Constructing Two-Dimensional Voronoi Diagrams via Divide-and-Conquer of Envelopes in Space*

Ophir Setter Micha Sharir Dan Halperin
School of Computer Science
Tel-Aviv University, Tel-Aviv, Israel
{ophirset,michas,danha}@post.tau.ac.il

Abstract

We present a general framework for computing two-dimensional Voronoi diagrams of different classes of sites under various distance functions. Most diagrams mentioned in the paper are in the plane. However, the framework is sufficiently general to support diagrams embedded on a family of two-dimensional parametric surfaces in $\mathbb{R}^3$. The computation of the diagrams is carried out through the construction of envelopes of surfaces in 3-space provided by CGAL (the Computational Geometry Algorithm Library). The construction of the envelopes follows a divide-and-conquer approach. A straightforward application of the divide-and-conquer approach for Voronoi diagrams yields algorithms that are inefficient in the worst case. We prove that through randomization, the expected running time becomes near-optimal in the worst case. We also show how to apply the new framework and other existing tools from CGAL to compute minimum-width annuli of sets of disks, which requires the computation of two Voronoi diagrams of two different types, and of the overlay of the two diagrams. We do not assume general position. Namely, we handle degenerate input, and produce exact results. Additional material is available at: http://acg.cs.tau.ac.il/projects/internal-projects/vd-via-dc-of-envelopes/project-page

1 Introduction

Voronoi diagrams were thoroughly investigated, and were used to solve many geometric problems, since introduced by Shamos and Hoey to the field of computer science [35] (although their origin dates back centuries ago — see [31]). The concept of Voronoi diagrams was extended to various kinds of geometric sites, ambient spaces, and distance functions. Among those are power diagrams of points in the plane, multiplicatively weighted Voronoi diagrams, additively weighted Voronoi diagrams (also known as Apollonius diagrams), Voronoi diagrams of line segments, and many others [3, 7, 31]. Different types of Voronoi diagrams have been unified under a generalized framework [23, 24].

Numerous approaches for computing Voronoi diagrams were developed: the divide-and-conquer algorithm by Shamos and Hoey [35], the sweep-line algorithm by Fortune [17], randomized incremental constructions [9, 24], a dynamic algorithm for planar convex objects [21], and more. Available exact implementations of Voronoi algorithms include the computation of Delaunay graphs dual to standard Voronoi diagrams, Apollonius diagrams, and segment Voronoi diagrams in CGAL [6, 13, 22], the Voronoi diagrams

*Work on this paper has been supported in part by the Hermann Minkowski–Minerva Center for Geometry at Tel-Aviv University. Work by Dan Halperin and Ophir Setter has also been supported in part by the Israel Science Foundation (Grant no. 236/06), and by the German-Israeli Foundation (grant no. 969/07). Work by Micha Sharir was also partially supported by NSF Grant CCF-05-14079, by a grant from the U.S.-Israeli Binational Science Foundation, by grant 155/05 from the Israel Science Fund, Israeli Academy of Sciences, and by a grant from the French-Israeli AFIRST program.

Table 1: Some of the types of Voronoi diagrams currently supported by our implementation, and their bisector classes.

Name	Sites	Distance function	Class of bisectors
Standard Voronoi diagram	points p_i	$\|\|x-p_i\|\|$	lines
Power diagram	circles (with center p_i and radius r_i)	$\sqrt{(x-p_i)^2-r_i^2}$	
Apollonius diagram	circles (with center p_i and radius r_i)	$\|\|x-p_i\|\|-r_i$	hyperbolic arcs
Möbius diagram	points p_i with scalars λ_i, μ_i	$\lambda_i(x-p_i)^2-\mu_i$	circles and lines
Anisotropic diagram	points p_i, with positive definite matrices M_i, and scalars π_i	$(x-p_i)^t M_i(x-p_i)-\pi_i$	conic arcs
Voronoi diagram of linear objects	interior-disjoint points, segments, rays, and lines	Euclidean distance	piecewise algebraic curves composed of line segments and parabolic arcs
Spherical Voronoi diagram	points on a sphere	geodesic distance	arcs of great circles (geodesic arcs)
Spherical power diagram	circles on a sphere	"spherical" power distance[1]	

of ellipses in CGAL [14], the construction of segment Voronoi diagrams in LEDA [8], and the implementation of the randomized algorithm for constructing abstract Voronoi diagrams in LEDA [34]. Approximated alternatives include the VRONI code for computing Voronoi diagrams of points, segments, and arcs in 2D [19], the use of the GPU (Graphics Processing Unit) to discretely compute Voronoi diagrams [20, 28], and others. Typically, the time complexity of constructing a Voronoi diagram that has linear complexity, using the above algorithms, is nearly linear.

Edelsbrunner and Seidel [11] established the connection between Voronoi diagrams in $\mathbb{R}^d$ and lower envelopes of the corresponding distance functions to the sites in $\mathbb{R}^{d+1}$ (see also Section 2), yielding a very general approach for computing Voronoi diagrams.

The Computational Geometry Algorithms Library (CGAL)[2] contains a robust and efficient implementation of a divide-and-conquer algorithm for constructing envelopes of surfaces in three dimensions [25]. The theoretical worst-case time complexity of constructing the envelope of n "well-behaved" surfaces in three dimensions using the algorithm is[3] $O(n^{2+\varepsilon})$ [2] (this is also an upper bound, almost tight in the worst case, on the combinatorial complexity of the envelope). As observed below, this near-quadratic running time can arise also in the case of envelopes that represent Voronoi diagrams of linear complexity. This fact poses an obstacle when attempting to utilize this algorithm for computing Voronoi diagrams that have linear complexity, as we aim for algorithms that run in near-linear time. We use randomization in the divide step to establish the theoretical aspect of our work and eliminate the above complexity obstacle, yielding an asymptotically efficient algorithm; see Section 3.

We present a general framework for computing various two-dimensional Voronoi diagrams, exploiting the efficient, robust, and general-purpose code of CGAL. Section 4 explains at a high level how to produce new types of Voronoi diagrams, and shows various examples of instantiations of our framework for computing numerous types of Voronoi diagrams together with other experimental results. More technical details on the software interface are provided in Appendix B. The framework is sufficiently general to support diagrams embedded on certain two-dimensional orientable parametric surfaces in $\mathbb{R}^3$, as the diagrams are

[1] Given a spherical point P and a spherical circle $C(Q,r)$, the spherical power "proximity" between P and C is defined to be $\frac{\cos d(P,Q)}{\cos r}$ where $d(P,Q)$ is the geodesic distance between P and Q [37].

