Exact Delaunay graph of smooth convex pseudo-circles: General predicates, and implementation for ellipses

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

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

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

ABSTRACT

We examine the problem of computing exactly the Delaunay graph (and the dual Voronoi diagram) of a set of, possibly intersecting, smooth convex pseudo-circles in the Euclidean plane, given in parametric form. Pseudo-circles are (convex) sites, every pair of which has at most two intersecting points. The Delaunay graph is constructed incrementally. Our first contribution is to propose robust end efficient algorithms for all required predicates, thus generalizing our earlier algorithms for ellipses, and we analyze their algebraic complexity, under the exact computation paradigm. Second, we focus on InCircle, which is the hardest predicate, and express it by a simple sparse 5×5 polynomial system, which allows for an efficient implementation by means of successive Sylvester resultants and a new factorization lemma. The third contribution is our CGAL-based C++ software for the case of ellipses, which is the first exact implementation for the problem. Our code spends about 98 sec to construct the Delaunay graph of 128 non-intersecting ellipses, when few degeneracies occur. It is faster than the CGAL segment Delaunay graph, when ellipses are approximated by k-gons for $k > 15$.

1. INTRODUCTION

Computing the Delaunay graph, and its dual Voronoi diagram, of a set of sites in the plane has been studied extensively due to its links to other important questions, such as medial axis computations, but also its numerous applications, including motion planning among obstacles, assembly, surface reconstruction, and crystallography [3]. Our work is also motivated by other problems besides the Delaunay graph: The predicates examined can be used to implement an algorithm for the convex hull of smooth convex pseudo-circles [8], whereas some of them appear in the computation of the visibility map among ellipses [13] [1].

[1]The software being developed for this problem uses our predicates.

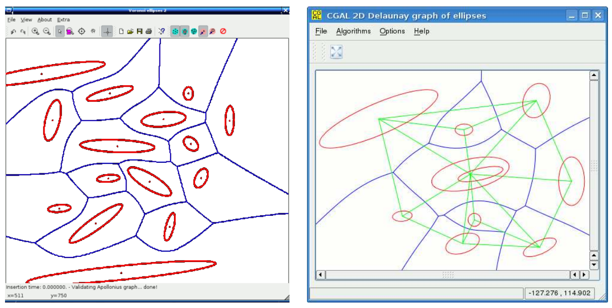
File View About Extra
insertion time: 0.000000. - Validating Apollonius graph... done!
CGAL 2D Delaunay graph of ellipses
File Algorithms Options Help
-127.276 , 114.902

Figure 1: Left: Voronoi diagram of 15 ellipses; Right: Voronoi diagram and Delaunay graph of 10 ellipses

In contrast to most existing approaches, our work guarantees the exactness of the Delaunay graph. This means that all combinatorial information is correct, namely the definition of the graph's edges, which, of course, have no geometric realization. The dual Voronoi diagram involves algebraic numbers for representing the vertices and bisector edges: our algorithms and software handle these numbers exactly. For instance, they can answer all point-location queries correctly. Hence, we say that our representation of the dual Voronoi diagram adheres to the principles of the exact computation paradigm, a distinctive feature in the realm of non-linear computational geometry. For drawing the diagram, our methods allow for an approximation of these numbers with arbitrary precision, and this precision need not be fixed in advance.

We consider smooth convex sites which intersect as pseudo-circles, i.e. there are at most two intersection points per pair. This is one of the hardest problems considered (and implemented) under the exact computation paradigm. Thus, our work explores the current limits of non-linear computational geometry. Naturally, purely algebraic techniques have to be enhanced by symbolic-numeric methods in order to arrive at practical implementations. The validity of our approach is illustrated by experiments with our software for the case of non-intersecting ellipses, which demonstrate that carefully implemented algebraic procedures incur a very reasonable overhead. In particular, our code is faster than constructing the Delaunay graph of closed curved objects each approximated by > 200 points roughly, or > 15 segments.

1.1 Previous work

Input sites have usually been linear objects, the hardest cases being line segments and polygons [15, 17]; moreover, only the latter yields an exact output. The approximation of smooth curved objects by (non-smooth) linear or circular segments may introduce artifacts and new branches in the Voronoi diagram, thus necessitating post-processing. It may even yield topologically incorrect results, as explained in [22].

The Voronoi diagram has been studied in the case of planar sites with curved boundaries [22, 2], where topological properties are demonstrated, including the type of bisector curves, though the predicates and their implementation are not considered. There are works that compute the planar Voronoi diagram approximately: In [14], curve bisectors are traced within machine precision to compute a single Voronoi cell of a set of rational C^1-continuous parametric closed curves. The runtime of their implementation varies between a few seconds and a few minutes. It is briefly argued that the method extends to exact arithmetic, but without elaborating on the underlying algebraic computations or the handling of degeneracies. In another work [23], the boundary of the sites is traced with a prescribed precision, while [21] suggests working with lower-degree approximations of bisectors of curved sites. Finally, error-bounded offset approximations of planar NURBS for the case of non-intersecting curves are considered in [24] and their implementation runtimes are withing a couple of seconds for up to three sites. See also [19] for the Delaunay graph of circles, which maintains topological consistency but not geometric exactness.

Curve-curve bisectors of parametric curves are considered in medial axis computations. In [1] the medial axis of a simply connected planar domain is computed, using a divide and conquer algorithm. The boundary of the domain is approximated via arc-splines. The correct medial axis up to a predefined input accuracy is computed.

Few works have studied *exact* Delaunay graphs for curved objects. In the case of circles, the exact and efficient implementation of [9] is now part of CGAL [4]. There is also the very efficient and robust implementation of [15], but relies on floating-point computations. For a more recent implementation that treats (non-intersecting) line segments and circular arcs, see [16]. Conics were considered in [3], but only in a purely theoretical framework. Moreover, the algebraic conditions derived were not optimal, leading to a prohibitively high algebraic complexity. In fact, the approach relied on eigenvector computations, hence was not exact.

In [18], the authors study the properties of smooth convex, possibly intersecting, pseudo-circles. They show that the Voronoi diagram of these sites belongs to the class of abstract Voronoi diagrams [20] and propose an incremental algorithm that relies on certain geometric predicates.

Our own previous work [11] studied non-intersecting ellipses, and proposed exact symbolic algorithms for the predicates required by the incremental algorithm of [18]. Predicates SIDEOFBISECTOR, DISTANCEFROMBITANGENT were implemented. We established a tight bound of 184 complex tritangent circles to 3 non-intersecting ellipses. The upper bound on the number of real circles is still open, but we have examples with 76 real tritangent circles, when the ellipses intersect [2]. Predicate INCIRCLE had been implemented in MAPLE, and used a different polynomial system than the one in this paper. Some of its properties had been observed without proof; this is rectified below to yield an optimal method for resultant computation, using a more suitable polynomial system.

In [12], the authors proposed a certified method for INCIRCLE, relying on a Newton-like numerical subdivision, which exploited the geometry of the problem and exhibits quadratic convergence for non-intersecting ellipses. The idea was to filter non-degenerate configurations. Its implementation used double precision interval arithmetic library ALIAS[3] to decide most non-degenerate configurations. In his work, we reimplement the proposed subdivision-based method in C++ using multi-precision floating point arithmetic. Experiments with state-of-the-art generic algebraic software on INCIRCLE [11], and generic numeric solvers [12], imply that our specialized C++ implementation is much more efficient, both in the exact as well as in the certified numeric part.

1.2 Our contribution

We extend previous results, which considered only non-intersecting ellipses, to smooth convex, possibly intersecting, pseudo-circles. This is the hardest step towards arbitrarily intersecting objects and requires re-working the predicates, especially INCIRCLE. At the same time, pseudo-circles are quite powerful. For instance, the Voronoi diagram in free (complementary) space of a set of arbitrarily intersecting convex objects coincides with the diagram of appropriate pseudo-circles [18]. We propose algorithms for all necessary predicates and examine the algebraic operations required for an efficient exact implementation. The analysis of the bit complexity of the required predicates, besides its theoretical importance, sheds light to the intrinsic complexity of computing with non-linear objects in an exact way. It allows us to bound the number of bits needed for an exact computation, to identify the time consuming parts of the algorithm, and finally, to discover the parts where symbolic-numeric techniques could be used to speed up the implementation. Moreover, the (sub-quadratic) bounds that we derived for curves of small degree are confirmed by our experiments.

To express the Voronoi circle of parametric curves, we introduce a new 5×5 polynomial system, where we trade number of equations and unknowns for polynomial degree. Although there are no determinantal formulae, in general, for the resultant of such a system, here we exploit their structure to find a succession of Sylvester determinants which yields a multiple of the resultant, where the extraneous factor is known *a priori*. This also bounds the degree of the algebraic numbers involved. In the case of arbitrary smooth parametric curves, cor. 7 provides almost all the extraneous factors of the resultant of the corresponding system. In the case of the conics we provide the complete factorization of the resultant (Th. 5). The technique that we used is quite general and could be used to other problems as well.

Finally, we present an implementation[4] in C++ for the case of non-intersecting ellipses (cf. fig. 1 left and 11 left), based on the incremental algorithm of [18], extending CGAL's existing implementation for circles. Our software is being extended to handle smooth convex pseudo-circles and, eventually, arbitrary smooth convex objects. One novelty is that we implement all the predicates in CGAL, by using certain algebraic libraries as explained in the sequel. This is the first implementation under the exact computation paradigm for

[2]G. Elber first showed us such examples.

[3]http://www-sop.inria.fr/coprin/logiciels/ALIAS/

[4]http://www.di.uoa.gr/~geotz/vorell/

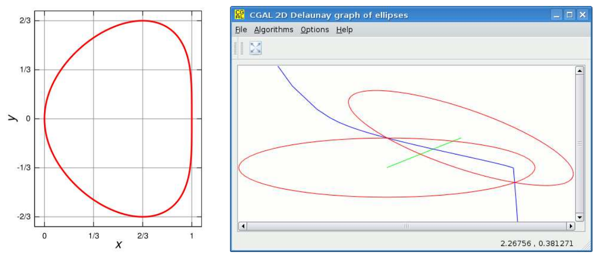

Figure 2: Left: The Bean curve $t \mapsto (\frac{1+t^2}{t^4+t^2+1}, \frac{t(1+t^2)}{t^4+t^2+1})$; Right: Voronoi diagram of two intersecting ellipses

sites more complex than circles, and illustrates the use of powerful algebraic techniques adapted to the underlying geometry. It spends about 98 sec for the Delaunay graph of 128 ellipses with few degeneracies. More importantly, it is faster than CGAL's segment and point Delaunay graph, when ellipses are approximated by 16-gons and, respectively, by > 200 points roughly.

