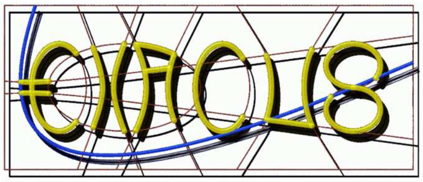
EXACUS: Efficient and Exact Algorithms for Curves and Surfaces ⋆

Eric Berberich[1], Arno Eigenwillig[1], Michael Hemmer[2], Susan Hert[3], Lutz Kettner[1], Kurt Mehlhorn[1], Joachim Reichel[1], Susanne Schmitt[1], Elmar Schömer[2], and Nicola Wolpert[1]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[2] Johannes-Gutenberg-Universität Mainz, Germany
[3] Serials Solutions, Seattle, WA., USA

Abstract. We present the first release of the EXACUS C++ libraries. We aim for systematic support of non-linear geometry in software libraries. Our goals are efficiency, correctness, completeness, clarity of the design, modularity, flexibility, and ease of use. We present the generic design and structure of the libraries, which currently compute arrangements of curves and curve segments of low algebraic degree, and boolean operations on polygons bounded by such segments.

1 Introduction



The EXACUS-project (*Efficient and Exact Algorithms for Curves and Surfaces*[4]) aims to develop efficient, exact (the mathematically correct result is computed), and complete (for all inputs) algorithms and implementations for low-degree non-linear geometry. Exact and complete methods are available in the algebraic geometry community, but they are not optimized for low-degree or large inputs. Efficient, but inexact and incomplete methods are available in the solid modeling community. We aim to show that exactness and completeness can be obtained at a moderate loss of efficiency. This requires theoretical progress in computational geometry, computer algebra, and numerical methods, and a new software basis. In this paper, we present the design of the EXACUS C++ libraries. We build upon our experience with CGAL [12, 22] and LEDA [25].

CGAL, the *Computational Geometry Algorithms Library*, is the state-of-the-art in implementing geometric algorithms completely, exactly, and efficiently. It deals mostly with linear objects and arithmetic stays within the rational numbers.

Non-linear objects bring many new challenges with them: (1) Even single objects, e.g., a single algebraic curve, are complex, (2) there are many kinds of degeneracies, (3) and point coordinates are algebraic numbers. It is not clear yet, how to best cope with these challenges. The answer will require extensive theoretical and experimental work.

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

[4] `http://www.mpi-inf.mpg.de/EXACUS/`

Therefore, we aim for a crisp and clear design, flexibility, modularity, and ease of use of our libraries.

The EXACUS C++ libraries compute arrangements of curves and curve segments of low algebraic degree, and boolean operations on polygons bounded by such segments. The functionality of our implementation is complete. We support arcs going to infinity and isolated points. We deal with all degeneracies, such as singularities and intersections of high multiplicity. We always compute the mathematically correct result. Important application areas are CAD, GIS and robotics. The recent Open Source release contains the full support for conics and conic segments, while existing implementations for cubic curves and a special family of degree four curves are scheduled for later releases. We work currently on extending our results to arrangements of general-degree curves in the plane [29] and quadric surfaces in space and boolean operations on them [4].

Here, we present the generic design and the structure of the EXACUS C++ libraries. The structure of the algorithms and algebraic methods is reflected in a layered architecture that supports experiments at many levels. We describe the interface design of several important levels. Two interfaces are improvements of interfaces in CGAL, the others are novelties. This structure and design has proven valuable in our experimental work and supports future applications, such as spatial arrangements of quadrics.

2 Background and Related Work

The theory behind EXACUS is described in the following series of papers: Berberich et al. [3] computed arrangements of conic arcs based on the LEDA [25] implementation of the Bentley-Ottmann sweep-line algorithm [2]. Eigenwillig et al. [10] extended the sweep-line approach to cubic curves. A generalization of Jacobi curves for locating tangential intersections is described by Wolpert [31]. Berberich et al. [4] recently extended these techniques to special quartic curves that are projections of spatial silhouette and intersection curves of quadrics and lifted the result back into space. Seidel et al. [29] describe a new method for general-degree curves. Eigenwillig et al. [9] extended the Descartes algorithm for isolating real roots of polynomials to bit-stream coefficients.

Wein [30] extended the CGAL implementation of planar maps to conic arcs with an approach similar to ours. However, his implementation is restricted to bounded curves (i.e. ellipses) and bounded arcs (all conic types), and CGAL's planar map cannot handle isolated points. He originally used Sturm sequences with separation bounds and switched now to the `Expr` number type of CORE [21] to handle algebraic numbers.