[2] `http://www.cgal.org`

[3] A bound of the form $O(f(n)\cdot n^{\varepsilon})$ means that the actual upper bound is $C_\varepsilon f(n)\cdot n^{\varepsilon}$, for any $\varepsilon > 0$, where C_ε is a constant that depends on ε, and generally approaches to infinity as ε goes to 0.

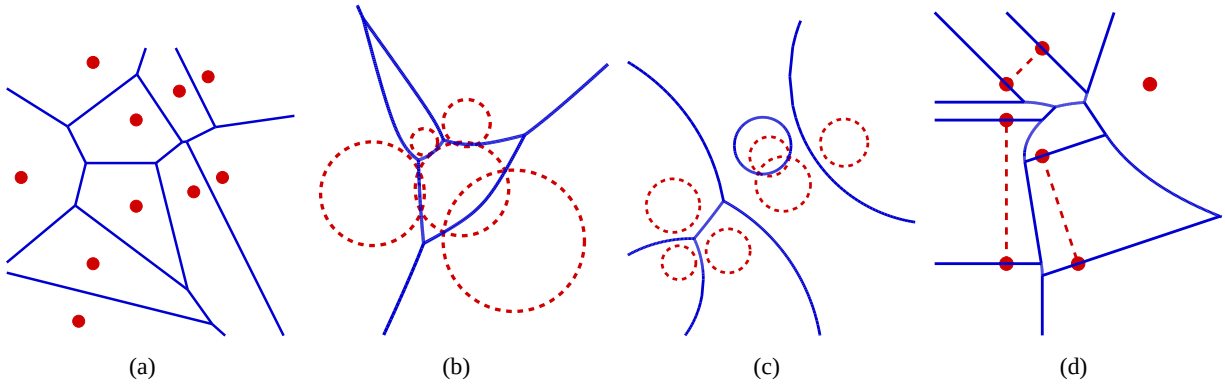


Figure 1: Various planar Voronoi diagrams computed with our software. (a) A standard Voronoi diagram (point sites with L_2 metric). (b) An additively weighted Voronoi (Apollonius) diagram with disk centers as sites and disk radii as weights. (c) A Möbius diagram with disk centers as sites. The distance from every point on the boundary of a disk to its corresponding site is zero. (d) A Voronoi diagram of segments. The sites in (b), (c), and (d) are illustrated with dashed curves. The figures in this paper are best viewed in color.

represented as arrangements of CGAL [5]. Section 5 generalizes an algorithm for computing a minimum-width annulus of a set of points in the plane [10] to a set of disks in the plane, and shows how the framework is applied for solving this problem.

The major strength of our approach is its completeness, robustness, and generality, that is, the ability to handle degenerate input, the agility to produce exact results, and the capability to construct diverse types of Voronoi diagrams. The code is designed to successfully handle degenerate input, while exploiting the synergy between generic programming and exact geometric computing, and the divide-and-conquer framework to construct Voronoi diagrams. Theoretically, the randomized divide-and-conquer envelope approach for computing Voronoi diagrams is efficient and it is asymptotically comparable to other (near-)optimal methods. However, the method uses constructions of bisectors and Voronoi vertices as elementary building blocks, and they must be exact, which makes the concrete running time of our exact implementation inferior to those of existing implementations of various dedicated (specific diagram type) implementations.

Our software practically supports any kind of Voronoi diagrams, provided that the user supplies a set of basic procedures for manipulating a small number of sites and their bisectors. The implementation of these procedures is an art in itself. However, recent tools to manipulate arbitrary algebraic plane curves [12] cover a wide range of bisector types, which enables the construction of different types of Voronoi diagrams. Table 1 lists some of the types of diagrams that are currently supported by our implementation. The figure above shows two types of Voronoi diagrams on the sphere computed with our software. Its left part shows a spherical Voronoi diagram of 14 points, and its right part shows a spherical power diagram of 10 circles. Both diagrams are composed of geodesic arcs. Figure 1 illustrates several types of planar Voronoi diagrams computed by our software.

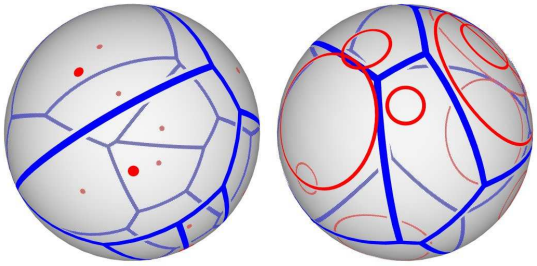


Figure 2: Voronoi diagrams on the sphere.

2 Preliminaries

2.1 A Divide-and-Conquer Algorithm for Constructing Voronoi Diagrams

Definition 1 (Lower envelope)**.** Given a set of bivariate functions $F = \{f_1, \ldots, f_n\}$, $f_i : \mathbb{R}^2 \to \mathbb{R}$, their *lower envelope* $\Psi(x,y)$ is defined to be their pointwise minimum: $\Psi(x,y) = \min_{1 \le i \le n} f_i(x,y)$.

The *minimization diagram* of S is the subdivision of the xy-plane into maximal relatively-open connected cells, such that the function (or the set of functions) that attains the lower envelope over a specific cell of the subdivision is the same.

Definition 2 (Voronoi diagram)**.** Let $O = \{o_1, \ldots, o_n\}$ be a set of n objects in the plane (also called *Voronoi sites*). The *Voronoi diagram* $\mathrm{Vor}_\rho(O)$ of O with respect to a given distance function ρ, is a partition of the plane into maximally connected cells, where each cell consists of points that are closer to one particular site (or a set of sites) than to any other site.

In certain cases, the distance to a site may depend on various parameters associated with the site; see, for example, the cases of Möbius diagrams or anisotropic diagrams in Table 1. The *bisector* $B(o_i, o_j)$ of two Voronoi sites $o_i, o_j \in O$ is the locus of all points that have an equal distance to both sites, that is

$$B(o_i, o_j) = \{x \in \mathbb{R}^2 \mid \rho(x, o_i) = \rho(x, o_j)\}.$$

From the above definitions it is clear that if we define $f_i : \mathbb{R}^2 \to \mathbb{R}$ to be $f_i(x) = \rho(x, o_i)$, for each $i = 1, \ldots, n$, then the minimization diagram of $\{f_1, \ldots, f_n\}$ is exactly the Voronoi diagram of O.

Computing envelopes of functions in $\mathbb{R}^3$ can be done with a divide-and-conquer algorithm [2]. The algorithm is adapted to Voronoi diagrams computation as follows: let S be a collection of n sites in the plane and let ρ be some distance function. We partition S into two disjoint subsets S_1 and S_2 of (roughly) equal size, construct their respective Voronoi diagrams $\mathrm{Vor}_\rho(S_1)$ and $\mathrm{Vor}_\rho(S_2)$, recursively, and then merge the two diagrams to obtain $\mathrm{Vor}_\rho(S)$.