The rest of the paper is organized as follows. The next section introduces notation, and examines some basic operations on the sites, whereas in section 3 we study the predicates in the case of general parametric curves from a computational point of view. In section 4 we deal with InCircle, proving certain geometric and algebraic properties leading to its efficient implementation. Section 5 presents our implementation in C++ and benchmarks to compare it against CGAL's implementation for circles as well as for points and polygons, concluding with future work.

2. PRELIMINARIES

2.1 Notation

Our input is smooth convex closed curves given in parametric form. For an example of such a curve we refer the reader to fig. 2 left. Smoothness allows the tangent (and normal) line at any point of the curve to be well-defined. We denote by C_t a smooth closed convex curve parametrized by t. We refer to a point p on C_t with parameter value $\hat{t}$ by $p_{\hat{t}}$, or simply by $\hat{t}$, when it is clear from context. By ${C_t}^{\circ}$ we denote the region bounded by the curve C_t. $\mathbf{C_t}$ is a smooth convex object (site), so that if p denotes a point in the plane, $p \in \mathbf{C_t} \iff p \in C_t \cup {C_t}^{\circ}$. When two sites intersect, we assume that their boundaries have at most two intersections, i.e. they form *pseudo-circles.* A curve C_t is given by the map

$$C_t : [a, b] \ni t \mapsto (X_t(t), Y_t(t)) = \left(\frac{F_t(t)}{H_t(t)}, \frac{G_t(t)}{H_t(t)}\right), \qquad (1)$$

but actual denominators *can differ;* we use (1) for simplicity in our proofs.

Here F_t, G_t and H_t are polynomials in $\mathbb{Z}[t]$, with degrees bounded by d and maximum coefficient bitsize bounded by τ. Moreover, $a, b \in \mathbb{Q} \cup \{\pm\infty\}$. All algorithms, predicates and the corresponding analysis are valid for any parametric curve, even when the polynomials have different degrees, though we use (1) for simplicity. We assume that $H_t(t) \neq 0$, for any $t \in [a, b]$. To simplify notation we write F_t instead of $F_t(t)$, and denote its derivative with respect to t as F'_t. When $d = 2$ the curves defined are conics: ellipses and circles are the only closed convex curves represented.

In what follows $\mathcal{O}_B$ means bit, while $\mathcal{O}$ means arithmetic complexity. The $\widetilde{\mathcal{O}}_B$ and $\widetilde{\mathcal{O}}$ notations mean that we ignore (poly-)logarithmic factors. For a polynomial $A = \sum_{i=1}^{d} a_i x^i$ in $\mathbb{Z}[x]$, $\mathtt{dg}(A)$ denotes its degree. By $\mathcal{L}(A)$ we denote an upper bound on the bitsize of the coefficients of A (including a bit for the sign). For $a \in \mathbb{Q}$, $\mathcal{L}(a) \geq 1$ is the maximum bitsize of the numerator and the denominator. We choose to represent real algebraic numbers, i.e. the real roots of an integer polynomial by the so-called *isolating interval* representation. For such a number α, this representation consists of a square-free polynomial, say A, which vanishes on α and a (rational) interval, say $[a, b]$, containing α and no other real root of A. We write $\alpha \cong (f, [a, b])$. To simplify notation, we also assume that for any polynomial it holds $\log(\mathtt{dg}(A)) = \mathcal{O}(\mathcal{L}(A))$.

2.2 Tangent and normal

Le $p_t(X_t, Y_t)$ be a point on the curve C_t. The equation of the tangent line at p_t is $(T_t) : (y - Y_t)X'_t - (x - X_t)Y'_t = 0$. After the substitution of the rational polynomial functions of X_t and Y_t, and elimination of the denominators, we get a polynomial in $\mathbb{Z}[x, y, t]$. By abuse of notation we denote this polynomial by $T_t(x, y, t)$: it is linear in x, y and of degree $\leq 2d - 2$ in t. If the maximum bitsize of F_t, G_t, H_t is τ, then the bitsize of T_t is $\leq 2\tau + 2\log(d)$.

The equation of the line that supports the normal at p_t is $(N_t) : (x - X_t)X'_t + (y - Y_t)Y'_t = 0$. As in the case of the tangent, after substitutions and elimination of the denominators we come up with a polynomial $N_t(x, y, t) \in \mathbb{Z}[x, y, t]$; which is linear in x and y, of degree $\leq 3d - 2$ in t and of bitsize $\leq 3\tau + 3\log(d)$.

2.3 Primitives PointInside and RelativePos

Given a curve C_t, if we consider rationals $t_1 \neq t_2$ in $[a, b]$, where the bitsize of a and b is bounded by σ, then the point $((X_t(t_1)+X_t(t_2))/2, (Y_t(t_1)+Y_t(t_2))/2)$ belongs to ${C_t}^{\circ}$; such a point is p in fig. 3. The bitsize of its coordinates is $\mathcal{O}(d\sigma + \tau)$, because $F_t(t_1)$ is of bitsize $\leq d\sigma + \tau + \log d$. The bit complexity of the primitive is dominated by the evaluations, hence $\widetilde{\mathcal{O}}_B(d^2\sigma + d\tau)$.

Now we characterize the relative position of sites $\mathbf{C_t}$, $\mathbf{C_r}$, i.e., whether they are separated, intersecting, externally or internally tangent, or if one is contained inside the other. The computation and characterization of all their bitangent lines suffices, due to the following properties: (i) C_t, C_r intersect as pseudo-circles iff they have at most two external bitangent lines. (ii) If one is contained inside the other, then there are 0 internal bitangent lines, and either 0 (boundaries separated) or up to a constant number (boundaries tangent) of external bitangent lines; additional computation is needed to distinguish this case from (i). (iii) If the sites do not intersect, then they have 2 internal and 2 external bitangent lines (only one internal bitangent if they are externally tangent). (iv) If the sites admit more than 2 external bitangent lines, then they do not form pseudo-circles. The same technique decides the relative position of a point and a site since, if a point is interior to a site, there are no supporting lines tangent to the site. Consequently, this property is used to identify a site contained inside another, by considering a boundary point of one of the sites. In the case of conics (i.e., ellipses) the method of counting and characterizing the bitangents involves solving a polynomial of degree 4.

There are cases where we need to compute the intersection

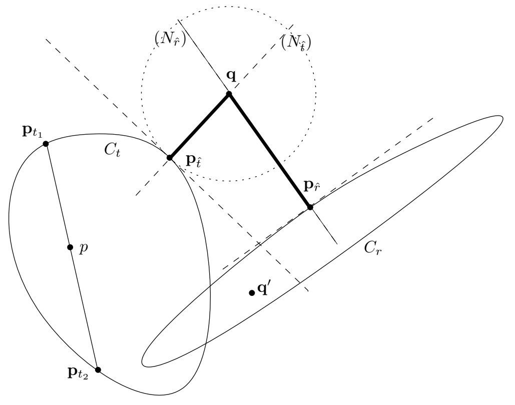

Figure 3: Deciding SideOfBisector

points of sites $\mathbf{C_t}, \mathbf{C_r}$. These are solutions of the resultant of $F_tH_r - F_rH_t$ and $G_tH_r - G_rH_t$ w.r.t. t. The resultant is a polynomial in r of degree d^2, after disregarding the redundant factor H_t^d.

3. BASIC PREDICATES

In this section we examine the predicates for the incremental algorithm of [18], the most important of which are: SideOfBisector that is used to perform nearest neighbor location, DistanceFromBitangent that is used to determine the existence of a Voronoi circle and also to determine if an infinite (unbounded) edge of the current Voronoi diagram will be modified, InCircle that determines if a vertex of the current Voronoi diagram is *in conflict* with the newly inserted site and shall disappear, and EdgeConflictType which is used to determine which part of an existing edge of the current Voronoi diagram will be modified due to the insertion of a new site. The randomized complexity of an insertion (in a diagram with n sites) is $\mathcal{O}(\log^2 n)$ for disjoint sites and $\mathcal{O}(n)$ for intersecting or hidden sites.

Computing an exact Delaunay graph implies that we identify correctly all degenerate cases, including Voronoi circles tangent to more than 3 sites.

The insertion of a new site consists roughly of the following: (i) Locate the nearest neighbor of the new site, (ii) Find a conflict between an edge of the current diagram and the new site, or detect that the latter is internal (hidden) in another site, in which case it does not affect the Delaunay graph nor the Voronoi diagram. (iii) Find the entire *conflict region,* defined as that part of the Voronoi diagram which changes due to the insertion of the new site, and update the dual Delaunay graph. We analyze predicates SideOfBisector, DistanceFromBitangent and EdgeConflictType, needed for the above three steps, while predicate InCircle is presented separately in the next section due to its higher complexity.

3.1 SideOfBisector

First, we recall from [18] the definition of distance. Given a site $\mathbf{C_t}$ and a point q in the plane, the (signed) distance $\delta(q, \mathbf{C_t})$ from q to $\mathbf{C_t}$ equals $min_{x\in C_t}||q-x||$ when $q \notin \mathbf{C_t}$ and to $-min_{x\in C_t}||q-x||$ when $q \in \mathbf{C_t}$, where $||\cdot||$ denotes the Euclidean norm. The absolute value of the distance equals the radius of the smallest circle centered at q tangent to $\mathbf{C_t}$.

