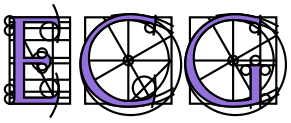
ECG

IST-2000-26473

Effective Computational Geometry for Curves and Surfaces



ECG Technical Report No. : *ECG-TR-361200-01*

An Empirical Comparison of Software for Constructing Arrangements of Curved Arcs (preliminary version)

Sylvain Pion, Monique Teillaud

Efraim Fogel, Dan Halperin, Ron Wein

Ioannis Emiris, Athanasios Kakargias, Elias Tsigaridas

Eric Berberich, Arno Eigenwillig, Michael Hemmer, Lutz Kettner, Kurt Mehlhorn, Elmar Schömer, Nicola Wolpert

Deliverable: 36 12 00 (new, item 01)
Sites: INRIA, MPI, TAU
Month: 36

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

An Empirical Comparison of Software for Constructing Arrangements of Curved Arcs (preliminary version) *

Efi Fogel, Dan Halperin, Ron Wein
Tel Aviv University

Monique Teillaud, Sylvain Pion
INRIA Sophia-Antipolis

Eric Berberich, Arno Eigenwillig, Michael Hemmer, Lutz Kettner,
Kurt Mehlhorn, Elmar Schömer, Nicola Wolpert
Max-Planck Institut für Informatik, Saarbrücken

Ioannis Emiris, Athanasios Kakargias, Elias Tsigaridas
National Kapodistrian University of Athens

April, 2004

Abstract

Arrangements of planar curves are fundamental structures in computational geometry. Algorithms for computing such arrangements consist of a topological part and a geometric part. For both parts different algorithmic approaches and implementations are possible. In ECG, we further developed and implemented these approaches. We followed modern software design and encapsulated our solutions into modules with well-defined and tight interfaces. In particular, we can combine different realizations of the topological part (we have two) with different realizations of the geometric part (we have three, which in turn are parametrized by different implementations of the underlying number types). The implementations of the geometric part follow quite different designs. In this report, we provide first comparisons of our different designs. In a later version of the report, we also plan to compare implementations outside the ECG-project.

The report is preliminary and rises more questions than it answers. We consider it as proof that our modular architecture is valuable and allows us to experiment and compare different approaches. We also consider it as proof for the close cooperation within the project.

1 Introduction

Given a set $\mathcal{C}$ of planar curves, the *arrangement* $\mathcal{A}(\mathcal{C})$ is the subdivision of the plane induced by the curves in $\mathcal{C}$ into maximally connected cells of dimension 0 (vertices), 1 (edges), and 2 (faces). The separation between the topological and geometric aspects of the planar arrangement software-package is advantageous, as it allows users to employ the package with their own special type of curves, without having any expertise in computational geometry on one hand, and it enables easier development of each module on the other. We take advantage of this separation to profile different models of the combinatorial module with a fixed geometric model, and different models of the geometric module with a fixed combinatorial model.

*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)

The study we describe here is in its very early stages. While we believe that the software components we use and describe here are the state-of-the-art in the field of our interest, this is the first time that all these different implementations are combined and compared in practice. In addition, not all the implementations have reached the same level of maturity. The test cases chosen exploit a common denominator of features proved to work properly on all implementations. The results of the study point at soft spots that are candidate for optimization, and are intended to be used as a base ground for future progress-monitoring.

Background

A planar arrangement can be constructed incrementally by inserting the curves one at a time into the arrangement data structures (in random order, when we also construct an efficient point-location structure, supported by the CGAL arrangement package [9]. For details on the CGAL arrangement package see [1, 9, 12, 13]).

However, for a large number of densely intersecting curves that form a cluttered arrangement it is more efficient to use an aggregate insertion-method, that sweeps the plane in a fixed direction (for example, left to right), and inserts the curves at once, known as a sweep-line algorithm (see for example [3]). In section 3 we present the results of experiments that compare two different implementations of sweep-line algorithms.