The merging step starts with overlaying $\mathrm{Vor}_\rho(S_1)$ and $\mathrm{Vor}_\rho(S_2)$. For each face f of the overlay, all its points have a fixed pair of nearest neighbors s_1 and s_2 from S_1 and S_2, respectively, where the bisector between s_1 and s_2 (restricted to f) partitions f into its portion of points nearer to s_1 and the complementary portion of points nearer to s_2. This results with pieces of the final Voronoi cells. Each feature of the refined overlay is labeled with the sites nearest to it. Finally, redundant features are removed (these are vertices and portions of edges from one Voronoi diagram that lie closer to a site in the other Voronoi diagram), and subcells of the same cell are stitched together, to yield the combined final diagram.

The asymptotic worst-case time complexity of the divide-and-conquer envelope algorithm (under the natural assumption that the objects and distance function have "constant description complexity") is $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$. Indeed, there are Voronoi diagrams in the plane that have quadratic complexity (such as Möbius diagrams), for which this construction is nearly worst-case optimal.

2.2 Constructing Envelopes in CGAL

CGAL is a library of efficient and reliable geometric data structures and algorithms [1]. It follows the exact geometric computation paradigm [39]. CGAL adheres to the generic programming paradigm [4] to achieve maximum flexibility without compromising with efficiency.

The `Arrangement_on_surface_2` package of CGAL supports the construction, the maintenance, and the manipulation of arrangements embedded on certain two-dimensional oriented parametric surfaces in three dimensions, such as spheres, cylinders, tori, etc. [5, 38].

The arrangement package is the basis of the `Envelope_3` package of CGAL, which implements the algorithm mentioned in the previous section to compute the lower (or the upper) envelope of a set of surfaces in three dimensions [25]. While insuring stability, the number of the exact (and slow) arithmetic operations is minimized by propagating known information to adjacent cells during the merge step of the algorithm. The package decouples the topology-related computation from the geometry-related computation, resulting in a generic and easy to reuse and adapt software. The `Envelope_3` package handles all degenerate situations, and when used with exact numeric types, achieves robustness and produces exact results.

The aforementioned definitions (Section 2.1) can be generalized to lower envelopes (or Voronoi diagrams) projected onto two-dimensional parametric surfaces. Furthermore, the divide-and-conquer algorithm can be used to compute lower (or upper) envelopes over these surfaces.

3 Near-Optimal Bound on the Expected Construction Time

The complexity of the merge step of the algorithm described in the Section above directly depends on the complexity of the overlay of the two partial diagrams. (The cost of the best algorithm for constructing the overlay is larger by a logarithmic factor than the combined complexity of the input diagrams and of the overlay.) Careless partition of the input sites into two subsets can dramatically slow down the computation. For example, consider the standard L_2-diagram for the following point-set in the plane, $S = \{(i,i)\}_{i=1}^{n/2} \cup \{(-i,i)\}_{i=1}^{n/2}$; see Figure 4 for an illustration. If we partition the set into two subsets, to the left and to the right of the *y*-axis, then in the final merge step, the overlay of the two sub-diagrams has $\Theta(n^2)$ complexity. Hence the algorithm runs in $\Omega(n^2)$ time, whereas the complexity of the final diagram is only $\Theta(n)$. We argue that when the partitioning is done *randomly*, the expected complexity of the overlay is comparable with the maximum complexity of the diagram for essentially any kind of sites and distance function, and for any possible input.

Theorem 3. *Consider a specific type of two-dimensional Voronoi diagrams, so that the worst-case complexity of the diagram of any set of at most n sites is $F(n)$. Let S be a set of n sites. If we randomly split S into two subsets S_1 and S_2, by choosing at random for each site, with equal probability, the subset it belongs to, then the expected complexity of the overlay of the Voronoi diagram of S_1 with the Voronoi diagram of S_2 is $O(F(n))$.*

For the proof see Appendix A.

The presented implementation uses the overlay method provided by the `Arrangement_on_surface_2` package. It is based on a sweep algorithm, the running time complexity of which is $O(F(n)\log n)$. This yields:

Corollary 4. *For a specific type of two-dimensional Voronoi diagrams, so that the worst-case complexity of the diagram of any set of at most n sites is $O(n)$, the divide-and-conquer envelope algorithm computes it in expected $O(n\log^2 n)$ time. If the worst-case complexity $F(n)$ is $\Omega(n^{1+\varepsilon})$ then the expected running time is $O(F(n)\log n)$.*

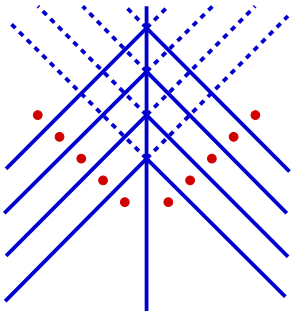
When the diagram is a convex subdivision, one can carry out the merge step more efficiently, in $O(F(n))$ expected time [18]. In particular, we have:

Corollary 5. *The L_2-Voronoi diagram of n points in the plane, or the power diagram of n disks in the plane, can be computed using the randomized divide-and-conquer envelope algorithm in expected optimal $O(n\log n)$ time.*

Remark. The analysis used to prove Theorem 3 can easily be extended to the case of lower envelopes of collections of bivariate functions (of constant description complexity). As a result, we get the following.

Corollary 6. *Let $\mathcal{G}$ be a collection of n bivariate functions of constant description complexity each, and let $F(m)$ be an upper bound on the complexity of the lower envelope of any subcollection of at most m functions. Then the expected complexity of the overlay of the minimization diagrams of two subcollections $\mathcal{G}_1$ and $\mathcal{G}_2$, obtained by randomly partitioning $\mathcal{G}$, as above, is $O(F(n))$. Consequently, the lower envelope of $\mathcal{G}$ can be constructed by the above randomized divide-and-conquer technique, in expected time $O(F(n)\log n)$, provided that $F(n) = \Omega(n^{1+\varepsilon})$, for some $\varepsilon > 0$. The expected running time is $O(n\log^2 n)$ when $F(n) = O(n)$.*

4 Examples and Experimental Results



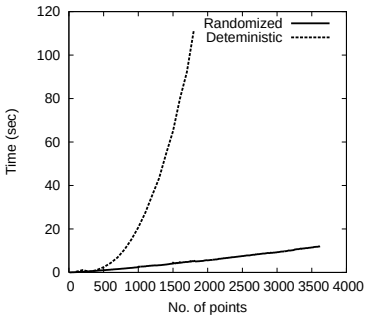


Figure 4 on the right demonstrates the effect of partitioning the sites randomly on worst-case inputs mentioned in Section 3. If we partition the set into two subsets, to the left and to the right of the y-axis ("deterministic" partitioning), the overlay of the two Voronoi diagrams has $\Theta(n^2)$ complexity. In the left part of the figure you can see such a case of 10 sites. The dashed segments are segments that exist in the overlay but will not be present in the final diagram. The right part of the figure shows that partitioning the sites randomly, results in nearly-linear running time in the number of sites. The graph shows the running time in seconds as a function of the number of sites (arranged in a worst case constellation) in both the worst-case deterministic and the randomized partitionings.