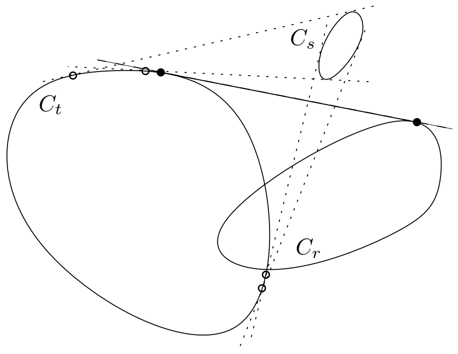

Figure 4: Deciding DistanceFromBitangent

Given two sites $\mathbf{C_t}$ and $\mathbf{C_r}$ and a point $q = (q_1, q_2) \in \mathbb{Q}^2$, this predicate decides which site is closest to the point. If $q \notin \mathbf{C_t}$ and $q \in \mathbf{C_r}$, then q is closer to $\mathbf{C_r}$. For example, in fig. 3, q' is closer to $\mathbf{C_r}$. Otherwise, if q lies outside or inside both sites, we have to compare the distances from q to the two curves. To find the site closest to q it suffices to compare the (squared) lengths of segments $qp_{\hat t}$ and $qp_{\hat r}$. For example in fig. 3 to conclude which site is closest to q, we have to compare $qp_{\hat t}$ and $qp_{\hat r}$.

In the sequel we analyze in detail the algebraic operations involved and their bit complexity. The goal is to compute the (squared) lengths of the segments $qp_{\hat t}$ and $qp_{\hat r}$, in isolating interval representation, and compare them. Let σ be the bitsize of the coordinates of q, and $N = \max\{d, \sigma, \tau\}$.

To compute the (squared) length $qp_{\hat t}$, we need to compute the coordinates of the point $p_{\hat t}$, for which we need the value of the parameter $\hat t$, which in turn is a real root of $N_t(q_1, q_2, t)$. The (squared) length of the segment(s) $q\,p_{\hat t}$ is

$$A_q(\hat t) = \|q - p_{\hat t}\|^2 = (X_t(\hat t) - q_1)^2 + (Y_t(\hat t) - q_2)^2.$$

Polynomial $N_t(q_1, q_2, t)$, belongs to $\mathbb{Q}[t]$, its degree is $\leq 3d-2$, and its bitsize is $\widetilde{\mathcal{O}}(\tau + \sigma)$. We compute the isolating interval representation of its real roots in $\widetilde{\mathcal{O}}_B(d^6 + d^4\tau^2 + d^4\sigma^2)$, the bitsize of the endpoints of the isolating intervals is $\mathcal{O}(d^2 + d\tau + d\sigma)$. For each of the real roots that we have computed, $\hat t$, we form the number $A_q(\hat t)$. All these numbers belong to a single algebraic extension $\mathbb{Q}(\hat t)$, and the smallest of them is the squared length $qp_{\hat t}$, i.e. the squared distance of q from the curve.

It remains to represent the smallest $A_q(\hat t)$ in isolating interval representation. For this we exploit the fact that we are working in the quotient ring $\mathbb{Z}[t]/N_t(q_1, q_2, t)$. The smallest squared distance that we are looking for, is the real algebraic number δ, which is the smallest positive real root of the polynomial $R \in \mathbb{Z}[x]$, given by $R(x) = \mathsf{Res}_t(x - A_q(t), N_t(q_1, q_2, t))$ (i.e., the resultant of the two polynomials w.r.t. t). We can compute the resultant in $\widetilde{\mathcal{O}}_B(d^4 + d^3\tau + d^3\sigma)$ [7]. It holds that $\deg(R) = \mathcal{O}(d)$ and $\mathcal{L}(R) = \mathcal{O}(d^2 + d\tau + d\sigma)$. We compute the isolating interval representation $\delta \cong (R_{red}, [0, c])$ in $\widetilde{\mathcal{O}}_B(d^6 + d^5\tau + d^5\sigma)$, where $\mathcal{L}(c) = \mathcal{O}(d^3 + d^2\tau + d^2\sigma)$.

We do the same for q and the other curve, and we compute the squared distance ζ. We compare δ and ζ in $\widetilde{\mathcal{O}}_B(d^6 + d^5\tau + d^5\sigma)$. The overall cost is $\widetilde{\mathcal{O}}_B(N^6)$, where $N = \max\{d, \sigma, \tau\}$.

The degree of the real algebraic numbers involved in the predicate, is $3d - 2$. For conics, this degree becomes 4, and is optimal, as shown in [11], where it was obtained using the pencil of two ellipses. Moreover, the complexity bound for the case of conics is $\widetilde{\mathcal{O}}_B(N)$.

3.2 DistanceFromBitangent

Consider two sites, $\mathbf{C_t}$ and $\mathbf{C_r}$, and their CCW bitangent line, which leaves both sites on the right hand-side, as we move from the tangency point of C_t to the tangency point of C_r; such a bitangent appears in fig. 4. This line divides the plane into two halfplanes and DistanceFromBitangent (abbreviated by DFB from now on) decides whether a third site, $\mathbf{C_s}$, lies in the same halfplane as the other two. The realization of this predicate consists in deciding the relative position of $\mathbf{C_s}$ with respect to the bitangent line.

We split the problem to two sub-problems. The first consists in computing the external bitangent of interest, while the second consists in deciding the relative position of the third site with respect to this bitangent.

To compute all the bitangents of C_t and C_r, we consider the polynomial that defines the equation of a tangent line to C_t, that also crosses C_r. For the line to be tangent to both sites, the discriminant of the corresponding polynomial should vanish. Among the real roots of the discriminant are the values of the parameter that correspond to the tangency points, which in turn allow us to compute the implicit equations of the bitangent lines. We characterize the bitangents as external or internal ones by computing their relative position w.r.t. to (rational) points inside the sites.

To decide the position of $\mathbf{C_s}$ w.r.t. the CCW bitangent line, we first check if the line crosses C_s. If this is the case, then the predicate is answered immediately, since C_s cannot lie within the same halfplane as C_t and C_r. If this line does not cross C_s, then to decide in which halfplane, C_s is lying, it suffices to compute the sign of the evaluation of the polynomials of the bitangent over an interior point of C_s. However, we can do better, as illustrated in fig. 4. We consider the tangency points of all the bitangents of C_t and C_s, shown with circular marks in fig. 4. We can then decide the position of C_s by the ordering of the aforementioned points *and* the tangency point of the bitangent and C_t, shown with solid circular mark in the same figure.

To analyze the bit complexity of the algebraic operations involved, recall that the degree of the defining polynomials of the curves is bounded by d and the maximum coefficient bitsize by τ. Let $T_t(x, y, t)$ be the polynomial of the tangent line to C_t. To compute the intersections of this line with C_r, we replace x and y with X_r and Y_r, from curve C_r. After elimination of the denominators we get a polynomial $T_{tr}(t, r) \in \mathbb{Z}[r, t]$, with total degree $\leq 3d - 2$ and $\mathcal{L}(T_{tr}) \leq 3\tau + 3\log(d)$. Moreover, the degree w.r.t. t, resp. r, is bounded by $2d - 2$, resp. d. For the line to be tangent to both curves, we demand the discriminant of T_{tr} w.r.t. r, $\Lambda_{tr} \in \mathbb{Z}[t]$, to vanish. The discriminant is computed in $\widetilde{\mathcal{O}}_B(d^4\tau)$, and it holds that $\mathsf{dg}(\Lambda_{tr}) \leq 4(d-1)^2$, and $\mathcal{L}(\Lambda_{tr}) = \mathcal{O}(d\tau)$ [7]. The real roots of the discriminant, say $\hat{t}$, are the values of the parameter that correspond to the points where a tangent line to C_t is also tangent to C_r. We can isolate the real roots of $\Lambda_{tr}(t)$ in $\widetilde{\mathcal{O}}_B(d^{12} + d^{10}\tau^2)$, and the endpoints of these intervals are of bitsize $\mathcal{O}(d^4 + d^3\tau)$ [10].

With each real root, $\hat{t}$, of $\Lambda_{tr}(t)$ we compute the implicit equation of a bitangent line, which is $T_{\hat{t}}(x, y) = T_t(x, y, \hat{t}) \in (\mathbb{Z}[\hat{t}])[x, y]$. We characterize it as external to both curves iff its equation yields the same sign when evaluated at an interior point of each site. An interior point, (u_1, u_2), can be computed using primitive PointInside, and the bitsize of its coordinates is $\mathcal{O}(d\sigma + \tau)$. Hence, $T_t(u_1, u_2, t)$ is of degree $\leq 2d - 2$ and bitsize $\widetilde{\mathcal{O}}(d\sigma + \tau)$. We compute the sign of $(T_t(u_1, u_2, \hat{t}))$ in $\widetilde{\mathcal{O}}_B(d^6\tau + d^6\sigma)$ [7].

It remains to compute the position of C_s w.r.t. the CCW external bitangent, by determining the ordering of the tangency points, as mentioned earlier. The tangency points of the bitangent depend on the real roots of $\Lambda_{t,r}(t)$, while the tangency points of all the bitangents of C_t and C_s, depend on the real roots of $\Lambda_{t,s}(t)$. Thus it suffices to compare the real roots of these polynomials, which can be done in $\widetilde{\mathcal{O}}_B(d^6\tau + d^6\sigma)$ [7].

Overall, the predicate can be decided in $\widetilde{\mathcal{O}}_B(d^{12} + d^{10}\tau^2 + d^6\sigma)$, or $\widetilde{\mathcal{O}}_B(N^{12})$, where $N = \max\{d, \sigma, \tau\}$. For conics, $d = 2$, so the bound depends only on τ and becomes $\widetilde{\mathcal{O}}_B(N^2)$. This can be improved, using specialized algorithms for small-degree algebraic numbers [11], to $\widetilde{\mathcal{O}}_B(N)$.

We conclude this section with EdgeConflictType. It determines the type of conflict-region (defined in [11]), and is reduced to InCircle and DFB. It also requires the OrderOnBisector sub-predicate, which determines the ordering of the centers of bitangent disks to 2 sites on their oriented bisector. The parametric representation allows for easy handling of tangency points. Therefore, OrderOnBisector becomes trivial, as bisector points are mapped to the footpoints on the boundary. The next section explains how tangency points of a Voronoi circle are represented. We have so far examined all predicates, except for InCircle, needed for the Delaunay graph, but also the convex hull, of smooth convex pseudo-circles. Their bit complexity is summarized as follows.

