Voronoi diagram of ellipses in CGAL

Ioannis Z. Emiris
University of Athens, Greece
emiris@di.uoa.gr

Elias P. Tsigaridas
INRIA Sophia-Antipolis, France
elias.tsigaridas@inria.fr

George M. Tzoumas
University of Athens, Greece
geotz@di.uoa.gr

Abstract

We present a CGAL-based implementation of the Voronoi diagram of ellipses in 2D. Based on the package for the Apollonius diagram in the plane and exploiting the generic programming principle, our main additions concern the implementation of the predicates. For this, we develop practical algebraic methods like trivariate system resultant computation, thus illustrating the concept of algebraic support to a geometric library via an algebraic kernel.

1 Introduction

This paper discusses our implementation of an algorithm for the Euclidean Voronoi diagram of ellipses, in the exact computation paradigm, cf. fig. 1. This is the first complete solution of how to implement the Voronoi diagram (and Delaunay graph) of ellipses under the exact computation paradigm, a non-trivial problem tackled in nonlinear computational geometry, because of the complexity of the algebraic operations involved. In particular, the real algebraic numbers involved in the InCircle predicate (defined later) are of degree 184.

We apply the incremental algorithm of [6] where the insertion of a new ellipse to the current Voronoi diagram consists of the following: (i) Find a conflict between an edge of the current diagram and the new ellipse, or detect that the latter is internal (hidden) in another ellipse, in which case it does not affect the diagram. (ii) Find the entire conflict region of the new ellipse and update the dual Delaunay graph.

We focus on *non-intersecting* ellipses, given *parametrically* (or *constructively*) in terms of their axes, center and rotation angle, which are all rational [3]. Our code is based on the CGAL package for the Apollonius (or Voronoi) diagram of circles [2], which uses the same incremental algorithm. CGAL follows the generic programming paradigm, hence the main issue was to analyze and implement all 4 predicates for ellipses: (κ_1) given two ellipses and a point, decide which ellipse is closest to the point; (κ_2) given two ellipses, decide the position of a third one relative to a specified external bitangent of the first two; (κ_3) given three ellipses, decide the position of a fourth one relative to one (external tritangent) Voronoi circle of the first three; this is the InCircle predicate; (κ_4) given four ellipses, compute the part of the Voronoi edge that changes due to the insertion of a new ellipse.

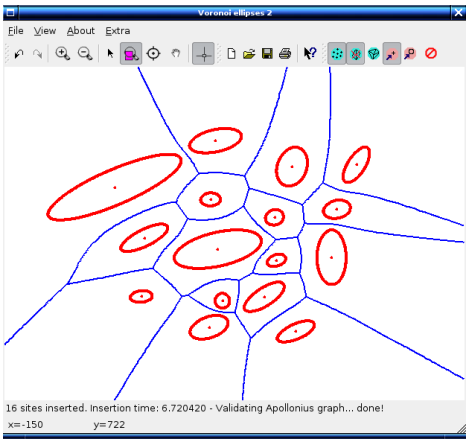

Figure 1: Voronoi diagram of ellipses.

The theoretical foundations of our approach were laid in [3]. A certified iterative subdivision algorithm with quadratic convergence was proposed in [4] for answering the InCircle predicate fast in non-degenerate instances using interval arithmetic and was a significant improvement over the corresponding algorithm in [3]. In [4], the authors worked with bisectors in the parametric space. So does the approach of [5] which is more general, since it applies to arbitrary Bézier or B-spline curves. However, this goes up to machine precision and is slower on ellipses.

Our main contribution is the first implementation[1] for the exact Voronoi diagram of ellipses, including a Graphical User Interface (GUI) for input, and visualization tools that can be used for any parametric curve. Our methods generalize to intersecting ellipses and pseudo-circles, but also to arbitrary closed smooth parametric curves. They can also compute an approximate Voronoi diagram of ellipses, with any predetermined precision. We implement certain algebraic methods in C++ to support InCircle, thus improving upon previous implementations. State-of-the-art *general* solvers are slow on InCircle, compared to our adapted solutions [3, 4].

The implemented algebraic methods rely, in specific

[1] www.di.uoa.gr/~geotz/vorell/

ways, on the algebraic library SYNAPS [2] and, through this software, library MPFR,[3] and NTL.[4] We thus illustrate the concept of providing algebraic support to a geometric computing library such as CGAL. Our algorithms are described in sec. 2.

Lastly, in sec. 4 we report on experiments when we vary the number of ellipses, their bitsize, and the degeneracy of their configuration. We also test our code on sets of circles, which allows us to compare it against the Apollonius package; our software is one to two orders of magnitude slower, which is to be expected because of the adapted methods employed for circles.