The geometric objects and the operations on these objects used by the (combinatorial) sweep-line algorithms (and perhaps by other operations supported by the relevant arrangement data-structure) is provided through a separate class referred to as traits, tailored to handle a specific family of curves. It defines the abstract interface between arrangements and the geometric primitives they use. In section 2 we present the results of experiments that compare the efficiency of three different traits-classes, and some derivatives, designed to handle conic arcs.

The comparative experiments reported in Section 2 and 3 are enabled by a continuing effort along a period of over a year and a half during the ECG project to streamline the interface of CGAL's arrangement package between the topological and geometric parts, and minimize the requirements of the traits class. In addition to intensive electronic discussions, we had two meetings in INRIA in December 2002 and in Dagstuhl in March 2003 where we studied these issues and developed the tight interface which is used here [10].

2 Comparison of Conic Traits by Experiments

We have conducted experiments to measure the performance of the construction of CGAL arrangements of conic arcs with three different approaches to handle conic arcs. Each approach, while implemented in its own context, provides a model of the traits concept of the CGAL arrangement data-structure that serves as a traits class. Some of these traits classes are driven by large software systems developed during the life cycle of the ECG project and beyond. The three traits-classes are:

- CGAL conic traits
- Curved-kernel conic traits
- EXACUS conic traits

2.1 CGAL Conic Traits

The traits class provided with the CGAL distribution can handle any type of bounded conic arcs, including full circles and ellipses and line segments, as lines can be viewed as degenerate conic curves, see [18]. The most advanced version of the traits class relies heavily on the CORE library for its algebraic infrastructure, see [14, 16]. It uses two number types: The `CORE::BigInt` type is used to represent the coefficients of the conic curves, and the `CORE::Expr` type is used to represent the coordinates of the arrangement vertices (which are roots of polynomials with integer coefficients). The traits class also stores additional combinatorial information with its geometric objects in order to filter out unnecessary expensive computations — a technique named *high-level filtering* [18].

2.2 Curved-Kernel Conic Traits

Our ultimate goal is to provide a kernel with curved objects for CGAL [17, 6], following a design similar to the current kernel which provides functionalities only for linear objects. One of the first target applications of this work is the CGAL arrangement package, applied to circle and conic arcs. In order to achieve this, we have written a thin layer between our kernel and the CGAL arrangements, which provides the traits class requirements.

In our kernel, circles and circle arcs have specific types (they do not share the representation of conic and conic arcs), which allows to benefit from a more memory efficient representation and improved primitives requiring lower degree computations. Therefore, we actually provide two traits classes for the CGAL arrangements : one for circle arcs, `Circular_arc_traits`, and one for conic arcs, `Conic_arc_traits` (note however that the current preliminary implementation of `Conic_arc_traits` only allows arcs of ellipses).

Concerning the representation, circles and conics are simply re-used from the current CGAL kernel, they store the coefficients of their equations. For both cases, an arc is essentially storing its two endpoints, and endpoints are represented as two supporting curves, plus algebraic numbers specifying their coordinates.

Algebraic numbers have a different type depending on their degrees (2 for circle arc endpoints, and up to 4 for conic arc endpoints). Let us describe more precisely how geometry interacts with algebra in the two different cases of circular arcs and conic arcs.

`Circular_arc_traits.` The geometric primitives implemented in the curved kernel for circular arcs rely on specialized methods for manipulations of algebraic numbers of degree 2 [4] (variants would be possible [15]). The implementation is generic, so, it allows us to benefit from some filtering at the arithmetic level when using a number type such as `CGAL::Lazy_exact_nt`. But it should be noticed that there is no filtering at the level of predicates and constructions in our current implementation.

