Computing the Voronoi Cells of Planes, Spheres and Cylinders in $\mathbb{R}^3$

Iddo Hanniel * and Gershon Elber

Department of Computer Science, Technion, Israel Institute of Technology

Abstract

We present an algorithm for computing the Voronoi cell for a set of planes, spheres and cylinders in $\mathbb{R}^3$. The algorithm is based on a lower envelope computation of the bisector surfaces between these primitives, and the projection of the trisector curves onto planes bounding the object for which the Voronoi cell is computed, denoted the base object. We analyze the different bisectors and trisectors that can occur in the computation. Our analysis shows that most of the bisector surfaces are quadric surfaces and five of the ten possible trisectors are conic section curves. We have implemented our algorithm using the IRIT library and the CGAL 3D lower envelope package. All presented results are from our implementation.

Key words: Constructive Solid Geometry, skeleton, medial axis

1 Introduction and Previous Work

The *Voronoi diagram* (VD) and *medial axis transform* (MAT) have proven to be useful tools in a variety of application domains [1–3]. In $\mathbb{R}^2$, many algorithms have been proposed for constructing Voronoi diagrams of low-order objects (points, line segments and polygons). There are also several results on constructing VDs of higher order objects such as circles [4], ellipses [5] and free-form curves [6].

* Corresponding author.
Email addresses: iddoh@cs.technion.ac.il (Iddo Hanniel), gershon@cs.technion.ac.il (Gershon Elber).

In $\mathbb{R}^3$, there is an extensive literature on the computation of VDs and MATs of polyhedra. Varying approaches were used in these works: thinning algorithms [7], distance field computations [8,9], surface sampling approaches [10] and algebraic methods [11]. However, extensions of these results to non-linear input surfaces, without resorting to their approximation by polyhedra, are rare.

Definition 1 *Given a set of disjoint bounded objects O_0, O_1, ... , O_n in $\mathbb{R}^3$, the* Voronoi cell *(VC) of object O_0, denoted the* base object*, is the set of all points closer to O_0 than to O_j, $\forall\ j > 0$. The* Voronoi diagram *(VD) is then the union of the Voronoi cells of all $(n+1)$ objects.*

In this paper, the term "Voronoi cell", unless otherwise noted, refers to the "boundary of the Voronoi cell".

Definition 2 *The locus of points that are equidistant from O_0 and O_j, for some $j > 0$, is called (the O_0O_j)* bisector*. The locus of points that are equidistant from O_0, O_i and O_j, for some $i, j > 0$ $(i \neq j)$, is called (the $O_0O_iO_j$)* trisector*. In general, bisectors in $\mathbb{R}^3$ are surfaces and trisectors are curves.*

Figure 1(a) demonstrates bisector surfaces between a plane and two spheres. Figure 1(b) demonstrates the trisector curve between a plane and two spheres. It can be seen in the figure that the trisector is the intersection of the two bisectors. Furthermore, in Figure 1(b) one can see that every point (see point p) on the trisector is equidistant from the three primitives. The bisectors in the example are paraboloids (as will be explained in Section 3.1) and the trisector is an ellipse (see Lemma (2) in Section 3.2).

There have been several results on the computation of bisectors in $\mathbb{R}^3$. Dutta and Hoffman [12] computed the bisectors of CSG primitives (plane, sphere, cylinder, cone and torus). In some of the cases they assumed special configurations of the surfaces (e.g., that the center of a sphere is located on the axis of a cylinder). Elber and Kim [13] extended these results and showed that bisector surfaces of CSG primitives in general configurations are also rational. For example, they compute the bisector of a sphere and any CSG primitive by reducing the problem to that of a bisector between a point and an offset surface, which is rational [14]. Peternell [15] gave a survey of the properties of bisector surfaces, and further extended the results described above. He proved that the bisector of two developable surfaces, which have a rational unit normal, is rational. In particular, cylinders and cones are such surfaces.

An implementation that constructs a medial axis in $\mathbb{R}^3$ of surfaces that are not polyhedra was presented by Ramanathan and Gurumoorthy [16]. In [16], which is based on their work in [17], they constructed the medial axis of extruded and revolved shapes. They exploited the fact that the 3D MAT of an extruded or revolved shape is closely related to the 2D MAT of its creating section curve. The output of their algorithm is still a piecewise linear approximation of the

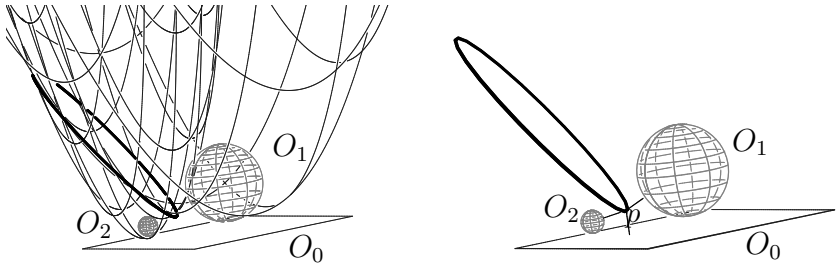


Fig. 1. An example of two bisector (surfaces) and a trisector (curve) between a plane and two spheres: (a) The bisector surfaces between the plane and each of the spheres, (b) The trisector curve (in bold black) between the plane and the two spheres, which is also the intersection of the two bisectors.

edges of the MAT.

Another direction of research, which has applications in molecular biology, has been on algorithms for constructing the VD of a set of spheres. Kim et al. [18] proposed an edge-tracing algorithm for computing the VD of a set of spheres in $\mathbb{R}^3$. Their algorithm assumes that the graph of the edges of each cell is connected, which is not true in general. An algorithm for computing a single Voronoi cell of a set of spheres was proposed by Will [19]. It is based on the insight that the projection of a trisector curve onto the base sphere is a circular arc. The algorithm is an incremental lower envelope construction, where the edges of the projection map are circular arcs.

In the computational geometry literature, there are related works on *additively weighted Voronoi diagrams*. The additively weighted Voronoi diagram of a set of points $\boldsymbol{p_1}, .., \boldsymbol{p_n}$ in $\mathbb{R}^d$ with respective weights $r_1, .., r_n$, is the Voronoi diagram under the additive distance function $dist(\boldsymbol{p}, \boldsymbol{p_i}) = ||\boldsymbol{p} - \boldsymbol{p_i}|| - r_i$ (see, for example, [20][Chapter 18.3.1]). When the spheres are disjoint, their Voronoi diagram is equivalent to the additively weighted Voronoi diagram in $\mathbb{R}^3$. Aurenhammer and Imai [21] showed that a cell of the additively weighted Voronoi diagram in $\mathbb{R}^d$ corresponds to the projection of the intersection of a cell of a suitably defined power diagram in $\mathbb{R}^{d+1}$ with a $d+1$-dimensional cone. Thus, the Voronoi cell can be computed by computing a power diagram in $\mathbb{R}^4$ and intersecting the power cells with a 4-dimensional cone. Boissonat and Delage [22] proposed a similar method to compute the additively weighted Voronoi diagram. Their algorithm is based on an incremental construction of the diagram. Computing a single Voronoi cell is reduced, by inversion, to computing an additively weighted convex hull problem, which is solved using the power diagram. They implemented their algorithm using filtered exact arithmetic.

In this work, we present an algorithm for computing the Voronoi cell of a set of objects in $\mathbb{R}^3$. The objects are the common CSG surfaces - (infinite) planes,

spheres and (infinite) cylinders. Given the input CSG primitives O_1, ... , O_n, and a base CSG primitive O_0, we can assume that the objects are translated and oriented so that the base object O_0 is positioned in a canonical manner. That is, if O_0 is a plane it is the xy-plane, if it is a sphere it is centered at the origin, and if it is a cylinder its axis is the y-axis. A user-defined bounding box is used to clip infinite objects. The algorithm outputs the trimmed surfaces of the Voronoi cell of O_0, as trimmed Bézier patches.

Similar to the algorithm of Will [19], our algorithm is based on a lower envelope algorithm of the bisector surfaces. However, unlike his algorithm, we project the trisectors onto a plane and not onto the sphere and therefore our algorithm can be extended to other base primitives that are not a sphere, namely, planes and cylinders (and probably others, e.g., cones). We also analyze, in Section 3, all the possibilities of trisectors between planes, spheres and cylinders, and show that of the ten trisector combinations five are, in fact, conic section curves. We use this fact in our algorithm. We implemented our algorithm using the IRIT solid modeling environment [23] and CGAL's lower envelope implementation [24,25], which is based on CGAL's arrangement package [26].

The rest of this paper is structured as follows: Section 2 outlines the basic idea of our algorithm and the lower envelope algorithm it is based on. Section 3 enumerates the different options of bisectors and trisectors for planes, spheres and cylinders, and discusses their representation. In Section 4, we describe the implementation of the lower envelope algorithm. Results from our implementation and a discussion appear in Section 5 while in Section 6, we discuss possible extensions of our work to other objects. In Section 7, we conclude the paper and discuss future improvements.

2 Basic Idea

The connection between Voronoi diagrams and lower envelopes is well known. For example, the Voronoi diagram of a set of curves in $\mathbb{R}^2$ corresponds to the projection of the lower envelope of the generalized cone surfaces in $\mathbb{R}^3$. The extended cones are constructed by extending 45-degree lines (with respect to the xy-plane) from the points on the curves (see, for example, [15]).