Emiris et al. [11] proposed a design for a support of non-linear geometry in CGAL focusing on arrangements of curves. Their implementation was limited to circles and had preliminary results for ellipsoidal arcs. Work on their kernel has continued but is not yet available. Their approach is quite different from ours: They compute algebraic numbers for both coordinates of an intersection point, while we need only the x-coordinate as algebraic number. On the other hand, they use statically precomputed Sturm sequences for algebraic numbers of degree up to four, an approach comparing favorably with our algebraic numbers. However, their algorithm for arrangement computations requires evaluations of the curve at points with algebraic numbers as coordinates.

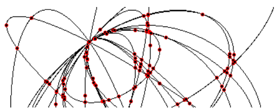
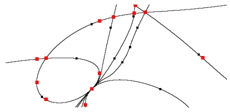
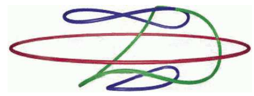
<table>
<tr><td colspan="2">
CONIX(CnX)</td><td colspan="2">
CUBIX(CbX)</td><td colspan="2">
QUADRIX(QdX)</td></tr>
<tr><td colspan="6">SWEEPX (SoX)</td></tr>
<tr><td colspan="6">NUMERIX (NiX)</td></tr>
<tr><td colspan="6">Library Support (LIS)</td></tr>
<tr><td>BOOST</td><td>GMP</td><td>CORE</td><td>LEDA</td><td>CGAL</td><td>QT</td></tr>
</table>

Fig. 1. Layered architecture of the EXACUS C++ libraries (with their name abbreviations).

The libraries MAPC [23] and ESOLID [8] deal with algebraic points and curves and with low-degree surfaces, respectively. Both libraries are not complete, e.g., require surfaces to be in general position.

Computer algebra methods, based on exact arithmetic, guarantee correctness of their results. A particularly powerful method related to our problems is Cylindrical Algebraic Decomposition invented by Collins [6] and subsequently implemented (with numerous refinements). Our approach to curve and curve pair analyses can be regarded as a form of cylindrical algebraic decomposition of the plane in which the lifting phase has been redesigned to take advantage of the specific geometric setting; in particular, to keep the degree of algebraic numbers low.

3 Overview and Library Structure

The design of the 94.000 lines of source code and documentation of the EXACUS C++ libraries follows the *generic programming paradigm* with C++ templates, as it is widely known from the *Standard Template Library*, STL [1], and successfully used in CGAL [12, 5, 19]. We use *concepts*[5] from STL (iterators, containers, functors) and CGAL (geometric traits class, functors) easing the use of the EXACUS libraries. C++ templates provide flexibility that is resolved at compile-time and hence has no runtime overhead. This is crucial for the efficiency and flexibility at the number type level (incl. machine arithmetic), however, is insignificant at higher levels of our design. In particular, a large part of our design can also be realized in other paradigms and languages.

EXACUS uses several external libraries: BOOST (interval arithmetic), GMP and CORE (number types), LEDA (number types, graphs, other data structures, and graphical user interface), CGAL (number types and arrangements), and Qt (graphical user interface). Generic programming allows us to avoid hard-coding dependencies. For example, we postpone the decision between alternative number type libraries to the final application code.

[5] A *concept* is a set of syntactical and semantical requirements on a template parameter, and a type is called a *model* of a *concept* if it fulfills these requirements and can thus be used as an argument for the template parameter [1].

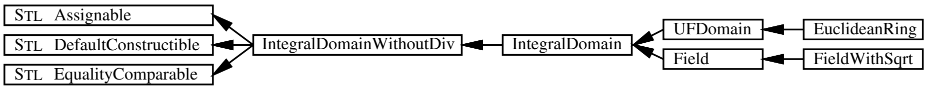


Fig. 2. The number type concepts in EXACUS. They refine three classical STL concepts.

We organized the EXACUS libraries in a layered architecture, see Figure 1, with external libraries at the bottom and applications at the top. In between, we have library support in LIS, number types, algebraic and numerical methods in NUMERIX, and the generic implementation of a sweep-line algorithm and a generic generalized polygon that supports regularized boolean operations on regions bounded by curved arcs in SWEEPX. Furthermore, SWEEPX contains *generic algebraic points and segments* (GAPS). It implements the generic and curve-type independent predicates and constructions for the sweep-line algorithm.