`Conic_arc_traits.` The underlying curved kernel for conic arcs is based on an algebraic kernel that is responsible for providing algebraic primitives like polynomials (univariate and bivariate), solve functions and comparisons of algebraic numbers of degree up to 4. This improves the modularity of the code. Since the code is in a very preliminary stage, the interaction between the geometric and the algebraic kernels is not optimal. Also, some caching techniques are implemented at the algebraic level but not at the geometric level, which could improve the performance.

At this point we have to say that the only existing model of algebraic kernel is based on the SYNAPS library, and our current implementation of the algebraic numbers in SYNAPS 2.1 (using methods described in [8, 7]) is not mature yet. In particular, it currently does not allow the

use of any arithmetic filtering technique but requires an exact multiprecision integral number type. So, we observe a lack of efficiency. Last, but not least, we have to mention that is the first attempt of interfacing CGAL with SYNAPS hence there are several implementation problems still open. All of the above are first priorities for improvements in the near future.

2.3 EXACUS Conic Traits

The EXACUS conic traits class is maintained by the Max-Planck-Institut für Informatik and provides the interface between curves in EXACUS[1] and their computation in CGAL `Planar_map_with_intersections`. EXACUS is a collection of Libraries for *Efficient and Exact Algorithms for Curves and Surfaces*. It started in 2002 as contribution of the Algorithms and Complexity Group (AG1) of the Max-Planck-Institute to the ECG project (*Effective Computational Geometry*).

One important layer of software in EXACUS is GAPS, which stands for *Generic Algebraic Points and Segments*. It is used to represent points on and segments of general algebraic curves. The implementation is complete, which means that we can handle all kind of degeneracies: isolated points, vertical segments and/or overlapping segments, high-degree intersections or several curves running through the same point. Furthermore, it is not restricted to the bounded case. Endpoints of unbounded curves (for example of the hyperbola $xy - 1 = 0$) are handled symbolically. To use GAPS one has to deliver the analysis of a pair of curves.

For conics this analysis is implemented in the module CONIX of EXACUS. It is based on the SUPPORT and the mathematical NUMERIX library of EXACUS. As exact number types we can choose at the moment between the LEDA and the GMP/CORE package.

The newest implementation of CONIX uses some ideas we developed for CUBIX [5] which replaces our first implementation described in [2]. Compared to this former implementation we gained an improvement on the running times of factor at least 10. And there are still ways to improve these times, because filters are currently used rarely: All arithmetic computations rely on the exact number types. Of course, we benefit from the internal filters of `leda::real` respectively `CORE::Expr`. Furthermore, we use modular arithmetic to filter gcd-computations. But in `ConiX`, there is no active floating point filter at all.

As input we allow complete conics: hyperbolas, ellipses, parabolas, (complex) line pairs and also single lines as degenerate conics. Besides complete conics, we deliver an exact way to input *segments* of conics, which do not need to be x-monotone and we also support the approximative input format for *conic arcs* of the CGAL conic traits. Before sweeping, we have to split the input into sweepable segments, which are either vertical or x-monotone, and always free of singular points in its interior.

The actual traits class `CGAL_Pmwx_2_for_GAPS_traits` used for the benchmarks simply calls or recombines already existing functions and functors of GAPS. With that traits class arrangements of curves in EXACUS (currently conics, cubics and projected intersection curves of quadrics) can be computed with the help of CGAL `Planar_map_with_intersections`. Compared to the EXACUS `sweep_curves` function of the module SWEEPX, the CGAL method enables additionally point location and incremental construction.

2.4 Empirical Results

The results listed below were produced by experiments conducted on a Pentium PC clocked at 1.8 GHz running `Linux` Fedora. The executables were compiled with GNU's `g++` compiler version 3.3.2 to maximum performance. For the exact number types we used either number

[1] `http://www.mpi-sb.mpg.de/projects/EXACUS/`

types provided by LEDA 4.4.1, CORE 1.6x, or internal number types provided by CGAL 3.0.1. We have restricted ourself to use only circles, ellipses, and combinations of these, as the curved-kernel conic traits was unable to handle general conics at the time the experiments took place.