In our context, computing the Voronoi cell corresponds to computing the lower envelope of the bisectors with respect to the distance function to the base object O_0. The authors have exploited this fact in [6], where they computed the Voronoi cell of a set of curves in $\mathbb{R}^2$ using a two-dimensional lower envelope algorithm. In this work, we use a three-dimensional lower envelope algorithm for computing the Voronoi cell (VC) of a set of CSG primitives. The stages of the algorithm are:

(1) Construct the bisector surfaces, trim the bisectors to within a manageable bounding box and purge away unnecessary branches (e.g., branches that are clearly serving no purpose in the Voronoi cell, see Section 3.1).
(2) Compute the lower envelope of the trimmed bisectors with respect to O_0 (see Section 4).
(3) Represent the faces of the Voronoi cell as trimmed parametric patches .

Figure 2 demonstrates the three stages of the algorithm on a base object that is a plane, and three input spheres.

It should be noted that the lower envelope we compute for O_0 is not always the regular lower envelope of the surfaces. Here, the lower envelope is computed with respect to the *distance functions of the bisectors from the base object*. This means that for each point on the base object, the corresponding point on the envelope is the point with minimal distance *along the base object's normal direction* (we assume the base object is not penetrated by the other objects, see Section 6 for a discussion on how to extend our algorithm to the case where the base object is penetrated). In the special case when the base object is a plane (as in Figure 2), the lower envelope with respect to the distance functions is equivalent to the regular lower envelope of the bisector surfaces.

In Section 2.1, we overview the standard divide and conquer lower envelope algorithm for surfaces in $\mathbb{R}^3$, while in Section 2.2, the projections used in the VC computation are described.

2.1 Lower Envelope Algorithm in $\mathbb{R}^3$

The divide and conquer lower envelope algorithm in $\mathbb{R}^3$ assumes the surfaces are xy-monotone [1] . Therefore, the surfaces are split into xy-monotone surfaces as a preprocess. In our context, this is performed as part of the purging step of unnecessary branches (see Section 4.2).

For a set of xy-monotone surfaces, the algorithm is as follows (see [25] for a full description):

(1) If there is just one input surface, return this surface. (Base of recursion).

Otherwise:
(2) Divide the surfaces into two groups.

[1] xy-monotone surfaces are surfaces for which any line perpendicular to the xy-plane intersects the surface at most once. In our context, as we will see in Section 2.2, the monotonicity of the surface is defined more generally and depends on the lines perpendicular to the base object.

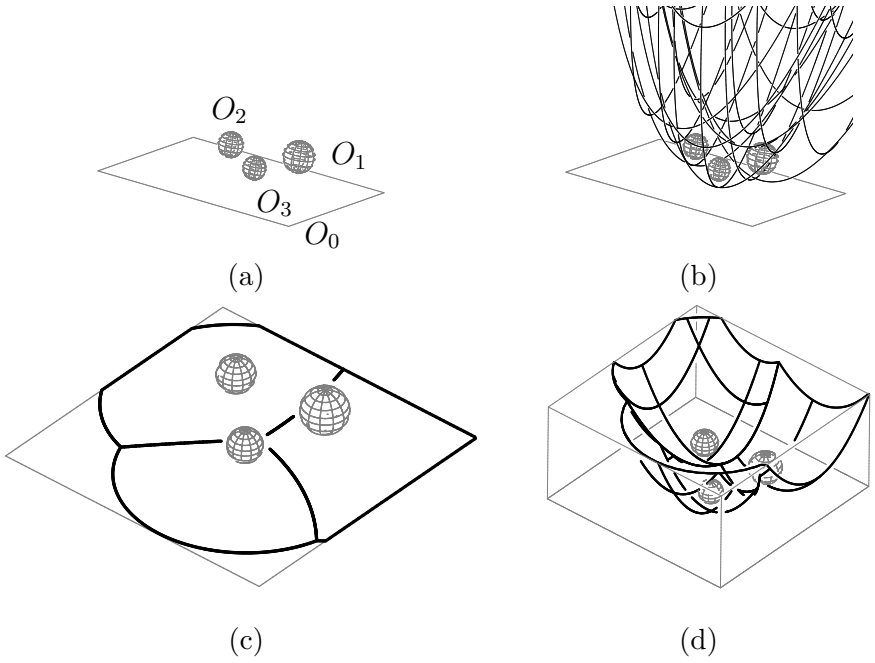


Fig. 2. An example of the stages of the algorithm with a base object, which is a plane, and three spheres: (a) The input CSG primitives, (b) The bisector surfaces, (c) The projection of the lower envelope onto the plane. (d) The faces of the Voronoi cell as trimmed parametric patches.

(3) Construct the lower envelope of each of the groups, recursively.
(4) Merge the lower envelopes.

The final merging stage consists of the following steps:

- Project each of the lower envelope boundaries (in our case, the trisector curves and bisector surface boundaries) onto the plane (in our case, the "plane" is not necessarily the xy-plane and we will discuss this projection later), resulting in two curve arrangements denoted *minimization diagrams* [25].
- Perform an overlay of the minimization diagrams. An overlay of two arrangements is the arrangement that is the result of combining both arrangements (see, for example, [27][Chapter 2]).
- For each face of the overlay arrangement, intersect the originating two surfaces (from each arrangement), project the intersection curves (if exist), and split the face accordingly, resulting in a final overlay arrangement in the plane.
- Each face of the arrangement now corresponds to at most two surfaces above it. The lower of the two faces, is the one we attach to the face of the resulting lower envelope arrangement. An inherent behavior of overlaying two projected lower envelopes is that neighboring faces might originate from the same surface. In such cases the neighboring faces are merged back.

The necessary operations for the lower envelope thus include:

(1) Intersections between bisector-surfaces (i.e., computing the trisectors).
(2) Projection of the intersection curves.
(3) The standard arrangement/overlay operations on the projected curves (as defined, e.g., by the CGAL arrangement package [26]).
(4) Comparing distances to the surfaces near an intersection curve (i.e., near the trisector).

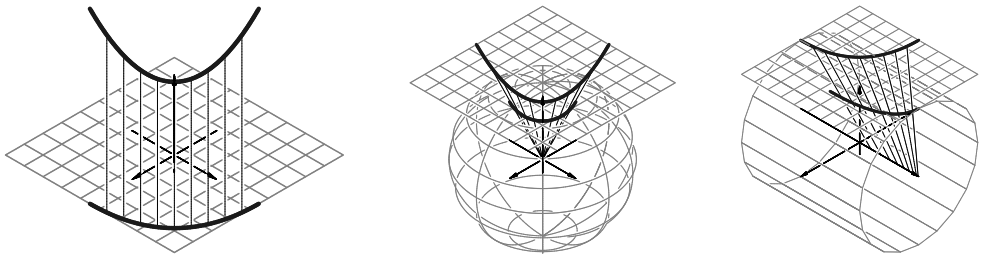
2.2 The Projection

There are three types of projections in our algorithm, depending on the type of the base objects in hand. Projecting the curves onto the base objects themselves can be difficult and produce non-rational curves. However, we can project them onto bounding planes that have a 1-1 correspondence with the base object. The projections are performed along the normal vectors of the base object. As we will see, the resulting projected curve is rational if the original curve was rational.

Let $\frac{(x(t),y(t),z(t))}{w(t)}$ be a trisector curve in $\mathbb{R}^3$ parameterized by some parameter t. If the base object is:

- **A plane (in canonical position):** the projected curve is simply $\frac{(x(t),y(t))}{w(t)}$.
- **A canonical sphere:** Consider the central projection of the curve onto the $z = 1$ plane, getting the projected curve $(\frac{x(t)}{z(t)}, \frac{y(t)}{z(t)}, 1)$, which is still a rational curve. In order to avoid problems with infinity, we project the curves in the sphere case onto the unit cube, i.e., we clip the curves projected onto $z = 1$ at $x = \pm 1$ and $y = \pm 1$, and perform similar projections and clippings to each of the remaining faces of the unit cube.
- **A canonical cylinder (where the cylinder's axis is the y-axis):** Project the curves onto $z = 1$. However, the (cylindrical) projection is done along the cylinder's normal vectors, i.e., the y-coordinate remains the same; therefore it is different from the one used for a base sphere. The projected curve is thus, $(\frac{x(t)}{z(t)}, \frac{y(t)}{w(t)}, 1)$ or $\frac{(w(t)x(t),z(t)y(t),z(t)w(t))}{z(t)w(t)}$, which is still a rational curve (although of a higher degree). The bounding planes for the canonical cylinder case will be $z = \pm 1$, $x = \pm 1$.

In summary, we use an orthogonal projection for a plane base object, a spherical (central) projection for a sphere and a cylindrical projection for a cylinder (see Figure 3).



(a) (b) (c)

Fig. 3. The three projections we use in our algorithm: (a) An orthogonal projection onto the xy-plane, (b) A central (spherical) projection onto the $z = 1$ plane, (c) A cylindrical projection onto the $z = 1$ plane.

3 Bisectors and Trisectors of Planes, Spheres and Cylinders

In the previous section, we described the general idea of our algorithm. In this section, we formulate the actual bisector surfaces (in Section 3.1) and trisector curves (in Section 3.2) that appear in the Voronoi cells of planes, spheres and cylinders.