In this paper, we focus on the interface design between NUMERIX, SWEEPX, GAPS, and the applications. Descriptions of the application layer can be found in [3, 10, 4].

4 NUMERIX Library

The NUMERIX library comprises number types, algebraic constructions to build types from types, such as polynomials (over a number type), vectors, and matrices, and a tool box of algorithms solving linear systems (Gauss-Jordan elimination), computing determinants of matrices (Bareiss and division-free method of Berkowitz [26]), gcds, Sylvester and Bézout matrices for resultants and subresultants, isolating real roots of polynomials (Descartes algorithm), and manipulating algebraic numbers. We import the basic number types integer, rationals and (optionally) real expressions from LEDA [25], GMP [17] and CORE [21], or EXT [28]. We next discuss core aspects of NUMERIX.

Number Type Concepts and Traits Classes: We aim for the Real RAM model of computation. An effective realization must exploit the tradeoff between expressiveness and efficiency of different number type implementations. In EXACUS, we therefore provide a rich interface layer of number type concepts as shown in Figure 2. It allows us to write generic and flexible code with maximal reuse. All number types must provide the construction from small integers, in particular from 0 and 1. `IntegralDomain`, `UFDomain`, `Field`, and `EuclideanRing` correspond to the algebraic concepts with the same name. `FieldWithSqrt` are fields with a square root operator. The concept `IntegralDomainWithoutDiv` also corresponds to integral domains in the algebraic sense; the distinction results from the fact that some implementations of integral domains, e.g., `CGAL::MP_Float`, lack the (algebraically always well defined) integral division. A number type may be ordered or not; this is captured in the `RealComparable` concept, which is orthogonal to the other concepts. The fields $\mathbb{Q}$ and $\mathbb{Z}/p\mathbb{Z}$ are ordered and not ordered respectively.

The properties of number types are collected in appropriate traits classes. Each concrete number type `NT` knows the (most refined) concept to which it belongs; it is encoded in `NT_traits<NT>::Algebra_type`. The usual arithmetic and comparison operators are required to be realized via C++ operator overloading for ease of use.

The division operator is reserved for division in fields. All other unary (e.g., sign) and binary functions (e.g., integral division, gcd) must be models of the standard STL `AdaptableUnaryFunction` or `AdaptableBinaryFunction` concept and local to a traits class (e.g., `NT_traits<NT>::Integral_div`). This allows us to profit maximally from all parts in the STL and its programming style.

Design Rationale: Our interface extends the interface in CGAL [12, 5, 19, 22] that only distinguishes `EuclideanRing`, `Field`, and `FieldWithSqrt`. The finer granularity is needed for the domain of curves and surfaces, in particular, the algebraic numbers and algorithms on polynomials. We keep the arithmetic and comparison operators for ease of use. Their semantic is sufficiently standard to presume their existence and compliance in existing number type libraries for C++. For the other functions we prefer the trouble-free and extendible traits class solution; this was suggested by the name-lookup and two-pass template compilation problems experienced in CGAL.

Polynomials: The class `Polynomial<NT>` realizes polynomials with coefficients of type `NT`. Depending on the capabilities of `NT`, the polynomial class adapts internally and picks the best implementation for certain functions (see below for an example). The number type `NT` must be at least of the `IntegralDomainWithoutDiv` concept. For all operations involving division, the `IntegralDomain` concept is required. Some functions require more, for example, the gcd-function requires `NT` to be of the `Field` or `UFDomain` concept. In general, the generic implementation of the polynomial class encapsulates the distinction between different variants of functions at an early level and allows the reuse of generic higher-level functions.

The `Algebra_type` of `Polynomial<NT>` is determined via template meta-programming by the `Algebra_type` of `NT`. It remains the same except in the case of `EuclideanRing`, which becomes a `UFDomain`, and both field concepts, which become an `EuclideanRing`. `NT` can itself be an instance of `Polynomial`, yielding a recursive form of multivariate polynomials. In our applications, we deal only with polynomials of small degree (so far at most 16) and a small number of variables (at most 3) and hence the recursive construction is appropriate. Some convenience functions hide the recursive construction in the bi- and trivariate case.

