store:

- The parameterization of the component.
- All quadrics the component \mathcal{C} lies on.
- A sorted list of all vertices on that component.

⁵ Nef polyhedra in d -dimensional space are the closure of half-spaces under Boolean set operations.

6.2. Algorithm

Though the final data structure represents each vertex by exactly one object, we do not guarantee this throughout the algorithm. In particular, we can not keep one sorted list of all vertices constructed so far, since this involves the comparison of two arbitrary vertices. Note that we have to avoid this kind of comparison since this can be too expensive, see Section 5.1. Instead we take advantage of the fact that we can efficiently match sequences of vertices representing the same set of points and that it is easy to compare vertices if they are defined on the same component.

In principal the overall algorithm has the following phases:

0. Initialization
1. Computation of all features induced by one quadric
2. Computation of all features induced by all pairs of quadrics
3. Computation of all features induced by all triples of quadrics
4. Sorting of vertices along algebraic components

In the sequel we discuss the details of the algorithm along the different phases. The most important phases are the second phase constructing all algebraic components and the third phase constructing nearly all vertices, that is, all vertices induced by the intersection of three quadrics. The unique representation of a vertex is guaranteed by the fourth phase, since it detects equality among vertices while sorting them along the algebraic components.

Remark: Note that we consider an isolated point as a separate algebraic component. The vertex representing the isolated point is defined as the only point on that algebraic component. At first glance this may seem odd, but it avoids a lot of special cases in the actual code as well as in the subsequent discussion of the algorithm.

Phase 0: Initialization.

This phase just ensures that the intersection of two or more surfaces from \mathcal{Q} will result in low dimensional features only, that is, it ensures that all surfaces are coprime. Consequently, \mathcal{Q} may also contain rational planes but this has no effect on the overall algorithm since we could interpret each rational plane as a quadric by defining it as a double plane. In the sequel we will not mention rational planes in particular.

Phase 1: Features Induced by One Quadric.

This phase introduces lower-dimensional features of singular input quadrics. In case of a cone this is just the rational point representing the apex of the cone. In the case of two intersecting planes we have to construct the rational line, which is the intersection of the two planes. Though the construction of these entities is rather trivial we have to keep track of them in the second phase.

For each quadric $Q \in \mathcal{Q}$ do:

- 1.1 If Q has inertia $(3, 0)$ or $(2, 1)$: Q is a complex or real cone with a rational point p being its singular locus. Construct a vertex representing p .
- 1.2 If Q has inertia $(2, 0)$ or $(1, 1)$: Q represents two complex or real planes intersecting in a real rational line ℓ . Construct ℓ and store it in the list of algebraic components.

Phase 2: Intersection of All Pairs of Quadrics.

This phase iterates over all pairs of quadrics and constructs all further algebraic components. The constructed vertices in this phase are the intersection points of two algebraic components, self intersections or isolated points. We ensure that each algebraic component is represented exactly once, see Section A.1.

For each pair (Q_S, Q_T) of quadrics out of \mathcal{Q} do:

- 2.1 Compute a unique representation of the pencil \mathcal{P} defined by Q_S and Q_T . Use the cache, see Section A.1 to avoid unnecessary reconstructions of algebraic components.
- 2.2 For a new pencil, use Dupont's algorithm to compute the intersection of Q_S and Q_T , see Section 2.
- 2.3 For each constructed cubic, conic or line check that the component is new. If it is not, unify it with the old representation as discussed in Section A.1.

Moreover, the second phase has to incorporate entities that have been constructed in the first phase:

- 2.4 If one of the two quadrics, say Q_S , is a cone let p be its singular point: if $p \notin Q_S \cap Q_T$ there is nothing to do. Otherwise, we have to determine a parameter value for each algebraic component $\mathcal{C} \subset Q_S \cap Q_T$ the point p lies on. Since p is a singular point of $Q_S \cap Q_T$ it is clear that \mathcal{C} is not a smooth quartic. Hence, \mathcal{C} has a rational parameterization and since p is rational it is easy to determine its parameter value, see also Section A.2.
- 2.5 If one of the two quadrics, say Q_S , represents two planes intersecting in the rational line ℓ : if $Q_S \cap Q_T$ is empty or contains the full line ℓ there is nothing to do. Otherwise, let seq_ℓ be the sequence of intersection points representing $\ell \cap Q_T$ on ℓ . Given the fact that ℓ is the singular locus of Q_S it is clear that seq_ℓ is a subset of the singular points of $Q_S \cap Q_T$. Hence, we can use Algorithm 3 to match seq_ℓ with the singular points in \mathcal{P} . An example of such case is presented on Figure 4.

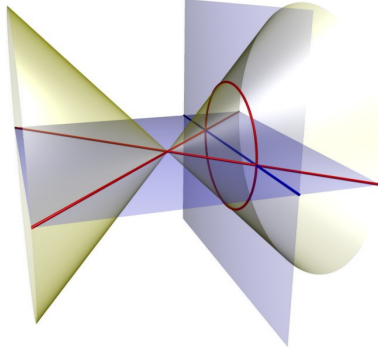


Figure 4. Q_S is a pair of intersecting planes with a singular line ℓ , depicted in blue, and Q_T a cone with its vertex on the horizontal plane but not on ℓ . Here there are three singular points on the intersection curve of Q_S and Q_T but only two of them on the singular line.

Phase 3: Intersection of all Triples.

This is the most important phase since it constructs nearly all vertices, that is, all vertices defined by the intersection of three quadrics. Note that it iterates over all triples of quadrics and not over all algebraic components. In this way we obtain all representations of points induced by three quadrics at once. This has the advantage that we can efficiently match these representations using Algorithm 3 as it has been discussed in Section 5.1.

For each triple (Q_S, Q_T, Q_U) of quadrics out of \mathcal{Q} do:

- 3.1. Intersect all algebraic components in $Q_S \cap Q_T$ with Q_U . For all resulting intersection points store their representations into the sequence seq_{ST} .
- 3.2. Unify representations in seq_{ST} representing the same point. This is necessary due to the fact that an intersection point in $Q_S \cap Q_T \cap Q_U$ may be common to several components in $Q_S \cap Q_T$. In this case this point is reported several times, once for each component it lies on. However, this is no problem, since the problematic points are already reported by the algorithm computing $Q_S \cap Q_T$.
- 3.3. In the same way compute the sequences seq_{SU} and seq_{TU} .
Use Algorithm 3 to match the representations in seq_{ST} , seq_{SU} and seq_{TU} .
- 3.4. For each algebraic component a vertex $v \in Q_S \cap Q_T \cap Q_U$ lies on, add v to the list of vertices of that algebraic component.

Phase 3 does not guarantee that each point is represented by exactly one vertex. For instance, it may happen that a point is the intersection point of four or more quadrics. In this case the third phase constructs four vertices representing this point, one vertex for each possible triple out of the four quadrics. The equality of these vertices is detected in the fourth phase.

Phase 4: Sorting and Unification of Vertices.

This phase sorts all points along their algebraic components, which is the main building block in order to initialize the edge-adjacency graph. As discussed in Section 5.2 this comparison is based on the parameter values of the vertices with respect to their common component.

An important side effect of this process is that it also detects equality among the vertices. If two vertices are equal we merge all the information stored in the representations of both vertices into one. At the end of the sorting process all equal vertices have exactly the same representation. This is due to the fact that all vertices have at least one algebraic component in common and that we sort along all algebraic components.