Creating a new type of Voronoi diagrams amounts only to the provision of a small set of geometric types and operations on a small number of sites and bisectors (e.g., determine which is the closer site to a given point, construct the bisector of two sites, etc.), gathered in a traits class. Despite the fact that the envelope code is used to compute Voronoi diagrams, the user does not have to define the surfaces explicitly, and does not have to know the algorithmic details of constructing minimization diagrams of envelopes of surfaces. More details can be found in Appendix B.

The required traits class must also be able to manipulate bisector curves, as required by the arrangement package. Among the already available traits classes in the arrangement package are traits for linear objects, circular arcs, conic arcs [38], and real algebraic plane curves [12]. These traits classes enable the creation of all the presented Voronoi diagrams and Voronoi diagrams with bisectors that can be represented by curves supported by these traits classes.

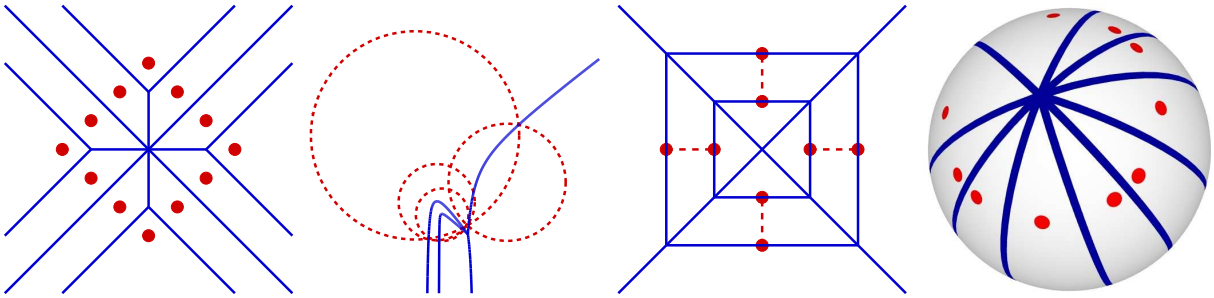


Figure 3: Degenerate Voronoi diagrams computed with our software. (a) A degenerate standard Voronoi diagram. (b) A degenerate additively weighted Voronoi (Apollonius) diagram. (c) A degenerate Voronoi diagram of segments. (d) A degenerate spherical Voronoi diagram.

We extended the `Envelope_3` package to support minimization diagrams that are embedded on two-dimensional parametric surfaces using the new `Arrangement_on_surface_2` package [5]. We extensively reuse code developed for planar Voronoi diagrams to compute Voronoi diagrams embedded on different surfaces. For example, using the framework together with the traits class for arcs of great circles on a sphere, we can compute Voronoi diagrams of points on a sphere and power diagrams on a sphere (see [15, 16] for more information and Figures 2, 3(d) for illustrations).

Figures 3 and 4 illustrate advantages of using CGAL arrangements as the base to the framework. The first is the handling of different scenarios of degenerate input. The latter is the immediate computation of

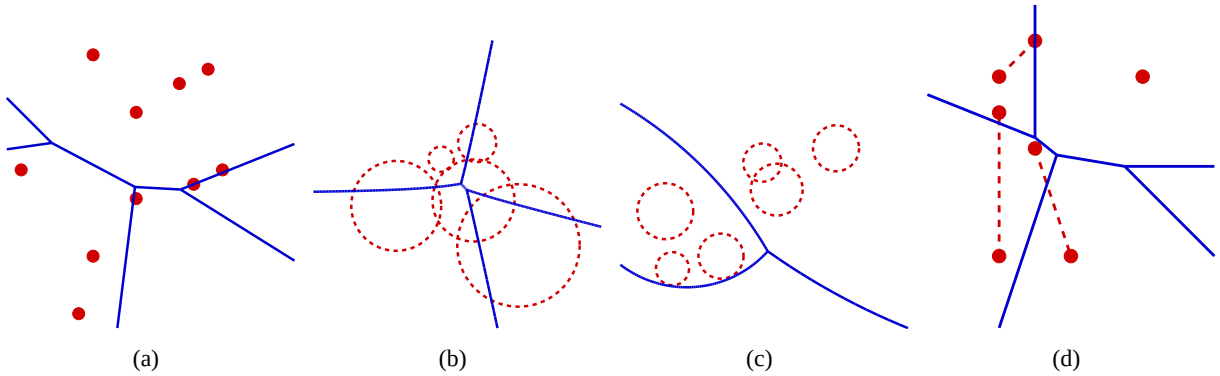


Figure 4: Farthest Voronoi diagrams of the same sites as in Figure 1 computed with our implementation. (a) A farthest point Voronoi diagram. (b) A farthest additively weighted Voronoi diagram. (c) A farthest Möbius diagram. (d) A farthest Voronoi diagram of segments.

both nearest-site Voronoi diagrams and farthest-site Voronoi diagrams which is enabled by computing lower and upper envelopes, respectively. The output is represented as an exact CGAL arrangement. This implies that we can easily traverse the vertices, edges, and faces of the diagram, and obtain the coordinates of the vertices of the diagram to any desired precision. We get plenty of additional functionality for free. For example, various point-location algorithms that come with the arrangement package.

Overlaying two (or more) Voronoi diagrams is also immediate, a fact that we used to implement the algorithm described below for computing a minimum-width annulus of a set of disks in the plane.

5 Minimum-Width Annulus of Disks

In this section we describe an application of our framework that demonstrates its generality and usefulness. We use our tools to consturct two different Voronoi diagrams and find a minimum-width annulus containing a set of disks in the plane.

An *annulus* is the bounded area between two concentric circles. The width of an annulus is the difference between the radii of the circles. Given a set of objects in the plane (disks in our case) the objective is to find a minimum-width annulus containing those objects, if one exists.[4] For point sites, a minimum-width annulus can be found by computing the overlay of the nearest-neighbor and farthest-neighbor Voronoi diagrams of the points. The center of the desired annulus must be a vertex of the overlay [10]. A similar approach handles the case of disk sites but the diagrams have to be defined more carefully.