The first set consists of six test cases featuring circles and ellipses, but not combinations of them, mostly in regular shapes and degenerate configurations, see Figure 1. Table 1 shows the number of input curves, the number of vertices, halfedges and faces of the resulting arrangement, and the running time in seconds it took to construct the arrangements by the various executables.

Series Name	Input Curves	Vertices	Halfedges	Faces	CGAL		CK		EXACUS	
					LEDA	CORE	Circle	Conic	LEDA	CORE
Box	41	507	2084	537	3.155	2.354	2.082	10.140	2.302	3.550
Circles	21	87	340	85	0.812	0.308	0.342	1.350	0.321	0.499
Rose	14	59	212	53	0.525	0.189	0.220	0.883	0.200	0.319
Random Circles	30	424	1576	367	4.466	1.915	1.981	7.300	1.266	2.356
Tangent Ellipses	9	44	184	50	0.799	0.278	-	1.199	0.137	0.283
Random Ellipses	20	221	808	185	1.318	18.510	-	10.200	0.590	0.855

Table 1: Time consumption of the construction of arrangements induced by full circles and ellipses.

The CGAL conic traits-class and the EXACUS conic traits-class come in two flavors. One is based on the `leda_real` number type offered by LEDA, and the other is based on the `CORE::Expr` number type offered by CORE. The CGAL conic traits-class based on CORE is a recent addition, and the figures above reveal that there is still plenty of room for optimization when many ellipses are present in the input.

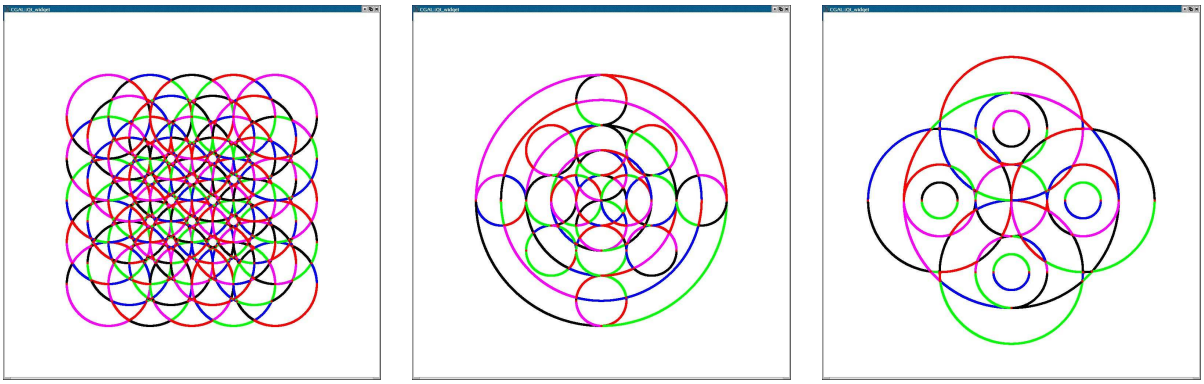
The curved-kernel conic traits also comes in two flavors as specified above. One is dedicated to circular arcs only and the other is designed to handle general conic arcs. The latter has not been optimized yet to its full potential. In particular, the predicates and constructions are not filtered at all. Recall that at the time the experiments took place it supported only circular and ellipsoidal arcs.

The second set consists of three test cases featuring different numbers of full circles and random full ellipses. Table 2 shows the same type of information as the previous table shows for these test cases. The execution with the CGAL CORE-based traits was suppressed, as it did not support this feature at the time these experiments took place.