We use polynomial remainder sequences (PRS) to compute the gcd of two polynomials (uni- or multivariate). Template meta-programming is used to select the kind of PRS: Euclidean PRS over a field and Subresultant PRS (see [24] for exposition and history) over a UFD. An additional meta-programming wrapper attempts to make the coefficients fraction-free, so that gcd computation over the field of rational numbers is actually performed with integer coefficients and the Subresultant PRS (this is faster). Gcd computations are relatively expensive. Based on modular resultant computation, we provide a fast one-sided probabilistic test for coprimality and squarefreeness that yields a significant speedup for non-degenerate arrangement computations [18]. We offer several alternatives for computing the resultant of two polynomials: via the Subresultant PRS or as determinant of the Sylvester or Bézout matrix [15, ch. 12]. Evaluating a Bézout determinant with the method of Berkowitz [26] can be faster than the Subresultant PRS for bivariate polynomials of small degrees over the integers. The Bézout determinant can also express subresultants, see e.g. [16, 20].

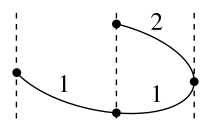
Algebraic Numbers: The `Algebraic_real` class represents a real root of a square-free polynomial. The representation consists of the defining polynomial and an isolating interval, which is an open interval containing exactly one root of the polynomial. Additionally, the polynomial is guaranteed to be non-zero at the endpoints of the interval. As an exception, the interval can collapse to a single rational value when we learn that the root has this exact rational value.

We use the Descartes Method [7, 27] to find isolating intervals for all real roots of a polynomial. We have a choice of different implementations, of which Interval Descartes [9] and Sturm sequences are ongoing work. We cross link all real roots of a polynomial, such that, for example, if one number learns how to factorize the defining polynomial, all linked numbers benefit from that information and simplify their representation. We learn about such factorizations at different occasions in our algorithms, e.g., if we find a proper common factor in the various gcd computations.

Interval refinement is central to the Descartes Method, to the comparison of algebraic numbers, and to some algorithms in the application layer [10]. We expose the refinement step in the following interface: `refine()` bisects the isolating interval. `strong_refine(Field m)` refines the isolating interval until m is outside of the closed interval. `refine_to(Field lo, Field hi)` intersects the current interval with the isolating interval (*lo, hi*). The global function `refine_zero_against` refines an isolating interval against all roots of another polynomial. The member function `x.rational_between(y)` returns a rational number between the two algebraic reals x and y. Here, it is desirable to pick a rational number of low-bit complexity.

5 SWEEPX Library

The SWEEPX library provides a generic sweep-line algorithm and generalized polygon class that supports regularized boolean operations on regions bounded by curved arcs. We based our implementation on the sweep-line algorithm for line-segments from LEDA [25], which handles all types of degeneracies in a simple unified event type. We extended the sweep-line approach with a new algorithm to reorder curve segments continuing through a common intersection point in linear time [3] and with a *geometric traits class* design for curve segments, which we describe in more detail here.



As input, we support full curves and, for conics, also arbitrary segments of curves, but both will be preprocessed into potentially smaller *sweepable segments* suitable for the algorithm. A *sweepable segment* is x-monotone, has a constant arc number in its interior (counting without multiplicities from bottom to top), and is free of one-curve events (explained in the next section) in its interior.

The *geometric traits class* defines the interface between the generic sweep-line algorithm and the actual geometric operations performed in terms of geometric types, predicates, and constructions, again realized as functors.[6] The idea of geometric traits classes goes back to CGAL [12, 5]. Our set of requirements is leaner than the geometric traits class in CGAL's arrangement and planar map classes [13], however, feedback from Emiris et al. [11] and our work is leading to improvements in CGAL's interface.

[6] We omit here the access functions for the functors for brevity, see [12] for the details.

A geometric traits class following the `CurveSweepTraits_2` concept needs a type `Point_2`, representing end-points as well as intersection points, and a type `Segment_2`, representing curve segments. In STL terminology, both types have to be default-constructible and assignable, and are otherwise opaque types that we manipulate only through the following predicates, accessors, and constructions. Observe the compactness of the interface.

Predicates:

- `Compare_xy_2` and `Less_xy_2` lexicographically compare two points, the former with a three-valued return type and the latter as a predicate.
- `Is_degenerate_2` returns true if a segment consists only of one point.
- `Do_overlap_2` tells whether two segments have infinitely many points in common (i.e., intersect in a non-degenerate segment).
- `Compare_y_at_x_2` returns the vertical placement of a point relative to a segment.
- `Equal_y_at_x_2` determines if a point lies on a segment (equivalent to testing `Compare_y_at_x_2` for equality, but may have a more efficient implementation).
- `Multiplicity_of_intersection` computes the multiplicity of an intersection point between two segments. It is used in the linear-time reordering of segments and hence used only for non-singular intersections.
- `Compare_y_right_of_point` determines the ordering of two segments just after they both pass through a common point. It is used for the insertion of segments starting at an event point.