Let $p \in \mathbb{R}^2$ be a point and $S \subset \mathbb{R}^2$ be a set of points. Consider the farthest-point distance function $\rho(p,S) = \max_{x \in S} ||p-x||$, which measures the farthest distance from the point p to the set S. Consider the farthest-neighbor Voronoi diagram with respect to this distance function. We call this diagram the "farthest-point farthest-site" Voronoi diagram. The distance function $\rho(p,S)$ becomes the Euclidean distance when the set S consists of a single point. However, this is not the case when the set S is a disk in the plane, say.

For a disk d in the plane with center c and radius r the farthest-point distance function is $\rho(p,d) = ||p-c|| + r$. Recall that the Apollonius distance function is defined to be $\rho(p,d) = ||p-c|| - r$. Next, observe that the farthest-point distance function is the same as the Apollonius distance with negative radii.

We prove that there is a minimum-width annulus (in case one exists), whose center is a vertex of the overlay of the Apollonius diagram and the farthest-point farthest-site Voronoi diagram of the disks.

Let $\mathscr{D} = \{d_1, \ldots, d_n\}$ be a collection of disks in the plane, and let C_I, C_O, and O denote the respective inner circle, outer circle, and center of an annulus containing $\mathscr{D}$. For simplicity of exposition, we assume

[4] When the width of the set is smaller than the width of any containing annulus there is no minimum-width annulus.

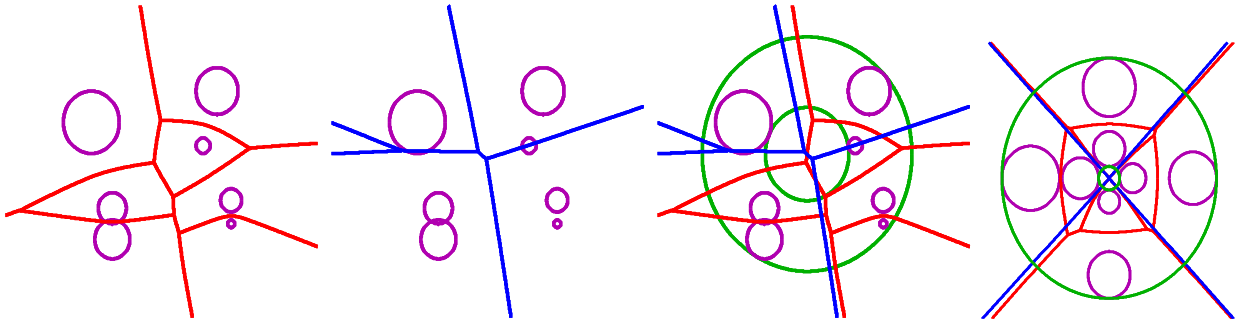


Figure 5: Computing a minimum-width annulus of a set of disks. (a) The Apollonius diagram of the set of disks. (b) The farthest-point farthest-site Voronoi diagram of the set of disks. (c) A minimum-width annulus of the set of disks. The center of the annulus is a vertex in the overlay of the Apollonius and the farthest-point farthest-site Voronoi diagrams. (d) A highly degenerate scenario for computing a minimum-width annulus of a set of disks. Four disks induce the vertex of the farthest-point farthest-site Voronoi diagram and other four disks induce a vertex of the Apollonius diagram at the same point.

here that $n \geq 3$; the cases of $n < 3$ are simple to handle. We show that there is a minimum-width annulus, whose bounding circles intersect the disks of $\mathscr{D}$ in at least 4 points in total, arguing as in [33].

Theorem 7. *If there is a minimum-width annulus containing $\mathscr{D}$, then there is also a minimum-width annulus such that the total number of intersection points between its bounding circles and the disks is at least 4.*

We omit the detailed proof in this extended abstract. Each minimum-width annulus must touch the disks of $\mathscr{D}$ in at least two points, as we can shrink C_O and expand C_I until each of them touches a disk. Thus, fixing O also fixes an annulus that touches the farthest point from (and the nearest point to) O contained in the disks of $\mathscr{D}$. In case N does not touch the disks of $\mathscr{D}$ in at least 4 points, we can move O and get a smaller width annulus.

From the above theorem it is clear that there are three cases:

1. C_I intersects at least 3 disks of $\mathscr{D}$. This means that O coincides with a vertex of the Apollonius diagram of the disks.
2. C_O intersects at least 3 disks of $\mathscr{D}$. This means that O coincides with a vertex of the farthest-point farthest-site Voronoi diagram of the disks.
3. Each of C_I and C_O intersects at least 2 disks of $\mathscr{D}$. In this case the annulus center O lies on a Voronoi edge of the Apollonius diagram and on an edge of the farthest-point farthest-site Voronoi diagram.

We therefore construct each of the diagrams, using our general divide-and-conquer machinery, and then overlay the two diagrams. For each vertex O of the overlay, we retrieve the four relevant disks (either the three touching the inner circle and the one touching the outer circle, in case 1, or the three touching the outer circle and the one touching the inner circle, in case 2, or the two pairs of disks touching respectively the inner and outer circles), and compute the width of the resulting annulus. We output the annulus of the smallest width. Figure 5 illustrates the algorithm for computing a minimum-width annulus of disks and a highly degenerate input that is being handled properly by our implementation.

Apollonius diagrams are of linear size in the number of sites, thus Corollary 4 is applied to yield an expected construction time of $O(n\log^2 n)$ for both diagrams. Overlaying the two diagrams with the sweep-based algorithm has $O((n+k)\log n)$ worst-case time complexity where k is the number of intersections between the diagrams. The total expected running time of the algorithm is therefore $O(n\log^2 n + k\log n)$, where k can be $\Theta(n^2)$.[5]

[5]It is reasonable to assume that the expected time complexity is, in many cases, smaller. Though not proven for disks, for

Table 2: Time consumption (in seconds) of minimum-width annuli computation and sizes of the corresponding constructed diagrams. V — the number of vertices of the diagram, E — the number of edges of the diagram, F — the number of faces of the diagram, T — the time consumed by the respective phase, C — the time consumed during the comparison of the candidates for the annulus center. "dgn_*" are input files in degenerate settings (see Figure 5(d) for an illustration), and "rnd_*" are input files in a random settings.

