Exact and efficient computations on circles in CGAL

Pedro M. M. de Castro, Sylvain Pion, Monique Teillaud

Exact and efficient computations on circles in CGAL (abstract) *†

Pedro M. M. de Castro‡ Sylvain Pion Monique Teillaud§

Abstract

CGAL *(Computational Geometry Algorithms Library)* is a large collection of geometric objects, data structures and algorithms. CGAL currently offers functionalities mostly for linear objects (points, segments, lines, triangles...).

The first version of a kernel for circles and circular arcs in 2D was recently released in CGAL 3.2. We show in this paper a variety of techniques that we tested to improve the efficiency of the 2D circular kernel. These improvements are motivated by applications to VLSI design, and real VLSI industrial data are used to measure the impact of the techniques used to enhance this kernel. The improvements will be integrated in CGAL 3.3.

1 Introduction

The goal of the CGAL Open Source Project is to provide easy access to efficient and reliable geometric algorithms to users in industry and academia. This is achieved in the form of the C++ *Computational Geometry Algorithms Library* [2]. The CGAL packages adopt the generic programming paradigm [12], making extensive use of C++ class templates and function templates, and their design is heavily inspired from the C++ Standard Template Library [5].

For instance, let us consider the case of geometric arrangements: an *arrangement* of a finite set of curves in the plane is the partition of the plane into faces, edges and vertices, that is induced by these curves [15, 3, 11]. A generic implementation of a data structure that handles arrangements is achieved by the `CGAL::Arrangement_2` class:

```
template <class Traits> class Arrangement_2
```

This class must be instantiated with a class, referred to as a *traits* class [19], that must define a type representing a certain family of curves, and some functions operating on curves of this family.

The CGAL *kernels* provide the users with basic geometric objects and basic functionalities on them. CGAL currently offers kernels for linear objects (points, segments, lines, triangles...), and the first version of a kernel for circles and circular arcs in 2D, called *2D circular kernel* in the sequel, was recently released in CGAL 3.2 [20]. A kernel can be wrapped into a traits class offering the interface for some geometric algorithms; this was done for the CGAL circular kernel and `CGAL::Arrangement_2`. However, a kernel is meant to offer general purpose functionalities, while a traits class offers the minimum set of functionalities required by a specific class.

Robustness, achieved through the exact geometric computation paradigm [23], is probably the first strength of CGAL. The CGAL arrangement package, completely redesigned for CGAL 3.2 [22, 21] offers a robust and efficient implementation. Other libraries like ESOLID [18] may crash on degenerate input data[1]. Real VLSI data sets consist of line segments and circular arcs, containing many degenerate, or close to degenerate, cases (junctions, identical intersection points, tangencies...) requiring highly robust code. Typically, the question is to compute boolean operations on these data, that can be easily performed on top of the computation of the input curves arrangement [13]. Efficiency is also a crucial quality for the use of CGAL on real industrial data. VLSI inputs consist of very large data sets, which leads to the need for improvements in the efficiency of the 2D circular kernel.

We show in this paper a variety of techniques from different nature that we tested to improve the 2D circular kernel, and experimental evidence of their impact is studied by benchmarking on real VLSI industrial data. The techniques resulting in measurable improvements will be integrated in CGAL 3.3.

2 The 2D Circular Kernel

To answer the need for robustness on manipulations of circular objects, a first version of the 2D circular kernel was released in CGAL 3.2.

The design of the 2D circular kernel [10] uses the

*The full version of this work is available at `https://hal.inria.fr/inria-00123259/en/` as INRIA research report RR-6091.

†This work is partially supported by the IST Programme of the 6th Framework Programme of the EU as a STREP (FET Open Scheme) Project under Contract No IST-006413 - ACS (Algorithms for Complex Shapes with certified topology and numerics) - `http://acs.cs.rug.nl/`

‡current address: Center of Computer Sciences, Universidade Federal de Pernambuco, Brazil. `pmmc@cin.ufpe.br`

§INRIA, BP93, 06902 Sophia Antipolis cedex, France, {`Sylvain.Pion,Monique.Teillaud`}`@sophia.inria.fr`, `http://www-sop.inria.fr/geometrica/team/`

[1] see `http://research.cs.tamu.edu/keyser/geom/esolid/`

extensibility and adaptability properties of the CGAL linear kernel [16]. The circular kernel is parameterized by, and inherits from, a `LinearKernel` parameter, for objects like points, circles and number types. It has a second parameter, `AlgebraicKernel`, providing algebraic operations that are necessary for computations on circles. The geometric level interface provides types already defined by the linear kernel, plus three new types: `Circular_arc_point_2` for points on circles, and `Circular_arc_2` and `Line_arc_2` respectively representing circular arcs and line segments delimited by such points.

