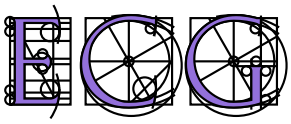
ECG

IST-2000-26473

Effective Computational Geometry for Curves and Surfaces



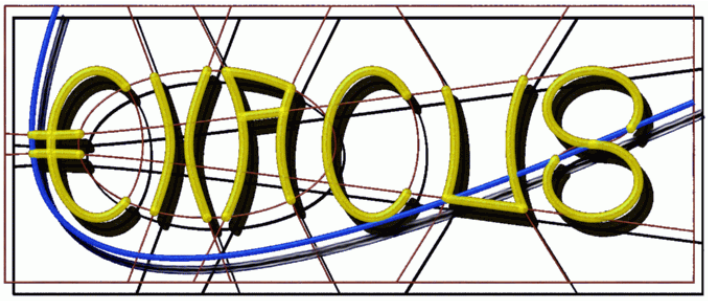
ECG Technical Report No. : *ECG-TR-361200-02*

EXACUS: Efficient and Exact Algorithms for Curves and Surfaces

Eric Berberich
Arno Eigenwillig
Michael Hemmer
Susan Hert
Lutz Kettner
Kurt Mehlhorn
Joachim Reichel
Susanne Schmitt
Elmar Schömer
Dennis Weber
Nicola Wolpert

Deliverable: 36 12 00 (new, item 02)
Site: MPI
Month: 36

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



EXACUS: Efficient and Exact Algorithms for Curves and Surfaces*

http://www.mpi-sb.mpg.de/projects/EXACUS/

Eric Berberich Arno Eigenwillig Michael Hemmer Susan Hert
Lutz Kettner Kurt Mehlhorn Joachim Reichel Susanne Schmitt
Elmar Schömer Dennis Weber Nicola Wolpert

Max-Planck-Institut für Informatik, Saarbrücken, Germany

July 15, 2005

Abstract

We present the first release of the C++ libraries of the EXACUS project of the Max-Planck-Institut für Informatik. We explain the structure of the libraries and the software design for the numerics library and the sweep-line algorithm library in more detail.

Overview

Since April 2002, EXACUS is a project at the Algorithms and Complexity Group (AG1) of the Max-Planck-Institut für Informatik. We work on a set of C++ libraries for efficient and exact algorithms for curves and surfaces. This work is part of the European Union's ECG project (*Effective Computational Geometry for Curves and Surfaces*).

EXACUS is a collection of C++ libraries, see Figure 1 for their layered architecture and a brief description. Our design follows the generic programming paradigm with C++ templates similar to well established design principles, for example, in the STL [MS94, Aus98] and in CGAL [FGK+00, BKSV00].

We give more details for the NUMERIX and the SWEEPX libraries below. The CONIX library contains the algorithm described in [BEH+02b] and ECG-TR-122103-01 [BEH+02a], but considerably improved with ideas developed for the CUBIX library as described in [EKSW04] and ECG-TR-182202-01 [ESW02]. The improvements are a factor of ten and more in the running time.

The libraries consist currently of about 75000 lines of code including the documentation that is embedded in the C++ source code. We use DOXYGEN to create the reference documentation. Since generic programming is not very well supported with DOXYGEN, for example, documenting requirements on template parameters, we use our own small post-processing script to document concepts similar to classes in DOXYGEN.

