COMPUTER-AIDED DESIGN & APPLICATIONS, 2017
VOL. 14, NO. 5, 572–581
http://dx.doi.org/10.1080/16864360.2016.1273576

Voronoi cells of non-general position spheres using the GPU

Zhongyin Hu [a], Xiang Li [a], Adarsh Krishnamurthy [b], Iddo Hanniel [c] and Sara McMains [a]

[a]University of California, Berkeley, USA; [b]Iowa State University, USA; [c]Technion, Israel Institute of Technology, Israel

ABSTRACT
We present a GPU algorithm for computing the Voronoi cells for a set of spheres in R^3. The algorithm is based on sampling rays from each sphere and finding the lower envelope of intersections of the rays with the sphere bisectors on the GPU. The presence of Voronoi vertices and edges is determined based on where neighboring rays intersect different bisectors. For accurate geometry, we present a numerical iteration approach to calculate the Voronoi vertices' locations within a user-defined tolerance. The algorithm robustly handles input in non-general position and large input sets of thousands of spheres.

KEYWORDS
Voronoi diagram; spheres; GPU; lower envelope

1. Introduction

The Voronoi diagram is a fundamental construct in computational geometry for partitioning space into regions that are closest to a set of points or surfaces, useful in a wide range of application domains. Many algorithms have been proposed for constructing Voronoi diagrams of lower order objects (points, line segments, and polygons) in R^3. However, Voronoi diagrams of higher order objects such as spheres in R^3 have not been explored as extensively, even though they have many applications. For example, to analyze pathways through molecular tunnels in proteins with their atoms represented by Van der Waals spheres, Voronoi diagrams of spheres are often used [11]. However, due to the lack of robust algorithms, many applications simplify the problem by assuming that all atoms have the same radius [11], or approximating larger atoms by collections of spheres with the radius of the smallest atom [3], so that existing codes for Voronoi diagrams of points can be used. Kim et al. [10] have successfully computed Voronoi diagrams of spheres in R^3, but their method is limited to input in general position.

In this paper, we present a GPU algorithm to compute Voronoi cells of spheres in R^3. Our main contributions include:

- A novel approach to compute Voronoi cells of spheres that exploits the parallelism of the GPU by shooting intersection rays at the bisector surfaces. We sample rays from each input sphere and find the lower envelope as points on Voronoi faces.
- Separation of the construction of the topology of the Voronoi cells from the calculation of the geometry. The topology is calculated directly from the face sample points.
- Accurate calculation of Voronoi vertices' geometry by using the samples to initialize Newton-Raphson iteration that can guarantee the accuracy of the vertices to be within a user-defined tolerance.
- Calculation of Voronoi cells of thousands of input spheres representing actual protein molecules (Fig. 1). Our algorithm is robust even for spheres not in general position, handling Voronoi vertices with degree greater than four.

2. Related work

The connection between Voronoi diagrams and lower envelopes is well known [4]. For example, the computation of the Voronoi diagram of a set of points (sites) in R^2 corresponds to the projection of the lower envelope of 45-degree cones whose apexes are located on the sites in the xy-plane. More generally, the Voronoi diagram of a set of curves in R^2 corresponds to the projection of the lower envelope of generalized cone surfaces that correspond to distance functions. This insight has led to several algorithms that construct the Voronoi diagram of objects in

CONTACT Sara McMains mcmains@me.berkeley.edu

R^2 by computing the lower envelope of their generalized cones (or distance functions) [2, 8, 13].

The connection between Voronoi diagrams and lower envelopes has inspired the use of GPU algorithms for Voronoi diagram computation, such as in the seminal work by Hoff et al. [9], in which the authors rendered a 3D polygonal mesh approximating the distance function, and used the z-buffer depth comparison to compute the lower envelope of these meshes and hence the Voronoi diagram. This approach has been far more successful for computing Voronoi diagrams in R^2 than in R^3 up to now, however, because in R^3 the Euclidean space lower envelope requires projections from surfaces in R^4 onto R^3.

Hanniel et al. computed the Voronoi cells of a set of CSG primitives (including spheres) using a three-dimensional lower envelope algorithm [6]. The implementation encountered robustness and efficiency issues beyond a relatively small number of input objects, however. One of the reasons is that a bivariate lower envelope algorithm inherently incurs degenerate cases; the vertices of the minimization diagram (the arrangement of curves that is the projection of the lower envelope) are an intersection of three curves meeting at the same point.