Intersections involving circles lead to manipulating algebraic numbers of degree two on this ring. Moreover, most geometric predicates on circular arcs are expressed as comparisons involving such roots. We rely on exact handling using polynomial representation of these roots and algebraic methods (like resultants and Descarte's rule of sign) which reduce the computations to comparisons of polynomial expressions [8, 17, 9]. It was shown that the latter choice, for this specific small degree two, was more efficient than using a general library like CORE or LEDA.A template class `Root_of_2<RT>` is provided for algebraic numbers of degree 2, using the following internal representation: three coefficients of type `RT` specifying the polynomial of degree 2, plus one boolean value specifying whether the smaller of the roots is considered, or the other root.

The CGAL arrangement package comes with a default traits class for line segments and circular arcs, called `Def_traits` in the sequel. We wrapped the circular kernel into a traits class offering the same interface. Since the arrangement package requires the traits to provide a unique type `Curve_2`, the default traits class does not offer separate types for line segments and circular arcs. The circular kernel, meant to be general purpose, offers two different types, `Circular_arc_2` and `Line_arc_2`. The traits class built on the circular kernel uses the `boost::variant` class[2] [1], that allows to wrap the two complex types in one unique type `Curve_2`. Moreover, since arrangements algorithms implemented in this package start by cutting all curves into x-monotone arcs, functionalities like intersection computations are provided only for x-monotone arcs in a traits class for arrangements. The circular kernel implements these functionalities for general arcs.

3 VLSI Data Sets and Conditions of Experiments

We conducted experiments to evaluate the practical influence of several improvement techniques on the CGAL 2D circular kernel. Our experiments consist in computing arrangements of real industrial data of VLSI models[3] representing electronic circuits, with the efficient sweep-line algorithm provided by the new CGAL arrangement package [22].

Input	CGAL 3.2	`Def_traits`	CGAL 3.3
vlsi_1	28.0	8.55	4.61
vlsi_2	48.0	2.59	1.31
vlsi_3	135	26.7	21.8
vlsi_4	569	26.9	25.4
vlsi_5	125	14.3	14.8
vlsi_6	611	137	134
vlsi_7	690	192	169
vlsi_8	3,650	220	136
vlsi_9	2,320	581	492
Very dense	335	77.9	76.2
Sparse	0.91	0.51	0.21

Table 1: Time, in seconds, spent to compute the arrangement with the CGAL 3.2 circular kernel, with `Def_traits`, and after the improvements presented in this paper.

The VLSI data have many cases of junctions, degeneracies and tangencies, causing approximate computation to fail due to rounding errors, and which make them appropriate for exact computation. See Figure 1 for an illustration. The table gives the sizes of the input files we are using for the experiments, together with the sizes of the corresponding arrangements.

We complete our experiments with synthetic input (see also Figure 1): a very dense distribution where circles are centered on a grid and all pairs of circles intersect (a), and a sparse distribution without intersection (b).

The hardware of our experiments was a Pentium 4 at 2.5 GHz with 1GB of memory, running Linux (2.4.20 Kernel). The compiler was g++4.0.2; all configurations were compiled with the -DNDEBUG -O2 flags.

In the sequel, the default traits class `Def_traits` will be used as a reference for measuring the performances of the circular kernel. Both of them are used with the same basic exact ring number type: `CGAL::MP_Float`. Note however that the default traits class is already optimized and uses arithmetic filtering (see Section 4.3.2) which makes it quite efficient. The CGAL 3.2 circular kernel is not yet filtered at all. Note also that the use of `boost::variant` (see Section 2) introduces a slight overhead in the running times obtained by the circular kernel.

These elements partly explain the poor performance of this kernel shown in Table 1. The rest of this work is devoted to show how several techniques can be combined to produce an important improvement in the running times.