2 The InCircle predicate

An ellipse is given in rational parametric form, which is suitable for following its boundary in the subdivision algorithm. Moreover, given the parametric form it is always possible to derive the implicit one using only rational arithmetic.

InCircle is clearly the most challenging predicate and the only one that had not been satisfactorily implemented in C++. In order to decide InCircle, we need to represent the Voronoi circle suitably. In [3], we proved that there can be 184 complex tritangent circles to three ellipses, hence this is the degree of the algebraic numbers involved in an algebraic approach. The corresponding algebraic system is very costly to solve, see [3], therefore we take a different track. First, we apply the certified subdivision solver of [4] and, if the predicate cannot be decided, we solve system $\{Q, B_1, B_3\}$ below. This happens at (near)degenerate configurations, where any fixed precision may not suffice.

We express the Voronoi circle by considering the intersection of three bisectors, namely the system: $B_1(t,r) = B_2(r,s) = B_3(s,t) = 0$, where t, r, s are the parameters of the three ellipses. This system is used in [4] for the subdivision algorithm with quadratic convergence. Experiments showed that it is very expensive to compute its resultant in order to use it for exact solving, due to the degrees of the B_i's.

We employ resultants, which expresses the solvability of a system of $n+1$ equations in n variables, as a condition on the coefficients. For the required notions from computer algebra, see [8].

Consider the following alternative polynomial system: $Q(t,r,s) = B_1(t,r) = B_3(s,t) = 0$. Here, Q is the condition that makes the three normals of each ellipse intersect at a single point. Q is a polynomial of total degree 12, four in each variable t, r, s. This system has a mixed volume of 432, like the system of $\{B_1, B_2, B_3\}$ above, but, nonetheless, it leads to an efficient way of computing the resultant. In fact, by exploiting the fact that not all variables appear in all equations, we can compute the system's resultant via two Sylvester resultants. The resultant that eliminates s from two polynomials is denoted by res_s.

$$\begin{aligned} R_1(t,r) &= \mathrm{res}_s(Q(t,r,s), B_3(s,t)) \\ &= \underbrace{(at^{28}r^{24} + \cdots)}_{\bar{R}_1} P_1(t)(1+t^2)^4, \\ R_2(t) &= \mathrm{res}_r(\bar{R}_1(t,r), B_1(t,r)) \\ &= R(t)[P_2(t)]^6(1+t^2)^{28} \qquad (1) \end{aligned}$$

where P_1, P_2 are univariate polynomials of degree 12, $\bar{R}_1$ is the "interesting" factor of R_1 and R is a univariate polynomial of degree 184 which is the polynomial we are looking for.

Lemma 1 *The factorization of $R_1(t,r)$, given in expression (1), is always true. The shown factors of $R_2(t)$ are always present; if they appear at the indicated powers, then expression (1) gives the full factorization of $R_2(t)$.*

The proof is omitted, but can be found in [1]. P_1 and P_2 correspond to the condition that the normals to two ellipses are parallel. The last factor in each factorization has no real roots.

We have no complete proof for the exponents of the extraneous factors in $R_2(t)$. Still, the shown exponents are confirmed by every example we have tried. In practice, we divide out these factors until we obtain the resultant with the optimal degree, namely 184.

The above lemma and discussion allow us to exploit the fact that both Sylvester determinants are factored. The factors which have no real roots, or whose roots correspond to the normals being collinear (which is a case handled apart), are divided out. Hence, our approach reduces to solving a univariate polynomial over the reals and comparing real algebraic numbers; the latter may require a univariate GCD computation to identify the case of common roots. To the best of our knowledge, there is no C++ implementation capable of computing efficiently the resultant that appears in InCircle.

The certified subdivision of [4] exploits several geometric properties of the problem, and allows us to decide InCircle before full precision has been reached. The algorithm approximates Voronoi circle's tangency points with precision up to 10^{-15} in up to 100 msec, when using standard floating-point arithmetic. When the specified precision is not enough (in near-degenerate or degenerate cases), we proceed with the resultant computation.

3 Implementation

For the implementation of the required algebraic operations, we have relied on algebraic library SYNAPS

[2] synaps.inria.fr/
[3] www.mpfr.org/
[4] www.shoup.net/ntl/

and, to be more precise, to library Mathemagix.[5] First, we implemented Newton's iteration with interval arithmetic as part of the `subdivix` package of Mathemagix (or SYNAPS). This is important for the numerical iterative algorithm for INCIRCLE. The floating-point types, and the associated arithmetic operations, are implemented in the MPFR library.