Anton et al. [1] showed how an alternate approach, relying on the empty circumsphere property of Voronoi vertices, can be made robust by using exact evaluation of the empty sphere predicate, showing also that the input must have six times more precision than that required for the empty sphere predicate exact computation.

To our knowledge, algorithms for computing Voronoi diagrams of spheres that show the best results for large data sets in practice are based on the edge tracing algorithm of Kim et al. [10]; however, this algorithm assumes non-degenerate input that is in general position, with all Voronoi vertices having degree four, and no disconnected Voronoi edges.

This paper is inspired by Hanniel et al. [7], which used the lower envelope of distance functions to compute the Hausdorff distance on the GPU by sampling points on the first surface and performing numerical iterations to compute the minimal distance from each sample to the other surface.

3. Algorithm description

Definition 1: Given a set of spheres $S_0, S_1, \ldots, S_n$ in R^3, the Voronoi cell of sphere S_i is the set of all points closer to S_i than to $S_j (\forall j \neq i)$. The Voronoi diagram is the union of the Voronoi cells of all $(n + 1)$ spheres.

Definition 2: The locus of points that are equidistant from S_i and S_j $(j \neq i)$ is called bisector $B_{i,j}$.

In general, bisectors in R^3 are surfaces, either a plane when the two spheres are the same size, or a hyperbolic surface when the two spheres are different sizes [6]. We assume that no input sphere is completely contained inside another, but our same framework handles intersecting spheres correctly. Voronoi faces are portions of such bisector surfaces between two spheres that have equal (minimal) distance from the two spheres. Please refer to Elber and Kim [5] for a detailed formulation of the bisectors.

For a single Voronoi cell, its Voronoi edges are formed where two of its Voronoi faces intersect and, in general, are segments of curves [10]. In general, three of the bisector surfaces contribute to each of the cell's Voronoi vertices, and this involves four spheres for each vertex.

In this paper, instead of analytically calculating Voronoi faces, we propose a sample-based approach to obtain sample points on Voronoi faces, by taking the lower envelope of the bisectors with respect to the distance function to the "base sphere" in each Voronoi cell. We shoot sample rays from this sphere in radial directions (Section 3.1), calculate the intersection of each ray with the implicit bisector surface functions (Section 3.2), and retain only the intersection with the minimum distance as our face sample point (Section 3.3).

We calculate the Voronoi vertices of each cell by using a marching-grid approach to locate the neighborhoods of the vertices. We virtually "color code" each Voronoi

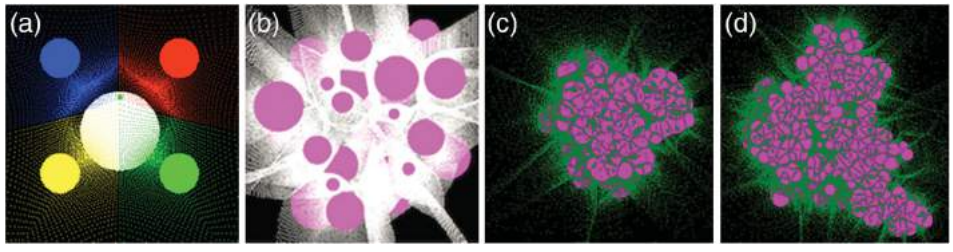

Figure 1. Example results: (a) A single Voronoi cell pictured, where the green Voronoi vertex is equidistant from the 5 spheres; the full Voronoi diagram we computed for (b) "Random Set A" [12]; (c) Protein ID 1crn-PQR consisting of 642 atoms (202 Cs, 55 Ns, 64 Os, 6 Ss, and 315 Hs) [14]; (d) Protein ID 1bh8-PQR consisting of 2161 atoms (680 Cs, 181 Ns, 203 Os, 10 Ss, and 1087 Hs) [14].

face sample point by giving it the same color as the bisector to which it belongs (Section 5). A Voronoi vertex occurs where there are three or four different colors out of four neighboring sample points, which means there is an intersection of three or more bisectors.

The stages of the algorithm to construct a Voronoi cell are:

1. Sample rays from the base sphere S_i.
2. Calculate the bisector surfaces functions between S_i and S_j ($j = 0, 1, \ldots, i-1, i+1, \ldots, n$).
3. Compute the intersection of each ray with all the bisectors and take the lower envelope of all the intersections to obtain the sample points on the Voronoi faces.
4. Find all grid cells of neighboring sample points that contain Voronoi vertices and use the Newton-Raphson method to calculate the vertex location.