- 4.1 For each smooth quartic \mathcal{C} split the list of vertices on that smooth quartic according to the connected components of \mathcal{C} . All other components consist of exactly one connected component.
- 4.2 For each connected component \mathcal{CC} ,
 - sort the list of vertices on that component,
 - erase duplicates from the sorted list of vertices,
 - initialize previous and next entries of the vertices with respect to \mathcal{CC} .

Output the edge-adjacency graph $\mathcal{A}(\mathcal{Q})$.

7. Implementation

Our software is implemented within the EXACUS project [4] and consists of two parts. The first part is an adaptor to the software QI [30], which implements the approach by Dupont et al. [16], see also Section 2. The second part is based on the intersection of an algebraic component with another quadric as discussed in Section 4.

As the whole EXACUS project, our software follows the generic programming paradigm with C++ templates similar to well-established design principles as they are used in the STL [2] or CGAL [11]. For instance, we can instantiate our algorithms on such a small scale as number types and arithmetic operations. The instantiation that currently results in the best performance is the set of number types provided by the library CORE.⁶ We use `CORE::BigInt` for integers of arbitrary length, `CORE::BigRat` in order to represent rational numbers. However, in most cases the coefficient type has to represent an algebraic extension of degree 2. This is provided by the special type `Sqrt_extension`. The type has been designed such that only values from the same extensions are interoperable. This has the advantage that it is possible to keep the representation of objects very simple, in particular it can be used as coefficient type in resultant and gcd computations. We also provide a nested form that is capable to represent algebraic extensions of degree 2×2 . For instance, this is used in the case of conics, see also Table 1. For the rare cases of algebraic extension of degree 3 or 4 we use the type `CORE::Expr`. The type `CORE::BigFloat` represents floats whose mantissa length can be changed at runtime. The type is used to implement the multiprecision floating point interval (MPFI) arithmetic as it is used within Algorithm 3. It is also used for filtering at various places.

A point stores a parameter value with respect to the parameterization of each algebraic component it lies on. Ignoring supplemental flags, this is the real root of some univariate polynomial, which is in general defined over some algebraic extension (Table 1). For the square-free factorization of this polynomial we have implemented a variant of Yun’s algorithm [38]. The gcd implementation [27] uses modular methods and incorporates ideas from [12], [21] and [29].

For the isolation of real roots we have integrated the so-called *bitstream Descartes* by Eigenwillig et al. [19]. The coefficients of the polynomial are converted to (potentially infinite) bitstreams and a variant of the Descartes Method is used to determine the isolating intervals of the real roots. The advantage of this method is that it is adaptive in the number of bits that are requested. The method is especially useful in case of algebraic coefficients.

Thus, a parameter value is essentially stored by a square-free polynomial and an isolating interval. This representation is encapsulated in a special type `Algebraic_real`. Given the design of our algorithm we only need to deal with real roots of polynomials defined over the same algebraic extension at a time. The relevant features are: testing whether a value is the root of another polynomial, comparison of two roots of two polynomials, and providing an approximation with respect to a given relative precision. The class `Algebraic_real` implements the quadratic refinement method by Abbott [1]. More precisely, it implements a variant that uses MPFI arithmetic, which reduces the costs of internal computations significantly. For further details on this implementation as well as an overview of EXACUS and CGAL we refer to the second part of [26].

⁶ A similar set of number types is provided by the library LEDA.

8. Comparison with the Projection Approach

We have not analyzed the worst case in the bit-complexity model, since we argue that our algorithms are adaptive in the bit-complexity and a worst-case analysis would not be representative. Instead, we want to show that our parameterization method is efficient and feasible. In particular we want to analyze the competitiveness of our approach with respect to the projection approach. Since both approaches were implemented within the EXACUS project, we were able to benchmark them in parallel.

For a given set $\mathcal{Q} = \{Q_1, \dots, Q_n\}$ of quadrics the projection approach as well as our approach aim for the computation of the full 3D arrangement induced by the set \mathcal{Q} . Unfortunately, none of them has reached this goal yet. Hence, it was not possible to compare the approaches on this final level. Our approach is currently only capable to compute the 3D edge-adjacency graph of the arrangement, while the projection approach only computes the different arrangements on the surface of each quadric.

Both approaches differ in their layout. Therefore, we first of all had to find a common level in order to compare both algorithms in a fair way. In case of the projection approach we decided to measure the computation of the arrangement on the first quadric. Hence, for our approach we modified the algorithm such that it also just computes the edge-adjacency graph of the arrangement on the first quadric. More precisely, the modified algorithm computes:

- The parameterization of all intersection curves $Q_1 \cap Q_i$ for $i = 2, \dots, n$.
- The intersection of these curves with the other quadrics, which results in different representations of the intersection points with respect to the different intersection curves they lie on.
- The matching of these representations and their merging into vertices.
- The sorting of the vertices along the curves.

We next give a more precise description of the projection approach by Berberich et al. [6]. Thereafter, we present the benchmarks instances. Finally, we present the comparison of the two approaches.

8.1. The Projection Approach by Berberich et al.

As in our case the goal of the projection approach is to compute the 3D arrangement which is induced by a given set $\mathcal{Q} = \{Q_1, \dots, Q_n\}$ of quadrics defined by rational coefficients of arbitrary size. The principal idea is to split the computation of the 3D arrangement into two steps. The first step is to compute for every quadric Q_i in \mathcal{Q} the 2D arrangement which is induced on the surface of Q_i by all other quadrics in \mathcal{Q} . This step is based on the *projection* of the appearing intersection curves onto the xy -plane. Note that these arrangements are computed separately for each quadric. In the second step, the plan is to use these arrangements in order to deduce the complete 3D arrangement. Though only the first step is presented in Berberich et al. [6], we consider this as a very promising approach.

In order to compute the arrangement on one quadric, say Q_1 , all intersection curves $Q_1 \cap Q_i$, $2 \leq i \leq n$, as well as the silhouette curve of Q_1 are *projected* onto the xy -plane by classical resultant computations eliminating z . The resulting curves are represented by polynomials over integer coefficients. The degree of these polynomials is at most 4. Due to the projection they lose the spatial information: branches on the upper and lower part of Q_1 are projected on top of each other. Therefore, each projected curve is first

of all decomposed into arcs with respect to the lower and upper part of Q_i . Thereafter, the planar arrangements for the upper and the lower part are computed separately by a variant of the Bentley-Ottmann sweep-line Algorithm [3]. Both arrangements together completely describe the arrangement of intersection curves on the surface of Q_1 .

The computation of the predicates which are needed by the sweep-line algorithm is reduced to the analysis of up to two projected curves at the same time, which is called a *Curve Pair Analysis* (CPA). This is more or less a cylindrical algebraic decomposition [13] of the two curves. First of all all event points are projected onto the x -axis by further resultant computations. The event points are the x -critical points of the curves and the intersection points of both curves. Although the projected curves are of algebraic degree 4, the coordinates of the intersection points of two curves are of algebraic degree 8. This is due to the fact that the up to 16 intersection points split into two groups of 8 points. Points in the first group are the true intersection points in \mathbb{R}^3 the others are the intersection points caused by the projection of the curves. Therefore, only points in the same group may be algebraic conjugates. The polynomials of degree 8 defining the coordinates of the intersection points are deduced by a multi-resultant computation using the three involved quadrics. After the computation of x -coordinates the CPA is completed by computing the intersections of y -slices with the curves. These slices are in general computed in between the x -coordinates, which has the advantage that the polynomials involved are not defined over any algebraic extension. Hence, the basic operations, such as root isolation or gcd computation, are in general performed over univariate polynomials with integer coefficients of degree up to 8.