Input	Apollonius				FPFS VD				Overlay				C	Total
	V	E	F	T	V	E	F	T	V	E	F	T		
rnd_50	84	128	45	6.74	10	21	12	2.40	126	213	88	0.44	0.57	10.16
rnd_100	162	242	81	17.47	17	35	19	5.36	238	395	158	0.81	1.46	25.12
rnd_200	317	460	144	44.07	15	31	17	11.16	416	659	244	1.28	1.33	57.86
rnd_500	672	967	296	136.46	16	33	18	29.30	775	1174	400	1.85	1.97	169.59
dgn_50	59	106	48	5.74	1	25	25	1.94	100	213	114	0.84	0.30	8.83
dgn_100	134	232	99	16.57	1	50	50	5.83	239	492	254	2.93	1.11	26.46
dgn_200	304	502	199	42.60	1	100	100	15.77	581	1156	576	7.60	2.78	68.76
dgn_500	785	1262	478	154.37	1	239	239	56.21	1645	3221	1577	37.93	7.94	256.47

The implementation of the Apollonius diagram is based on the algebraic plane curves traits for the arrangement package [12] as the bisector of two Apollonius sites is one branch of a hyperbola. We choose the correct branch of the hyperbola by lexicographically sorting the x-monotone sub-curves of the full curve, and then considering the centers and radii of the disks.

Given this implementation of the Apollonius traits class, the general framework allows us to easily compute the farthest-point farthest-site Voronoi diagram by computing the farthest Voronoi diagram of the disks with negative radii. The overlay operation is readily available in the arrangement package.

Table 2 shows the sizes of the constructed diagrams in each phase of the algorithm, and the time consumption (in seconds) of the execution of each phase. The vertices of the overlay are the candidates for the center of the annulus. The experiments were carried out on an Intel® Core™2 Duo 2GHz processor with 1GB memory running Linux operating system.

6 Conclusions and Future Work

Our framework together with the existing traits classes of the arrangement package provide the means to produce various Voronoi diagrams in an exact and robust way. Moreover, the theoretical bound on the expected running time of our algorithm is nearly optimal.

Future work is to enrich the variety of Voronoi diagrams computed with our software to include Voronoi diagrams of circular arcs, Bregman Voronoi diagrams [29], Voronoi diagrams in the hyperbolic Poincaré half-plane [32] or in the Poincaré hyperbolic disk [30], Hausdorff Voronoi Diagrams [31], and Voronoi diagrams on different surfaces, for example, cylinders or tori. Some of the above diagrams have bisectors that can be handled with existing traits classes available by the arrangement package. Another objective is to generalize the computation of minimum-width annuli to other types of objects.

Acknowledgements

The authors are grateful to Efi Fogel and Eric Berberich for many helpful suggestions and fruitful cooperation, and to Ron Wein, Michal Meyerovitch, and Barak Raveh for their cooperation and support of this work.

random point sets it is known that the expected number of intersections between the farthest and the nearest Voronoi diagrams is linear.

References

[1] CGAL *User and Reference Manual*, 3.4 edition, 2008. `http://www.cgal.org/Manual/3.4/doc_html/cgal_manual/packages.html`.

[2] P. K. Agarwal, O. Schwarzkopf, and M. Sharir. The overlay of lower envelopes and its applications. *Disc. Comput. Geom.*, 15:1–13, 1996.

[3] F. Aurenhammer and R. Klein. Voronoi diagrams. In J.-R. Sack and J. Urrutia, editors, *Handb. Comput. Geom.*, chapter 5, pages 201–290. Elsevier, 2000.

[4] M. H. Austern. *Generic Programming and the STL*. Addison-Wesley, 1999.

[5] E. Berberich, E. Fogel, D. Halperin, K. Mehlhorn, and R. Wein. Sweeping and maintaining two-dimensional arrangements on surfaces: A first step. In *Proc. 15th Annu. Eur. Symp. Alg.*, pages 645–656, 2007.

[6] J.-D. Boissonnat, O. Devillers, S. Pion, M. Teillaud, and M. Yvinec. Triangulations in CGAL. *Comput. Geom. Theory Appl.*, 22(1-3):5–19, 2002.

[7] J.-D. Boissonnat, C. Wormser, and M. Yvinec. Curved Voronoi diagrams. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*. Springer, 2006.

[8] C. Burnikel, K. Mehlhorn, and S. Schirra. How to compute the Voronoi diagram of line segments: Theoretical and experimental results. In *Proc. 2nd Annu. Eur. Symp. Alg.*, pages 227–239, 1994.

[9] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Disc. Comput. Geom.*, 4(5):387–421, 1989.

[10] H. Ebara, N. Fukuyama, H. Nakano, and Y. Nakanishi. Roundness algorithms using the Voronoi diagrams. In *Proc. 16th Canadian Conf. on Comput. Geom.*, volume 41, 1989.

[11] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Disc. Comput. Geom.*, 1:25–44, 1986.

[12] A. Eigenwillig and M. Kerber. Exact and efficient 2D-arrangements of arbitrary algebraic curves. In *Proc. 19th Annu. ACM-SIAM Symp. Disc. Alg.*, pages 122–131. Soc. for Industrial and Applied Math. (SIAM), 2008.

[13] I. Z. Emiris and M. I. Karavelas. The predicates of the Apollonius diagram: Algorithmic analysis and implementation. *Comput. Geom. Theory Appl.*, 33(1-2):18–57, 2006.

[14] I. Z. Emiris, E. P. Tsigaridas, and G. Tzoumas. Voronoi diagram of ellipses in CGAL. In *Abstracts of 24th Eur. Workshop Comput. Geom.*, pages 87–90, 2008.

[15] E. Fogel, O. Setter, and D. Halperin. Exact implementation of arrangements of geodesic arcs on the sphere with applications. In *Abstracts of 24th Eur. Workshop Comput. Geom.*, pages 83–86, 2008.

[16] E. Fogel, O. Setter, and D. Halperin. Movie: Arrangements of geodesic arcs on the sphere. In *Proc. 24th Annu. ACM Symp. Comput. Geom.*, pages 218–219, 2008.

[17] S. J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, pages 153–174, 1987.

[18] L. J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Disc. Comput. Geom.*, 2:175–193, 1987.

[19] 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(2):95–123, 2001.

[20] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proc. 26th Annu. Int. Conf. on Comput. Graphics and Interactive Techniques*, pages 277–286. ACM Press, 1999.

[21] M. Karavelas and M. Yvinec. The Voronoi diagram of planar convex objects. In *Proc. 11th Annu. Eur. Symp. Alg.*, pages 337–348, 2003.

[22] M. I. Karavelas. A robust and efficient implementation for the segment Voronoi diagram. In *Proc. 1st Int. Symp. on Voronoi Diagrams in Sci. and Engineering*, pages 51–62, 2004.

[23] R. Klein. *Concrete and Abstract Voronoi Diagrams*, volume 400 of *LNCS*. Springer, 1989.