3.1 Bisectors

Voronoi faces are bisectors of two objects, i.e., the surfaces that have equal (minimal) distance from two objects. A formulation of the bisectors can be found in [13,15]. These formulations, in general, use the parametric representation of the two objects to compute the bisectors. Another representation can be computed from the distance functions:

- **The plane distance function:**

$$r = \frac{(ax + by + cz + d)}{\sqrt{a^2 + b^2 + c^2}}, \tag{1}$$

  where (a, b, c, d) are the plane's coefficients, (x, y, z) is the Euclidean point and r is the distance of the point from the plane. Equation (1) returns a signed distance, and therefore care should be taken to the orientation of the plane.
- **The sphere distance function:**

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = (r + r_0)^2, \tag{2}$$

  where (x_0, y_0, z_0) is the sphere's center point, r_0 is the sphere's radius and (x, y, z) is the Euclidean point, with r the distance of the point from the sphere's surface.

- **The cylinder distance function:** The cylinder distance function is derived from the point-line distance function [2] in $\mathbb{R}^3$, (hereafter bold letters will denote vector values):

$$(r + r_0)^2 = \langle \boldsymbol{p} - \boldsymbol{c_0}, \boldsymbol{p} - \boldsymbol{c_0} \rangle - \frac{\langle \boldsymbol{p} - \boldsymbol{c_0}, \boldsymbol{d_0} \rangle^2}{\langle \boldsymbol{d_0}, \boldsymbol{d_0} \rangle}, \tag{3}$$

  where $\boldsymbol{p}$ is the point (x, y, z), $\boldsymbol{d_0} = (d_{0x}, d_{0y}, d_{0z})$ is the direction of the cylinder's axis and $\boldsymbol{c_0}$ is a point (c_{0x}, c_{0y}, c_{0z}) on the cylinder's axis (i.e., the parametric line $\boldsymbol{c_0} + t\boldsymbol{d_0}$ is axis of the cylinder). r_0 is the cylinder's radius.

The bisector surfaces are the simultaneous solutions of two distance equations. In most cases (four out of six cases, see Table 1), eliminating r is trivial and results in a quadric surface. However, in two of the six cases, namely the sphere-cylinder bisector and cylinder-cylinder bisector, r cannot be easily eliminated unless the radius r_0 is identical for both objects. Thus, in general, the bisector is not a quadric surface. From Equations (2) and (3), it can be seen that, by subtracting the distance equations, r can be formulated as a quadratic function in x, y, and z. Hence, the sphere-cylinder and cylinder-cylinder bisectors are, in the general case, degree-4 implicit surfaces.

Plane-Plane	Plane.
Plane-Sphere	Quadric - Paraboloid.
Plane-Cylinder	Quadric - Elliptic cone [13] [3].
Sphere-Sphere	Quadric - Hyperboloid of two sheets.
Sphere-Cylinder	Degree-4 implicit surface (admits a rational representation [13]).
Cylinder-Cylinder	Degree-4 implicit surface (admits a rational representation [15]).

Table 1
Bisectors between planes, spheres and cylinders.

Table 1 summarizes the different combinations of bisectors between planes, spheres and cylinders. Note that all bisectors between planes, spheres, and cylinders admit a rational parameterization [13,15].

[2] See for example, http://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html

[3] We denote by *elliptic cone* a right elliptic cone, i.e., a cone whose elliptical base is perpendicular to the cone's axis.

3.2 Trisectors

The Voronoi edges are trisectors of three objects, i.e., the curves in $\mathbb{R}^3$ that have an equal distance from three objects. In this section, we analyze the different cases of trisectors in $\mathbb{R}^3$ between planes, spheres, and cylinders, and prove that of the ten possible trisector combinations, five are in fact conic section curves (see also Table 2).

Plane-Plane-Plane	*Line.*
Plane-Plane-Sphere	*Ellipse or parabola.*
Plane-Plane-Cylinder	*Conic section.*
Plane-Sphere-Sphere	*Ellipse or parabola.*
Plane-Sphere-Cylinder	Intersection of an elliptic cone and a paraboloid.
Plane-Cylinder-Cylinder	Intersection of two elliptic cones.
Sphere-Sphere-Sphere	*Conic section.*
Sphere-Sphere-Cylinder	Intersection of a hyperboloid and a degree-4 implicit surface.
Sphere-Cylinder-Cylinder	Intersection of two degree-4 implicit surfaces.
Cylinder-Cylinder-Cylinder	Intersection of two degree-4 implicit surfaces.

Table 2
Trisectors between planes, spheres and cylinders. The conics are in *italics*.

3.2.1 Trisectors of Three Spheres

Kim et al. [18] and Will [19] have already proven that the trisector of three spheres is a conic section. We repeat their result for completeness.

Lemma 1 *The trisectors of three spheres in* $\mathbb{R}^3$ *are conic sections [18,19].*

Proof: Following [18], the fact that the trisector of three spheres is a conic section can be proved using the definitions of Dupin cyclides [28,29]. The Dupin cyclide is the envelope of spheres simultaneously tangent to three fixed spheres. The trisector is the locus of centers of spheres simultaneously tangent to three fixed spheres. Therefore, the trisector is the locus of the sphere centers constructing the Dupin cyclide (or the "spine" of the Dupin cyclide). However, a different definition of the Dupin cyclide is "The envelope of spheres with centers on a conic and touching a sphere" [29]. Therefore, the spine of the Dupin cyclide is a conic section i.e., the trisector is a conic section. ■

A different, constructive, way to prove Lemma 1, which also follows [18], is by solving the three distance equations in the following manner:

Proof: Given three spheres, we can always shrink them by the smallest radius and remain with a point and two spheres of smaller radius. Without loss of generality we can assume the point is at the origin (if not, we translate the whole system). The three distance equations are therefore:

$$\begin{aligned} &x^2 + y^2 + z^2 = r^2, \\ &(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 = (r + r_1)^2, \\ &(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = (r + r_2)^2, \end{aligned} \tag{4}$$

where (x_i, y_i, z_i) is the center of the ith sphere and r_i is its radius. Plugging the first equation into the second and the third, we end up with two linear equations in x, y, z, and r. Eliminating r we get the plane equation:

$$\begin{aligned} &2\left(\frac{x_1}{r_1} - \frac{x_2}{r_2}\right)x + 2\left(\frac{y_1}{r_1} - \frac{y_2}{r_2}\right)y + 2\left(\frac{z_1}{r_1} - \frac{z_2}{r_2}\right)z + \\ &\left(\frac{{x_2}^2 + {y_2}^2 + {z_2}^2}{r_2} - \frac{{x_1}^2 + {y_1}^2 + {z_1}^2}{r_1} + r_2 - r_1\right) \\ &= 0. \end{aligned} \tag{5}$$

Thus, the trisector curve is on the resulting plane, and intersecting this plane with one of the hyperboloid bisectors will give us a conic section. ■

By choosing an appropriate coordinate system on the plane, $\mathcal{P}$, described by Equation (5), i.e., two vectors $\boldsymbol{u} = (u_x, u_y, u_z) \in \mathcal{P}$ and $\boldsymbol{v} = (v_x, v_y, v_z) \in \mathcal{P}$, and a point $(x_0, y_0, z_0) \in \mathcal{P}$, we can represent $\mathcal{P}$ in parametric form as:

$$\begin{aligned} x &= x(u, v) = x_0 + u_x u + v_x v, \\ y &= y(u, v) = y_0 + u_y u + v_y v, \\ z &= z(u, v) = z_0 + u_z u + v_z v. \end{aligned} \tag{6}$$

Plugging these parametric plane equations into an implicit representation of the hyperboloid will give us an implicit form of the conic section in the plane parametric uv-space of Equation (6). That is, we will get an implicit quadratic polynomial, $Q(u, v) = 0$, which can then be represented as a parametric conic section in uv-space $\boldsymbol{c}_{uv}(t) = (u(t), v(t))$ (see, for example, [30][Chapter 3] on transforming conic sections from implicit to parametric form). Assigning $(u(t), v(t))$ into Equation (6) results in the parametric trisector space curve:

$$x(t) = x(u(t), v(t)),$$
$$y(t) = y(u(t), v(t)),$$
$$z(t) = z(u(t), v(t)).$$

3.2.2 Trisector of Two Spheres and a Plane

Similarly, the following lemma holds for the plane-sphere-sphere trisector:

Lemma 2 *The trisectors of two spheres and a plane is either an ellipse or a parabola.*

Proof: Given the plane equation $ax + by + cz + d = 0$ we have the equations:

$$\begin{aligned} &\frac{ax + by + cz + d}{\sqrt{a^2 + b^2 + c^2}} = r, \\ &(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 = (r + r_1)^2, \\ &(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = (r + r_2)^2. \end{aligned} \tag{7}$$

Subtracting the left and right sides of the last two equations, we eliminate x^2, y^2, z^2, and r^2, and are left with two linear equations in x, y, z, and r. We can thus eliminate r and remain with a plane equation, which means the trisector is planar. Intersecting the plane equation with the plane-sphere bisector, which is a paraboloid (see Section 3.1), yields a conic section, which is the trisector. Since the intersection between a plane and a paraboloid is either a parabola (when the intersecting plane is parallel to the paraboloid axis) or an ellipse (when the intersecting plane intersects the paraboloid axis), the lemma is proven. ■

3.2.3 Trisector of a Sphere and Two Planes

For the plane-plane-sphere trisector, the following lemma holds:

Lemma 3 *The trisectors of two planes and a sphere is an ellipse or a parabola.*

Proof: This is immediate since the bisector of two planes is a plane and the bisector of a plane and a sphere is a paraboloid. Thus, the trisector is an intersection of a plane and a paraboloid, which is an ellipse or a parabola. ■

3.2.4 Trisector of a Cylinder and Two Planes

For the plane-plane-cylinder trisector, the following lemma holds:

Lemma 4 *The trisectors of two planes and a cylinder is a conic section.*

Proof: The bisector of two planes is a plane and the bisector of a plane and a cylinder is an elliptic cone (see [13] and Section 3.1 above). Thus, the trisector is an intersection of a plane and a quadric, which is a conic section. ■

3.2.5 Combinations of All Trisectors

Table 2 summarizes all possible combinations of trisectors between planes, spheres and cylinders.