The main disadvantage of the approach is that in case of covertical events on the same curve a shear of the coordinate system must take place and everything has to be recomputed with respect to the new coordinate system. Moreover, it does not provide an explicit parameterization of the appearing intersection curves.

8.2. Benchmark Instances

In order to show different aspects within the benchmarks we generated three different families of instances:

- rnd Each instance in this family contains random quadrics of a fixed bit-size. The instances differ in the number of contained quadrics. The quadrics are given by random integer coefficients of 50 bits each. In order to avoid empty quadrics and decrease the likelihood of empty intersection, each quadric is guaranteed to intersect the $[-100, 100]^3$ cube at least once. All other quadrics have been discarded.
- deg As for the rnd-instances this family varies in the number of quadrics and has also been used within Berberich et al. [6]. The quadrics within this family have been generated by interpolation. In order to achieve degenerate situations, several quadrics share values for partial derivatives and higher-order derivatives at common intersection points. The coefficients of this family are given by integers with 73 bits on average.
- bits Each instance of this family contains a constant number of random quadrics. The number of quadrics is 50. The instances differ in the bit-size of the coefficients of the quadrics. The number of bits varies from 10 to 190 bits. The quadrics are generated in the same way as for the rnd-family.

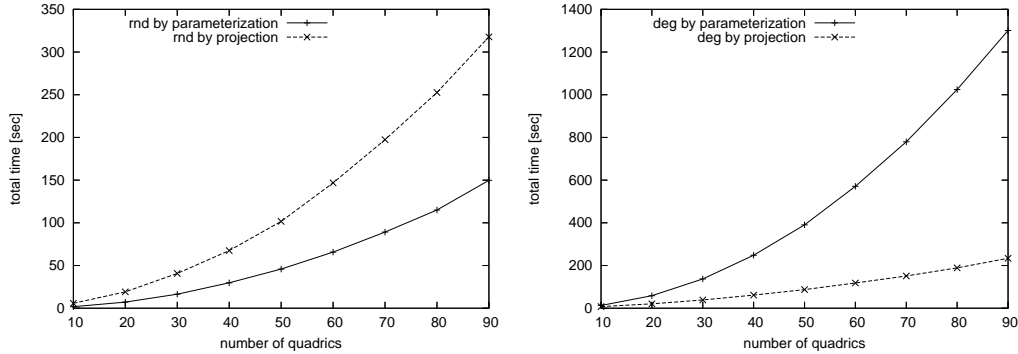


Figure 5. The parameterization approach compared to the projection approach. The left plot shows the timings for the random instances (rnd). The right plot shows the timings for the degenerated instances (deg).

8.3. Parameterization vs. Projection

We compared our approach to the projection approach on the random instances (rnd) as well as on the degenerate instances (deg). The results are presented in Figure 5.

The runtime of both algorithms seems quadratic in the number of quadrics. This is due to the fact that it is dominated by the quadratic number of intersections. In case of the projection algorithm these are the pairwise intersections of the n projected curves. In our case these are the intersections of all algebraic components on Q_1 with the other quadrics in \mathcal{Q} . For the chosen instances the number of vertices in the arrangement on Q_1 is quadratic in the number of quadrics. Therefore, both algorithms are linear in the number of vertices. However, due to the fact that the projection approach induces artificial vertices, it was not possible to show this effect within one plot. Note that we can expect that the projection approach is output-sensitive due to the use of the sweep-line algorithm. We cannot state this for our approach because we have not yet integrated any high level geometric filter. However, using a Bounding Volume Hierarchy as in e.g. Schömer et al. [34] should resolve this issue.

The right plot in Figure 5 shows that for the degenerate instances the projection approach performs better than our parameterization approach. This is due to the fact that for the degenerate situations it is not possible to avoid the computation of very expensive gcds, as it is the case for equal algebraic numbers (e.g. equal intersection points) or non-square-free resultants (e.g. tangential intersection points). These computations cannot be avoided by both approaches. However, in our case this is much more expensive for the following two reasons.

- (1) Our intersection curves are represented in the parameter space of the chosen quadric Q_R , see Section 2. This introduces a considerable amount of extra bits, for instance, the number of bits in the resultant polynomials for input quadrics with 50 bit integers is 4500 compared to just 1000 bits for the corresponding polynomial in the projection approach.
- (2) In our case most polynomials are defined over an algebraic extension of degree 2, which increases the number of exact arithmetic operations significantly. For instance the multiplication of two numbers of the type `Sqrt_extension` requires 5

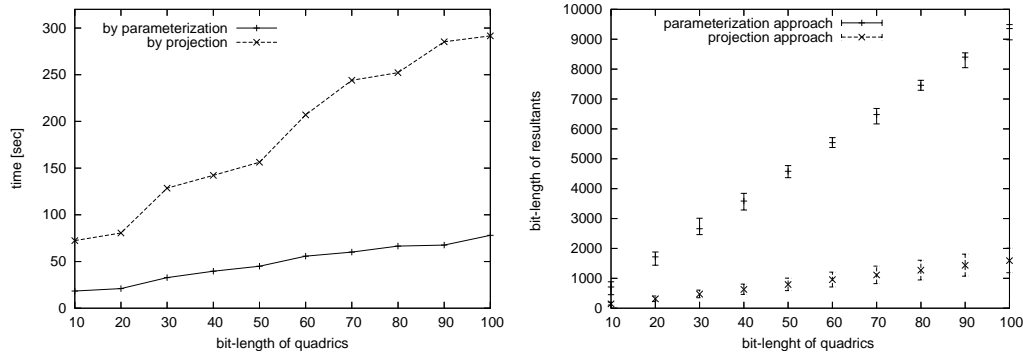


Figure 6. Projection approach compared to the parameterization approach on random instances with growing bits. The left plot shows the timings. The right plot shows the bit size of the coefficients in the resultant polynomial.

multiplications and 2 additions:

$$(a + b\sqrt{c})(a' + b'\sqrt{c}) = (aa' + bb'c) + (ab' + ba')\sqrt{c}.$$

Moreover, we cannot apply the more efficient algorithms for square-free factorization and gcd computation as they are used for the integer polynomials in the projection approach. Indeed, for the degenerate instances, the time for the gcd computation contributes about 80% to the total runtime of the parameterization approach.

The left plot in Figure 5 shows that for the random instances the parameterization approach is significantly faster than the projection approach. This effect is even amplified if the number of bits increases as it is shown in the left plot of Figure 6. Note that this is in a sense paradoxical, since in our case the number of bits increases 4.5 times faster than for the projection approach. This can be observed in the right plot of Figure 6. This shows that for the generic case we have been able to widely decouple our approach from the bit-size of the input. This can be first of all explained by the consistent use of multiprecision floating point interval arithmetic (MPFI) within our approach. This is possible due to the available parameterization of the intersection curves. By contrast, the projection approach has to apply a root isolation algorithm to the resultant polynomial on the x -axis but also to the polynomials appearing due to the y -slices within the CPA.