We now describe the steps in detail.

3.1. Sampling rays from base spheres

For simplicity of calculation, we assume each base sphere in turn is a unit sphere with radius 1 located at the origin. If not, we translate and scale the coordinate system so that the base sphere fulfills this requirement. The algorithm to sample normal rays from the sphere surface uses a parametric representation of the ray. Denoting **O** as the vector of the ray origin on the sphere surface, **n** as the unit normal of the ray, and t as the scalar distance along the ray from the origin, the representation of the ray **r**(t) in parameter t is given in Eq. 1. With respect to the base sphere, **O** is on the surface of the base sphere, **n** is in the radial direction from the base sphere, and t represents the distance from the ray origin to the intersection of the ray with the bisector surface (Fig. 2(a)).

$$\mathbf{r}(t) = \boldsymbol{r}(x(t), y(t), z(t)) = \boldsymbol{O} + t \cdot \boldsymbol{n} \tag{1}$$

For sampling, we parameterize the six faces of the axis-aligned bounding cube of the base sphere. Each face of the bounding cube is taken as a uniformly subdivided parameterized domain expressed in variables u and v (Fig. 2(b)).

Each of the six u-v domains corresponds to 1/6 of the base sphere. For each sample on each u-v domain, there is a ray shooting from the surface of the base sphere. Fig. 2(b) shows an example of sampling based on the top face $z - 1 = 0, -1 \le x \le 1, -1 \le y \le 1$. The u-v domain for the top face has u and v ranges as $-1 \le u \le 1, -1 \le v \le 1$. Each (u, v) sample corresponds to a ray (u, v, 1) with the origin on the sphere surface, normalized as unit vector $\frac{1}{\sqrt{(u^2+v^2+1^2)}} \cdot (u, v, 1)$. The sample ray origin and unit normal on the base sphere are both $\frac{1}{\sqrt{(u^2+v^2+1^2)}} \cdot (u, v, 1)$.

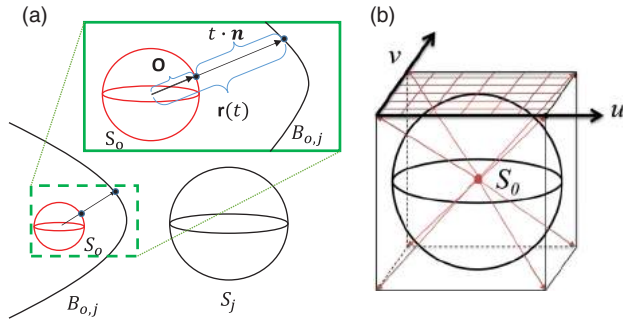

Figure 2. (a) Illustration of ray intersection with bisector surfaces. (b) Mapping sphere to six u-v parametric surfaces on the bounding cube; uniform parametric sampling of top surface shown.

3.2. Calculating the bisector functions

The bisector surfaces between two spheres are the simultaneous solutions of two distance equations (Eq. 2), where P(*x*,*y*,*z*) is any Euclidean point on the bisector surface, (C_{x1},C_{y1},C_{z1}), (C_{x2},C_{y2},C_{z2}) are the centers of the two spheres, and R_1, R_2 are the radii of the two spheres:

$$\sqrt{(x - C_{x1})^2 + (y - C_{y1})^2 + (z - C_{z1})^2} - R_1 = \sqrt{(x - C_{x2})^2 + (y - C_{y2})^2 + (z - C_{z2})^2} - R_2. \tag{2}$$

Eliminating the radius in Eq. 2 results in an implicit quadratic surface equation:

$$Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J = 0 \tag{3}$$

where $A = 4R^2 - 4D_x^2$, $B = 4R^2 - 4D_y^2$, $C = 4R^2 - 4D_z^2$, $D = -8D_xD_y$, $E = -8D_yD_z$, $F = -8D_zD_x$, $G = -8R^2C_{x1} - 4KD_x$, $H = -8R^2C_{y1} - 4KD_y$, $I = -8R^2C_{z1} - 4KD_z$, and $J = 4R^2(C_{x1}^2 + C_{y1}^2 + C_{z1}^2) - K^2$, where $R = R_1 - R_2$, the difference in the radii; $D_x = C_{x1} - C_{x2}$, $D_y = C_{y1} - C_{y2}$, $D_z = C_{z1} - C_{z2}$, the distance between the centers; and $K = (C_{x2}^2 - C_{x1}^2) + (C_{y2}^2 - C_{y1}^2) + (C_{z2}^2 - C_{z1}^2) - R^2$.