Of the ten combinations, five are known to be conic sections (or a line) and can therefore be represented as rational curves. Of the other five, two (plane-cylinder-cylinder and plane-sphere-cylinder) are intersections of (rational) quadric surfaces. Intersection curves of quadric surfaces do not, in general, have a rational parameterization [31]. The other three (sphere-sphere-cylinder, sphere-cylinder-cylinder and cylinder-cylinder-cylinder) are general intersections of degree-four implicit surfaces. Intersection curves of degree-four implicit surfaces also do not admit a rational parameterization, in general.

For the trisector cases that do not admit a rational parameterization, we approximate the trisectors to a given precision. For example, by intersecting the two rational bisector surfaces, or by solving the distance Equations (1)-(3) directly using the IRIT multivariate solver [23,32]. Then, we fit a spline through them (see discussion in Section 4.3), and use this approximation, in the algorithm.

4 Implementing the Lower Envelope Algorithm

We now describe the implementation of our algorithm that is based on the IRIT solid modeling system [23] and on CGAL's 3D lower envelope implementation [24,25], which is based on the CGAL arrangement package [26].

This section starts by describing CGAL's arrangement package and how it is interfaced with IRIT. Then, in Section 4.2, we discuss the lower envelope implementation and how it is used in the implementation of our algorithm. Finally, Section 4.3 discusses our implementation of trisectors that are not conic sections.

As discussed in Section 2.1, in computing the 3D lower envelopes, we mainly compute overlays of the 2D projections of the boundaries of smaller lower envelopes. Thus, 2D curve arrangements are the basic building blocks of the 3D lower envelope algorithm. In this section, we describe the curve arrangement package of CGAL and how we used it in our implementation.

CGAL, the Computational Geometry Algorithms Library, is a collaborative effort of several sites in Europe and Israel, aiming to provide a generic and robust, yet efficient, implementation of widely used geometric data structures and algorithms.

CGAL's arrangements package [26] is a generic data structure supporting various operations. Given a set of (not necessarily x-monotone and possibly intersecting) curves, it constructs the arrangement data structure using the sweep-line or incremental construction algorithm. The arrangement data structure enables easy traversal over the edges, faces and vertices of the arrangement and efficient locating of points in the arrangement. For a full survey of the operations supported by the package refer to the CGAL manual [26].

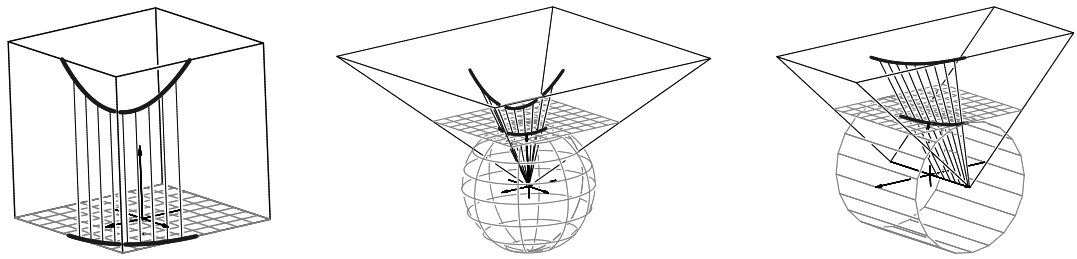
CGAL uses the generic programming paradigm [33] in order to achieve flexibility and genericity. Every algorithm and data structure in CGAL is parameterized by a so-called geometric traits class [34], in which the basic geometric functionality is implemented. Thus, users can employ CGAL's algorithm's with their own objects without having to re-implement the algorithm. All that is needed is to implement the relevant traits class and parameterize the algorithm with it. For further information on the design and implementation of CGAL, and on the usage of generic programming within it see, for example, [35].

The arrangement traits specification enables arrangements of different types of curves. The main operations required by an implementation of an arrangement traits class are: comparing points by their coordinates, determining whether a point is above, below or on a given x-monotone curve, computing the intersections between two curves and comparing two intersecting curves directly to the right of their intersection point. The full list of requirements can be found in the arrangement documentation.

In order to interface between the CGAL arrangement package and IRIT, this work implements a traits class for points and curves in IRIT. In this class, the aforementioned basic geometric functions are implemented using the functions provided by the IRIT libraries.

CGAL's 3D lower envelope package [24,25], implements the divide and conquer 3D lower envelope algorithm sketched in Section 2.1. It is based on the arrangement package described in Section 4.1 above and extends its traits mechanism. The lower envelope implementation can handle all types of input surfaces, provided certain functionalities on them are given by the user. The 2D curve arrangement functionality is provided by the IRIT traits class for free-form curves described above. Except for the regular functionalities on curves needed by CGAL's arrangement package, the 3D lower envelope package also requires the following functionalities (function names are following CGAL style):

- *Surface* and *Xy_monotone_surface* types: These are the bisector surfaces. The surfaces hold references to their originating CSG primitives as well as the rational free-form parametric representation (as computed by IRIT). This enables one to use both the parametric form and the implicit form of the bisectors. Interested only in the portions of the surfaces that correspond to the relevant trimmed bisectors, it is achieved either by trimming unwanted branches when possible (e.g., the unwanted branch of the two-sheet hyperboloid that is the sphere-sphere bisector) or by implicitly referring to the branch we are interested in. For example, if a ray from the origin intersects two branches, we consider only the intersection point that corresponds to the trimmed bisector. This is done by computing the minimal distance from the point to the two objects and purging away the points that do not comply to the minimal distance constraints.
- *Construct_projected_boundary_curves*: This construction is implemented by intersecting the bisector surface with a 3D bounding frustum, and projecting the resulting curves onto the 2D projection plane (i.e., the $z = 0$ plane for a base object that is a plane, and the $z = 1$ plane for the sphere and cylinder base objects). The frustum's shape depends on the projection we use (i.e., it depends on the base object). For a plane it is a box, for a sphere it is a square pyramid (with its apex at the origin) and for a cylinder it is a prismatic wedge (see Figure 4).
- *Construct_projected_intersections*: The intersections between the bisector surfaces are, in fact, the trisector curves. We compute them in the way described in Section 3.2 and project them onto the 2D projection plane(s).
- *Compare_distance_to_envelope*: This function compares the height of two surfaces above a given point. In our context, of a projective lower envelope, we cast a ray through the 2D point, along the projection direction. That is, we cast a vertical ray through the point when the base object is a plane, a ray from the origin through the point when the base object is a sphere and for a cylinder, a ray from the orthogonal projection of the point onto the cylinder axis. The distance from the ray-surface intersection point is



(a) (b) (c)

Fig. 4. The three projections with their corresponding bounding frustums: (a) A bounding box for a base object that is a plane, (b) A square pyramid for a base object that is a sphere, (c) A wedge for a base object that is a cylinder. See also Figure 3.

the "height" in our context (the larger the distance in the surface normal direction, the higher the surface is above the point).

- *Compare_distance_to_envelope_above*:
  This function compares the height of two surfaces above a point, which is an ϵ distance in the y-direction away from the projected intersection curve. That is, it compares the slope of the surfaces near their intersection curve. We handle this by constructing the tangent planes to the surfaces at a point on the intersection curve and comparing the slopes of the tangent planes.

4.3 Approximation of Trisectors that are not Conic Sections

Non conic trisectors are computed using the multivariate solver [32]. They can be computed in several ways: One way is to follow the general path for parametric bisectors [32] and construct distance and orthogonality constraints for the three objects using their parametric domains. This amounts to eight equations in nine unknowns, namely $(u_0, v_0, u_1, v_1, u_2, v_2, x, y, z)$, where (u_i, v_i) are the parameters of the the CSG primitive surfaces and (x, y, z) are the points on the trisector curve in $\mathbb{R}^3$. The eight equations are formed out of two distance equality equations:

$$||(x, y, z) - \boldsymbol{S_0}||^2 = ||(x, y, z) - \boldsymbol{S_1}||^2,$$

$$||(x, y, z) - \boldsymbol{S_0}||^2 = ||(x, y, z) - \boldsymbol{S_2}||^2,$$

and six orthogonality constraints (three in u and three in v):

$$\left\langle \frac{\partial \boldsymbol{S_i}(u_i, v_i)}{\partial u_i}, (x, y, z) - \boldsymbol{S_i}(u_i, v_i) \right\rangle = 0,$$

$$\left\langle \frac{\partial \boldsymbol{S_i}(u_i, v_i)}{\partial v_i}, (x, y, z) - \boldsymbol{S_i}(u_i, v_i) \right\rangle = 0,$$

where $\boldsymbol{S_i}(u_i, v_i) = (x_i(u_i, v_i), y_i(u_i, v_i), z_i(u_i, v_i))$ is the ith CSG primitive. The disadvantages of such a large system of equations are clear.