Some comments are in order. First, we believe performance is important for both degenerate and non-degenerate input. Indeed, in many geometric applications of interest both situations will occur and there will be a large fraction of each. We have actually verified that this is the case in preliminary experiments on a boundary evaluation loop of quadric CSG models based on the parameterization approach. Second, it should be stressed that the degenerate instances benchmark we have used maximizes in some sense the negative effects of the parameterization approach. Indeed, only degree 8 polynomials were kept, representing situations where two smooth quartics become tangent. So we have the combined effect of large bit complexity of parameterizations/degree 8 polynomials (the smooth quartic case is the dominating case bit-complexity-wise), polynomials defined over an extension (the coefficients of the parameterization of a smooth quartic almost always have a square root) and polynomials of maximum degree among degenerate

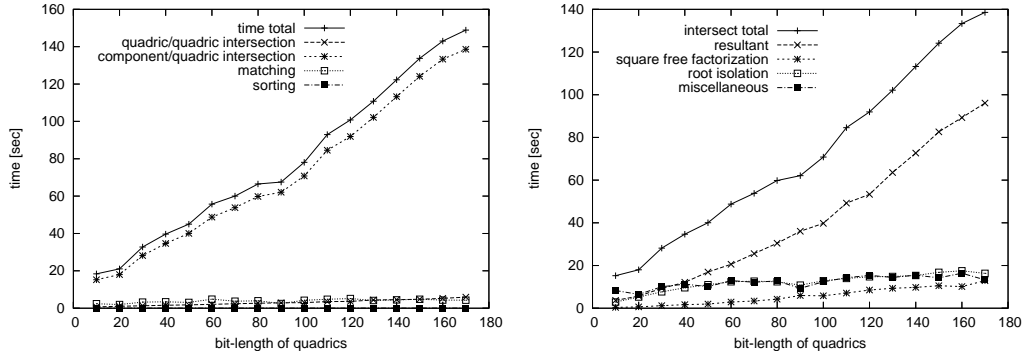


Figure 7. Detailed timings for the parameterization approach on the random instances with growing bits. The left plot shows the decomposition of the overall runtime. The right plot shows the runtime decomposition for the component/quadric intersection. In each plot the first curve is the sum of the others.

data, inducing difficult gcd calculations. In any geometric application, with a mix of non-degenerate and degenerate situations of all kinds (from degree 2 to degree 8 polynomials, from low to high bit complexity, with or without square roots), the performance gap between the projection approach and our parameterization approach is overall certainly much smaller than what Figure 5 suggests.

8.3.1. Details for Generic Instances

We chose the random instances with growing bits in order to give a more detailed analysis for the generic case. The left plot in Figure 7 shows that the total runtime is dominated by the time spent within the algebraic component/quadric intersection. The runtime for other important steps within our algorithm, namely the intersection of quadrics as well as the matching and sorting of intersection points can be neglected.

- For the quadric/quadric intersection this is first of all caused by the fact that the coefficient size within this step is still quite small. Moreover, this step is only performed a linear⁷ number of times.
- For the sorting and matching this is due to the efficient implementation of the type `Algebraic_real` and the use of MPFI arithmetic in Algorithm 3 as discussed in Section 5.1. In particular, these steps can be considered as independent from the bit size.

The right plot in Figure 7 shows a detailed analysis of the time spent within the intersection of the algebraic components with the other quadrics. The total time is broken down in the times for the resultant computation, the square-free factorization and the root isolation. The last curve covers the remaining time.

- The curve representing the time spent within the resultant computation is the most significant curve and shows that the performance of the resultant computation is still depending on the number of involved bits. For the biggest instance, with input quadrics represented by 170 bit integer coefficients, the computation of the resultant contributes about 70% to the total runtime. This is due to the fact that all computations leading to the resultant polynomial are performed using traditional exact integer arithmetic, i.e. the used coefficient type within `Sqrt_extension` is `leda::integer` or `CORE::BigInt`.

⁷ Note that this refers to the modified algorithm computing the edge-adjacency graph on the first quadric only.

- The next important step is the square-free factorization of the resultant. First of all note that for the used instances (bits) all appearing polynomials are already square-free. Hence, the time spent within the square-free factorization is basically just the time that the modular gcd algorithm needs to detect that the computed polynomial is trivial. This is usually the case right after the first prime number. Hence, we can almost neglect the time spent in this part.
- The used approach within the root isolation is the Bitstream Descartes Method by Eigenwillig et al. [19]. The plot shows that this method is independent from the bit size of the coefficients. This is due to the fact that the method has been designed such that it uses only as many leading bits of the coefficients as needed in order to isolate the roots. We have been able to apply this approach for integer coefficients and for `Sqrt_extension` coefficients as well.
- The last curve shows the time spent within the remaining task in order to compute the correct arc corresponding to a real root of the resultant. These are mainly some ‘is root of’ tests and some sign computations of polynomials at the root, see also Algorithm 2 in Section 4.2. The plot shows that this is almost independent from the bit size as well. This has been achieved by the use of the MPFI arithmetic in the sign computations and the modular filter used within the ‘is root of’ tests.

8.3.2. Details for Degenerate Instances

The left plot of Figure 8 shows that, as in the generic case, the total time is again dominated by the time spent within the intersection of the algebraic components with the other quadrics. This is even more significant than in the generic cases. However, the right plot in Figure 8 shows that the time spent within the component/quadric intersection is dominated by the square-free factorization of the resultant. This is due to the fact that for the degenerate instances the resultant is not always square-free. In these cases the modular filter fails and Yun’s algorithm has to call several very expensive gcd computations.

We abstained from introducing another family of degenerate instances with growing bits as the results for the existing family are already significant enough. In particular the costs for the gcd computations depend on the number of involved bits and will dominate all other costs within our approach. The used gcd is based on modular methods. For more details on these methods and further benchmarks we refer to [27].

All benchmarks were measured on a Pentium(R) M processor 1.7 GHz with 512 kB cache under Debian Linux and the GNU C++ compiler v4.1 with optimizations (-O2).

8.4. Summary of Benchmarks

Both approaches to the quadrics arrangement problem, the projection approach as well as our approach, have their own strengths and weaknesses.

- Though the parameterization approach in general introduces an algebraic extension of degree two and a considerable amount of extra bits the parameterization allows an easy application of MPFI arithmetic. In combination with the modular filter and the Bitstream Descartes Method this has the consequence that our approach is faster than the projection approach for the generic instances. However, in the degenerate cases it is not possible to avoid the very expensive gcd computations and we have to pay the bill for the extra bits introduced by the parameterization. Hence, we don’t think that our approach will ever be faster than the projection approach for the degenerate

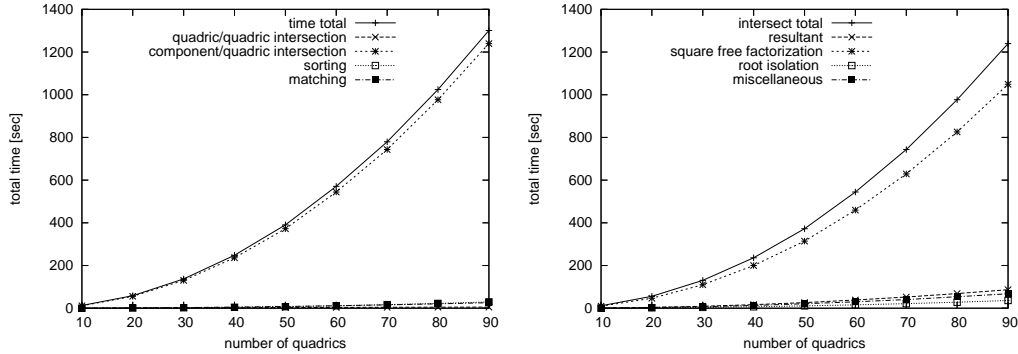


Figure 8. Detailed timings for the parameterization approach on the degenerate instances. The left plot shows the decomposition of the overall runtime. The right plot shows the runtime decomposition for the component/quadric intersection. In each plot the first curve is the sum of the others.

situations. Even though the modular gcd methods may not be fully optimized it is just impossible that a gcd over algebraic extensions with more bits is faster than a gcd for polynomials with integer coefficients and fewer bits.