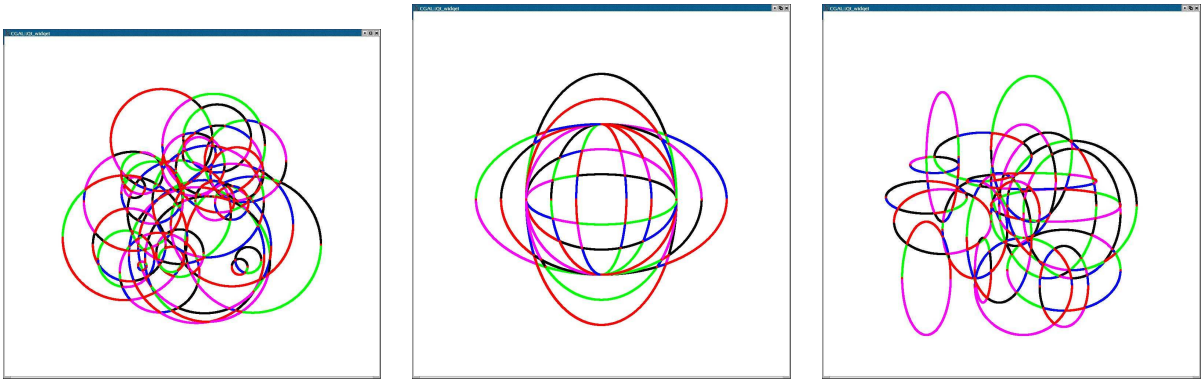
We conclude with the running times of a carefully hand-constructed instance that consists of circular arcs in degenerate positions. Table 3 lists the results and Figure 2 shows a screenshot of the arrangement produced.

2.5 Conclusion and Remarks

While there are a few fluctuations that will have to be investigated, it seems that most of the results can be reasoned. The EXACUS implementation is the most advanced, as it supports all types of conic curves and conic arcs, and exhibits better stability, but the figures extracted from executables produced from optimized code do lie in the same neighborhood, even though only the



Box | Circles | Rose



Random circles | Tangent ellipses | Random ellipses

Figure 1: Snapshots of arrangements constructed by full circles and ellipses.

Series Name	Input Curves	Vert-ices	Half-edges	Faces	CGAL LEDA	CK Conic	EXACUS LEDA	EXACUS CORE
Rand 1	32	783	3078	758	8.300	57.630	2.770	5.390
Rand 2	48	1656	6516	1604	27.810	118.960	6.400	12.740
Rand 3	64	2866	11330	2801	46.820	211.530	12.210	23.180

Table 2: Time consumption of the construction of arrangements induced by combination of circles and ellipses.

Series Name	Input Curves	Vert-ices	Half-edges	Faces	CGAL LEDA	CGAL CORE	CK Circle	CK Conic	EXACUS LEDA	EXACUS CORE
Circular Arcs	40	211	808	195	2.146	1.157	0.959	5.220	1.083	1.518

Table 3: Time consumption of the construction of arrangements induced by circular arcs.

CGAL conic trait-class was developed at the same site the arrangement framework was developed at. This establishes the validity of each one of the approaches above, and provides certain assurance regarding its worthwhileness. As our approaches rest on quite different algebraic methods and hence are partially orthogonal to each other, we feel that substantial further improvements should be possible by using several methods. This will be the subject of further research. We remark that it is too early for combining the implementations as we do not know yet which appraoch works best in which situation. More experimentation is needed to answer this question.

In the course of these experiments we also realized that the different traits classes not only differ in the algebraic techniques but also in the caching strategies deployed. Caching is typically used to store the analysis of a single curve or of pairs of curves to avoid costly recomputations, for example, when indeed all four intersection points of a pair of ellipses show up in the arrangement. We propose to revise the benchmarks to compare the algebraic methods without caching sideeffects. This can be done with the interface that the GAPS part of EXACUS provides, namely the analysis of pairs of curves, or some similar caching strategy integrated in the CGAL arrangement software. An alternative can be to augment the current implementations in their predicate levels and print a trace of all the predicate calls with their parameters. Running conic arrangement bechmarks would then lead to application benchmark data for the algebraic layer, and the different implementations can be compared on that layer ignoring caching.