A more feasible way is to use the parametric equations of the *bisector*, which results in three equations in four variables (u_1, v_1, v_2, v_2) where (u_i, v_i) are the parameters of the bisector surfaces. The equations are now:

$$\begin{aligned} x_1(u_1, v_1) &= x_2(u_2, v_2), \\ y_1(u_1, v_1) &= y_2(u_2, v_2), \\ z_1(u_1, v_1) &= z_2(u_2, v_2), \end{aligned}$$

where $(x_i(u_i, v_i), y_i(u_i, v_i), z_i(u_i, v_i))$, $i = 1, 2$, represent the ith bisector surface. The result is a univariate solution i.e., a set of points (u_1, v_1, u_2, v_2) in the uv-domains that corresponds to points on the trisector. These points can be interpolated and assigned into the surface equations to result in the 3D Euclidean curves.

The problem with the second method is the high degree of the equations in the parametric domain. A better way for primitive surfaces is to work directly using the Euclidean distance functions whenever available (e.g., the ones described in Section 3.1) within the prescribed bounding box.

For example, the sphere-sphere-cylinder trisector is the solution of the following equations:

$$\begin{aligned} &\langle \boldsymbol{p} - \boldsymbol{c_1}, \boldsymbol{p} - \boldsymbol{c_1} \rangle - (r + r_1)^2 = 0, \\ &\langle \boldsymbol{p} - \boldsymbol{c_2}, \boldsymbol{p} - \boldsymbol{c_2} \rangle - (r + r_2)^2 = 0, \\ &\langle \boldsymbol{p} - \boldsymbol{c_3}, \boldsymbol{p} - \boldsymbol{c_3} \rangle - \frac{\langle \boldsymbol{p} - \boldsymbol{c_3}, \boldsymbol{d_3} \rangle^2}{\langle \boldsymbol{d_3}, \boldsymbol{d_3} \rangle} - (r + r_3)^2 = 0, \end{aligned} \tag{8}$$

where $\boldsymbol{p} = (x, y, z)$ and r is the distance to the CSG primitives, and $\boldsymbol{c_i}, i = 1, 2$, is the center of the ith sphere and r_i is its radius. $\boldsymbol{c_3}$ and $\boldsymbol{d_3}$ are the point and direction of the cylinder's axis and r_3 is the cylinder's radius.

Equation (8) amounts to three distance equations (of degree 2) in four unknowns (x, y, z, r). The result of Equation (8) is a set of points on the trisector with their corresponding distance value r.

Furthermore, we can eliminate r from these distance functions, resulting in two equations in three unknowns (x, y, z). In this case, however, the maximal degree of the equations is 4 (see Section 3.1).

5 Experimental Results

As described in Section 4, we implemented the proposed algorithm for extracting the Voronoi cell of a set of planes, spheres and cylinders in $\mathbb{R}^3$ using the IRIT modeling environment combined with the CGAL 3D lower envelope package. In this section, results from some test cases are presented. The input consists of a base object, a set of planes, spheres and cylinders, and a bounding box for clipping infinite bisectors and trisectors. The input CSG primitives are not approximated and neither are the untrimmed bisector surfaces. Whenever the trisectors can be represented as a conic section (see Table 2), we do so. When the trisectors cannot be represented as a conic section, we use the IRIT multivariate solver to solve the three constraints in four unknowns, as described in Section 4.3.

Computation of the presented Voronoi cells took from a fraction of a second to several seconds. Slower times were recorded for inputs that did not have an analytical representation and hence required approximations, resulting in trisector curves containing hundreds of sampled points. All the experiments were carried out on an Intel Quad CPU Q6600 2.4 GHz computer (single processor use) with 2GB RAM using the IRIT environment and the CGAL library.

It should be noted that the arrangements and lower envelope packages of CGAL adopt, as does CGAL in general, the *exact computation* paradigm [36]. Namely, the algorithms are certified to produce correct results only if the arithmetic used is exact. If the arithmetic is not exact, the algorithms is not certified and can produce incorrect results or fail due to an inconsistent state. CGAL does not make an effort to recover from possible inconsistent states that arise from non-exact arithmetic.

We implemented our algorithm using the IRIT solid modeling environment, which is implemented in floating point arithmetic using tolerances. Such arithmetic can fail to produce correct results in degenerate or near-degenerate cases. A more robust implementation could have used interval arithmetic or implemented a more careful implementation of the lower envelope algorithm, which takes into account possible inconsistencies due to floating point arithmetic (see Held [37], for such a sophisticated implementation of Voronoi diagrams of line segments in $\mathbb{R}^2$). Still, even such implementations cannot guarantee the robustness of the algorithm for all inputs. In any case, such implementations are beyond the scope of this work.

In our context, exact arithmetic was not an option since some of the operations used in the algorithm cannot yet be implemented using current state-of-the-art exact arithmetic. In Section 7 we discuss possible improvements to the

algorithm, which may enable usage of exact arithmetic in the future.

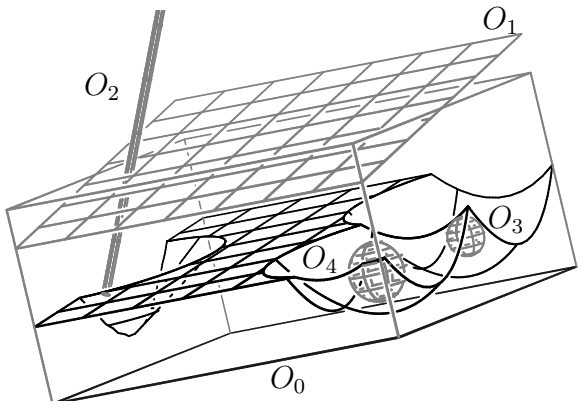


Fig. 5. The Voronoi cell (in black) of a base object that is a plane (bottom of the bounding box) with a plane, O_1, a cylinder, O_2, and two spheres, O_3 and O_4. Computation of the algorithm took 0.76 seconds.

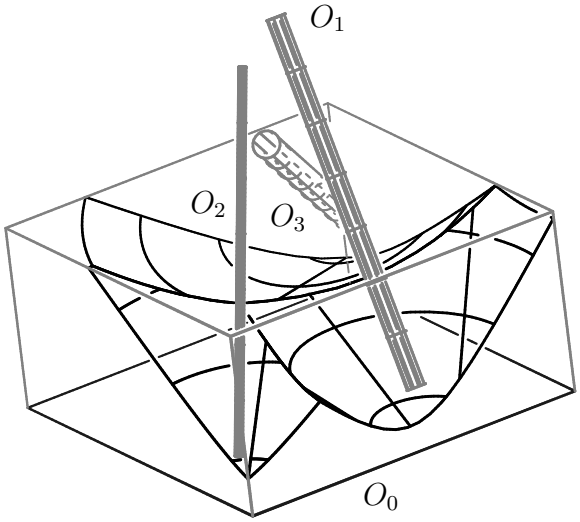


Fig. 6. The Voronoi cell (in black) of a base object that is a plane with three cylinders O_1, O_2, O_3 (in gray). Computation of the algorithm took 14.8 seconds.

Figure 5 shows the Voronoi cell of a plane, a cylinder, and two spheres with a base object that is a plane. The face of the Voronoi cell corresponding to the plane is a trimmed plane, the face corresponding to the cylinder is a trimmed elliptic cone, and the faces corresponding to the spheres are trimmed paraboloids, all merged together into one VC. Computing the Voronoi cell for this input took 0.76 seconds.

Figure 6 shows the Voronoi cell of a base plane, where the input geometry consists of three cylinders. The trisectors in this example are plane-cylinder-cylinder trisectors, which do not admit a rational parameterization, and therefore they are all approximated. Computing the Voronoi cell for this input took

14.8 seconds.

Figure 7 demonstrates a case where the base object is a sphere. The input primitives are three spheres. In order to demonstrate the way we compute the cell for a sphere, the figure shows only 1/6 of the Voronoi cell that corresponds to the upper face of the unit cube on which the trisectors are projected.

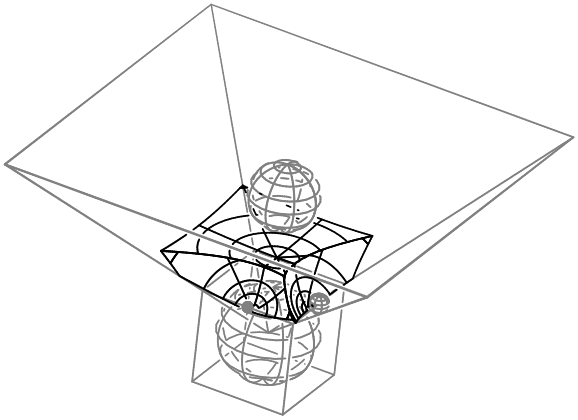


Fig. 7. One sixth of the Voronoi cell (in black) of a base object that is a sphere with three other spheres (in gray). The bounding pyramid corresponding to the upper face of the cube is also displayed.