- A clear disadvantage of our approach is that it is not extendable to more than just quadrics. This is due to the fact that the idea of ruled surfaces in the pencil is not applicable to higher degree surfaces. By contrast the projection approach is quite generic. In principal the main work is done by the curve pair analysis which is used to answer all predicates within the Bentley-Ottmann sweep-line algorithm. The other important step is to lift the projected arcs back to the correct part of the surface. Note that at least the first part is already available due to [17,18] and the second part seems feasible using a similar ray shooting technique as it has already been applied for the projection approach.

9. Conclusion and Future Work

We have presented a major milestone towards the construction of the 3D arrangement of a given set \mathcal{Q} of quadric surfaces, namely the computation of the edge-adjacency graph connecting the vertices of the arrangement. Our prototype is implemented within the framework of the EXACUS project and is based on a parameterization of quadric intersections. It is *complete* in the sense that it can handle all kinds of inputs including all degenerate ones, where intersection curves have singularities or pairs of curves intersect with high multiplicity. It is *exact* in the sense that it always computes the mathematical correct result. It is *efficient* measured in running times and compares somewhat favorably to the only previous implementation, namely the projection approach by Berberich et al. [6].

One lesson learned during the course of this work is that working in parameter space has a non-negligible cost. An important question is whether the impaired performance induced by the use of the parameterization approach is somehow balanced by having a parametric representation of edges and vertices of the adjacency graph. While it is too early to have a definite answer, we believe there are many issues involved in dealing with “real” applications of arrangements or subarrangements of quadrics which may

require or be easier to deal with if an explicit representation of edges and vertices is available. For instance, consider the algorithm for computing the medial axis (surface) of a polyhedron due to Culver et al. [15]. Such a medial surface is piecewise quadratic and each quadric surface is the bisector of two boundary features (vertex, edge, or face). A fundamental step in the Culver algorithm is called seam tracing (cf. [15, Sec. 6]): a seam is the intersection of quadric bisectors and seam tracing is essentially the problem of ordering points along an algebraic space curve of degree up to 4. Obviously this is easier to achieve if the curve is given in explicit form. On a different level, the (largely unexplored) problem of computing a fixed-precision “embedding” of an arrangement of quadrics, i.e. a correct machine realization, may be simpler to attack with a parametric representation.

Despite our important first step, the computation of the 3D arrangement is still missing. However, the data structure representing the edge-adjacency graph $\mathcal{G}(\mathcal{Q})$ can be considered as a preliminary stage towards the data structure that represents the arrangement $\mathcal{A}(\mathcal{Q})$. The idea is to store the arrangement $\mathcal{A}(\mathcal{Q})$ in a variant of a structure used to represent *Nef-Polyhedra* as presented in [9,23]. The structure is a vertex-oriented structure that stores the local neighborhood around each vertex, which could be modeled by an arrangement on an ε -sphere⁸ around the vertex. Each vertex on the sphere corresponds to an incident edge of the 3D arrangement and in the same way each edge on the sphere corresponds to an incident face. The most powerful idea of this approach is to delay the construction of the final data structure and to perform all operations within the sphere maps of the vertices first. In the case of Boolean set operations it is possible to first of all identify the relevant vertices, i.e. those which are part of the output, and to perform the set operations separately for each relevant sphere map. Other information such as the nesting of the shells could be deduced by ray shooting.

A. Unifying Equal Algebraic Components: Details

We now give details of the unification step (Section 3) aimed at identifying and handling duplicates among intersection components. The comparison of algebraic components is discussed in Section A.1 and the rescuing of points on components tagged as duplicates is discussed in Section A.2.

A.1. Comparing Algebraic Components

Even though the cache avoids a second construction of quartics, we discuss the comparison for all algebraic component types.

- **Quartics:** For quartics, as well as for cubics and conics, the comparison is based on the construction history of each algebraic component, that is, each algebraic component stores at least the pencil it has been constructed from. Due to the fact that a quartic unambiguously defines its pencil it suffices to test the equality of the pencil.
- **Cubics:** In the case of cubics we use the fact that each pencil contains up to one cubic only. Hence, it is sufficient to test whether the first cubic is contained in the pencil of the second cubic. To test this the parameterization of the first cubic is plugged into the implicit equations of the two defining quadrics in the pencil of the second cubic. The cubics are equal if both resulting polynomials vanish.

⁸ The ε -sphere is just one option, for further work on this see also Limbach [31].

- **Conics:** For conics the situation gets a bit more involved due to the fact that a pencil may contain up to two regular conics. However, if one of the pencils contains only one regular conic we proceed as in the case of cubics.

It remains to discuss the case that both pencils contain two regular conics. Let \mathcal{C} and \mathcal{C}' be the two compared conics and $\mathcal{P} = \text{pencil}(Q_S, Q_T)$ and $\mathcal{P}' = \text{pencil}(Q_{S'}, Q_{T'})$ be their defining pencils, respectively.

- If $\mathcal{P} \neq \mathcal{P}'$ we know that both pencils contain up to one common conic only. This is due to the fact that two conics unambiguously define a pencil. Hence, the two conics are equal if $\mathcal{C} \subset Q_{S'} \cap Q_{T'}$ and $\mathcal{C}' \subset Q_S \cap Q_T$. This is tested in the same way as in the case of cubics.
- If $\mathcal{P} = \mathcal{P}'$ the algorithm by Dupont et al. [16] will choose the only existing quadric of rank 2 of the pencil in order to parameterize the conics. And since the rest of the algorithm is deterministic it will always produce exactly the same parameterization for the conics. Hence, comparing the conics in this case is trivial.

Lines.

In the case of lines, we have to compare lines defined in an algebraic extension of similar degree only, since the degree is guaranteed to be optimal (see [16, Table 1, Part III]). In case of degree one and two, we compare the lines via Plücker Coordinates [36] using explicit arithmetic⁹. In case of a degree 4 extension we can guarantee a unique representation of the line by construction due to the fact that the pencil is uniquely defined by the line and its three algebraic conjugates. In the case of lines defined in a degree 3 extension the situation is more involved, since the pencil is not unique due to the fourth rational line.

Lines in an Algebraic Extension of Degree 3.

First of all note that this is a very rare case (cf. see [16, Part III]). It implies that the intersection is made of four lines, three living in an extension of degree 3 and the fourth being rational. Due to the low relevance of this case we abstained from implementing a very efficient method. Instead, we currently just use Plücker Coordinates with `leda::real` or `CORE::Expr` as coefficient type. Note that this is quite inefficient in case of equality due to the fact that the involved numbers have to be refined until the separation bound is reached.