Lemma 1. *We can decide* SideOfBisector*,* DFB*, and the corresponding primitives in $\widetilde{\mathcal{O}}_B(N^{12})$. In the case of conics, the bound becomes $\widetilde{\mathcal{O}}_B(N)$, where $N = \max\{d, \tau, \sigma\}$.*

4. INCIRCLE

This section introduces a polynomial system for expressing the Voronoi circle, leading to a robust and fast implementation of our main predicate. Recall that the Voronoi circle is centered at a Voronoi vertex whose radius equals the distance between the vertex and the sites closest to it (i.e., the circle is tangent to the sites). A Voronoi disk is a disk defined by a Voronoi circle.

Given sites $\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_s}$ in this order, we denote their associated Voronoi disk by V_{trs} iff their tangency points on the disk are in CCW direction. In this case, V_{trs} is a CCW Voronoi disk, and V_{tsr} is a CW Voronoi disk. Since the Voronoi diagram of smooth convex pseudo-circles is an abstract Voronoi diagram, given 3 sites, there may exist at most one CCW Voronoi disk and at most one CW Voronoi disk. Moreover, these disks may be either external (externally tangent to the sites) or internal (internally tangent).

Let us now adapt the definition of conflict from [18], (see also fig. 6 and 7):

Definition 2. *Given sites $\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_s}$, let V_{trs} be their Voronoi disk and $\mathbf{C_h}$ be a query site. If V_{trs} is an external Voronoi disk, then $\mathbf{C_h}$ is in conflict with V_{trs} iff V_{trs} is intersecting $C_h{}^\circ$. If V_{trs} is an internal Voronoi disk, then $\mathbf{C_h}$ is in conflict with V_{trs} iff V_{trs} is included in $C_h{}^\circ$.*

Given $\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_s}$, InCircle decides if a newly inserted site $\mathbf{C_h}$ is in *conflict* with V_{trs}. A degeneracy arises when $\mathbf{C_h}$ is also tangent to V_{trs}. Given that V_{trs} exists, the predicate is computed as follows: (i) Solve the algebraic system that expresses the Voronoi circle. Among the solutions (which correspond to various tritangent circles, cf. fig. 5 middle), find V_{trs}. (ii) Determine the relative position of $\mathbf{C_h}$ w.r.t. V_{trs}. Each step is explained in the subsections that follow.

4.1 Computing the Voronoi circle

The polynomial system expressing all circles tangent to $\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_s}$ is:

$$\begin{array}{rcl} N_t(x,y,t) = N_r(x,y,r) = N_s(x,y,s) & = & 0 \\ M_{tr}(x,y,t,r) = M_{ts}(x,y,t,s) & = & 0. \end{array} \tag{2}$$

The first 3 equations correspond to normals at points t, r, s on the 3 given sites. All normals go through the Voronoi vertex (x, y). The last two equations force (x, y) to be equidistant from the sites: each one corresponds to the bisector of the segment between two footpoints (cf. fig. 5 left). This system was also used in [21]. Elimination of x, y from M_{tr}, N_t, N_r yields the bisector of two sites w.r.t. t, r.

System (2) shall be solved over $\mathbb{R}$, thus yielding a set of solution vectors in $\mathbb{R}^5$. Only one solution vector corresponds to V_{trs}. There exist solution vectors with CCW orientation of the tangency points, and solution vectors with CW orientation which do not correspond to the Voronoi circle we are looking for. They just correspond to some tritangent circle (cf. fig. 5 middle).

4.1.1 Solving the system

The *resultant* of $n+1$ polynomials in n variables is an irreducible[5] polynomial in the coefficients of the polynomials which vanishes iff the system has a complex solution. In particular, sparse (or toric) resultants express the existence of solutions in $(\mathbb{C}^*)^n$ [6]. We employ vectors $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n$, where $|\alpha|$ is the 1-norm and we write x^α for $\Pi_i x_i^{\alpha_i}$.

Proposition 3. *We are given polynomials $F_0, \ldots, F_n \in \mathbb{K}[x_1, \ldots, x_n]$ over a field $\mathbb{K}$, such that $F_i = \sum_{0 \le |\alpha| \le d_i} u_{i,\alpha} x^\alpha$ is of total degree d_i, where $0 \le i \le n$. Their resultant w.r.t. $x_1, \ldots, x_n$ is homogeneous of degree $d_0 \cdots d_{j-1} d_{j+1} \cdots d_n$ in $u_{j,\alpha}$, where $0 \le |\alpha| \le d_j$, and $j \in [0, n]$. This means, for any $\lambda \in \mathbb{K}$, that $\mathrm{Res}(F_0, \ldots, \lambda F_j, \ldots, F_n) = \lambda^{d_0 \cdots d_{j-1} d_{j+1} \cdots d_n}$ $\mathrm{Res}(F_0, \cdots, F_n)$. The total degree of the resultant is $\sum_{j=0}^n d_0 \cdots d_{j-1} d_{j+1} \cdots d_n$.*

Lemma 4. *Let $f(x) = T^2 a_n x^n + T a_{n-1} x^{n-1} + \sum_{i=0}^{n-2} a_i x^i$ and $g(x) = T^2 b_n x^n + T b_{n-1} x^{n-1} + \sum_{i=0}^{n-2} b_i x^i$. Then the resultant of f and g w.r.t. x is a multiple of T^4.*

It is impossible to compute the resultant of 5 arbitrary polynomials as a determinant, so we apply successive Sylvester determinants, i.e., optimal resultant formulae for $n = 1$. This typically produces extraneous factors but, by exploiting the fact that some polynomials are linear, and that none contains all variables, we shall provide the *complete* factorization of the computed polynomial; we focus on conics for simplicity, but our approach holds for any parametric curve. We denote by $\Pi(t)$ the resultant of (2) when eliminating all variables except t: it is, generally, an irreducible univariate polynomial and vanishes at the values of t that correspond to the complex tritangent circles. Recall that the curves are defined by (1).

[5]Irreducibility occurs for generic coefficients; otherwise, resultants can be factorized.

Theorem 5. *If $\Pi(t)$ is the resultant of (2) as above, then $\mathrm{Res}_{xy}(R_1, R_2, N_t) = \Pi(t) H_t^{40} (G_t H_t' - G_t' H_t)^{36}$, where, $R_1 = \mathrm{Res}_r(M_{tr}, N_r)$, $R_2 = \mathrm{Res}_s(M_{ts}, N_s)$ and the degree of Π is 184.*

Proof. All polynomials belong to $\mathbb{Z}[x, y, t, r, s]$: we shall eliminate x, y, r, s to obtain the univariate resultant in $\mathbb{Z}[t]$. Polynomials N_t, N_r, and N_s are of total degree 5, linear in x, y and of degree 4 in the parameter. Polynomials M_{tr} and M_{ts} are of total degree 9, linear in x and y and of degree 4 in the parameters. First, we eliminate r from M_{tr} and N_r:

$$R_1(x,y,t) = \mathrm{Res}_r(M_{tr}(x,y,t,r), N_r(x,y,r)).$$

It is of total degree 22, of degree 6 in x, in y, and in x, y, and 16 in t. We do the same for M_{ts}, N_s and obtain:

$$R_2(x,y,t) = \mathrm{Res}_s(M_{ts}(x,y,t,s), N_s(x,y,s)),$$

which follows the same degree pattern. It remains to compute the final polynomial in $\mathbb{Z}[t]$:

$$R_3(t) = \mathrm{Res}_{x,y}(R_1(x,y,t), R_2(x,y,t), N_t(x,y,t)).$$

Let us write $N_t = D(t)y + A(t)x + C(t)$, where $D(t) = H_t(G_t H_t' - G_t' H_t)$ of degree 4. To compute R_3, we shall solve N_t for y and substitute in R_1, R_2. This introduces denominators, eliminated by multiplying by $D(t)^6$. Then, we can take the Sylvester resultant w.r.t. x. During these operations we apply prop. 3 as follows:

$$\begin{array}{ll} \mathrm{Res}_{xy}(D^6 R_1, D^6 R_2, y + \frac{Ax+C}{D}) & = \\ D^{36} D^{36} \mathrm{Res}_{xy}(R_1, R_2, y + \frac{Ax+C}{D}) & = \\ = D^{36} \mathrm{Res}_{xy}(R_1, R_2, D(y + \frac{Ax+C}{D})) & = D^{36} R_3. \end{array} \tag{3}$$

From prop. 3, the degree of the resultant w.r.t. t is $16(6 \cdot 1) + 16(6 \cdot 1) + 4(6 \cdot 6) = 336$. The previous discussion suggests that $(H_t(G_t H_t' - G_t' H_t))^{36}$, the leading coefficient of y in N_t, appears as an extra factor.

Moreover, after substitution of y in R_1 and R_2, we obtain polynomials of the form $H_t^2 c_6 x^6 + H_t c_5 x^5 + c_4 x^4 + c_3 x^3 + \ldots + c_0$. From lemma 4, it follows that the resultant of two such polynomials contains H_t^4 as a factor. Therefore, the degree of Π is $336 - 4 \cdot 36 - 2 \cdot 4 = 184$. $\square$

The above theorem provides an upper bound of 184 complex tritangent circles to 3 conics. Numeric examples show that the bound is tight. This bound was also obtained in [11] using a different system. Now, we describe the factorization for arbitrary degree curves based on prop. 3,

Corollary 6. *We are given $R_0, R_1, R_2 \in \mathbb{K}[x, y]$, where the total degree of R_1 and R_2 is n in x, in y, and in x and y together, and $R_0 = Dy + Ax + C$, where $AD \neq 0$, then $\mathrm{Res}_x(\mathrm{Res}_y(R_0, R_1), \mathrm{Res}_y(R_0, R_2)) = D^{n^2} \mathrm{Res}_{xy}(R_0, R_1, R_2)$.*

Corollary 7. *The degree of the resultant of (2) for general parametric curves, as in (1), is bounded by*

$$(3d-2)(5d-2)(9d-2),$$

after dividing out the factor of $(H_t(G_t H_t' - G_t' H_t))^{(5d-2)^2}$.