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

[25] M. Meyerovitch. Robust, generic and efficient construction of envelopes of surfaces in three-dimensional space. In *Proc. 14th Annu. Eur. Symp. Alg.*, pages 792–803, 2006.

[26] M. Meyerovitch, R. Wein, and B. Zukerman. 3d envelopes. In C. E. Board, editor, *CGAL User and Reference Manual*. 3.4 edition, 2008.

[27] N. Myers. "Traits": A new and useful template technique. In S. B. Lippman, editor, *C++ Gems*, volume 5 of *SIGS Reference Library*, pages 451–458. 1997.

[28] F. Nielsen. An interactive tour of Voronoi diagrams on the GPU. In *ShaderX*^{6}*: Advanced Rendering Techniques*. Charles River Media, 2008.

[29] F. Nielsen, J.-D. Boissonnat, and R. Nock. On bregman voronoi diagrams. In *Proc. 18th Annu. ACM-SIAM Symp. Disc. Alg.*, pages 746–755, Philadelphia, PA, USA, 2007. Soc. for Industrial and Applied Math. (SIAM).

[30] Z. Nilforoushan and A. Mohades. Hyperbolic Voronoi diagram. In *Int. J. of Comput. Geom. Appl.*, pages 735–742, 2006.

[31] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, 2nd edition, 2000.

[32] K. Onishi and N. Takayama. Construction of Voronoi diagram on the upper half-plane. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 79(4):533–539, 1996.

[33] T. J. Rivlin. Approximating by circles. *Computing*, 21:93–104, 1979.

[34] M. Seel. Eine Implementierung abstrakter Voronoidiagramme. Master's thesis, Universität des Saarlandes, 1994.

[35] M. I. Shamos and D. Hoey. Closest-point problems. In *Proc. 16th IEEE Symp. Found. Comput. Sci.*, pages 151–162, 1975.

[36] B. Stroustrup. *The C++ Programming Language, Third Edition*. Addison-Wesley, 1997.

[37] K. Sugihara. Laguerre voronoi diagram on the sphere. *J. for Geom. Graphics*, 6(1):69–81, 2002.

[38] R. Wein, E. Fogel, B. Zukerman, and D. Halperin. 2D arrangements. In CGAL Editorial Board, editor, CGAL *User and Reference Manual*. 3.4 edition, 2008.

[39] C.-K. Yap and T. Dubé. The exact computation paradigm. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*, pages 452–492. World Scientific, Singapore, 2nd edition, 1995.

A Proof of Theorem 3

Theorem 3. *Consider a specific type of two-dimensional Voronoi diagrams, so that the worst-case complexity of the diagram of any set of at most n sites is $F(n)$. Let S be a set of n sites. If we randomly split S into two subsets S_1 and S_2, by choosing at random for each site, with equal probability, the subset it belongs to, then the expected complexity of the overlay of the Voronoi diagram of S_1 with the Voronoi diagram of S_2 is $O(F(n))$.*

Proof. Each vertex of the overlay is either a vertex of $\mathrm{Vor}_\rho(S_1)$, a vertex of $\mathrm{Vor}_\rho(S_2)$, or a crossing between an edge of $\mathrm{Vor}_\rho(S_1)$ and an edge of $\mathrm{Vor}_\rho(S_2)$ (non exclusive disjunction.) The number of vertices of the first two kinds is $O(F(n))$, so it suffices to bound the expected number of crossings between edges of $\mathrm{Vor}_\rho(S_1)$ and of $\mathrm{Vor}_\rho(S_2)$. Such a crossing is a point u that is defined by four sites, $p_1, q_1 \in S_1$ and $p_2, q_2 \in S_2$, so that u lies on the Voronoi edge $b(p_1, q_1)$ of $\mathrm{Vor}_\rho(S_1)$ that bounds the cells of p_1 and q_1, and on the Voronoi edge $b(p_2, q_2)$ of $\mathrm{Vor}_\rho(S_2)$ that bounds the cells of p_2 and q_2. Without loss of generality, assume that $\rho(u, p_1) = \rho(u, q_1) \leq \rho(u, p_2) = \rho(u, q_2)$ (the inequality is strict if we assume general position).

A simple but crucial observation is that u must also lie on the Voronoi edge between the cells of p_1, q_1 in the *overall* diagram $\mathrm{Vor}_\rho(S)$. Indeed, if this were not the case then there must exist another site $s \in S$ so that u is nearer to s than to p_1, q_1. But then s cannot belong to S_1, for otherwise it would prevent u from lying on the Voronoi edge of p_1, q_1. For exactly the same reason, s cannot belong to S_2 — it would then prevent u from lying on the Voronoi edge of p_2, q_2 in that diagram. This contradiction establishes the claim.

Define the *weight* k_u of u to be the number of sites s satisfying

$$\rho(u, p_1) = \rho(u, q_1) < \rho(u, s) < \rho(u, p_2) = \rho(u, q_2).$$

Clearly, all these k_u sites must be assigned to S_1.

In other words, for any crossing point u between two Voronoi edges $b(p_1, q_1)$, $b(p_2, q_2)$, with weight k_u (with all the corresponding k_u sites being farther from u than p_1, q_1 and nearer than p_2, q_2), u appears as a crossing point in the overlay of $\mathrm{Vor}_\rho(S_1)$, $\mathrm{Vor}_\rho(S_2)$ if and only if the following three conditions (or their symmetric counterparts, obtained by reversing the roles of S_1, S_2) hold: (i) $p_1, q_1 \in S_1$; (ii) $p_2, q_2 \in S_2$; and (iii) all the k_u sites that contribute to the weight are assigned to S_1. This happens with probability $\frac{1}{2^{k_u+3}}$.

Hence, if we denote by N_w (resp., $N_{\leq w}$) the number of crossings of weight w (resp., of weight at most w), the expected number of crossings in the overlay is

$$\sum_{w \geq 0} \frac{N_w}{2^{w+3}} = O\left(\sum_{w \geq 0} \frac{N_{\leq w}}{2^w} \right), \tag{1}$$

where the right-hand side is obtained by substituting $N_w = N_{\leq w} - N_{\leq w-1}$, and by a simple rearrangement of the sum.

We can obtain an upper bound on $N_{\leq w}$ using the Clarkson-Shor technique [9]. Specifically, denote by $N_w(n)$ (resp., $N_{\leq w}(n)$) the maximum value of N_w (resp., $N_{\leq w}$), taken over all sets of n sites. Then, since a crossing is defined by four sites, we have

$$N_{\leq w}(n) = O\left(w^4 N_0(n/w)\right).$$

Note that if a crossing u, defined by p_1, q_1, p_2, q_2, has weight 0 then p_1, q_1, p_2, q_2 are the four nearest sites to u. The number of such quadruples is thus upper bounded by the complexity of the *fourth-order* Voronoi diagram of some set S_0 of n/w sites.