The algorithm is demonstrated for the whole VC in Figure 8, where six spheres surround the base object, which is the central sphere. In Figure 8(b), the projections of the VC curves onto the unit cube bounding the base sphere object are visualized. The curves are central spherical projections of sphere-sphere-sphere trisectors, which are conic sections (by Lemma 1) and are therefore conic sections themselves. Computing the Voronoi cell for this input took 2.9 seconds.

In Figure 9, another Voronoi cell for a base object that is a sphere is shown, with two (infinite) planes and two spheres surrounding it. The surfaces in this VC are paraboloids (plane-sphere bisectors) and hyperboloids (sphere-sphere bisectors). The trisector curves are conic sections – plane-plane-sphere, plane-sphere-sphere and sphere-sphere-sphere trisectors. Computing the Voronoi cell for this input took 0.65 seconds.

Figure 10, shows a Voronoi cell for a base object that is a cylinder, with two (infinite) planes surrounding it and a sphere. The cylinder is the canonical cylinder along the y-axis, clipped at $y = \pm 1$. The surfaces in this VC also contain a non-quadric bisector – the sphere-cylinder bisector, which is a degree-4 implicit surface. The trisector curves contain conic sections (plane-plane-cylinder) and non-conic section curves (plane-sphere-cylinder trisectors). In Figure 10(b) the cylindrical projections of these curves onto the bounding planes are shown. Computing the Voronoi cell for this input took 4.1 seconds.

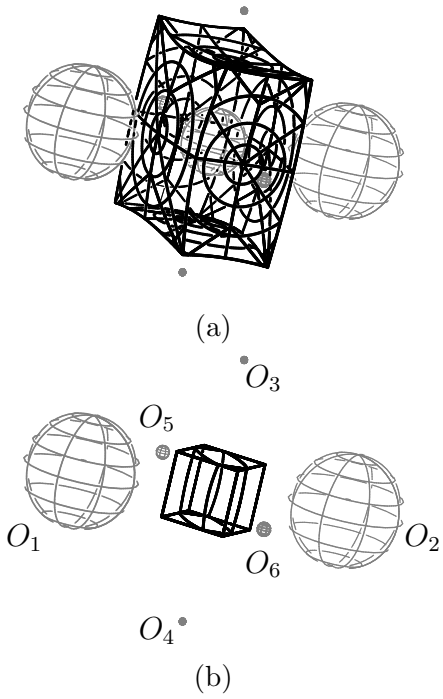


Fig. 8. A Voronoi cell of a base object that is a sphere with six surrounding spheres: (a) The full VC (in black) with the original input spheres (in gray), (b) The spherical projections of the VC curves (in black) onto the unit cube bounding the base object. Computation of the algorithm took 2.9 seconds. Compare with Figure 7.

In Table 3 we show the running times of computing the Voronoi cell as a function of the number of objects. The base object is a plane and the objects are spheres of different radii (see Figure 12 for the final configuration). The running time for these inputs (which do not contain approximated trisectors) corresponds roughly to $O(nlog(n))$ (see Figure 11), which is in accordance with the expected asymptotic runtime of the lower envelope algorithm.

Table 3
Constructing the Voronoi cell of a base object which is a plane, for a varying number of spheres (see also Figure 11 and Figure 12). Times are given in seconds.

Objects Num	4	6	8	10	12	16	18	20	22	24
Time	0.13	0.24	0.38	0.53	0.71	1.05	1.16	1.36	1.56	1.72

6 Extending the Algorithm to Other Objects

There are several requirements that need to be fulfilled in order to extend our work to VC computation for other types of objects. Our algorithm is basically a lower envelope algorithm of the bisector surfaces. In theory, the lower envelope should be computed by projecting the curves onto the base

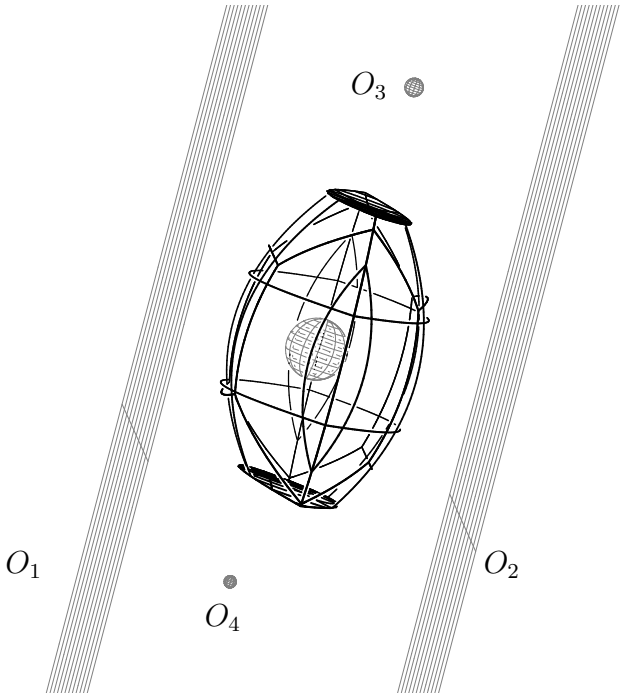


Fig. 9. The full VC (in black) of a base object that is a sphere with two planes, O_1 and O_2, and two spheres, O_3 and O_4 (in gray) surrounding it. Computation of the algorithm took 0.65 seconds.

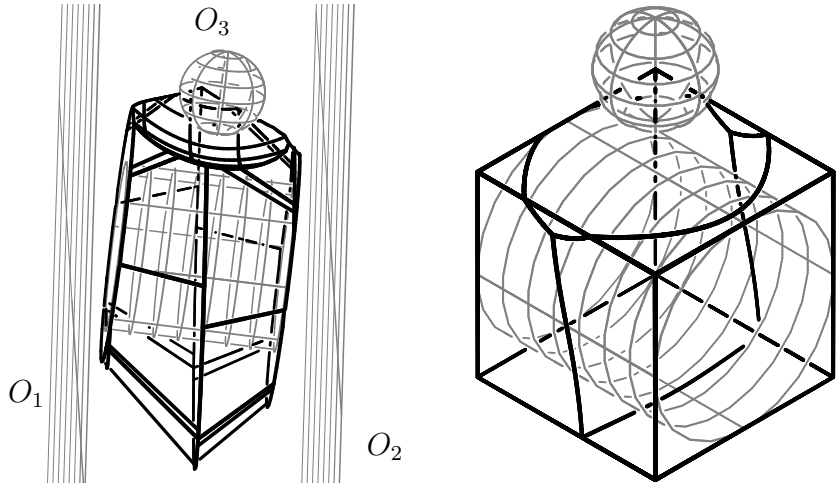


Fig. 10. A Voronoi cell of a base object that is a cylinder with two planes, O_1 and O_2, and a sphere, O_3: (a) The full VC (in black) with the original input planes and sphere (in gray) is shown, (b) The cylindrical projections of the VC curves (in black) onto a box bounding the base object is presented. The two planes were removed from (b) for a better visualization. Computation of the algorithm took 4.1 seconds.

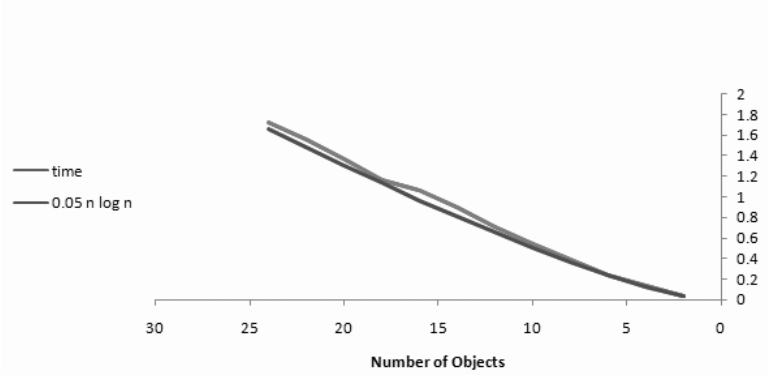


Fig. 11. The running time of the algorithm as a function of the number of input objects. The dark gray line is the running time in seconds and the light gray line is the graph of the function $0.05nlog(n)$ (see also Table 3 and Figure 12).

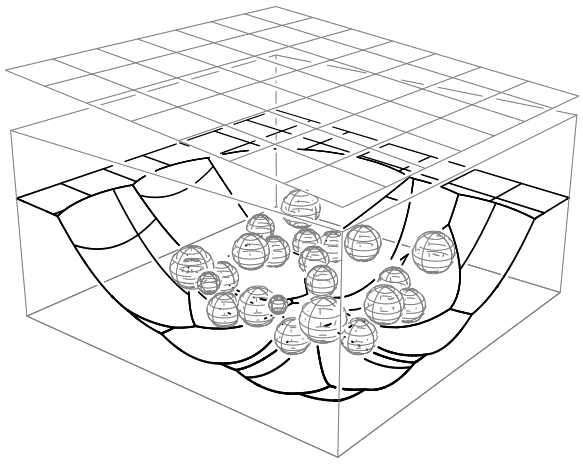


Fig. 12. The full VC (in black) of a base object that is a plane with twenty four sphere objects (see also Table 3 and Figure 11).

object. Instead, we compute it by a 1-1 projection of the curves onto planes, which have a 1-1 correspondence with the base object.