We are aware of the limitations and small number of test data sets used in the comparison so far. We have test data for larger random data sets and for degenerate arrangements of ellipse, i.e., several ellipses intersect in the same point or intersect tangentially. We also generated a family of test data for arrangements of ellipses with increasing bitsize of the coefficients. However, only EXACUS was able to handle these test instances. The other implementations need further improvements before we can move to these interesting cases.

The experiments described above were generated, executed, and presented using a testing and profiling toolkit described in details in [11]. The toolkit was employed to produce a handful of results, out of which only the highlights are presented in this report in Tables 1, 2, and 3. The complete sets of results along with some other data related to the experiments can be found at `http://www.cs.tau.ac.il/~efif/ECG/empirical_cmp`.

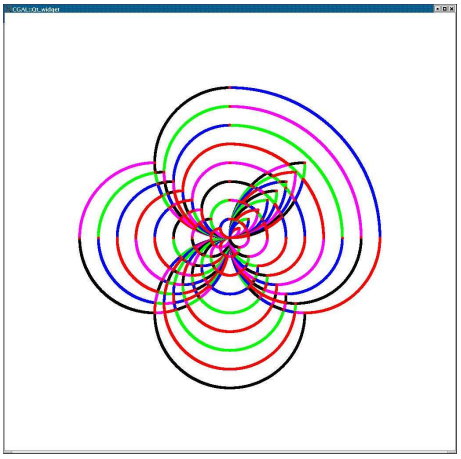


Figure 2: Circular arcs in degenaret positions

3 Experimental Comparison of the EXACUS and CGAL Sweep Line Algorithms

We report on a few first experiments comparing the relative efficiency of the sweep line algorithms for arrangement computation of CGAL's publicly available `Planar_map_with_intersections` (Pmwx) class (CGAL-3.0.1, member function `.insert(begin, end)`) with the `SoX::sweep_curves()` function of EXACUS (to be released internally with ECG-TR-361200-02) based on EXACUS' CubiX predicates for cubic curves (ECG-TRs 182202-01 and 242107-01). This means we compare two different implementations of the high-level (combinatorial) algorithm based on the same set of geometric predicates and operations. Unlike the rest of this report, our test instances in this section are sets of cubic curves, not conics.

The experiments have been conducted on a Pentium IIIM clocked at 1.2 GHz running `Linux` 2.4 using executables that have been compiled with GNU's `g++` compiler (version 3.3.3, with the flags `-O2 -DNDEBUG`). We use LEDA 4.4.1 for the exact number types (mostly integer and rational). The arrangement computations have been run from within CubiX' graphical demo program `xcubi`. To avoid potentially unfair influences of the slightly different interfaces, we have avoided input data with overlapping segments, and we made `SoX::sweep_curves()` call `Intersect_right_of_2` to compute intersections of segments one at a time (and not all at once, as `Intersect_2` would have done). One difference with respect to reordering segments persists (see below).

The first series of benchmarks consists of random sets of n cubic curves. Each curve f is defined by interpolation through 9 points chosen uniformly at random from a set of $9n$ random points on the $\{-128, \ldots, 127\}^2$ integer grid. Every interpolation point results in a homogeneous linear condition on the 10 unknown coefficients of f, so that generically 9 conditions determine the equation of a curve uniquely, up to a constant factor. Before sweeping, each curve has to be broken into sweepable segments, which are in particular x-monotone.

Table 5 shows the size of the input data (number of sweepable segments), size of the resulting arrangement (number of nodes and half-edges), and the running time in seconds (excluding splitting, averaged over three runs). The average bit length of a curve's longest coefficient is about 100 bits for all instances.