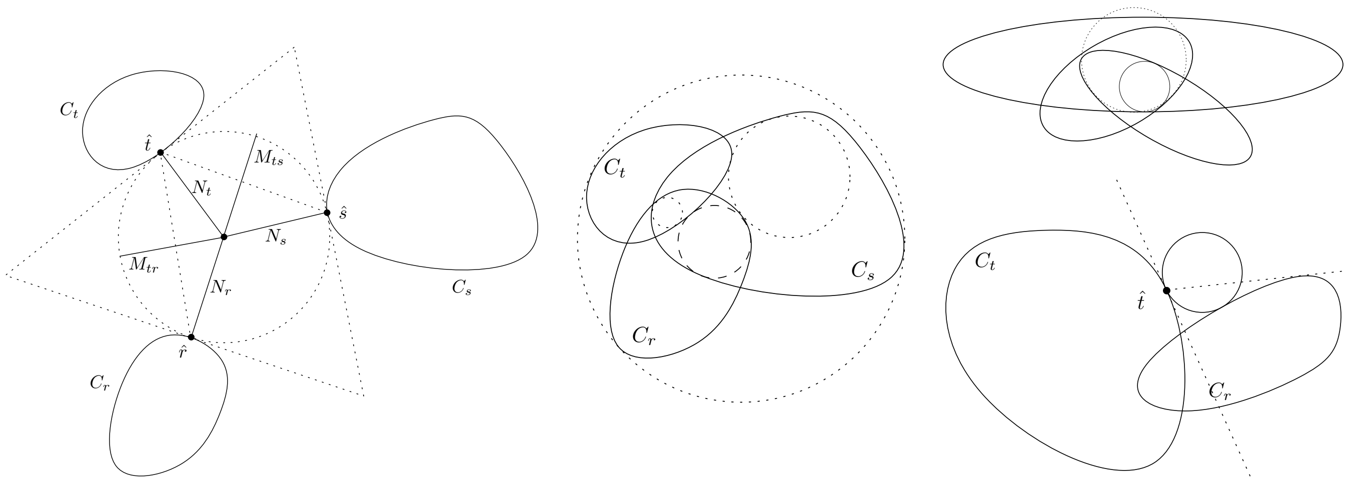

Figure 5: Left: An external tritangent circle; Middle: Various tritangent circles. Dotted line: CCW(t,r,s), Dashed line: CW(t,r,s); Right: top: An internal tritangent circle, bottom: Finding an external bitangent circle

A more careful analysis may exploit term cancellations to yield a tighter bound.

If we solve the resultant of system (2), we obtain one coordinate of the solution vectors (in isolating interval representation). There are methods to obtain the other variables, too. For instance, plugging a value of t in the bisector equation of sites C_t and C_r allows us to find the corresponding value for r (see section 4.2 for how to choose the appropriate r-solution).

Moreover, having obtained the resultant allows us to detect *degenerate* configurations, i.e., all 4 sites being tangent to the same Voronoi disk. Consider the triplets $\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_s}$ and $\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_h}$. Let $\Pi_1(t), \Pi_2(t)$ be the resultants of (2) for these triplets respectively. If the triplets admit an identical Voronoi circle, then $gcd(\Pi_1, \Pi_2) \neq 1$. Conversely, if $gcd(\Pi_1, \Pi_2) \neq 1$, the triplets *may* have an identical solution vector (which can be verified by looking at the other coordinates, in a way analogous to the one presented in [11]).

4.1.2 Choosing the proper solution

In this subsection we consider the question of choosing, among all solutions of the polynomial system corresponding to the tangency points $(\hat{t}, \hat{r}, \hat{s})$ of a tritangent circle, the one that corresponds to Voronoi circle V_{trs}.

First, we eliminate irrelevant solutions. Consider the tangency points $p_{\hat{t}}$, $p_{\hat{r}}$, $p_{\hat{s}}$ for a solution triplet $\hat{t}, \hat{r}, \hat{s}$. The tangency points corresponding to V_{trs} satisfy CCW($p_{\hat{t}}$, $p_{\hat{r}}$, $p_{\hat{s}}$) and we disregard the rest of the solution vectors.

Now we distinguish an external and an internal tritangent circle from the rest of the tritangent circles. The tangency points define the former iff the tangent line of the Voronoi circle at each tangency point separates its adjacent site from the other two tangency points, see fig. 5 left. Even if the tangent line intersects the other sites (not shown in the example), the tangency points are still separated. Checking that the tangency points correspond to an internal circle can be performed by applying lemma 11 (cf. section 4.2.2). Note that in this case, an argument symmetric to the external case (i.e., the internal tritangent circle is such that all three tangency points are on the same side of the tangent line as the site, and inside the site) does not apply, due to the fact that the internal tritangent circle may be “locally” inside the curve, but not “globally” (i.e., the dotted circle of fig.5 top right).

4.2 Deciding conflict

Since the Voronoi circle is expressed algebraically, it is not clear how to decide its relative position w.r.t. the query site, i.e., by primitive RELATIVEPOS. Therefore, we model the disk inclusion test of definition 2 as a circle inclusion test.

4.2.1 Conflict with external Voronoi disk

The following can be obtained from property 5.1 [12], see fig. 6.

LEMMA 8. *Given convex sites $\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_s}$, let V_{trs} be an external Voronoi disk of theirs and $\hat{t}$ its tangency point on C_t. Let $\mathbf{C_h}$ be a query site. If $\hat{t} \in \mathbf{C_h}$, then $\mathbf{C_h}$ is in conflict with V_{trs} (by definition). Otherwise, let B_{th} be an external bitangent disk of $\mathbf{C_t}$ and $\mathbf{C_h}$, tangent at $\hat{t}$ (and $\hat{h}$ resp.). Then $\mathbf{C_h}$ is in conflict with V_{trs} iff B_{th} is strictly contained in V_{trs}.*

The check for inclusion is performed by comparing the corresponding radii, which can be done simply by comparing the coordinates of the center, since the circles share a common tangency point. Note that in case B_{th} does not exist (i.e., $\hat{t}$ does not lie in the interior of the convex hull of $\mathbf{C_t}$ and $\mathbf{C_h}$), $\mathbf{C_h}$ is not in conflict with V_{trs}. Now it remains to compute the external bitangent circle B_{th}. There may exist up to 6 bitangent circles, tangent at a given $\hat{t}$, for the case of ellipses [11], and up to a constant number for arbitrary smooth convex sites. In [11], simple geometric tests are provided to isolate the external bitangent circle among all bitangent circles to non-intersecting ellipses that also apply to general convex sites. For completeness we provide the corresponding lemma, omitting the proof.

Consider two sites $\mathbf{C_t}$ and $\mathbf{C_r}$ and a point $\hat{t}$ on C_t, where $\hat{t} \notin \mathbf{C_r}$ and t lies in the interior of the convex hull of $\mathbf{C_t}$ and $\mathbf{C_r}$. Then there exist two tangent lines to C_r passing through $\hat{t}$. Let r_1, r_2 be the points where they touch C_r.

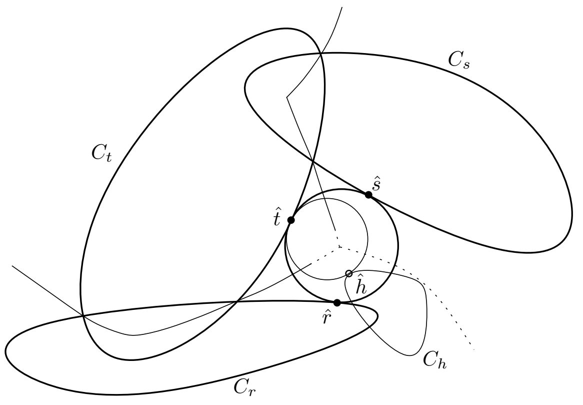

Figure 6: Lem. 8: conflict of query site $\mathbf{C_h}$ with external Voronoi disk

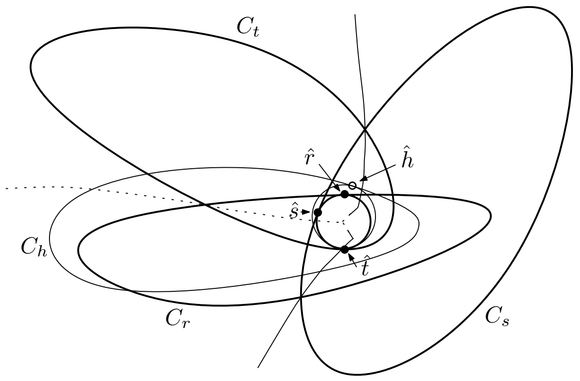

Figure 7: Lem. 10: conflict of $\mathbf{C_h}$ with internal Voronoi disk; the Voronoi edges are solid, the conflict region (edge) is dotted

The chord r_1r_2 separates C_r into two arcs, of which the arc on the same side as $\hat{t}$ called a *visible arc.*

LEMMA 9. *Consider the tangent line ϵ of C_t at point $\hat{t}$. If ϵ does not intersect C_r, then only the visible arc of $\hat{t}$ contains the tangency point of the external bitangent circle. Otherwise, this tangency point is contained in the arc whose endpoints are: the intersection of ϵ with C_r and the endpoint of the visible arc which lies on the opposite side of C_t w.r.t. ϵ.*

Note that lemma 9 also holds when intersecting sites $\mathbf{C_t}$ and $\mathbf{C_r}$ admit an external bitangent circle tangent at point $\hat{t}$ of C_t (cf. fig. 5 bottom right). This is because (i) $\hat{t} \notin C_t \cap C_r$ since it is the tangency point of an externally tangent circle, (ii) $\hat{t}$ lies on the arc of C_t bounded by the convex hull of $\mathbf{C_t}$ and $\mathbf{C_r}$ (iii) the tangent line of C_t at $\hat{t}$ intersects C_r.

Overall, it is always possible to obtain an arc on C_r that contains the tangency point of the *externally* tangent circle, and answer the predicate by applying lemma 8.

Lastly, let us note that with some configurations, it is possible to determine the conflict immediately by looking at the ordering of the tangency points of the Voronoi circles of C_t, C_r, C_s and C_t, C_r, C_h. This is due to the fact that the radii of the external bitangent circles, as one external bitangent line is deforming to the other, are monotonically decreasing and increasing [11, Lemma 6].