Hence, for a mature implementation we propose the following approach. Let ℓ and ℓ' be the two lines to be compared. Moreover, let $\mathcal{P} = \text{pencil}(Q_S, Q_T)$ and $\mathcal{P}' = \text{pencil}(Q_{S'}, Q_{T'})$ be the defining pencils of ℓ and ℓ' respectively. First of all note that ℓ is one of three algebraic conjugate lines in the pencil \mathcal{P} . The fourth line in \mathcal{P} is rational. All lines intersect in one real rational point p , namely the singular locus of \mathcal{P} . Since ℓ is not rational this is the only rational point of ℓ . Of course the same holds for ℓ' and its rational point p' . Hence, $p \neq p'$ implies that $\ell \neq \ell'$.

From now on assume that $p = p'$. Construct a rational plane \mathcal{H} not containing p . Thereafter, compute the conics $\mathcal{C}_S = \mathcal{H} \cap Q_S$ and $\mathcal{C}_T = \mathcal{H} \cap Q_T$. Use the CONIX library [5] to compute their intersection $\mathcal{C}_S \cap \mathcal{C}_T$. Each point in $\mathcal{C}_S \cap \mathcal{C}_T$ corresponds to one intersection of a line in \mathcal{P} with \mathcal{H} . Use Algorithm 3 to identify the point $p_\ell \in \mathcal{C}_S \cap \mathcal{C}_T$ corresponding to ℓ . In the same way compute $p_{\ell'}$ corresponding to ℓ' . The lines are equal iff $p_\ell = p_{\ell'}$.

⁹ For instance, `leda::rational` or `NiX::Sqrt_extension` (Section 3).

A.2. Redefinition of Points on X'_C

Let \mathcal{C} be a cubic, conic or line that has been constructed with a pencil \mathcal{P} . Now we want to consider the case that \mathcal{C} has just been constructed a second time with another pencil \mathcal{P}' . Note that \mathcal{P}' also induces some other algebraic components, for instance, another conic in case \mathcal{C} is a conic. Let X'_C be the new and X_C the old parameterization of \mathcal{C} respectively. Since we want to delete X'_C we first of all have to redefine the points on X'_C created during the second construction with respect to X_C . There are at most 2 points which correspond to intersections of \mathcal{C} with the other component(s) induced by \mathcal{P}' .

Unfortunately, it is in general not trivial to provide a function that just maps the parameter values of points on X'_C to the corresponding values on X_C . This is hindered by two reasons. First, in case of conics and cubics the mapping is not linear. Second, X'_C and X_C may be given in two different algebraic extensions. Therefore, we decided to follow a general approach which is applicable to all but a very few special cases which just involve rational lines or rational points. To begin with, we start with these two special cases. Thereafter, we identify the remaining cases and discuss the general approach.

In the sequel let \mathcal{P}' be the pencil that led to the construction of X'_C . Moreover, let \mathcal{Q} be the set of m points on \mathcal{C} that have been induced by \mathcal{P}' and let $seq' = \{p'_1, \dots, p'_m\}$, $m \leq 2$ be the sequence of representations for these points with respect to X'_C .

Rational Line.

Let \mathcal{C} be a rational line. It is clear that X_C and X'_C are rational as well. Let the new parameterizations of \mathcal{C} be given by

$$X'_C(\lambda, \mu) = \lambda p' + \mu q', \quad \text{with } p', q' \in \mathbb{P}^3(\mathbb{Q})$$

Since the points p' and q' are rational we can compute their parameter values for X_C by solving a set of linear equations. Let these values be given by $(\lambda_{p'}, \mu_{p'}) \in \mathbb{P}^1(\mathbb{Q})$ and $(\lambda_{q'}, \mu_{q'}) \in \mathbb{P}^1(\mathbb{Q})$ for p' and q' respectively. Then the matrix

$$A = \begin{pmatrix} \lambda_{p'} & \lambda_{q'} \\ \mu_{p'} & \mu_{q'} \end{pmatrix} \in \mathbb{Q}^{2 \times 2}$$

provides the mapping of the parameter values for X'_C onto those for X_C . Since A is rational it can be used to convert all representations in seq' to the desired representations with respect to X_C , even though the parameter values are defined in some algebraic extension.

Rational Point.

Let $p \in \mathcal{Q}$ be a rational point on X'_C . In order to compute its parameter value on X_C we just recompute the parameter value on X_C without using the parameter value on X'_C at all. Since \mathcal{C} is not a smooth quartic, $X_C(\xi)$ is given in terms of rational functions:

$$X_C(\xi) = [f_1(\xi), f_2(\xi), f_3(\xi), f_4(\xi)]^T,$$

where $f_i \in \mathbb{K}[\xi]$ and where \mathbb{K} is the algebraic extension field over which $X_C(\xi)$ is defined. Since p is a valid point in \mathbb{P}^3 , let w.l.o.g. p_4 be the coordinate that is not zero. Then the parameter value for p is given by the real roots of

$$g(\xi) = gcd(s_{14}(\xi), s_{24}(\xi), s_{34}(\xi)), \quad \text{where } s_{ij}(\xi) = f_i(\xi)p_j - f_j(\xi)p_i.$$

due to the fact that p is rational this is defined over the algebraic extension of X_C , that is, $g \in \mathbb{K}[\xi]$.

Remaining Cases.

The principal idea for the remaining cases is to characterize \mathcal{Q} as the intersection of \mathcal{C} with the singular locus of \mathcal{P}' and to use this characterization to compute a sequence $seq = \{p_1, \dots, p_m\}$ representing the m points in \mathcal{Q} with respect to $X_{\mathcal{C}}$. Thereafter, these representations are matched with those in $seq' = \{p'_1, \dots, p'_m\}$ using Algorithm 3 as discussed in Section 5.1. It remains to compute $seq = \{p_1, \dots, p_m\}$ representing the points in \mathcal{Q} with respect to $X_{\mathcal{C}}$. However, this is only possible if \mathcal{C} itself is not part of the singular locus of \mathcal{P}' .

In the classification of pencils and their intersection curve in Dupont et al. [16], there are three cases of components which are contained in the singular locus of the intersection curve:

- when the intersection curve is a double regular conic: There is nothing to do due to the fact that \mathcal{Q} is the empty set.
- when the intersection curve is composed by two double lines: The two lines are defined in a rational plane and intersect in a rational point.
- when the intersection curve is composed by two lines and a double line. The double line is a rational line. The two other lines are not contained in the singular locus of \mathcal{P}' .

Hence, from now on we can assume that \mathcal{C} is not part of the singular locus of \mathcal{P}' and that \mathcal{P}' is a regular pencil. We shall now give a proper characterization of S with respect to $X_{\mathcal{C}}$. The set \mathcal{Q} is comprised of the intersection points of \mathcal{C} with the other algebraic components induced by \mathcal{P}' . Since \mathcal{C} itself is not part of the singular locus these are exactly the singular points on \mathcal{C} . Let Q_S and Q_T be two regular quadrics contained in \mathcal{P}' , the singular locus of \mathcal{P}' is defined as $\{p \in Q_S \cap Q_T \mid \nabla Q_S(p) = \nabla Q_T(p)\}$, this is well defined due to the fact that Q_S and Q_T are regular. By the fact that $\nabla Q_S(p) = S \cdot p$ we can define \mathcal{Q} as follows:

$$\mathcal{Q} := \{p \in \mathcal{C} \mid Sp = Tp\}$$

In terms of the parameterizations $X_{\mathcal{C}}$ this is:

$$\mathcal{Q} := \{X_{\mathcal{C}}(\xi) \mid SX_{\mathcal{C}}(\xi) = TX_{\mathcal{C}}(\xi), \xi \in \mathbb{P}^1\}$$

Now write $SX_{\mathcal{C}}(\xi)$ and $TX_{\mathcal{C}}(\xi)$ as follows:

$$\begin{aligned} [f_1(\xi), f_2(\xi), f_3(\xi), f_4(\xi)]^T &:= SX_{\mathcal{C}}(\xi) \\ [g_1(\xi), g_2(\xi), g_3(\xi), g_4(\xi)]^T &:= TX_{\mathcal{C}}(\xi) \end{aligned}$$

Then the parameter values for the points in \mathcal{Q} are given by the real roots of

$$\gcd(s_{12}(\xi), s_{13}(\xi), s_{14}(\xi), s_{23}(\xi), s_{24}(\xi), s_{34}(\xi)),$$

where $s_{ij}(\xi) = f_i(\xi)g_j(\xi) - f_j(\xi)g_i(\xi)$.