Series Name	Input Curves	Segments	Vertices	Halfedges	SoX	Pmwx
random	30	266	2933	11038	6.7	8.0
random	60	454	11417	44440	27.7	34.6
random	90	680	26579	104474	67.5	81.2

Table 4: Time consumption of sweeping random sets of cubic curves.

Profiling the executions on the "random 30" instance exhibits three clear differences in the number of predicate invocations: First, Pmwx invokes `compare_right_of_common_point()` ~ 200 times as often as SoX. The dominant reason for this is certainly reordering by comparison (Pmwx) as opposed to reordering by intersection multiplicities (SoX, see below). Furthermore, Pmwx invokes the lexicographic comparison of points ~ 5 times as often (including indirect calls through comparison operators) and `y_order()` ~ 10 times as often. The reason for this is not evident at this point.

Secondly, we offer a benchmark to look at the effects of high-degree vertices with intersections of multiplicity higher than 1. Its data sets are obtained in a similar fashion, except that

1) There are only 32 interpolation points.
2) For each interpolation point p, we pick random values for slope m_p and curvature κ_p. Whenever a curve f is interpolated through p, we make its slope equal to m_p (yielding one additional linear condition) and, with probability $\sqrt{0.5}$, we also make its curvature equal to κ_p (yielding another condition).
Now the intersections at interpolation points will have multiplicity 2 or 3 (due to prescribing slope and maybe curvature) and involve many curves (due to the small number of interpolation points). In principle, this setting favors the `SoX::sweep_curves()` function, because it can reorder segments passing through a high-degree vertex in linear time based only on their previous order and the intersection multiplicities of adjacent curves. However, as seen from the numbers of nodes given below, most intersections still occur outside interpolation points and will almost certainly involve only 2 curves and have multiplicity 1. So the relation of running times is similar to the "rand" instances above. The overall slowdown in the "hdeg" instances is due to the more costly curve pair analysis in degenerate cases, see Table 5.

Series Name	Input Curves	Segments	Vertices	Halfedges	SoX	Pmwx
hdeg	30	254	2356	8816	12.9	14.9
hdeg	60	496	8068	31140	52.9	59.8

Table 5: Time consumption of sweeping sets of cubic curves that produce high degree intersection vertices of multiplicity higher than 1.

We conclude with 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. A screenshot of this instance is shown in Figure 3. Here `SoX::sweep_curves()` can exhibit the full benefit of linear-time reordering. The timings are (in seconds): 1.7 for SoX, 4.3 for Pmwx.

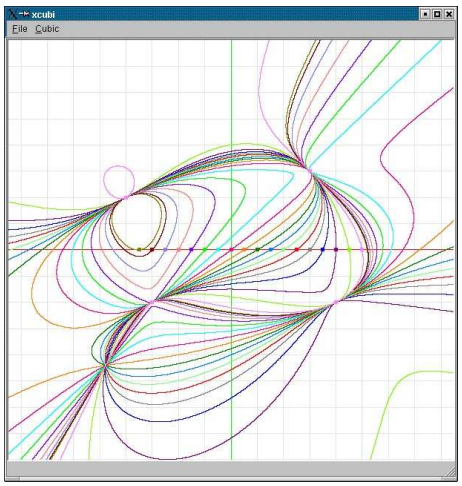


Figure 3: A pencil of 18 curves intersecting with multiplicities 1 or 2, respectively, in 5 distinct points.

Conclusion and Further Questions

What can we learn from these experiments? We can manufacture an example where the linear time reordering really matters. The "hdeg" examples originally intended for this purpose show instead that a small fraction of high degree nodes compared to the overall number of nodes, 1.4% and 0.4% respectively, does not matter much. Actually, just to the contrary, here the Pmwx sweep is 13-16% slower, while for the "rand" instances it is 19-25% slower.

All experiments show an advantage for the SoX implementation, but then, this is the native combination developed to work together. We took care not to use examples that would penalize the Pmwx implementation with the interface mapping to the CubiX implementation (see discussion above), but it has to be said that this interface is still experimental, so that the figures above do not allow a definitive judgment on the relative merits of the two implementations.