Therefore, the lower envelope algorithm can be applied and the VC can be computed for any objects that meet the following general requirements:

(1) A "projection plane" (or a set of "projection planes") that has a 1-1 correspondence with the base objects and with the bisector surfaces.
(2) A representation of the bisector surfaces that provides the following functionality:
   - Given a point on the projection plane, one should be able to compute the corresponding point on the bisector (the inverse of the projection onto the plane). More precisely, given two points on two different bisectors corresponding to the same point on the projection plane, we need to be able to determine which of the points is closer to the base object.
   - Similarly, one should be able to compute the normal of the bisector at

a given point or at least compare normals of two bisectors along their mutual trisector.

(3) A representation of the trisector curves that can be projected onto the projection plane. The projected trisectors should support the 2D arrangement functionalities, as described in Section 4.1.

In principle, the presented algorithm can be extended to cones as well. The projection for a cone, however, is more complicated. As in the cylinder case, the projection for cones would be onto the cone axis but with an angle, following the cone's apex angle. The distance function for a cone is also more complicated than the functions described in Section 3.1, and is also dependant on the apex angle of the cone.

Assume cone $\mathcal{C}$ is positioned so that its axis is along the y-axis, its apex is the origin, and let θ be the apex angle of the cone. Without loss of generality, we assume that the cone extends towards the positive y-direction only. Then, the radius r_y of the cone at a given y-coordinate [4] is $r_y = y\tan(\theta)$. The distance function between a point (x, y, z) and the cone (i.e., the distance to its orthogonal projection onto the cone) is thus:

$$r = cos(\theta)(\sqrt{x^2 + z^2} - r_y) = \cos(\theta)\sqrt{x^2 + z^2} - \sin(\theta)y. \tag{9}$$

Using basic trigonometry, we can derive the following projection onto the plane $z = 1$. The "cone projection" of a point (x, y, z) onto the cone axis (the y-axis in this example) can be developed as follows:

$$\tan(\theta) = \frac{\Delta y}{\sqrt{x^2 + z^2}},$$

where Δy is the shift of the projected point in the y-direction. This can be seen if we look at the plane going through (x, y, z) and the cone axis and project the point orthogonally to the lines of the cone. Therefore, $\Delta y = \tan(\theta)\sqrt{x^2 + z^2}$. From triangle similarities we have that $\Delta y_{z=1}$ - the shift in the y-coordinate when projecting onto the $z = 1$ plane is:

$$\frac{\Delta y}{\Delta y_{z=1}} = \frac{z}{z - 1}.$$

We therefore end up with:

$$\Delta y_{z=1} = \tan(\theta)\sqrt{x^2 + z^2}\left(1 - \frac{1}{z}\right).$$

[4] We use the convention that r_y is negative for a negative y, this will simplify our following computations.

For a curve $(x(t), y(t), z(t))$, the final "cone projection" onto the $z = 1$ plane is thus:

$$\left(\frac{x(t)}{z(t)}, y(t) + \tan(\theta)\sqrt{x(t)^2 + z(t)^2} \left(1 - \frac{1}{z(t)} \right), 1 \right)$$

Unlike the other projections in this work, and because of the square-root, the cone-projection presented here is not rational, i.e., a projection of a rational curve will not yield a rational plane curve.

In order to extend this work to Voronoi cells for trimmed CSG surfaces, the lower envelope algorithm needs to be enhanced by representations of curve-curve and surface-curve bisectors. The trimming curves induce curve-curve bisectors (between pairs of trimming curves) and curve-surface bisectors (between surfaces and trimming curves). For example, consider a clipped cylinder and a sphere (similar to the ones in Figure 10(b)). Except for the (trimmed) sphere-cylinder bisector, there are also sphere-curve bisectors to be computed between the cylinder cross sections (which are circles) and the sphere.

CSG objects that intersect each other can also be handled in principle. If the base object is not intersected by any of the other objects, then our algorithm can work without modification even if the other objects are nested or interpenetrate. In the case where the base object is penetrated, the intersecting object needs to be divided into two trimmed CSG surfaces — a trimmed surface that is inside and a trimmed surface that is outside of the base object (or on the positive and negative sides for a base object that is a plane). The intersection curves between the base object and the penetrating object need to be computed and they are the trimming curves of the trimmed CSG object. Computing the Voronoi cell for interseecting objects then reduces to computing the Voronoi cells for the trimmed CSG surfaces, as described in the previous paragraph. We need to compute the Voronoi cell for the trimmed surfaces outside the base object and the Voronoi cell of the trimmed surfaces inside the base object.

The projection plane need not necessarily be a bounding plane in Euclidean space. The important thing is that there is a 1-1 correspondence between the plane and the bisectors and a 1-1 correspondence between the plane and the base object. Thus, the "projection plane" can be in parameter space of the base object. For example, for a space curve $c(t)$, a possible projection plane can be the $t\theta$-plane, where t is the parameter of c and θ is the angle between the vector $\boldsymbol{p} - c(t)$, $\boldsymbol{p}$ being a point on the bisector, and some vector $\boldsymbol{v}$. A possible choice for $\boldsymbol{v}$ can be the normal vector $\boldsymbol{N}$ of the curve's Frenet frame (see for example [30][Chapter 4]). Given a point $\boldsymbol{p} = (x, y, z)$ on a 3-space curve-curve bisector, the plane point (t, θ) can be computed by projecting $\boldsymbol{p}$ onto c and computing the angle of the vector between $\boldsymbol{p}$ and the projection point.

Singular points, where $\boldsymbol{N}$ is not defined, will have to be handled separately.

In a similar way, the Voronoi cell of a torus can be computed by projecting the trisectors onto the uv-parameter plane. The uv-parameter plane might also be used as the projection plane for computing Voronoi cells of general free-form surfaces. For this, a suitable representation of the bisectors and trisectors is needed. Elber and Kim [38] propose such a representation as a set of constraints on the parametric uv-domain.

A problem that arises in general surfaces is that for a non-convex surface the 1-1 correspondence between points on the bisector and the base object is lost. That is, there can be more than one projection of the bisector point onto the base object. This should be taken into consideration, for example by trimming the bisector or limiting the maximal distance supported, in a preprocessing step so that every point on the trimmed bisector will have exactly one projected point.

7 Conclusions and Future Work

We have presented an algorithm for computing the Voronoi cells of planes, spheres and cylinders in $\mathbb{R}^3$. The algorithm is based on a lower envelope algorithm of the bisector surfaces, which projects the trisector curves onto planes bounding the base object. We have analyzed the different bisectors and trisectors that can occur in the computation of the Voronoi cell. Our analysis showed that four of the six bisectors are quadric surfaces and five of the ten trisectors are conic section curves. We have implemented our algorithm using the IRIT library and the CGAL 3D lower envelope package.

In this work we compute the Voronoi cell of a single object. In principle, this enables us to compute the whole Voronoi diagram by computing the Voronoi cell for all objects. However, there is some redundancy in computing the Voronoi diagram in such a manner, since, for example, the bisectors are computed twice. There are probably better algorithms that compute the whole Voronoi diagram in a more efficient manner. Such algorithms are subject to future research.

Our algorithm can be improved for trisectors that do not have a rational representation. While intersection curves of quadric surfaces do not, in general, have a rational representation, they admit a homogeneous parameterization of the type [39,31]:

$$c(t) = a(t) + b(t)\sqrt{s(t)},$$

where $a(t)$ and $b(t)$ are four dimensional vectors of polynomials (i.e., in homogeneous space) and $s(t)$ is a scalar polynomial. We can use the parameterization above (with the square root) to project these trisectors and then use them in the algorithm as described above. This, however, requires that we develop special functions for the arrangement operations (intersection, splitting into x-monotone pieces etc.) that will overcome the difficulties of this representation due to the square root. Methods for quadric intersections such as the ones presented by Lazard et al. [40] and by Berberich et al. [41] might be used.

Another improvement can use the information on the CSG primitives for operations of the 2D arrangements. In the current implementation, the curves in $\mathbb{R}^2$ are computed explicitly as projections of the trisectors. When the trisectors do not admit a rational parameterization we work on approximating curves and perform the 2D operations (e.g., curve intersections) on the approximations. However, instead of using approximations we can use an implicit representation of the trisector curves and store references to their originating primitives. We can then perform the 2D arrangement operations using this knowledge. For example, vertices of the arrangement, which are projections of points that are equidistant from four primitives, can be computed by solving four distance equations in four unknowns (x, y, z, r).

Computing x-tangency points is also one of the basic operations in planar arrangements. In order to compute x-tangency points without approximating the curves, the derivatives of the trisector curves need to be computed using the originating primitives.