In order to solve the degree-184 univariate polynomials for INCIRCLE, we use the fast implementation of Continued Fractions [7], available in SYNAPS. This is comparable and, in most cases, faster than state-of-the-art exact univariate solvers such as RS, but also numeric solvers such as MPSOLVE.[6] The algorithm isolates all real roots in rational intervals, which also allows us to compare roots from different resultants.

We implemented bivariate and univariate polynomial interpolation in SYNAPS, in order to compute the resultant of the system of bisectors, refer to eq. (1). This is essentially a Chinese remaindering algorithm in the ring of polynomials, where the moduli are values of the polynomial and the output are its coefficients. After computing the numeric Sylvester determinants by NTL (see below), we use lemma 1 to divide out the values of the factors described in that lemma. Therefore, the number of moduli does not depend on the total degree of the determinant but rather on the total degree of the factor of interest, considered as a univariate or bivariate polynomial.

When two roots are equal, we compute the GCD of the defining polynomials. For this, we use NTL which is, to the best of our knowledge, the only open source C++ library that provides efficient implementation of asymptotically fast algorithms for polynomial GCD's and univariate Sylvester-resultant computation. We developed an interface between NTL and SYNAPS. The interface eliminates the need for unnecessary copies, and allows us to work directly with NTL objects in SYNAPS (and vice versa). We take advantage of the fact that integer arithmetic in NTL can be based on GMP, and we built a wrapper around these objects.

The concept of an `Algebraic_kernel` for the Voronoi diagram of ellipses is quite demanding. It requires, just to name the most important operations, symbolic univariate polynomial resultant and GCD computations, real root isolation and comparison of real algebraic number and computation of the resultant of a trivariate polynomial system.

In connecting to algebraic software, it was important to separate the geometric from the algebraic operations. This work is in line with the concept of kernels, adopted by CGAL, where different types of operations are grouped into separate modules.

For the combinatorial part of the algorithm, we relied on CGAL. CGAL follows the generic programming paradigm, hence the main issue is to implement the 4 predicates for ellipses and generalize circular sites to ellipses. We are based on the `Apollonius_graph_2` package. Besides the predicates, the most important class is the `Apollonius_site_2`, which represents the sites of the Voronoi diagram; in our case the ellipses. We modified the corresponding class so as to inherit from our `Ellipse` class. A snapshot of the corresponding code is as follows:

```
template < class K >
class Apollonius_site_2: public Ellipse
{
public:
  typedef K                          Kernel;
  typedef typename K::Point_2        Point_2;
  typedef Apollonius_site_2<K>       Self;
  typedef typename K::FT             FT;
// Field Number Type
  typedef typename K::RT             RT;
// Ring Number Type
.... };
```

4 Experiments

With elliptic sites. Let us now see several results regarding elliptic sites. These experiments have been carried out on a P4 2.6-GHz machine with 1.5GB of RAM.

First, we consider the overall time for the construction of the combinatorial structure of the dual (Delaunay) graph. While the first few sites are inserted almost instantly, subsequent ones cause many updates on the graph and require about a second for their insertion. Total timings are shown in the first two columns of table 1 (top). The runtime increases roughly linearly with 15 sites or more and is about one second for each ellipse. This is in accordance with theoretical bounds, although larger instances shall have to be tested to verify this.

We have also measured the performance of the first three predicates with varying bitsize (cf. the subsequent columns of the table). Using randomly perturbed coefficients by either adding or subtracting 10^{-e}, we obtain large rational numbers. All runtimes appear to grow subquadratically in e. For predicates κ_1 and κ_2 the timings vary from a couple milliseconds for $e = 20$ to 280 msec for $e = 100$. For INCIRCLE, the subdivision algorithm was used, since the situation was non-degenerate, and was roughly 10 times slower. In case of degeneracies, the runtime of INCIRCLE is dominated by the resultant computation, varying from 25 sec for $e = 4$ to 231 sec for $e = 20$. Note that the resultant computation is over 50 times slower than the time required by the subdivision algorithm.