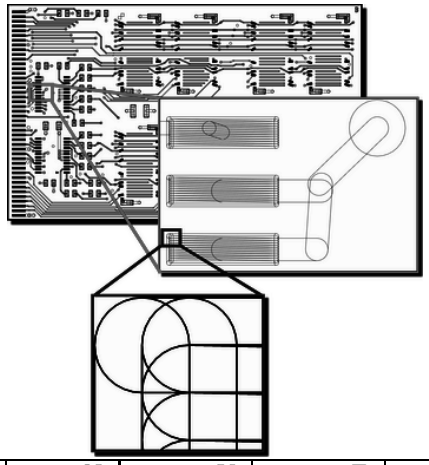
[2] `http://www.boost.org/libs/variant/index.html`

[3] Thanks to Mania Barco and GeometryFactory.

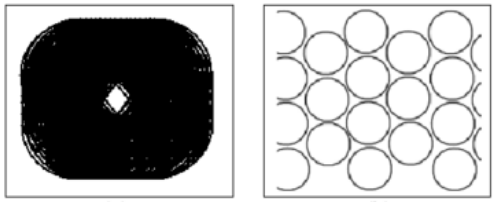
4 Tuning the 2D Circular Kernel

4.1 Caching Using Bit-Field

The results of some costly operations can be stored to avoid recomputing them several times. When those operations return symbolic values, like *boolean*, the memory space used can be very low: the results can be efficiently stored in a *bit-field* which consists in the manipulation of individual bits of an entire block of data [4].



VLSI	N	V	E	F
vlsi_1	11,929	20,649	26,468	6,385
vlsi_2	9,663	8,556	9,439	887
vlsi_3	35,449	101,758	163,316	61,887
vlsi_4	12,937	81,096	143,049	61,986
vlsi_5	4,063	40,636	77,598	36,965
vlsi_6	74,052	547,250	1,016,460	470,480
vlsi_7	89,918	495,209	878,799	383,871
vlsi_8	123,175	370,304	555,412	190,031
vlsi_9	139,908	1,433,248	2,690,530	1,257,684



Distributions	N	V	E	F
very dense (a)	400	160,400	320,000	159,602
sparse (b)	441	882	882	442

Figure 1: An example of VLSI data (*vlsi_7*), composed of 22,406 polygons and 294 circles, for a total number of 89,918 arcs. The zooms show the complexity of the data. The bottom picture shows the two synthetic data sets. The tables give the characteristics of the data sets: N is the number of arcs (line segments or circular arcs), V is the number of vertices of the arrangement, E is the number of edges and F the number of faces.

We introduce a bit-field as an additional data member of `Circular_arc_2`, to store whether an arc is x-monotone, the complement of an x-monotone arc (same for y), and whether its endpoints are on the upper part of the supporting circle (same with left part). The bit-field has 2 bytes (in fact, only 12 from the total 16 bits are used).

The bit-field can be quickly computed when arcs are computed by cutting previous arcs into monotone arcs.

4.2 Caching Intersections of Supporting Circles

We tried to use a `std::map` which takes a pair of circles and returns their (at most two) intersection points. Experiments on VLSI files showed that this was not an interesting contribution.

4.3 Enhancing the Algebraic Number Type

The `Root_of_2` number type was improved by following two directions.

4.3.1 Optimizing Particular Cases

Every time when the rationality of a `Root_of_2` is detectable in constant time, we use a specific constructor that will allow to take advantage of this property. Those cases appear in the intersection of two tangent circles, in the intersection of a line and a circle that are tangent, in the intersection of a vertical/horizontal line with a circle.

4.3.2 Arithmetic Filtering

The general idea of *filtering* comparisons consists in computing an approximate result, together with an error bound. If the error bound is small enough, comparing the approximate values is enough to give the result of the comparison of the exact values, which allows to conclude very quickly. Otherwise, we say that the filter *fails*, and the computation is done using exact arithmetic. Clearly, the errors need to be kept as small as possible to avoid too many filter failures, since expensive exact computation must be used in that case, and the computations of error bound only induce an overhead [6]. The combination of exact computation and filtering techniques allows to get both fast and exact results.

4.4 Reference Counting

After the previous improvements, we profiled the circular kernel using GPROF[4] and VALGRIND[5] and we discovered that around 15% of the whole running time

[4] `http://www.gnu.org/software/binutils/manual/gprof-2.9.1`
[5] `http://valgrind.kde.org/`

was spent on calling the `CGAL::MP_Float` copy constructor. A good option to handle this copy construction bottleneck is to use a *reference counting* technique [7].

4.5 Different Algebraic Number Type Representation

In spite of the overall good performance obtained with the previous improvements, high execution times are obtained on the *vlsi_8* data set, which is a showstopper for the use of the circular kernel on industrial data.