The normal direction of a bisector surface at a given point $\boldsymbol{p}$ on the surface, is equal to the difference vector between the two footpoints of $\boldsymbol{p}$ [15]. Therefore, the tangent direction of a trisector curve, at a point $\boldsymbol{p}$ on the curve, is the intersection of two tangent planes of two bisectors at $\boldsymbol{p}$. The tangent can thus be computed by a cross product of the bisector's normal vectors. Denote by $\boldsymbol{p}_i(x, y, z)$ the ith footpoint, which are rational functions depending on the primitive. Then, the trisector's tangent direction vector $\boldsymbol{T}(x, y, z)$ is given by:

$$\boldsymbol{T}(x, y, z) = (\boldsymbol{p}_1 - \boldsymbol{p}_2) \times (\boldsymbol{p}_1 - \boldsymbol{p}_3).$$

Thus, x-tangency points in the 2D arrangement can be computed by solving four equations in four unknowns - the three trisector distance equations and an additional constraint for the vertical tangency constraint. For example, for a simple plane projection this amounts to equating the x coordinate of $\boldsymbol{T}$ to zero.

The direct usage of such implicit representations for trisectors that do not admit an explicit representation, can have several advantages. First, it can work faster since approximations tend to have large representations, and the

implicit representations postpone the actual evaluation to when it is actually needed (lazy evaluation). Second, it reduces cascaded errors due to work with approximations. Third, unlike approximations, working with implicit representations opens (in some cases) the way to exact arithmetic. Implementation of such representations is left for future work.

Acknowledgment

This research was partly supported by the New York metropolitan research fund, Technion, in part by the Israeli Ministry of Science (grant No. 3-4642), and in part by the Israel Science Foundation (grant No. 346/07).

References

[1] C. G. Armstrong, Modeling requirements for finite element analysis, Computer Aided Design 26 (7) (1994) 573–578.

[2] G. S. Bajaj, E. Thiel, (3-4)-weighted skeleton decomposition for pattern representation and description, Pattern Recognition 27 (8) (1994) 1039–1049.

[3] H. Blum, Biological shape and visual science (Part I), Journal of Theoretical Biology 38 (1973) 205–287.

[4] D.-S. Kim, D. Kim, K. Sugihara, Voronoi diagram of a circle set from Voronoi diagram of a point set: II. Geometry., Computer Aided Geometric Design 18 (6) (2001) 563–585.

[5] I. Z. Emiris, E. P. Tsigaridas, G. M. Tzoumas, The predicates for the Voronoi diagram of ellipses., in: Proceedings of the 22nd ACM Symposium on Computational Geometry, 2006, pp. 227–236.

[6] I. Hanniel, M. Ramanathan, G. Elber, M.-S. Kim, Precise Voronoi cell extraction of freeform rational planar closed curves, in: V. Shapiro, L. Kobbelt (Eds.), Proceedings of the ACM Symposium on Solid and Physical Modeling, 2005, pp. 51–59.

[7] Y. Y. Zhang, P. S. P. Wang, Analytic comparison of thinning algorithms, Int. J. Pattern Recognit. Artif. Intell. 7 (1993) 1227–1246.

[8] M. Foskey, M. Lin, D. Manocha, Efficient computation of a simplified medial axis, in: Proceedings of the ACM Symposium on Solid Modeling and Applications, 2003, pp. 96–107.

[9] K. Hoff, T. Culver, J. Keyser, M. Lin, D. Manocha, Fast computation of generalized Voronoi diagrams using graphics hardware, in: Proceedings of ACM SIGGRAPH, 1999, pp. 277–286.

[10] N. Amenta, S. Choi, R. Kolluri, The power crust, in: Proceedings of the ACM Symposium on Solid Modeling and Applications, 2001, pp. 139–146.

[11] M. Etzion, S. A. Rappoport, Computing the Voronoi diagram of a 3-d polyhedron by separate computation of its symbolic and geometric parts, in: Proceedings of the ACM Symposium on Solid Modeling and Applications, 1999, pp. 167–178.

[12] D. Dutta, C. M. Hoffmann, On the skeleton of simple CSG objects, ASME Transactions: Journal of Mechanical Design 115 (1993) 87–94.

[13] G. Elber, M. S. Kim, Rational bisectors of CSG primitives, in: Proceedings of the ACM Symposium on Solid Modeling and Applications, 1999, pp. 159–166.

[14] G. Elber, M.-S. Kim, Computing rational bisectors, IEEE Computer Graphics and Applications, 19 (6).

[15] M. Peternell, Geometric properties of bisector surfaces, Graphical Models 62 (3) (2000) 202–236.

[16] M. Ramanathan, B. Gurumoorthy, Constructing medial axis transform of extruded and revolved 3D objects with free-form boundaries., Computer-Aided Design 37 (13) (2005) 1370–1387.

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

[18] D.-S. Kim, Y. Cho, D. Kim, Euclidean Voronoi diagram of 3D balls and its computation via tracing edges., Computer-Aided Design 37 (13) (2005) 1412–1424.

[19] H.-M. Will, Fast and efficient computation of additively weighted Voronoi cells for applications in molecular biology., in: Algorithm Theory - SWAT '98, 6th Scandinavian Workshop on Algorithm Theory, Vol. 1432 of LNCS, Springer-Verlag, 1998, pp. 310–321.

[20] J.-D. Boissonnat, M. Yvinec, Algorithmic geometry, Cambridge University Press, UK, 1998, translated from French by Hervé Brönnimann.
URL `http://cgal.inria.fr/Publications/1998/BY98`

[21] F. Aurenhammer, H. Imai, Geometric relations among Voronoi diagrams., in: STACS 87, 4th Annual Symposium on Theoretical Aspects of Computer Science, Vol. 247 of LNCS, Springer-Verlag, 1987, pp. 53–65.

[22] J.-D. Boissonnat, C. Delage, Convex hull and Voronoi diagram of additively weighted points., in: Proceedings of the 13th Annual European Symposium on Algorithms (ESA), Vol. 3669 of LNCS, Springer-Verlag, 2005, pp. 367–378.

[23] G. Elber, The IRIT 9.5 User Manual, http://www.cs.technion.ac.il/∼irit (2005).

[24] M. Meyerovitch, Robust, generic and efficient construction of envelopes of surfaces in three-dimensional space, M.Sc. thesis, Department of Computer Science, Tel Aviv University, Tel Aviv, Israel (May 2006).

[25] M. Meyerovitch, Robust, generic and efficient construction of envelopes of surfaces in three-dimensional space, in: Proceedings of the 14th Annual European Symposium on Algorithms (ESA), Vol. 4168 of LNCS, Springer-Verlag, 2006, pp. 792–803.

[26] R. Wein, E. Fogel, B. Zukerman, D. Halperin, 2D arrangements, in: CGAL Editorial Board (Ed.), CGAL-3.3 User and Reference Manual, 2007, http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/Arrangement_2/Chapter_main.html.

[27] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry, Algorithms and Applications, 2nd Edition, Springer, New York, 1998.

[28] D. Dutta, R. Martin, M. Pratt, Cyclides in surface and solid modeling, IEEE Computer Graphics and Applications 13 (1) (1993) 53–59.

[29] A. Hirst, Blending cones and planes by using cyclides of Dupin, Bulletin of the Institute of Mathematics and its Applications 26 (3) (1990) 41–46.

[30] E. Cohen, R. F. Riesenfeld, G. Elber, Geometric Modeling with Splines: An Introduction, A. K. Peters, 2001.

[31] W. Wang, R. N. Goldman, C. Tu, Enhancing Levin's method for computing quadric-surface intersections., Computer Aided Geometric Design 20 (7) (2003) 401–422.

[32] G. Elber, M.-S. Kim, Geometric constraint solver using multivariate rational spline functions, in: Proceedings of the ACM Symposium on Solid Modeling and Applications, 2001, pp. 1–10.

[33] M. H. Austern, Generic Programming and the STL: Using and Extending the C++ Standard Template Library, Addison-Wesley, 1998.

[34] N. Myers, A new and useful template technique: "Traits", in: S. B. Lippman (Ed.), C++ Gems, Vol. **5** of SIGS Reference Library, 1997, pp. 451–458.

[35] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, S. Schönherr, On the design of CGAL a computational geometry algorithms library, Software – Practice and Experience 30 (11) (2000) 1167–1202.
URL citeseer.ist.psu.edu/fabri99design.html

[36] C. K. Yap, T. Dubé, The exact computation paradigm, in: D. Z. Du, F. K. Hwang (Eds.), Computing in Euclidean Geometry, 2nd Edition, Vol. **1** of Lecture Notes Series on Computing, World Scientific, Singapore, 1995, pp. 452–492.

[37] M. Held, Voronoi diagram and offset curves of curvilinear polygons, Computer-Aided Design 30 (4) (1998) 287–300.

[38] G. Elber, M.-S. Kim, A computational model for nonrational bisector surfaces: Curve-surface and surface-surface bisectors, in: Proceedings of Geometric Modeling and Processing 2000 (GMP2000), 2000, pp. 364–372.

[39] J. Levin, A parametric algorithm for drawing pictures of solid objects composed of quadric surfaces, Commun. ACM 19 (10) (1976) 555–563.

[40] S. Lazard, L. Mariano Penaranda, S. Petitjean, Intersecting quadrics: An efficient and exact implementation, Computational Geometry, Theory and Applications 35 (1-2) (2006) 74–99.
URL http://hal.inria.fr/inria-00000380/en/

[41] E. Berberich, M. Hemmer, L. Kettner, E. Schömer, N. Wolpert, An exact, complete and efficient implementation for computing planar maps of quadric intersection curves, in: Proceedings of the 21st ACM Symposium on Computational Geometry, 2005, pp. 99–106.