This results in the desired sequence $seq = \{p_1, \dots, p_m\}$ representing the points in \mathcal{Q} with respect to $X_{\mathcal{C}}$.

B. Multiplicities of Points on a Smooth Quartic

As discussed in Section 4 our overall algorithm does not require the multiplicity of the intersection points. Hence, Theorem 2 omits statements about the multiplicity of the

intersection points. The following theorem is given in order to close this gap and may be seen as an addendum to Theorem 2.

In what follows if $p(\xi)$ is a homogeneous polynomial in $\xi = (u, v)$ we denote by $p^{(i,j)}(\xi)$ the partial derivative $\frac{\partial^i p}{\partial u^j \partial v^{i-j}}$.

Theorem 3. Let f, τ_ε, g and $res \neq 0$ be defined as in Equations (3), (4), (5) and (6) respectively. Let ξ_0 denote a real root of the resultant such that $\Delta(\xi_0) > 0$ and let τ_ε be a valid parameterization for ξ_0 . Moreover, let m_0 denote the multiplicity of ξ_0 in res . Then:

- (1) If $s_{12}(\xi_0) \neq 0$ then ξ_0 corresponds to a unique real intersection point of multiplicity m_0 ;
- (2) Otherwise let $i_c > 0$ denote the smallest integer such that there exists an integer j with $s_{12}^{(i_c, j)}(\xi_0) \neq 0$.
 - (a) There are two real intersection points corresponding to ξ_0 with multiplicity i_c and $m_0 - i_c$ respectively.
 - (b) If $i_c \neq m_0 - i_c$, the point with multiplicity $m_0 - i_c$ is the one on arc $\mathbf{X}_\varepsilon(\xi)$, where $\varepsilon = \text{sign}(s_{12}^{(i_c, j)}) \text{sign}((a_1 s_{12} - 2a_2 s_{02})^{(i_c, j)})|_{\xi_0}$.

Proof. The first case does not necessitate much comment: there is a single real intersection point by Theorem 2 and it gets all the multiplicity.

Now to the second case. Observe that $\forall i \in \{i \in \mathbb{N} \mid i < i_c\}$ the i -th derivative

$$\begin{aligned} g(\xi, \tau_\varepsilon(\xi))^{(i, j)} &= (2a_1 s_{12} - 4a_2 s_{02} - 2\varepsilon s_{12} \sqrt{\Delta})^{(i, j)}, \\ &= (2a_1 s_{12} - 4a_2 s_{02})^{(i, j)} - (2\varepsilon s_{12} \sqrt{\Delta})^{(i, j)}, \\ &= (2a_1 s_{12} - 4a_2 s_{02})^{(i, j)} - 2\varepsilon s_{12}^{(i, j)} \sqrt{\Delta} - \dots - 2\varepsilon s_{12} (\sqrt{\Delta})^{(i, j)}, \end{aligned}$$

evaluated at ξ_0 , is independent of ε . That is, both points have at least multiplicity i_c . If $i_c = m_0 - i_c$ it is clear that both points have exactly multiplicity i_c . Otherwise, consider $g(\xi, \tau_\varepsilon(\xi))^{(i_c, j)}$ evaluated at ξ_0 :

$$g(\xi_0, \tau_\varepsilon(\xi_0))^{(i_c, j)} = (2a_1 s_{12} - 4a_2 s_{02})^{(i_c, j)}(\xi_0) - 2\varepsilon s_{12}^{(i_c, j)}(\xi_0) \sqrt{\Delta(\xi_0)}.$$

This is not independent from ε . And since $i_c \neq m_0 - i_c$ it is clear that

$$g(\xi_0, \tau_\varepsilon(\xi_0))^{(i_c, j)} = 0$$

has exactly one solution, namely

$$\varepsilon = \text{sign}(s_{12}^{(i_c, j)}) \text{sign}((a_1 s_{12} - 2a_2 s_{02})^{(i_c, j)})|_{\xi_0}.$$

□

Remark: In case $\tau_\varepsilon(\xi)$ is not a valid parameterization for ξ_0 a symmetric consideration for $\tilde{\tau}_\varepsilon(\xi)$ leads to:

$$g(\xi_0, \tilde{\tau}_\varepsilon(\xi_0))^{(i_c, j)} = (4a_0 s_{02} - 2a_1 s_{01})^{(i_c, j)}(\xi_0) - 2\varepsilon s_{01}^{(i_c, j)}(\xi_0) \sqrt{\Delta(\xi_0)}.$$

Hence, if $m_0 - i_c \neq i_c$ the point with multiplicity $m_0 - i_c$ is the one on arc $\mathbf{X}_\varepsilon(\xi)$, where $\varepsilon = \text{sign}((2a_0 s_{02} - a_1 s_{01})^{(i_c, j)}) \text{sign}(s_{01}^{(i_c, j)})|_{\xi_0}$. Note that at least one parameterization remains valid, see also Section 2.