4.2.2 Conflict with internal Voronoi disk

Intersecting sites may also admit an internally tritangent Voronoi circle. In this case InCircle can be answered by the following lemma:

LEMMA 10. *Given sites $\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_s}$, let V_{trs} be their internal Voronoi disk, and $\hat{t}$ its tangency point on C_t. Let $\mathbf{C_h}$ be a query site. If $\hat{t} \notin \mathbf{C_h}$, then $\mathbf{C_h}$ is not in conflict with V_{trs} (by definition). Otherwise, let B_{th} be an internal bitangent disk of $\mathbf{C_t}$ and $\mathbf{C_h}$, tangent at $\hat{t}$ (resp. $\hat{h}$). Then $\mathbf{C_h}$ is in conflict with V_{trs} iff V_{trs} is strictly contained in B_{th}.*

PROOF. See fig. 7. ($\Leftarrow$). If V_{trs} is strictly contained in B_{th}, then it lies in the interior of B_{th}, except point $\hat{t}$. This means that $V_{trs} \in {C_h}^\circ$, as B_{th} and V_{trs} share the common tangency point $\hat{t}$ and both circles are internally tangent to C_t. Thus, according to def. 2, V_{trs} is in conflict with $\mathbf{C_h}$.
($\Rightarrow$). If $\mathbf{C_h}$ is in conflict with V_{trs}, then $V_{trs} \in {C_h}^\circ$. Circles B_{th} and V_{trs} have in common the tangency point $\hat{t}$ and since B_{th} touches C_h (at $\hat{h}$), V_{trs} is strictly contained in B_{th}. □

Let us now identify an internal bitangent circle (among all bitangent circles). First, we observe that if an internal bitangent circle at point $\hat{t}$ of C_t exists, then $\hat{t} \in \mathbf{C_t} \cap \mathbf{C_r}$ and $\hat{r} \in \mathbf{C_t} \cap \mathbf{C_r}$ (the converse may not be necessarily true).

LEMMA 11. *Given intersecting sites $\mathbf{C_t}$ and $\mathbf{C_r}$, consider their bitangent circle B_{tr} at points $\hat{t}$ and $\hat{r}$ respectively with $\hat{t}, \hat{r} \in \mathbf{C_t} \cap \mathbf{C_r}$. Then B_{tr} is an internal bitangent circle iff B_{tr} has the smallest radius among all bitangent circles of C_t and C_r tangent at $\hat{t}$, and the radius of B_{tr} is bounded by the radius of curvature of C_t at $\hat{t}$ and the radius of the self-bitangent circle of C_t at $\hat{t}$.*

PROOF. ($\Leftarrow$). Let B_{tr} be an internal bitangent circle which means that $B_{tr} \in \mathbf{C_t}$. Then the radius of B_{tr} is smaller than the radius of curvature of C_t at $\hat{t}$, since the circle is inside C_t around $\hat{t}$ [5]. Moreover, since B_{tr} does not cross C_t, its radius is no greater than that of the self-bitangent circle of C_t at $\hat{t}$ (that's the distance from $\hat{t}$ to its corresponding point on the medial axis of C_t). Now assume that there exists another circle B'_{tr} internal to C_t and tangent at $\hat{t}$ with radius smaller than that of B_{tr}. Then B'_{tr} does not have any common points with C_r since B_{tr} is internally tangent to C_r. Therefore B_{tr} has the smallest radius among all bitangent circles of C_t and C_r tangent at $\hat{t}$.
($\Rightarrow$). Let B_{tr} have the smallest radius among all bitangent circles tangent at $\hat{t}$, not greater than the radius of curvature of C_t at $\hat{t}$, neither than the radius of the self-bitangent circle of C_t at $\hat{t}$. Then locally (around $\hat{t}$) B_{tr} is inside C_t. Also, (globally) B_{tr} does not cross C_t, thus $B_{tr} \in \mathbf{C_t}$. Since $\hat{t} \in \mathbf{C_t} \cap \mathbf{C_r}$ it follows that B_{tr} is internally tangent to C_r. □

The above lemma completes lemma 10 when internal bitangent circle B_{th} does not exist. In that case the corresponding circle with radius equal to the radius of curvature or that of the self-bitangent circle is considered instead of B_{th}. In an implementation, the curvature constraint can be forced by comparing with the evolute point [14]. The self-bitangent circles can be computed by considering the bisector of a curve and itself, then by computing the (self-)bitangent circles and finally choosing the one with the smallest radius (in case of identical self-bitangent circle we can consider the evolute point).

4.3 Existence

In some cases, it is required to know whether the CCW Voronoi circle V_{trs} exists or not. This is a generalization of EXISTENCE of [9] to pseudo-circles.

There is a straightforward way to determine if V_{trs} exists. Solve the algebraic system and look for V_{trs}. If it is not found, then we may conclude that V_{trs} does not exist. However, in some cases one can check the existence of V_{trs} in an easier way, avoiding solving the system. Moreover, we shall be able to determine its type (i.e., external or internal).

If one site is contained in another one (hidden), V_{trs} cannot exist. Thus, we assume that there are no hidden sites.

Observe that DFB is a special case of INCIRCLE, where the CCW bitangent line is an infinite CCW Voronoi circle. Geometrically, a conflict of this type means that the insertion of $\mathbf{C_s}$ into the Voronoi diagram will invalidate part of the bisector of $\mathbf{C_t}$ and $\mathbf{C_r}$. In this case, the predicate evaluates to negative, i.e., $\mathrm{DFB}(\mathbf{C_t}, \mathbf{C_r}, \mathbf{C_s}) < 0$, otherwise $\mathrm{DFB}(\mathbf{C_t}, \mathbf{C_r}, \mathbf{C_s}) \geq 0$.

LEMMA 12. *Given sites $\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_s}$, let κ be the number of conflicts of* DFB*, when evaluated at triplets ($\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_s}$), ($\mathbf{C_r}$, $\mathbf{C_s}$, $\mathbf{C_t}$) and ($\mathbf{C_s}$, $\mathbf{C_t}$, $\mathbf{C_r}$). Then (i) If $\kappa < 2$ then V_{trs} does not exist. (ii) If $\kappa \geq 2$ then: (a) If $\mathbf{C_t} \cap \mathbf{C_r} \cap \mathbf{C_s} \neq \emptyset$ and there exists a $[t, r, s]$ sequence of arcs on its boundary (i.e., a CCW sequence of arcs respectively belonging to $\mathbf{C_t}$, $\mathbf{C_r}$ and $\mathbf{C_s}$), then either V_{trs} is internal or it does not exist. (b) Otherwise, V_{trs} exists and is external.*

PROOF. (i) Case $\kappa < 2$. Without loss of generality we assume that $\mathrm{DFB}(\mathbf{C_t}, \mathbf{C_r}, \mathbf{C_s}) \geq 0$ and $\mathrm{DFB}(\mathbf{C_s}, \mathbf{C_t}, \mathbf{C_r}) \geq 0$. If $\mathbf{C_t} \cap \mathbf{C_s} = \emptyset$ (cf. fig. 8 left), then $\mathbf{C_t}$,$\mathbf{C_r}$,$\mathbf{C_s}$ do not admit an external tritangent circle touching them in this order. Since they have no common intersection, an internal tritangent circle does not exist either. Therefore V_{trs} does not exist. If $\mathbf{C_t} \cap \mathbf{C_s} \neq \emptyset$ (fig. 8 top right), then, given all possible positions of $\mathbf{C_s}$, there cannot be an intersection of all three sites with a $[t, r, s]$ sequence of arcs on its boundary, since $\mathbf{C_s}$ would have to not intersect the CW external bitangent of $\mathbf{C_t}$ and $\mathbf{C_r}$ when it is not possible, or it would have to intersect some site more than twice. Therefore, an internal V_{trs} does not exist. With similar arguments we show that neither does an external V_{trs} exist.

(ii) Case $\kappa \geq 2$. Without loss of generality we may assume that $\mathrm{DFB}(\mathbf{C_t}, \mathbf{C_r}, \mathbf{C_s}) < 0$ and $\mathrm{DFB}(\mathbf{C_s}, \mathbf{C_t}, \mathbf{C_r}) < 0$. (a) Case $\mathbf{C_t} \cap \mathbf{C_r} \cap \mathbf{C_s} \neq \emptyset$ and there *exists* a $[t, r, s]$ sequence of arcs. V_{trs} cannot be external, since in this case it would either be $\mathrm{DFB}(\mathbf{C_r}, \mathbf{C_t}, \mathbf{C_s}) < 0$ (i.e., $\mathbf{C_s}$ intersecting the CW bitangent line of $\mathbf{C_t}$ and $\mathbf{C_r}$), or $\mathbf{C_s}$ would have to intersect some site more than twice. Therefore, if V_{trs} exists it should be internal. In fact, an internal tritangent circle does not exist iff each (internal) maximal disk of of $\mathbf{C_s}$ (tangent at some point of the s-arc) is contained in $C_t{}^\circ \cup C_r{}^\circ$ (this is related to the *medial axis location* operation described in [18]). (b) If $\mathbf{C_t} \cap \mathbf{C_r} \cap \mathbf{C_s} = \emptyset$ then V_{trs} cannot be internal. It follows (cf. fig. 8 left) that V_{trs} exists and is external. Finally, if the intersection is nonempty and a $[t, r, s]$ sequence of arcs does not exist, then with arguments symmetric to case (ii,a) we are left only with the case that V_{trs} exists and it can only be external (cf. fig. 8 bottom right). ☐

5. IMPLEMENTATION & EXPERIMENTS

This section describes our efficient and exact implementation for non-intersecting ellipses in the plane (fig. 1 left), which is now being extended to handle pseudo-circles (fig. 2 right), or sites fully contained in other ones (fig. 1 right), based on the aforementioned algorithms. Our code is based on the existing CGAL Apollonius package for the combinatorial part of the algorithm. Since CGAL follows the generic programming paradigm, the main issue was to implement the predicates for ellipses, generalizing the circular sites developed for the Apollonius diagram.