Finally, we measured the time needed for the subdivision algorithm to reach a precision of 2^{-b} when using MPFR floats. This version currently lacks some

[5] www.mathemagix.org/

[6] www.dm.unipi.it/cluster-pages/mpsolve/

# sites	insertion	e	κ_1	κ_2	κ_3
5	0.752	20	0.0270	0.033	0.49
10	5.916	40	0.0696	0.076	1.02
15	13.000	60	0.1286	0.135	1.70
20	19.200	80	0.1996	0.197	2.49
25	25.602	100	0.2848	0.285	3.40

e	resultant	b	subdivision
4	25.12	53	0.20
8	57.24	900	0.58
12	102.63	2000	1.06
16	160.44	12000	10.65
20	231.19	24000	31.61

Table 1: Performance of the predicates on elliptic sites. Timings are in seconds.

optimizations making it about 2 times slower than the one in [4] using ALIAS.[7] Standard floating-point precision is achieved in about 0.2s, while 1 second suffices for almost 2000 bits of precision. However, higher approximations slow down considerably, such as the 24k-bit approximation that needs about half a minute. This raises some questions on whether the theoretical separation bound of several million bits can be achieved in practice by any implementation.

Against the CGAL Apollonius package. We performed experiments per predicate, when the input is restricted to circles. We used three data sets, involving degenerate inputs, near-degenerate inputs (by randomly perturbing the degenerate ones by 10^{-e}) and random inputs. These experiments were carried out on a 1.83GHz Core 2 Duo processor with 1GB of RAM. The timings for the Apollonius package were several milliseconds, while for our implementation varied from several milliseconds to a few seconds.

In most cases, degenerate inputs are solved faster than near-degenerate ones because the former, from the algebraic point of view, imply lower degree algebraic numbers. This is not the case for the subdivision-based algorithm, since in the degenerate cases it has to use large precision in order to make a decision. Another interesting observation is that runtime increases linearly in the bitsize, especially with perturbed input. This is because we implemented algorithms of constant arithmetic time complexity, which do not depend on how close to degeneracy the configuration lies.

It seems that our current implementation of the predicates for ellipses is up to two orders of magnitude slower that the dedicated one, when we restrict to circles. The worst relative performance is observed, as expected, for INCIRCLE, which is the most expensive predicate and, in the case of circles, has been optimized. The best relative performance occurs for κ_2, because the two approaches follow similar algorithms. The difference of performance is not surprising, since the case of circles reduces to computations with real algebraic numbers of degree 2 and the Apollonius predicates are specifically designed to exploit this.

We may conclude that specialized implementations for predicates involving small degree algebraic numbers, combined with algorithms exploiting the geometric characteristics of the problem, are more efficient than generic approaches.

[7] www-sop.inria.fr/coprin/logiciels/ALIAS/

Acknowledgments. George Tzoumas is partially supported by State Scholarship Foundation of Greece, Grant No. 4631. All authors acknowledge partial support by IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-006413-2 (ACS - Algorithms for Complex Shapes). We thank Athanasios Kakargias for help with the overall code design, Michael Hemmer for suggestions on kernel design, and Costas Tsirogiannis for help with the GUI.

References

[1] I. Z. Emiris, E. P. Tsigaridas, and G. M. Tzoumas. A CGAL-based implementation for the Voronoi diagram of ellipses. Submitted. Manuscript available from http://www.di.uoa.gr/~geotz/.

[2] I.Z. Emiris and M.I. Karavelas. The predicates of the Apollonius diagram: algorithmic analysis and implementation. *Comp. Geom.: Theory & Appl., Spec. Issue on Robust Geometric Algorithms and their Implementations*, 33(1-2):18–57, 2006.

[3] I.Z. Emiris, E.P. Tsigaridas, and G.M. Tzoumas. The predicates for the Voronoi diagram of ellipses. In *Proc. Annual ACM Symp. on Computational Geometry*, pages 227–236, June 2006.

[4] I.Z. Emiris and G.M. Tzoumas. A real-time implementation of the predicates for the Voronoi diagram of parametric ellipses. In *Proc. ACM Symp. Solid Physical Modeling*, pages 133–142, Beijing, 2007.

[5] I. Hanniel, R. Muthuganapathy, G. Elber, and M.-S. Kim. Precise Voronoi cell extraction of free-form planar piecewise c^1-continuous closed rational curves. In *Proc. ACM Symp. Solid Phys. Modeling*, pages 51–59, Cambridge, Mass., 2005. (Best paper award).

[6] M.I. Karavelas and M. Yvinec. Voronoi diagram of convex objects in the plane. In *Proc. Europ. Symp. Algorithms*, LNCS, pages 337–348. Springer, 2003.

[7] E.P. Tsigaridas and I.Z. Emiris. On the complexity of real root isolation using continued fractions. *Theor. Comp. Science, Spec. Issue Comput. Algebraic Geom. & Applications*, 392(1-3):158–173, February 2008.

[8] C.K. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, New York, 2000.