Accessors and Constructions:

- `Source_2` and `Target_2` return the source and target point of a curve segment.
- `Construct_segment_2` constructs a degenerate curve segment from a point.
- `New_endpoints_2` and `New_endpoints_opposite_2` replace the endpoints of a curve segment with new representations and return this new segment. The latter functor also reverses the orientation of the segment. They are used in the initialization phase of the sweep-line algorithm where equal end-points are identified and where the segments are oriented canonically from left to right [25].
- `Intersect_2` constructs all intersection points between two segments in lexicographical order.
- `Intersect_right_of_point_2` constructs the first intersection point between two segments right of a given point. It is used only for validity checking or if caching is disabled.

We also offer a geometric traits class for CGAL's planar map and arrangement classes, actually implemented as a thin adaptor for GAPS described next. Thus, we immediately get traits classes for CGAL for all curve types available in EXACUS. Our traits class supports the incremental construction and the sweep-line algorithm in CGAL. This solution is independent of LEDA.

6 Generic Algebraic Points and Segments (GAPS)

The generic point and segment types are essentially derived from the idea of a cylindrical algebraic decomposition of the plane and a number type for *x*-coordinates, while

y-coordinates are represented implicitly with arc numbers on supporting curves. Studying this more closely [10], one can see that all predicates and constructions from the previous section can be reduced to situations with one or two curves only.

The software structure is now as follows: The application libraries provide a curve analysis and a curve pair analysis, which depend on the actual curve type. Based on these analyses, the Gaps part of SweepX implements the generic algebraic points and segments with all predicates and constructions needed for the sweep-line algorithm, which is curve-type independent generic code. Small adaptor classes present the Gaps implementation in suitable geometric traits classes for the sweep-line implementation in SweepX or the CGAL arrangement.

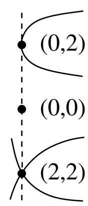
Gaps requires two preconditions; squarefreeness of the defining polynomial and coprimality of the defining polynomials in the curve pair analysis. We allow to defer the check to the first time at which any of the analysis functions is called. The application libraries may impose further restrictions on the choice of a coordinate system. If a violation is detected, an exception must be thrown that is caught outside SweepX. The caller can then remove the problem and restart afresh. ConiX does not restrict the choice of coordinate system. CubiX and QuadriX assume a generic coordinate system to simplify the analysis, see [10, 4] for details.

We continue with curve and curve pair analysis, and how we handle unboundedness.

Curve Analysis: A planar curve is defined as zero set of a bivariate squarefree integer polynomial f. We look at the curve in the real affine plane per x-coordinate and see arcs at varying y-values on the fictitious vertical line $x = x_0$ at each x-coordinate x_0. At a finite number of events, the number and relative position of arcs changes. Events are x-extreme points, singularities (such as self-intersections and isolated points) and vertical asymptotes (line components, poles).

The result of the curve analysis is a description of the curve's geometry at these events and over the open intervals of x-coordinates between them. This description is accessible through member functions in the `AlgebraicCurve_2` concept. It consists of the number of events, their x-coordinates given as algebraic numbers, the number of arcs at events and between them, an inverse mapping from an x-coordinate to an event number, and it gives details for each event in objects of the `Event1_info` class.

The `Event1_info` class describes the topology of a curve over an event x-coordinate x_0. The description answers four questions on the local topology of the curve over the corresponding x-coordinate: (1) How many real arcs (without multiplicities) intersect the (fictitious) vertical line $x = x_0$? (2) Does the curve have a vertical line component at x_0? (3) Is there a vertical asymptote at x_0, and how many arcs approach it from the left and right? (4) For any intersection point of the curve with the vertical line $x = x_0$, how many real arcs on the left and right are incident to it? In the example figure, (0,0) stands for an isolated point, (0,2) for a left x-extreme point or cusp, and (2,2) for a singular point.

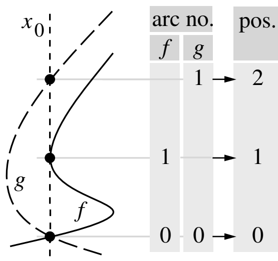


Curve Pair Analysis: We study ordered pairs of coprime planar algebraic curves f and g. Similar to the curve analysis, we look at the pair of curves in the real affine plane per x-coordinate and see arcs of both curves at varying y-values above each x-coordinate. At a finite number of events, the number and relative position of arcs changes: There are intersections at which arcs of both curves run together, there are one-curve events, and there are events that are both.