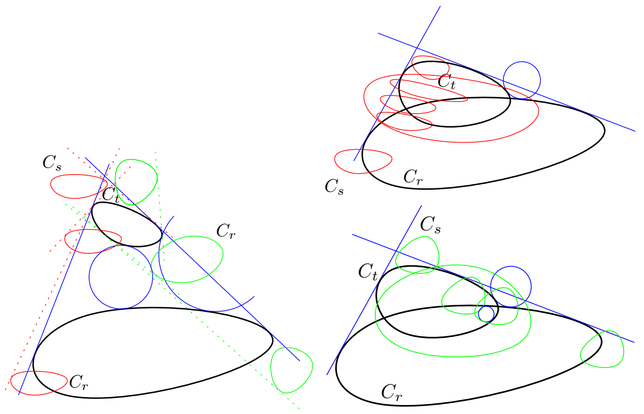

Figure 8: $\mathbf{C_t} \cap \mathbf{C_r} = \emptyset$ (left), or $\mathbf{C_t} \cap \mathbf{C_r} \neq \emptyset$ (right). Red sites: $\kappa < 2$, Green sites: $\kappa \geq 2$

For the required algebraic operations, we relied on SYNAPS[6], an algebraic library which features state-of-the-art implementations for real solving, used to solve the degree-184 univariate polynomials.

We have implemented polynomial interpolation to compute the resultant of system (2). To speed up the implementation, we use NTL[7] which is an open source C++ library providing asymptotically fast algorithms for polynomial GCD and Sylvester resultants. We apply thm. 5 to eliminate the extraneous factors at each evaluation, hence obtaining the optimal-degree polynomial at the reconstruction phase with the optimal number of 185 evaluations.

For INCIRCLE, we have incorporated the subdivision method of [12], using interval arithmetic provided by SYNAPS and multi-precision floating point numbers by MPFR[8]; thus we no longer rely on ALIAS. This is a filter, which answers INCIRCLE before full precision is achieved at non-degenerate cases, especially when looking for the external Voronoi circle; however, its convergence for arbitrary intersecting smooth curves may no longer be quadratic. When the MPFR precision is not enough, we fall back to the exact algebraic method. At the heart of the subdivision method we have implemented a univariate interval Newton solver, which handles polynomials with interval coefficients, thus allowing us to quickly solve multivariate equations by plugging in interval approximations of each variable.

The subdivision algorithm can approximate the tangency points that correspond to the Voronoi circle with arbitrarily high precision, hence it is used as a filter. The use of resultant allows us to determine if there exists a degenerate Voronoi circle, tangent to 4 sites, since it can provide us with the true separation bound (for more details see [12]). This holds in all computations in the sense that we perform "diffi-

[6]http://www-sop.inria.fr/galaad/synaps/
[7]http://www.shoup.net/ntl/
[8]http://www.mpfr.org/

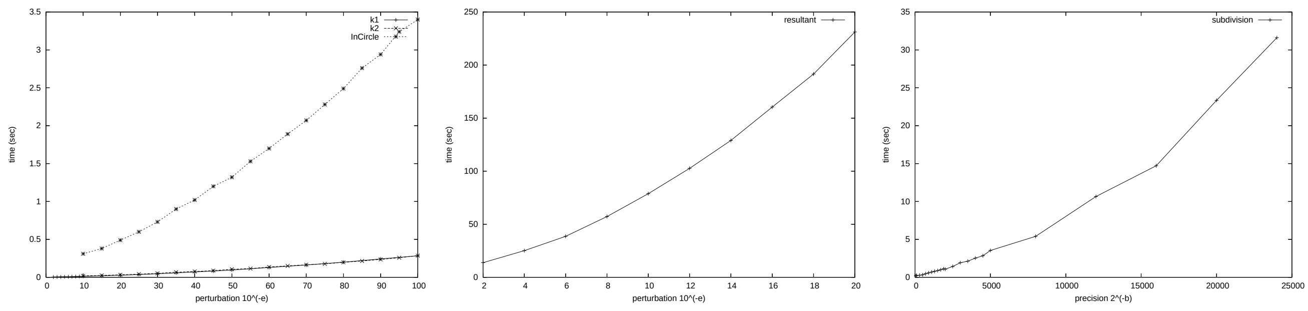

Figure 9: Left: Predicates; Middle: Resultant computation; Right: Subdivision

cult" algebraic operations using interval arithmetic. During an algebraic computation with interval arithmetic, degeneracies or near-degeneracies lead to uncertain sign evaluations. Then, the resultant provides us with the necessary information in order to decide (increasing our precision by a constant number of bits, but significantly less than having to go up to the theoretical separation bound).

Lastly, we implemented a visualization algorithm for the bisector of two ellipses. Since the implicit equation has a total degree of 28 in Cartesian space, we trace the equivalent implicit curve in parametric space, which is of degree 12; however, this routine is subject to further optimization.

Overall, our software design, is generic so as to distinguish the geometry from the algebra part. This is important when connecting to the algebraic libraries, as described above, but also in order to use alternative libraries. In fact, our code is able to use any algebraic library as long as certain interface requirements are satisfied. Currently, we are also using CGAL's Algebraic Kernel which provides univariate real solving, multivariate polynomial handling and resultant computation via interpolation.

Now, we present various experimental results. All runtimes are obtained on a Pentium-4 2.6 GHz machine with 1.5GB of RAM, unless otherwise specified.

We have measured the performance of SideOfBisector, DFB and InCircle with varying bitsize. Left fig. 9 corresponds to ellipses with randomly perturbed parameters (axes, rotation and center of ellipses) by adding / subtracting 10^{-e}, with varying e, to small (10-bit) random input parameters; this forces the polynomials computed during each predicate evaluation to have coefficients of large bitsize, since the coefficients depend on the input parameters (axes, rotation, center). All runtimes appear to grow subquadratically in e, which is expected since SideOfBisector, DFB have constant arithmetic complexity and InCircle is handled by the subdivision algorithm with quadratic convergence, hence computes τ bits in $\mathcal{O}(\log(\tau))$ operations. In case of degeneracies, the runtime of InCircle is dominated by the resultant computation, shown in middle fig. 9. The sub-quadratic behavior of the first two predicates agrees with the theoretical complexity bound derived in lemma 1.

Finally, we measured the time needed for the subdivision algorithm to reach a precision of 2^{-b}, using MPFR floats, in right fig. 9. This (multi-precision) version currently lacks some optimizations making it about 2 times slower than the one using ALIAS, when standard floating point precision (53-bit) is employed. This precision is achieved in about 0.2 sec, and 1 sec suffices for almost 2000 bits of precision, whereas the 24k-bit approximation needs about 0.5 min. This shows that the theoretical separation bound of several million bits [11] cannot be achieved efficiently, hence the usefulness of resultant-based methods. On the other hand, resultants, even with 10-bit input coefficients can be about 70 times slower than the subdivision algorithm using the standard floating point precision of 2^{-53}. In short, both methods have to be combined for a robust and fast solution.

The overall time for the construction of the Delaunay graph (and the structure representing its dual diagram) is shown in right fig. 10 (solid line). It takes, for instance, 98 sec to compute the exact Delaunay graph of 128 non-intersecting ellipses. More importantly, it is about linear in the number of sites for up to this number of non-intersecting ellipses.

5.1 Comparing with point approximations

An alternative way to solve problems with curved sites is to approximate them by simpler objects such as polygons or even sets of points. However, a good approximation may require a large number of input sites.

Each ellipse is approximated by a constant number of k points taken uniformly on its boundary (just like the vertices of the polygons in fig. 11, right). These points have rational coordinates, as they are obtained using (1).

Using GNU rational arithmetic (Gmpq in CGAL), we compare against 3 variations of the incremental algorithm of CGAL for the Delaunay triangulation: (i) without filtering,[9] (ii) with filtering, (iii) with filtering and improved nearest neighbor location. Points are inserted in CCW order for each ellipse, so the Delaunay face of the lastly inserted point is given as a hint for the insertion routine.

Our own implementation for ellipses uses Gmpq but no filtering, except for InCircle that uses a subdivision-based method. However, the latter also uses some slower exact computations which could be accelerated further.

Left fig. 10 presents results concerning 32 ellipses, with k varying from 8 to 320. We see that the Delaunay graph computation of 32 ellipses is faster for variations (i),(ii),(iii) of the Delaunay triangulation of points for $k \geq 120$, $k \geq 160$ and $k \geq 240$ respectively. The corresponding Delaunay triangulations have 3840, 5120 and 7680 vertices respectively. There are Delaunay edges corresponding to pairs of points

[9]Filtering is a technique where the predicates are answered using double arithmetic, falling back to slower exact arithmetic only when the filter fails to produce an answer. This implies that there is some mechanism of determining that the outcome of the filter is correct or not.

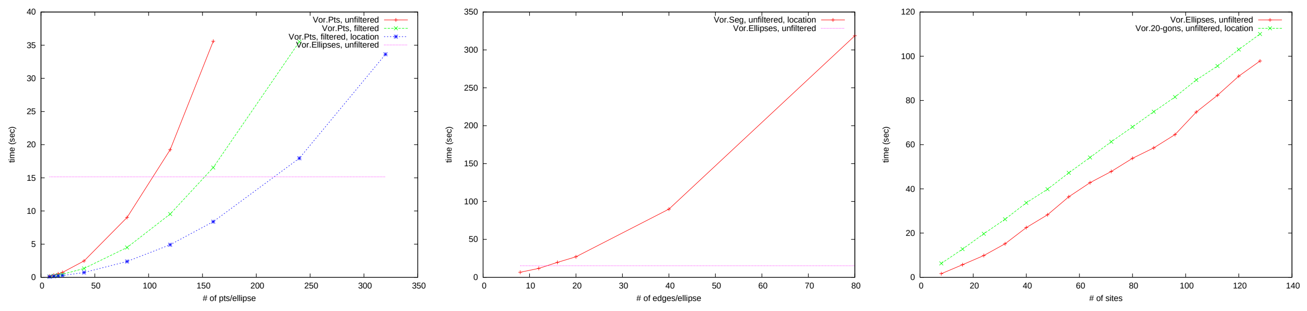

Figure 10: Delaunay graph of: 32 ellipses vs point approximations with increasing number of points per ellipse (left), 32 polygons with increasing number of edges (middle), polygons and ellipses (right)

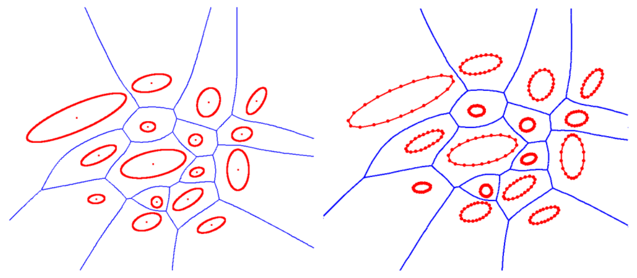

Figure 11: Left: Voronoi diagram of 16 ellipses. Right: Voronoi diagram of 16 20-gons approximating each ellipse (320 segments in total)

on the same ellipse (with dual Voronoi edges in the ellipses' interior). These should be discarded which induces an extra overhead not measured. This is also true for all but one Delaunay edge between neighboring ellipses.