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

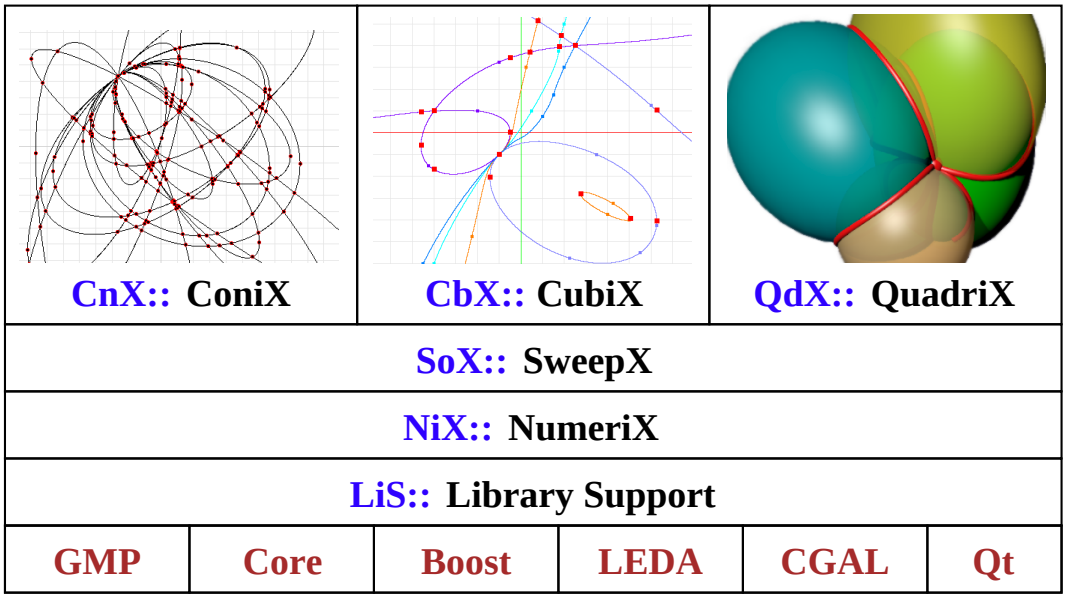


Figure 1: Library layers of the EXACUS project: At the bottom we have the external libraries that we detect and can use in EXACUS. Library Support provides the foundation, such as configuration and memory management. NUMERIX adds number type support, symbolic algebra and numerical algorithms. SWEEPX contains a generic sweep line algorithm suitable for segments of all type of curve arcs and boolean operations based on it. The top layer contains the three applications pursued so far; CONIXand CUBIXare applications for curves in the plane. QUADRIX extends our work to arrangements of quadric surfaces in space. This first EXACUS software release contains Library Support, NUMERIX, SWEEPX, and CONIX.

We use CVS for version control and distributed collaboration. Configuration and build management is done with the Gnu family of tools `autoconf`, `automake`, and `libtool`. The supported compiler is `g++` starting with version 3.1 and the supported platforms are Linux and Solaris. We strive for C++ Standard conformance of our code, but experience shows that porting to more compilers and platforms can sometimes be easy and sometimes not.

We detect a couple of other libraries during configuration. EXACUS does not depend on these libraries in various core parts of its functionality. However, in other parts, we did not reinvent the wheel and depend on existing implementations. Generic programming allows us to stay flexible and postpone the decision between several alternatives to the final user application code, for example, the choice of number types. The other libraries are: LEDA for its number types, the graph data structure and others in the SWEEPX library, and a window for visualization in the CONIX applications. CGAL for its number types, kernel geometry, and the planar map, just recently supported from the CONIX predicates with a new geometric traits class. Qt for visualization of the CUBIX applications. GMP and CORE for their number types. BOOST for the interval arithmetic.

We run an automatic test-suite daily from CVS based on our own scripts. We always maintain a stable revision tagged `PROVEN`. New contributions are tagged `CONJECTURE` and—after a successful run of the test-suite—they are marked `PROVEN` (after all our background is in theory ;-).

We have already released a prototype application based on the CONIX library[1] and another prototype application based on the CUBIX library[2]. This is now our source code release of the CONIX library and the layers below it. At the time of the writing of this report the release has *alpha* status and we restrict the access to ECG project partners only. A few items in the documentation and tutorial examples for CONIX will be added before making the *beta* release publicly available.

[1] `http://www.mpi-sb.mpg.de/EXACUS/ConiX/`
[2] `http://www.mpi-sb.mpg.de/EXACUS/CubiX/`

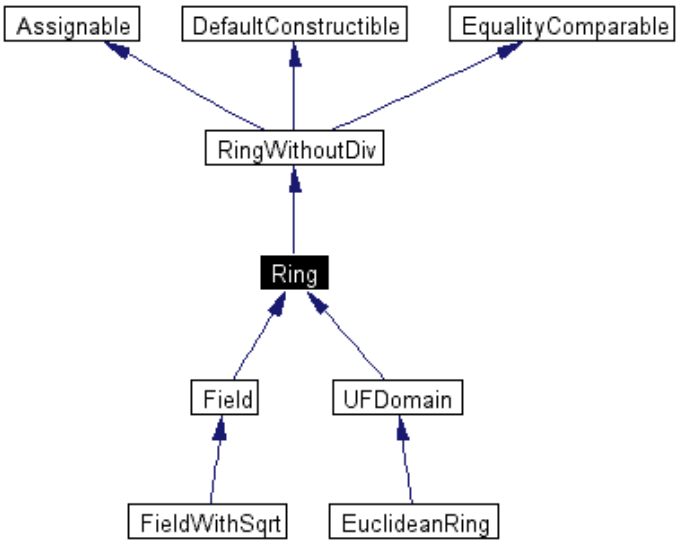


Figure 2: The number type concepts in Exacus. The arcs show the refinement relationship among number types. In particular, each number type is a refinement from the classical Stl concepts `Assignable`, `DefaultConstructible`, and `EqualityComparable`.