We claim that the complexity of the fourth-order Voronoi diagram of n sites is $O(F(n))$. Indeed, any quadruple p_1, q_1, p_2, q_2 of four nearest sites to some point u can be charged to a face of the fourth-order

diagram (the one whose projection contains u). Each such face can in turn be charged either to one of its vertices, or to its rightmost point, or to a point at infinity on one of its edges. Assuming general position, each such boundary point can be charged at most $O(1)$ times. Now another simple application of the Clarkson-Shor technique shows that the number of these vertices and boundary points is $O(F(n))$ — each of them becomes a feature of the (0-order) Voronoi diagram if we remove a constant number of sites, which happens with large probability when we sample a constant fraction of the sites.

In other words, we have $N_{\leq w}(n) = O(w^4 F(n/w)) = O(w^4 F(n))$. Substituting this into (1), we obtain an upper bound of $O(F(n))$ on the complexity of the overlay, as claimed. □

B Software Interface: Adding New Diagrams

This appendix contains technical details on what needs to be supplied by the user of the framework in order to implement a new type of Voronoi diagrams. The interface consists of several functions, each operating on a small number of user-defined types (sites or bisector curves). Providing these functions does not require knowledge of the underlying algorithms. In this appendix we assume some familiarity of the reader with generic programming [4] and the C++ programming language [36].

The `Envelope_3` package is fairly general and can deal with surfaces that have two-dimensional intersections. Therefore, the `Envelope_3` can also compute Voronoi diagrams with two-dimensional bisectors[6]. However, most types of Voronoi diagrams have only one-dimensional bisectors. We used this fact to reduce and simplify the interface that is used by the `Envelope_3` code for diagrams with one-dimensional bisectors. In order to construct Voronoi diagrams with two-dimensional bisectors we advise the user to refer to the `Envelope_3` documentation for further information [26].

We require the user of our framework to define a set of geometric types and operations that will be used by the algorithm. This way the user can adapt the algorithm to compute the desired type of Voronoi diagrams. We emphasize that a user wishing to add a new type of diagrams does not have to know the algorithmic details of computing the minimization diagram of the envelope of surfaces.

Following the generic programming approach, our algorithm is parameterized with a *traits* class [27]. A traits class should provide certain predefined types and methods, and is passed as a parameter to a class template. In our case, the algorithm is the class template, which accepts a traits class that encapsulates the geometric types and operations that the algorithm uses. The set of requirements of types, functions, and operators is called a *concept* [4]. The concept of the traits class for our Voronoi algorithm is called *EnvelopeVoronoiTraits_2*.

The first step in creating a new type of Voronoi diagrams is to define the embedding surface of the diagram. The creator of the traits class picks the embedding surface of the Voronoi diagram by defining the *Topology_traits* type, which encapsulates the topology of the surface on which the diagram is embedded. Currently implemented topology traits classes in the `Arrangement_on_surface_2` package include topology traits classes for the bounded or unbounded plane, a topology traits class for quadric surfaces, and a specially tailored topology traits class for the sphere.

The *EnvelopeVoronoiTraits_2* concept refines the *ArrangementTraits_2* concept, defined in the `Arrangement_on_surface_2` package, which is used when constructing two-dimensional arrangements. We need these additional predicates and operations to be able to manipulate bisector curves of Voronoi sites. A Voronoi site is represented by the user-defined `Site_2` type.

Another important type that a Voronoi traits class needs to define is a functor type named `Construct_bisector_2` and an appropriate method to obtain an instance of that functor. `Construct_bisector_2` is a functor that when given two `Site_2` variables and an output iterator, returns objects of type `X_monotone_curve_2` that form the bisector of the two Voronoi sites.

[6] For example, two point sites in the L_1-Voronoi diagram can have a two-dimensional bisector.

Table 3: Required types and functors by EnvelopeVoronoiTraits_2.

Name	*Input*	*Output*	*Description*
`Site_2`	—	—	A type that represents a Voronoi site.
`Construct_bisector_2`	Two `Site_2` objects	An output iterator with values of type of `X_monotone_curve_2`	Returns `X_monotone_curve_2` objects that together form the bisector of the two input sites.
`Compare_distance_above_2`	Two `Site_2` objects and an `X_monotone_curve_2`	`Comparison_result`	Determines which of the given Voronoi sites is closer to the area "above" the given *x*-monotone curve, where "above" is the area that lies to its left when the curve is traversed from its *xy*-lexicographically smaller end to its *xy*-lexicographically larger end.
`Compare_distance_at_point_2`	Two `Site_2` objects and a `Point_2`	`Comparison_result`	Determines which of the given Voronoi sites is closer to the given point.
`Construct_point_on_x_monotone_2`	`X_monotone_curve_2`	`Point_2`	Constructs an interior point on the given *x*-monotone curve.
`Compare_dominance_2`	Two `Site_2` objects	`Comparison_result`	Determines which of the sites dominates the other in case that there is no bisector between the two sites.

Other required functors are used to determine on which side of the bisector, if there is one[7], each site dominates. The `Compare_distance_above_2` accepts two sites and an *x*-monotone curve, which is part of their bisector and determines which site dominates the area above the *x*-monotone curve, where "above" is defined to be the area to the left of the curve when it is traversed from the *xy*-lexicographically smaller end-point to the *xy*-lexicographically larger end-point. Due to the fact that each of the two sites dominates one side of the bisector, the framework can use the "above" functor to implement the "below" version needed by the envelope code. If there is no bisector between the two sites then the `Compare_dominance_2` functor is used to determine which of the sites dominates the other.

The above functor is used to determine on which side of the bisector each site dominates, but the algorithm also needs to know which site dominates a given point on the two-dimensional surface. The `Compare_distance_at_point_2` functor is used to determine which of two given Voronoi sites is closer to a given point. The functor is used together with the `Construct_point_on_x_monotone_2` functor, which is used to construct an interior point on a given *x*-monotone curve.

After the user has satisfied all the requirements of the *EnvelopeVoronoiTraits_2* concept, the `Voronoi_2_to_Envelope_3_adaptor` class is used to adapt the Voronoi traits class to a full `Envelope_3` traits class [26].

Table 3 summarizes the types and functors that are needed in order to implement a new Voronoi diagram type. Notice that `Point_2` and `Comparison_result` are standard types in CGAL.

[7] There are cases where there is no bisector between two Voronoi sites. For example, two Apollonius sites where one is completely contained inside the other have no bisector.