Exact and efficient computations on circles in CGAL
Pedro M. M. de Castro, Sylvain Pion, Monique Teillaud

To cite this version:
Pedro M. M. de Castro, Sylvain Pion, Monique Teillaud. Exact and efficient computations on circles in CGAL. 23rd European Workshop on Computational Geometry, 2007, Graz, Austria. hal-02196933

HAL Id: hal-02196933
https://inria.hal.science/hal-02196933
Submitted on 29 Jul 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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

\(^1\)see [http://research.cs.tamu.edu/keyser/geom/esolid/](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 interface level provides types already defined by the linear kernel, plus three new types: CircularArcPoint2 for points on circles, and CircularArc2 and LineArc2 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, CircularArc2 and LineArc2. The traits class built on the circular kernel uses the boost::variant class[^1] [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].

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).

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

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.

[^1]: 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].

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\_{\text{of}\_2} number type was improved by following two directions.

4.3.1 Optimizing Particular Cases

Every time when the rationality of a Root\_{\text{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\textsuperscript{4} and valgrind\textsuperscript{5} and we discovered that around 15% of the whole running time

\begin{table}
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
VLSI & N & V & E & F \\
\hline
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 & 2,690,550 & 1,257,684 \\
\hline
\end{tabular}
\end{table}

\begin{table}
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
Distributions & N & V & E & F \\
\hline
very dense (a) & 400 & 160,400 & 320,000 & 159,602 \\
sparse (b) & 441 & 882 & 882 & 442 \\
\hline
\end{tabular}
\end{table}

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.

\textsuperscript{4}http://www.gnu.org/software/binutils/manual/gprof-2.9.1
\textsuperscript{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 data set, which is a showstopper for the use of the circular kernel on industrial data.

A number of type Root₂ 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 \( \alpha, \beta, \gamma \) as numbers of a field type FT, such that \( \alpha + 2 \beta \sqrt{\gamma} \) 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