The license for the release will be the Q Public License version 1.0 (QPL) for most files with a few exceptions in SweepX regarding the sweep-line algorithm that is derived from work in Leda. For these exceptions we have special license conditions offered by Algorithmic Solutions Software GmbH[3].

The NumeriX Library

The NumeriX library contains number type support, symbolic algebra and numerical algorithms. Its primary goal is not a stand-alone general purpose library in its area, but to optimally support our applications. This gives us more freedom to specialize methods towards our applications and experiment with different options.

In particular we highlight here the number type concepts we distinguish and the various traits classes associated with number types. They allow us to write generic and flexible code with maximal reuse. As a model example we discuss the polynomial class and the algebraic numbers in a bit more detail. Some aspects of the NumeriX library that we just mention briefly here are linear algebra classes and algorithms for determinants, Sylvester and Bezout matrices for resultants and subresultants, linear system solver, the number type for quadratic extension fields, modular arithmetic, and floating point filter support.

Figure 2 shows the refinement relationship among the number type concepts that we distinguish in Exacus. The additional concept `RealComparable` is orthogonal to the other concepts.

The requirements on number type concepts come in three flavors: (1) We require an (explicit) constructor from small integer values ranging from -128 to $+127$. It is used in particular to create the zero and the one element. (2) We require suitable operator overloading for the different arithmetic and comparison operators. We reserve the `operator/()` for the division without remainder in a field. Other divisions, such as division with remainder, are provided in the third set of requirements. (3) All other requirements are local to the `NiX::NT_traits<NT>` class that provides suitable specializations for each number type `NT`. For example, `typename NiX::NT_traits<NT>::Algebra_type` refers to a tag that encodes the most refined concept that the number type `NT` fulfills, e.g., `NiX::Ring_tag` if it is a model of a `Ring`. Another example is `typename NiX::NT_traits<NT>::Integral_div`, which must be a model of the standard Stl `AdaptableBinaryFunction` concept that implements the integral division for that number type `NT`.

The number type concepts are in detail:

[3] `http://www.algorithmic-solutions.com/`

- **RealComparable:** (Orthogonal to the other number type concepts) Concept comprising comparisons and related operations for number types representing real numbers. Requires that `NiX::NT_traits<NT>` contains suitable functors `Abs`, `Sign`, and `Compare` for a three-valued comparison result, and all the comparison and equality operators for the type `NT`.

- **RingWithoutDiv:** The most basic number type concept; a ring with 0, 1, + , *, that represents elements of an integral domain, i.e., a commutative ring with unity free of zero divisors. A model of this concept is required to have a constructor from small integer values and suitable operators.

- **Ring:** Concept of a ring with an integral division operation provided through the `AdaptableBinaryFunction` functor `NiX::NT_traits::Integral_div`.

- **UFDomain:** The ring is a unique factorization domain (a.k.a. UFD or factorial ring), meaning that every non-zero non-unit element has a factorization into irreducible elements (a.k.a. prime elements) that is unique up to order and up to multiplication by invertible elements (units). (An irreducible element is a non-unit ring element that cannot be factored further into two non-unit elements.) In a UFD, any two elements, not both zero, possess a greatest common divisor (gcd). It is computed by the `AdaptableBinaryFunction` functor `NiX::NT_traits::Gcd`.

- **EuclideanRing:** The ring affords a suitable notion of minimality of remainders such that given x and $y \neq 0$ we obtain an (almost) unique solution to $x = qy + r$ by demanding that a solution (q, r) is chosen to minimize r. In particular, r is chosen to be 0 if possible. The most prominent example of an Euclidean ring are the integers. Whenever both x and y are positive, then it is conventional to choose the smallest positive remainder r. In other cases, there seems to be no universally observed convention on how to choose the sign. (In particular, the ISO C++ Standard fixes none for the modulo operation `%` on the builtin integral types.) This concept requires that `NiX::NT_traits<NT>` contains suitable functors `Div`, `Mod`, and `Div_mod` where the latter returns both, (q, r), together.