The slowdown in running time is smaller than the ratio of predicate invocations. For the given set of predicates, this is not surprising in the light of caching: The refinement of isolating intervals of x-coordinates means that redundant comparisons are decided at once from the boundaries alone (and identity can be cached by merging equal representations). The caching of curve pair analysis means that repeated evaluations of the same geometric predicate do not cause repeated execution of costly symbolic computations. Then, the question why does the Pmwx implementation make redundant calls in the first place remains.

References

[1] The CGAL project homepage. `http://www.cgal.org/`.

[2] E. Berberich, A. Eigenwillig, M. Hemmer, K. M. S. Hert, and E. Schömer. A computational basis for conic arcs and boolean operations on conic polygons. In *Proc. European Symp. on Algorithms*, pages 174–186. Springer, 2002. LNCS 2461, Berlin.

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

[4] O. Devillers, A. Fronville, B. Mourrain, and M. Teillaud. Algebraic methods and arithmetic filtering for exact predicates on circle arcs. *Comput. Geom. Theory Appl.*, 22:119–142, 2002.

[5] A. Eigenwillig, L. Kettner, E. Schömer, and N. Wolpert. Complete, exact and efficient computations with cubic curves. In *Proc. 20th Annu. ACM Sympos. Comput. Geom*, 2004.

[6] I. Z. Emiris, A. V. Kakargias, S. Pion, M. Teillaud, and E. P. Tsigaridas. Towards an open curved kernel. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, 2004. to appear.

[7] I. Z. Emiris and E. P. Tsigaridas. Comparison of fourth-degree algebraic numbers and applications to geometric predicates. Technical Report ECG-TR-302206-03, INRIA Sophia-Antipolis, 2003.

[8] I. Z. Emiris and E. P. Tsigaridas. Methods to compare real roots of polynomials of small degree. Technical Report ECG-TR-242200-01, INRIA Sophia-Antipolis, 2003.

[9] 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).

[10] E. Fogel, D. Halperin, R. Wein, M. Teillaud, E. Berberich, A. Eigenwillig, S. Hert, and L. Kettner. Specification of the traits classes for cgal arrangements of curves. Technical Report ECG-TR-241200-01, INRIA Sophia-Antipolis, 2003.

[11] E. Fogel and A. Sturm. Testbed implementations of exact and approximate algorithm. Technical Report ECG-TR-36121501-01, Freie Universität Berlin, 2003.

[12] E. Fogel, R. Wein, and D. Halperin. Code flexibility and program efficiency by genericity: Improving CGAL's arrangements. Menuscript, Tel-Aviv univ., 2004.

[13] I. Hanniel and D. Halperin. Two-dimensional arrangements in CGAL and adaptive point location for parametric curves. In *Proc. of the 4th Workshop of Algorithm Engineering*, volume 1982 of *Lecture Notes Comput. Sci.*, pages 171–182, Saarbrücken, 2000. Springer-Verlag.

[14] V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. A core library for robust numeric and geometric computation. In *15th ACM Symp. on Computational Geometry, 1999*, pages 351–359, 1999.

[15] M. I. Karavelas and I. Z. Emiris. Root comparison techniques applied to computing the additively weighted Voronoi diagram. In *Proc. 14th ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 320–329, 2003.

[16] C. Li and C. Yap. A new constructive root bound for algebraic expressions. In *12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 496–505, 2001.

[17] S. Pion and M. Teillaud. Towards a cgal-like kernel for curves. Technical Report ECG-TR-302206-01, MPI Saarbrücken, INRIA Sophia-Antipolis, 2003.

[18] R. Wein. High-level filtering for arrangements of conic arcs. In *Proc. ESA 2002*, pages 884–895. Springer-Verlag, 2002.