A number of type `Root_of_2` is a root of a polynomial ax^2+bx+c and is represented by the three coefficients a, b, c of a ring type `RT` and a boolean (see Section 2). The basic computations on such an algebraic number reduce to manipulations of the coefficients. When `RT` is a multi-precision number type (which is necessary to achieve exact computations), the time complexity of these manipulations grows with the length of the numbers. The length increases when computations are cascaded.

Storing α, β, γ as numbers of a field type `FT`, such that $\alpha+\beta\sqrt{\gamma}$ is the root, allows to reduce the lengths of multi-precision numbers.

4.6 Geometric Filtering

We mentioned interval arithmetic filtering techniques in Section 4.3.2. A similar scheme can be applied at the geometric level [14] for filtering predicates: fast approximate computation is done first; most of the time, the error bounds allow to certify the result; in bad cases, the filter fails and the result is computed exactly. Using axis-aligned bounding boxes as enclosing shapes is very appropriate for our application.

References

[1] Boost, C++ libraries. `http://www.boost.org`.

[2] Cgal, Computational Geometry Algorithms Library. `http://www.cgal.org`.

[3] Pankaj K. Agarwal and Micha Sharir. Arrangements and their applications. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[4] Paul Anderson and Gail Anderson. *Navigating C++ and Object-Oriented Design*. Prentice Hall, 2003.

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

[6] H. Brönnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, 109:25–47, 2001.

[7] George E. Collins. A method for overlapping and erasure of lists. *Communications of the ACM*, 3(12):655–657, December 1960.

[8] Olivier Devillers, Alexandra Fronville, Bernard Mourrain, and Monique Teillaud. Algebraic methods and arithmetic filtering for exact predicates on circle arcs. *Comput. Geom. Theory Appl.*, 22:119–142, 2002.

[9] I. Emiris and E. P. Tsigaridas. Computing with real algebraic numbers of small degree. In *Proc. 12th European Symposium on Algorithms, LNCS 3221*, pages 652–663. Springer-Verlag, 2004.

[10] Ioannis Z. Emiris, Athanasios Kakargias, Sylvain Pion, Monique Teillaud, and Elias P. Tsigaridas. Towards an open curved kernel. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 438–446, 2004.

[11] Efi Fogel, Dan Halperin, Lutz Kettner, Monique Teillaud, Ron Wein, and Nicola Wolpert. Arrangements. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*. Springer-Verlag, Mathematics and Visualization, 2006.

[12] Efi Fogel and Monique Teillaud. Generic programming and the CGAL library. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*. Springer-Verlag, Mathematics and Visualization, 2006.

[13] Efi Fogel, Ron Wein, Baruch Zukerman, and Dan Halperin. 2D regularized boolean set-operations. In CGAL Editorial Board, editor, *CGAL-3.2 User and Reference Manual*. 2006.

[14] Stefan Funke and Kurt Mehlhorn. Look: A lazy object-oriented kernel for geometric computation. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 156–165, 2000.

[15] D. Halperin. Arrangements. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 21, pages 389–412. CRC Press LLC, Boca Raton, FL, 1997.

[16] Susan Hert, Michael Hoffmann, Lutz Kettner, Sylvain Pion, and Michael Seel. An adaptable and extensible geometry kernel. *Computational Geometry: Theory and Applications*, To appear. Special issue on CGAL.

[17] Menelaos I. Karavelas and Ioannis 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.

[18] J. Keyser, T. Culver, M. Foskey, S. Krishnan, and D. Manocha. ESOLID - a system for exact boundary evaluation. *Computer-Aided Design*, 26(2):175–193, 2004.

[19] N.C. Myers. Traits: a new and useful template technique. *C++ Report*, June 1995.

[20] Sylvain Pion and Monique Teillaud. 2D circular kernel. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.2 edition, 2006.

[21] Ron Wein, Efi Fogel, Baruch Zukerman, and Dan Halperin. 2D arrangements. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.2 edition, 2006.

[22] Ron Wein, Efi Fogel, Baruch Zukerman, and Dan Halperin. Advanced programming techniques applied to CGAL's arrangement package. *Computational Geometry: Theory and Applications*, To appear. Special issue on CGAL.

[23] C. K. Yap and T. Dubé. The exact computation paradigm. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 452–492. World Scientific, Singapore, 2nd edition, 1995.