5.2 Comparing with polygonal approximations

We compare against the CGAL package for the segment Delaunay graph (and the dual Voronoi diagram) [17], which is followed by the removal of edges between consecutive segments on the same ellipse.

We replaced each ellipse (fig.11 left) by a 20-gon (fig.11 right). Right fig. 10 shows the time for the incremental construction of the Voronoi diagram of polygons by CGAL (dashed line) compared to that of ellipses (solid line) when the number of sites varies from 4 to 128. The cost of an insertion is roughly $\mathcal{O}(\log^2 k)$ [17, 18], where k is the number of already inserted sites. Care has been taken to perform smartly the nearest neighbor queries. In particular, since each segment is added in CCW order around each ellipse, a probable nearest neighbor is the lastly inserted segment. This is given as a hint to the insertion routine. The segment Delaunay implementation measured uses Gmpq arithmetic and no filtering. We didn't count the time required for the deletion of artificial edges. Interestingly, the Delaunay graph of polygons is slower with > 15 segments per ellipse.

Middle fig. 10 shows the required time to construct the Delaunay graph of 32 ellipses (solid line) and that of 32 polygons approximating each ellipse with a varying number of edges (dotted line). As the number of edges per ellipse increases, the squared-logarithmic cost per insertion becomes non-negligible. The Delaunay graph of polygons is faster only for approximations of < 16 edges per ellipse.

5.3 Restriction to circles

We performed experiments on circles per predicate, juxtaposing our software and the CGAL package for the Delaunay graph, equivalently the Apollonius (Voronoi) diagram, of circles, on a 1.83 GHz Core2 Duo with 1GB RAM. The inputs were: (i) degenerate: instances with a moving query circle; (ii) near degenerate: like (i), but the the input is perturbed randomly by $\pm 10^{-e}$, $e \in \{2, 4, 6\}$; (iii) random: centers uniformly distributed in a predefined interval.

Runtime increases almost linearly with bitsize for all predicates. This is more evident in (ii), because both approaches rely on algorithms of constant arithmetic complexity, which do not depend on how close to degeneracy the configuration lies. Both implementations behave almost identically on dataset (i) and (ii) $e = 2$, since bitsize is the same, and neither approach depends on separation bounds.

It seems that our implementation is up to 2 orders of magnitude slower that the dedicated one, when we restrict to circles. The difference of performance is not surprising, since the case of circles reduces to computations with real algebraic numbers of degree 2. The best performance occurs for DFB, because both approaches follow an algorithm with the same algebraic complexity. The worst performance is observed, as expected, for INCIRCLE, which, in the case of circles, has been optimized.

5.4 Experiments with general parametric curves

While our C++ implementation covers only ellipses for now, we have applied the proposed approach for the resultant computation on various types of curves using MAPLE. Some preliminary results are summarized in table 1. The first column shows the type of curve, the second its degree, the third the time in sec, the fourth the degree of the resultant and the last column shows the (non-tight) bound of our general formula from corollary 7.

First, we took the *bean* curve of left fig. 2 and applied simple affine transformations yielding very small (5-bit) coefficients. We computed the resultant of such triplets. The long runtimes indicate that working with high-degree curves requires very efficient implementations in order to be practical. We additionally considered the case of 3 random *conics* with small (10-bit) coefficients. In this case where $d = 2$, we have a tight bound of 184, while the general formula yields 512. We also tested the resultant computation on polynomial branches of degree 2 and 3 (*B-splines*), with very small (5-bit) coefficients. The runtime is less than 1 sec for $d = 2$ and less than 10 sec when $d = 3$. These experiments indicate

Curves	d	**time**	**resultant**	**bound**
Beans	4	530.00	2632	6120
Conics	2	7.90	184	512
B-splines	3	9.20	404	2275
B-splines	2	0.65	93	512

Table 1: Resultant degree for various curves

that an efficient exact implementation may still be possible for larger bitsizes, and we can benefit from the increased flexibility that piecewise functions offer (with a trade-off between the total number of non-linear objects and the complexity of the algebraic operations).

6. FUTURE WORK

We can adjust our algorithms for the predicates so as to compute the Delaunay graph (or the convex hull) of convex sites with richer parametric representation, such as piecewise smooth parametric curves (NURBS). The main issue is to identify efficiently the curve, or piece, which matters, and to apply the current predicates. This can be achieved fast by numerical certified methods, such as our subdivision algorithm for the case of InCircle.

7. ACKNOWLEDGMENTS

I.E. is partially supported by Marie-Curie Network "SAGA", FP7 contract PITN-GA-2008-214584 and 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). E.T. is partially supported by contract ANR-06-BLAN-0074 "Decotes". G.T. is partially supported by the State Scholarship Foundation of Greece, Grant No. 4631.

8. REFERENCES

[1] O. Aichholzer, W. Aigner, F. Aurenhammer, T. Hackl, B. Jüttler, and M. Rabl. Medial axis computation for planar free-form shapes. *Comp. Aided Design*, (to appear), 2009.

[2] H. Alt, O. Cheong, and A. Vigneron. The Voronoi diagram of curved objects. *Discr. and Comput. Geometry*, 34(3):439–453, 2005.

[3] F. Anton. *Voronoi diagrams of semi-algebraic sets.* PhD thesis, The University of British Columbia, January 2004.

[4] CGAL: Computational geometry algorithms library. http://www.cgal.org.

[5] J. J. Chou. Voronoi diagrams for planar shapes. *IEEE Comput. Graph. Appl.*, 15(2):52–59, 1995.

[6] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry.* Number 185 in GTM. Springer, New York, 2nd edition, 2005.

[7] D. I. Diochnos, I. Z. Emiris, and E. P. Tsigaridas. On the asymptotic and practical complexity of solving bivariate systems over the reals. *J. Symbolic Computation*, (to appear), 2009.

[8] G. Elber, M.-S. Kim, and H.-S. Heo. The convex hull of rational plane curves. *Graph. Models*, 63(3):151–162, 2001.

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

[10] I. Z. Emiris, B. Mourrain, and E. P. Tsigaridas. Real algebraic numbers: complexity analysis and experimentation. In P. Hertling, C. Hoffmann, W. Luther, and N. Revol, editors, *Reliable Implementations of Real Number Algorithms: Theory and Practice*, volume 5045 of *LNCS*, pages 57–82. Springer Verlag, 2008.

[11] I. Z. Emiris, E. P. Tsigaridas, and G. M. Tzoumas. Predicates for the exact Voronoi diagram of ellipses under the euclidean metric. *Intern. J. Computational Geometry & Applications*, 18(6):567–597, 2008. Spec. Issue on SoCG'06.

[12] I. Z. Emiris and G. M. Tzoumas. Exact and efficient evaluation of the InCircle predicate for parametric ellipses and smooth convex objects. *Comput. Aided Des.*, 40(6):691–700, 2008.

[13] L. Habert. Computing bitangents for ellipses. In *Proc. 17th Canad. Conf. Comp. Geom.*, pages 294–297, 2005.

[14] I. Hanniel, R. Muthuganapathy, G. Elber, and M.-S. Kim. Precise Voronoi cell extraction of free-form rational planar closed curves. In *Proc. ACM Symp. Solid Phys. Modeling*, pages 51–59, Cambridge, MA, 2005.

[15] M. Held. Vroni: An engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. *Comput. Geom. Theory Appl.*, 18:95–123, 2001.

[16] M. Held and S. Huber. Topology-oriented incremental computation of Voronoi diagrams of circular arcs and straight line-segments. *Comp. Aided Design*, (to appear), 2009. Spec. issue on Voronoi diagrams.

[17] M. I. Karavelas. A robust and efficient implementation for the segment Voronoi diagram. In *Proc. Int. Symp. Voronoi Diagrams*, pages 51–62, 2004.

[18] 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.

[19] D.-S. Kim, D. Kim, and K. Sugihara. Voronoi diagram of a circle set from Voronoi diagram of a point set: II. Geometry. *Comp. Aid. Geom. Des.*, 18(6):563–585, 2001.

[20] R. Klein, K. Mehlhorn, and S. Meiser. Randomised incremental construction of abstract Voronoi diagrams. *Comput. Geom.: Theory & Appl.*, 3(3):157–184, 1993.

[21] R. Ramamurthy and R. Farouki. Voronoi diagram and medial axis algorithm for planar domains with curved boundaries - II: detailed algorithm description. *J. Comput. Appl. Math.*, 102(2):253–277, 1999.

[22] R. Ramamurthy and R. Farouki. Voronoi diagram and medial axis algorithm for planar domains with curved boundaries I. theoretical foundations. *J. Comput. Appl. Math.*, 102(1):119–141, 1999.

[23] M. Ramanathan and B. Gurumoorthy. Constructing medial axis transform of planar domains with curved boundaries. *Comp.-Aided Design*, 35(7):619–632, 2003.

[24] J.-K. Seong, E. Cohen, and G. Elber. Voronoi diagram computations for planar nurbs curves. In *Proc. ACM Symp. Solid & Phys. modeling*, pages 67–77, NY, 2008.