- **Field:** Concept of a field where every non-zero element has a multiplicative inverse, so, `NiX::NT_traits::Integral_div` is defined for any divisor $\neq 0$. This functor is now also required to be available with the global `operator/`.

- **FieldWithSqrt:** Concept for the field of algebraic expressions (FAE) limited to real square root expressions. The corresponding `AdaptableUnaryFunction` functor is `NiX::NT_traits<NT>::Sqrt`.

The requirements on the small integer constructor and the operator definitions are fulfilled for all suitable number types in our context that we know of. The `NiX::NT_traits` class requirements are easily provided for existing and new number types.

Realistically, we have currently only two main branches of number types that can fill our taxonomy of number types (in particular the `FieldWithSqrt` concept). These are the number types in the LEDA library [MN00], and the number types in GMP [Gra96] plus the CORE library [KLPY99]. We support both branches and provide the `NiX::NT_traits` class specializations.

In addition to the `NiX::NT_traits` class, we provide a couple of other traits classes, also in the namespace `NiX::`, for more specialized functionality:

- `Arithmetic_traits` is a set of compatible number types used to parameterize algorithms and data structures. Compatibility means that conversions are defined where they make sense mathematically, e.g. from integers to rationals.

- `Fraction_traits<NT>` converts between fractional and integer representations, i.e., the numerator and denominator of a fraction `NT`. It is useful in converting a polynomial with fractional coefficients into a common integer divisor and a numerator polynomial with integer coefficients, which allows certain operations of polynomials to be much faster.
- `Scalar_factor_traits<NT>` extracts common scalar factors from nested algebraic compound objects `NT` like polynomials over one-root numbers with integer coefficients to simplify them.
- `Modular_traits<NT>` maps a value of number type `<NT>` to its value of our modular arithmetic type. It is used for the modular arithmetic filter of algebraic numbers.

The class `Polynomial<NT>` in NumeriX represents polynomials with coefficients of type `NT`. Depending on the capabilities of `NT`, the polynomial class adapts and picks the best implementation for certain functions. For this, the polynomial class looks at the number type concept that the actual argument for `NT` fulfills, and it uses the other traits classes for simplifying coefficients or switching from fractional coefficients to integral coefficients and back. The number type `NT` must be at least a model of the `RingWithoutDiv` concept. For all operations naturally involving division, the `Ring` concept is required. Some functions need more than the `Ring` concept, for example, gcds computed from polynomial remainder sequences require the `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 a functions at an early level and allows the reuse of generic higher-level functions.

Template meta-programming is used to determine the `Algebra_type` of `Polynomial<NT>` as a function of `NiX::NT_traits<NT>::Algebra_type` according to the following table:

`NT`	`Polynomial<NT>`
`RingWithoutDiv`	`RingWithoutDiv`
`Ring`	`Ring`
`UFDomain`	`UFDomain`
`EuclideanRing`	`UFDomain`
`Field`	`EuclideanRing`
`FieldWithSqrt`	`EuclideanRing`

The number type `NT` can itself be an instance of `NiX::Polynomial`, yielding a form of multivariate polynomials. Some convenience functions help to make the bivariate case based on this representation more fluent to use.

Similarly, the `NiX::Algebraic_number` class adapts to the number types used in its defining polynomial. The related `Real_roots` functor creates algebraic numbers for all real roots of a polynomial at once. It links all algebraic numbers together, so that, for example, whenever one number learns how to factorize the defining polynomial, all algebraic numbers of that polynomial learn from that information.

The SweepX Library

The sweep-line algorithm as described in [BEH+02b, MN00, MN94] is at the core of the SweepX library. It uses data structures from Leda for the y- and the x-structure, and for the output graph. On top of the sweep line algorithm we have generalized polygons in the SweepX library, i.e., a polygon representation that is closed under regularized boolean operations [MN94].

The sweep-line algorithm is a generic implementation that is independent of the geometric primitives, for example, the degree of the curve. We implemented this using geometric traits classes following the principles in Cgal [FGK+00]. Such a geometric traits class contains types for

the geometric objects and functors[4] for the different geometric predicates and constructions needed in the sweep. The types and functors are:

Types

- `Point_2` used for end-points and intersection points of segments.
- `Segment_2` used as curve segment in computing the arrangement.

We require for points and segments to be default-constructible and assignable. In addition, to be able to make use of the LEDA data structures, we also need a global `ID_Number` function overloaded for points and segments.