The result of the curve pair analysis is a description of the curves' geometry at these events and over the open intervals of x-coordinates between them. This description is accessible through member functions in the `AlgebraicCurvePair_2` concept. It consists of various mappings of event numbers to indices of roots in the resultants $\mathrm{res}(f, f_y)$, $\mathrm{res}(g, g_y)$, and $\mathrm{res}(f, g)$, to x-coordinates and reverse, and it gives details for each event in objects of the `Event2_slice` class.

An `Event2_slice` is a unified representation of the sequence of arcs over some x-coordinate x_0, be there an event or not [10, 4]. A slice of a pair (f, g) at x_0 describes the order of points of f and g lying on the fictitious vertical line $x = x_0$, sorted by y-coordinate. Points are counted without multiplicities.

Let the *arc number* of a point on f be its rank in the order amongst all points of f on $x = x_0$, and respectively for points on g. Let the *position number* of a point on f or g be its rank in the order amongst all points of $f \cup g$ on $x = x_0$. All ranks are counted, starting at 0, in increasing order of y-coordinates. An important function in the slice representation is the mapping of arc numbers, used in the point and sweepable segment representations, to position numbers, used to compare y-coordinates of points on different curves f and g. For the example in the figure, this mapping will tell us that the second point of g is the third amongst all points (w.r.t. y-coordinates), i.e., it maps arc number 1 to position 2.



Unbounded Curves and Segments: The implementation is not restricted to bounded curves and segments. We define symbolic "endpoints" at infinity, such that unbounded curves and segments, e.g., the hyperbola $xy - 1 = 0$ or any part of it, can be included in a uniform representation of sweepable segments. We use two techniques, compactification and symbolic perturbation: We add minus- and plus-infinity symbolically to the range of x-coordinate values. Then, we perturb x-coordinates to represent "endpoints" of curves that approach a pole and unbounded "endpoints" of vertical lines and rays. Let $\varepsilon > 0$ be an infinitesimal symbolic value. We perturb the x coordinate of an endpoint of a curve approaching a pole from the left by $-\varepsilon$, the lower endpoint of a vertical line by $-\varepsilon^2$, the upper endpoint of a vertical line by ε^2, and an endpoint of a curve approaching a pole from the right by ε. We obtain the desired lexicographic order of event points.

We also extend the curve analyses with the convention that for x-coordinates of minus- or plus-infinity we obtain the number and relative position of arcs before the first or after the last event, respectively. Note that this is not "infinity" in the sense of projective geometry.

7 Evaluation and Conclusion

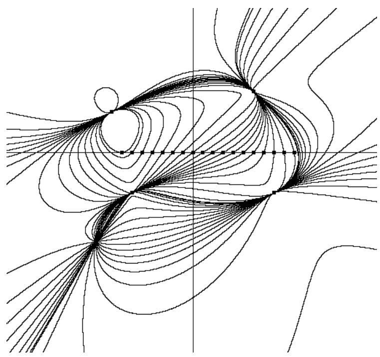
The libraries are extensively tested and benchmarked with data sets showing runtime behavior, robustness, and completeness. In particular for completeness we manufactured test data sets for all kinds of degeneracies [3, 10, 4]. A preliminary comparison [14] between Wein [30], Emiris et al. [11], and us showed that our implementation was the only robust and stable and almost always the fastest implementation at that time. In particular, the other implementations could not handle large random instances, degenerate instances of several ellipses intersecting in the same point or intersecting tangentially, and instances of ellipses with increasing bit-size of the coefficients. A new comparison is planned when the other implementations have stabilized.

The report [14] also contains a comparison of our sweep-line algorithm with the CGAL sweep-line algorithm used for the planar map with intersections. We briefly summarize the result in the table that shows running times in seconds of both implementations measured for three random instances of cubic curves of 30, 60, and 90 input curves respectively on a Pentium IIIM at 1.2 GHz, Linux 2.4, `g++` 3.3.3, with `-O2 -DNDEBUG`, LEDA 4.4.1, and CGAL 3.0.1.

Data set	Segs	Vertices	Halfedges	SoX	CGAL
random30	266	2933	11038	6.7	8.0
random60	454	11417	44440	27.7	34.6
random90	680	26579	104474	67.5	81.2

Profiling the executions on the “random30” instance exhibits a clear difference in the number of predicate calls. The dominant reason for this is certainly reordering by comparison (CGAL) as opposed to reordering by intersection multiplicities (SoX).