References

- [1] J. Abbott, Quadratic interval refinement for real roots, poster at ISSAC 2006, <http://www.dima.unige.it/~abbott/>.
- [2] M. H. Austern, *Generic Programming and the STL*, Addison-Wesley, 1998.
- [3] J. L. Bentley, T. A. Ottmann, Algorithms for reporting and counting geometric intersections, *IEEE Trans. Comput.* C-28 (9) (1979) 643–647.
- [4] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, L. Kettner, K. Mehlhorn, J. Reichel, S. Schmitt, E. Schömer, N. Wolpert, Exacus: Efficient and exact algorithms for curves and surfaces, in: G. S. Brodal, S. Leonardi (eds.), *13th Annual European Symposium on Algorithms (ESA 2005)*, vol. 3669 of LNCS, European Association for Theoretical Computer Science (EATCS), Springer, Palma de Mallorca, Spain, 2005, pp. 155–166.
- [5] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, K. Mehlhorn, E. Schömer, A computational basis for conic arcs and Boolean operations on conic polygons, in: R. Möhring, R. Raman (eds.), *Proc. ESA 2002*, vol. 2461 of LNCS, Springer, Rome, Italy, 2002, pp. 174–186.
- [6] E. Berberich, M. Hemmer, L. Kettner, E. Schömer, N. Wolpert, An exact, complete and efficient implementation for computing planar maps of quadric intersection curves, in: *Proc. SoCG 2005*, ACM Press, New York, NY, USA, 2005, pp. 99–106.
- [7] E. Berberich, M. Kerber, M. Sagraloff, Exact geometric-topological analysis of algebraic surfaces, in: M. Teillaud, E. Welzl (eds.), *Proceedings of the 24th ACM Symposium on Computational Geometry*, ACM, College Park Maryland, USA, 2008, pp. 164–173.
- [8] E. Berberich, M. Sagraloff, A generic and flexible framework for the geometrical and topological analysis of (algebraic) surfaces, in: E. Haines, M. McGuire (eds.), *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling*, ACM, Stony Brook, USA, 2008, pp. 171–182.
- [9] H. Bieri, W. Nef, A sweep-plane algorithm for computing the volume of polyhedra represented in Boolean form, *Linear Algebra and its Applications* 53 (1983) 69–97.
- [10] T. J. Bromwich, *Quadratic forms and their classification by means of invariant factors*, Cambridge Tracks in Mathematics and Mathematical Physics, 1906.
- [11] H. Brönnimann, L. Kettner, S. Schirra, R. Veltkamp, Applications of the generic programming paradigm in the design of CGAL, in: M. Jazayeri, R. Loos, D. Musser (eds.), *Generic Programming—Proceedings of a Dagstuhl Seminar*, LNCS 1766, Springer-Verlag, 2000, pp. 206–217.
- [12] W. S. Brown, The subresultant PRS algorithm, *ACM Trans. Math. Softw.* 4 (3) (1978) 237–249.
- [13] G. E. Collins, Quantifier elimination for real closed fields by cylindrical algebraic decomposition, in: *Proc. 2nd GI Conf. on Automata Theory and Formal Languages*, vol. 6, LNCS, Springer, Berlin, 1975, pp. 134–183, reprinted with corrections in: B. F. Caviness and J. R. Johnson (eds.), *Quantifier Elimination and Cylindrical Algebraic Decomposition*, 85–121. Springer, 1998.
- [14] T. Culver, J. Keyser, M. Foskey, S. Krishnan, D. Manocha, ESOLID - A system for exact boundary evaluation, *Computer-Aided Design* 36 (2) (2004) 175–193.
- [15] T. Culver, J. Keyser, D. Manocha, Exact computation of the medial axis of a polyhedron, *Computer Aided Geometric Design* 21 (1) (2004) 65–98.
- [16] L. Dupont, D. Lazard, S. Lazard, S. Petitjean, Near-optimal parameterization of the intersection of quadrics: I+II+III, *J. of Symbolic Computation* 43 (3) (2008) 168–232.
- [17] A. Eigenwillig, M. Kerber, Exact and efficient 2D-arrangements of arbitrary algebraic curves, in: *Proc. 18th ACM-SIAM Sympos. Discrete Algorithms*, 2008, pp. 122–131.
- [18] A. Eigenwillig, M. Kerber, N. Wolpert, Fast and exact geometric analysis of real algebraic plane curves, in: C. W. Brown (ed.), *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, ACM, ACM, Waterloo, Ontario, Canada, 2007, pp. 151–158.

- [19] A. Eigenwillig, L. Kettner, W. Krandick, K. Mehlhorn, S. Schmitt, N. Wolpert, A Descartes algorithm for polynomials with bit-stream coefficients, in: V. G. Ganzha, E. W. Mayr, E. V. Vorozhtsov (eds.), *Computer Algebra in Scientific Computing: 8th International Workshop, CASC 2005*, vol. 3718 of LNCS, Springer, Kalamata, Greece, 2005, pp. 138–149.
- [20] A. Eigenwillig, L. Kettner, E. Schömer, N. Wolpert, Exact, efficient, and complete arrangement computation for cubic curves, *Computational Geometry* 35(1-2) (2006) 36–73.
- [21] M. J. Encarnacion, Computing GCDs of polynomials over algebraic number fields, *J. Symbolic Computation* 20 (1995) 299–313.
- [22] E. Flato, D. Halperin, I. Hanniel, O. Nechushtan, E. Ezra, The Design and Implementation of Planar Maps in CGAL, *The ACM Journal of Experimental Algorithmics* 5, also in LNCS Vol. 1668 (WAE '99), pages 154–168.
- [23] M. Granados, P. Hachenberger, S. Hert, L. Kettner, K. Mehlhorn, M. Seel, Boolean operations on 3D selective Nef complexes: Data structure, algorithms, and implementation, in: *ESA 2003*, LNCS 2832, Springer, 2003, pp. 654–666.
- [24] M. Greenberg, J. Harper, *Algebraic Topology: A First Course*, Benjamin-Cummings, Reading, MA, 1981.
- [25] D. Halperin, Arrangements, in: J. E. Goodman, J. O'Rourke (eds.), *Handbook of Discrete and Computational Geometry*, 2nd ed., chap. 24, Chapman & Hall/CRC, 2004, pp. 529–562.
- [26] M. Hemmer, Exact computation of the adjacency graph of an arrangement of quadrics, Ph.D. thesis, Johannes Gutenberg Universität Mainz (2008).
- [27] M. Hemmer, D. Hülse, Generic implementation of a modular GCD over algebraic extension fields, in: *25th European Workshop on Computational Geometry*, Université Libre de Bruxelles, Brussels, Belgium, 2009, p. 4.
- [28] T. Y. Lam, *The Algebraic Theory of Quadratic Forms*, W. A. Benjamin, MA, 1973.
- [29] L. Langemyr, S. McCallum, The computation of polynomial GCD's over an algebraic number field, *J. Symbolic Computation* 8 (1989) 429–448.
- [30] S. Lazard, L. M. Peñaranda, S. Petitjean, Intersecting quadrics: an efficient and exact implementation, in: *SCG '04: Proceedings of the Twentieth Annual Symposium on Computational geometry*, ACM, New York, NY, USA, 2004, pp. 419–428.
- [31] S. Limbach, Continued Work on the Computation of an Exact Arrangement of Quadrics, Master's thesis, Universität des Saarlandes (Dec. 2008).
- [32] S. McCallum, G. E. Collins, Local box adjacency algorithms for cylindrical algebraic decompositions, *Journal of Symbolic Computation* 33 (3) (2002) 321–342.
- [33] B. Mourrain, J. P. Tócourt, M. Teillaud, Sweeping of an arrangement of quadrics in 3D, *Computational Geometry: Theory and Applications* 30 (2005) 145–164, special issue, 19th European Workshop on Computational Geometry.
- [34] E. Schömer, J. Reichel, T. Warken, C. Lennerz, Efficient collision detection for curved solid objects, in: K. Lee, N. M. Patrikalakis (eds.), *Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications*, Association of Computing Machinery (ACM), ACM, Saarbrücken, Germany, 2002, pp. 321–328.
- [35] E. Schömer, N. Wolpert, An exact and efficient approach for computing a cell in an arrangement of quadrics, *Computational Geometry: Theory and Applications* 33 (2006) 65–97, special Issue on Robust Geometric Applications and their Implementations.
- [36] J. Stolfi, *Oriented Projective Geometry: A Framework for Geometric Computations*, Academic Press, New York, NY, 1991.
- [37] R. Wein, High level filtering for arrangements of conic arcs, in: *ESA 2002*, LNCS 2461, 2002, pp. 884–895.
- [38] D. Y. Y. Yun, On square-free decomposition algorithms, in: *SYMSAC '76: Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation*, ACM Press, New York, NY, USA, 1976, pp. 26–35.