Predicates

- `Compare_xy_2` and `Less_xy_2` provide lexicographic comparison of two points, the former with a three-valued return type and the latter as a boolean predicate.
- `Is_degenerate_2` tells true, if the segment is degenerate.
- `Do_overlap_2` determines if two curve segments share a continuous segment of the underlying supporting curve.
- `Compare_y_at_x_2` determines the vertical placement of a point relative to a segment.
- `Equal_y_at_x_2` determines if a point lies on a segment (This functor can have a faster implementation than testing `Compare_y_at_x_2` for equality).
- `Multiplicity_of_intersection` computes the multiplicity of an intersection point between two segments.
- `Compare_y_right_of_point` determines the ordering of two segments just after they both pass through a common point.

Accessors and Constructions

- `Source_2` returns the source point of a curve segment.
- `Target_2` returns the 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 curve 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 [MN00].
- `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. Used only for validity checking and when the intersection dictionary for caching of already computed intersections is not used.

[4] Functors plus access functions that we omit here for brevity, see [FGK+00] for details.

With this well-specified interface it was easy to implement various models for this geometric traits class: We have one model for straight line segments that allows the comparison of the generic implementation in the SweepX library with the version in Leda that is specialized on straight line segments only. We have another model for circular arcs, where we get away with a `FieldWithSqrt` number type and do not need algebraic numbers. And of course, we have models for conic segments and cubic curves. The current release contains only the code for conic arcs.

A newer part in the SweepX library is the GAPS module, which stands for *Generic Algebraic Points and Segments*. It bridges the gap between the analysis of plane algebraic curves and pairs of them on the one hand, and the notion of sweepable segments and points used in the geometric traits class above. It consists mainly of class templates that are models of the `Point_2`, `Segment_2` concepts, and models of the functors required in the above geometric traits class concept. The point and segment class templates come in two flavours, one with and one without genericity assumptions on the choice of coordinates. (The latter may eventually subsume the former.)

We also offer a geometric traits class for Cgal's planar map with intersections. Since the traits class is implemented as an adaptor for the GAPS module, we get immediately a traits class for ConiX and CubiX.

The genericity of our implementation then supported the (preliminary) empirical study comparing the different approaches for arrangements of curves developed in Cgal see ECG-TR-361200-01 [FHW+04].

Acknowledgements

We would like to acknowledge contributions by Michael Seel and discussions with Sylvain Pion.

References

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

[BEH+02a] 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. Technical Report ECG-TR-122103-01, MPI Saarbrücken, 2002. Submitted to ESA 2002.

[BEH+02b] Eric Berberich, Arno Eigenwillig, Michael Hemmer, Susan Hert, Kurt Mehlhorn, and Elmar Schömer. A computational basis for conic arcs and boolean operations on conic polygons. In Rolf Möhring and Rajeev Raman, editors, *Algorithms - ESA 2002: 10th Annual European Symposium*, volume 2461 of *Lecture Notes in Computer Science*, pages 174–186, Rome, Italy, September 2002. Springer.

[BKSV00] Hervé Brönnimann, Lutz Kettner, Stefan Schirra, and Remco 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.

[EKSW04] Arno Eigenwillig, Lutz Kettner, Elmar Schömer, and Nicola Wolpert. Complete, exact, and efficient computations with cubic curves. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, 2004.

[ESW02] Arno Eigenwillig, Elmar Schömer, and Nicola Wolpert. Sweeping arrangements of cubic segments exactly and efficiently. Technical Report ECG-TR-182202-01, MPI Saarbrücken, 2002.

[FGK+00] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL a computational geometry algorithms library. *Softw. – Pract. Exp.*, 30(11):1167–1202, 2000.

[FHW+04] 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.

[Gra96] T. Granlund. *GNU MP, The GNU Multiple Precision Arithmetic Library, version 2.0.2*, June 1996.

[KLPY99] Vijay Karamcheti, Chen Li, Igor Pechtchanski, and Chee Yap. *The CORE Library Project*, 1.2 edition, 1999. http://www.cs.nyu.edu/exact/core/.

[MN94] Kurt Mehlhorn and Stefan Näher. Implementation of a sweep line algorithm for the straight line segment intersection problem. Report MPI-I-94-160, Max-Planck-Institut Inform., Saarbrücken, Germany, 1994.

[MN00] Kurt Mehlhorn and Stefan Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 2000.

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