Additionally, we report the running time of a hand-constructed instance: a pencil of 18 curves intersecting in five distinct points, in four of them with multiplicity 2, and nowhere else. Here `SoX::sweep_curves()` can exhibit the full benefit of linear-time reordering; 1.7 seconds for SoX and 4.3 seconds for CGAL.



All experiments show an advantage for our implementation. We took care not to use examples that would penalize the CGAL implementation with the interface mapping to the CUBIX implementation (see [14]), but the implementations are too complex to allow a definitive judgment on the relative merits of the two implementations based on above numbers.

We conclude with a few remarks on efficiency: Arithmetic is the bottleneck, so we use always the simplest possible number type, i.e., integers where possible, then rationals, LEDA real or CORE, and algebraic numbers last. However, using LEDA reals naively, for example, to test for equality or in near equality situations, can be quite costly, which happened in the first CONIX implementation, and using algebraic numbers or a number type that extends rationals with one or two fixed roots instead can be beneficial. The released CONIX implementation has been improved considerably following the ideas developed for the CUBIX implementation. Furthermore, we use caching of the curve analysis and we compute all roots of a polynomial at once. All roots of the same polynomial are cross linked, and if one root happens to learn about a factorization of the polynomial, all other roots will also benefit from the simplified representation. We use modular arithmetic as a fast filter test to check for inequality.

So far we have almost exclusively worked with exact arithmetic and optimized its runtime. Floating-point filters have been only used insofar that they are integral part of some number types. More advanced filters are future work.

We use the sweep-line algorithm for computing arrangements, but since arithmetic is the obvious bottleneck, it might pay off to consider incremental construction with its lower algebraic degree in the predicates and the better asymptotic runtime.

CGAL plans a systematic support for non-linear geometry. We contribute to this effort with our design experience and implementations presented here.

Acknowledgements

We would like to acknowledge contributions by Michael Seel, Evghenia Stegantova, Dennis Weber, and discussions with Sylvain Pion.

References

1. M. H. Austern. *Generic Programming and the STL.* Addison-Wesley, 1998.
2. J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, September 1979.
3. E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, K. Mehlhorn, and E. Schömer. A computational basis for conic arcs and boolean operations on conic polygons. In *ESA 2002, LNCS 2461*, pages 174–186, 2002.
4. E. Berberich, M. Hemmer, L. Kettner, E. Schömer, and N. Wolpert. An exact, complete and efficient implementation for computing planar maps of quadric intersection curves. In *Proc. 21th Annu. Sympos. Comput. Geom.*, pages 99–106, 2005.
5. H. Brönnimann, L. Kettner, S. Schirra, and R. Veltkamp. Applications of the generic programming paradigm in the design of CGAL. In M. Jazayeri, R. Loos, and D. Musser, editors, *Generic Programming—Proceedings of a Dagstuhl Seminar*, LNCS 1766, pages 206–217. Springer-Verlag, 2000.
6. G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proc. 2nd GI Conf. on Automata Theory and Formal Languages*, volume 6, pages 134–183. LNCS, Springer, Berlin, 1975. Reprinted with corrections in: B. F. Caviness and J. R. Johnson (eds.), Quantifier Elimination and Cylindrical Algebraic Decomposition, 85–121. Springer, 1998.
7. G. E. Collins and A.-G. Akritas. Polynomial real root isolation using Descartes' rule of sign. In *SYMSAC*, pages 272–275, 1976.
8. T. Culver, J. Keyser, M. Foskey, S. Krishnan, and D. Manocha. Esolid - a system for exact boundary evaluation. *Computer-Aided Design (Special Issue on Solid Modeling)*, 36, 2003.
9. A. Eigenwillig, L. Kettner, W. Krandick, K. Mehlhorn, S. Schmitt, and N. Wolpert. A Descartes algorithm for polynomials with bit-stream coefficients. In *Proc. 8th Int. Workshop on Computer Algebra in Scient. Comput. (CASC)*, LNCS. Springer, 2005. to appear.
10. A. Eigenwillig, L. Kettner, E. Schömer, and N. Wolpert. Complete, exact, and efficient computations with cubic curves. In *Proc. 20th Annu. Sympos. Comput. Geom.*, pages 409–418, 2004. accepted for Computational Geometry: Theory and Applications.
11. I. Z. Emiris, A. Kakargias, S. Pion, M. Teillaud, and E. P. Tsigaridas. Towards and open curved kernel. In *Proc. 20th Annu. Sympos. Comput. Geom.*, pages 438–446, 2004.
12. A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL, the computational geometry algorithms library. *Softw. – Pract. and Exp.*, 30(11):1167–1202, 2000.
13. E. Flato, D. Halperin, I. Hanniel, O. Nechushtan, and E. Ezra. The design and implementation of planar maps in CGAL. *ACM Journal of Experimental Algorithmics*, 5, 2000. Special Issue, selected papers of the Workshop on Algorithm Engineering (WAE).

14. E. Fogel, D. Halperin, R. Wein, S. Pion, M. Teillaud, I. Emiris, A. Kakargias, E. Tsigaridas, E. Berberich, A. Eigenwillig, M. Hemmer, L. Kettner, K. Mehlhorn, E. Schömer, and N. Wolpert. Preliminary empirical comparison of the performance of constructing arrangements of curved arcs. Technical Report ECG-TR-361200-01, Tel-Aviv University, INRIA Sophia-Antipolis, MPI Saarbrücken, 2004.
15. I. M. Gelfand, M. M. Kapranov, and A. V. Zelevinsky. *Discriminants, Resultants and Multidimensional Determinants*. Birkhäuser, Boston, 1994.
16. R. N. Goldman, T. W. Sederberg, and D. C. Anderson. Vector elimination: A technique for the implicitization, inversion, and intersection of planar parametric rational polynomial curves. *CAGD*, 1:327–356, 1984.
17. T. Granlund. *GNU MP, The GNU Multiple Precision Arithmetic Library, version 2.0.2*, 1996.
18. M. Hemmer, L. Kettner, and E. Schömer. Effects of a modular filter on geometric applications. Technical Report ECG-TR-363111-01, MPI Saarbrücken, 2004.
19. S. Hert, M. Hoffmann, L. Kettner, S. Pion, and M. Seel. An adaptable and extensible geometry kernel. In *Proc. 5th Workshop on Algorithm Engineering (WAE'01)*, LNCS 2141, pages 76–91, Arhus, Denmark, August 2001. Springer-Verlag.
20. X. Hou and D. Wang. Subresultants with the Bézout matrix. In *Proc. Fourth Asian Symp. on Computer Math. (ASCM 2000)*, pages 19–28. World Scientific, Singapore New Jersey, 2000.
21. V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. A core library for robust numeric and geometric computation. In *Proc. 15th Annu. Sympos. Comput. Geom.*, pages 351–359, 1999.
22. L. Kettner and S. Näher. Two computational geometry libraries: LEDA and CGAL. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Disc. and Comput. Geom.*, pages 1435–1463. CRC Press, second edition, 2004.
23. J. Keyser, T. Culver, D. Manocha, and Shankar Krishnan. MAPC: A library for efficient and exact manipulation of algebraic points and curves. In *Proc. 15th Annu. Sympos. Comput. Geom.*, pages 360–369, 1999.
24. R. Loos. Generalized polynomial remainder sequences. In B. Buchberger, G. E. Collins, and R. Loos, editors, *Computer Algebra: Symbolic and Algebraic Computation*, pages 115–137. Springer, 2nd edition, 1983.
25. K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 2000.
26. G. Rote. Division-free algorithms for the determinant and the pfaffian: algebraic and combinatorial approaches. In H. Alt, editor, *Computational Discrete Mathematics*, pages 119–135. Springer-Verlag, 2001. LNCS 2122.
27. F. Rouillier and P. Zimmermann. Efficient isolation of polynomial's real roots. *J. Comput. Applied Math.*, 162:33–50, 2004.
28. S. Schmitt. The diamond operator – implementation of exact real algebraic numbers. In *Proc. 8th Internat. Workshop on Computer Algebra in Scient. Comput. (CASC 2005)*, LNCS. Springer, 2005. to appear.
29. R. Seidel and N. Wolpert. On the exact computation of the topology of real algebraic curves. In *Proc. 21th Annual Symposium on Computational Geometry*, pages 107–115, 2005.
30. R. Wein. High level filtering for arrangements of conic arcs. In *ESA 2002, LNCS 2461*, pages 884–895, 2002.
31. N. Wolpert. Jacobi curves: Computing the exact topology of arrangements of non-singular algebraic curves. In *ESA 2003, LNCS 2832*, pages 532–543, 2003.