If the two spheres are different sizes, the bisector is a hyperbolic surface; otherwise it is a plane. If the two spheres are of the same size, the coefficients A, B, C, D, E, and F are zero and Eq. 3 simplifies to the linear equation:

$$Gx + Hy + Iz + J = 0 \tag{4}$$

where $G = 2(C_{x1} - C_{x2}), H = 2(C_{y1} - C_{y2}), I = 2(C_{z1} - C_{z2})$, and $J = C_{x2}^2 - C_{x1}^2 + C_{y2}^2 - C_{y1}^2 + C_{z2}^2 - C_{z1}^2$.

3.3. Finding intersections and lower envelope

The Euclidean distance function r of a point (x, y, z) from a sphere surface is $(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = (r + r_0)^2$, where (x_0, y_0, z_0) is the center of the sphere with radius r_0, and r is the distance of the point (x, y, z) from the spherical surface. When solving for the intersection of the ray and the bisector surface, we substitute the ray representation (Eq. 1) in terms of t into the bisector implicit function (Eq. 3); which can be simplified as a quadratic equation in parameter t when the bisector is a hyperbolic surface (Eq. 5), or a linear equation when the bisector surface is a plane (Eq. 6).

$$a \cdot t^2 + b \cdot t + c = 0 \quad (5)$$

$$d \cdot t + e = 0 \quad (6)$$

While solving Eq. 5, depending on the value of the discriminant $(b^2 - 4ac)$, there will be zero, one, or two solutions to the equation, corresponding to the respective cases where the ray does not intersect the bisector, intersects it once, or intersects it twice. In addition, the solution to the hyperbolic bisector equation has two possible sheets, and depending on the relative radius of the second sphere to the base sphere, only one of the sheets should be retained. As shown in Fig. 3, when the smaller sphere (the red sphere on the left) is the base sphere, the closer of the intersections corresponds to the correct bisector surface and therefore, the closer solution of Eq. 5 is used. Conversely, when the larger sphere is the base sphere (shown in green), the farther solution is used.

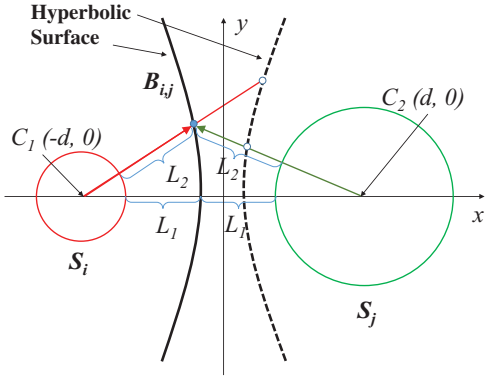

Figure 3. Culling the incorrect sheet of the two hyperboloid sheets.

When the ray does not intersect the bisector surface, we use a distance value of infinity to unify the lower envelope calculation. For each ray, the lowest distance to all the bisectors is stored, to output the final sample point on the Voronoi face the ray intersects. For infinite Voronoi cells, some of the final sample points will be at infinity (these can be thought of as intersections with a bisector surface between the base sphere and infinity, allowing us to process them like any other bisector for calculating Voronoi vertices).

In the case of intersecting spheres, the bisector surface will pass through the interior of both spheres (Fig. 4). (Intersecting spheres are actually quite common in the case of molecular input where atoms' influence spheres often overlap within a single protein, for example.) For simplicity of implementation, we thus calculated all distances along sample rays from the origin of the base sphere rather than its surface, so that the lower envelope calculations need only consider positive distances to bisectors.

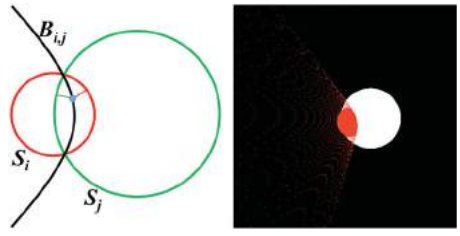

Figure 4. Bisector surface for two intersecting spheres.

4. Calculating Voronoi cells

We use a marching-grid approach to locate the neighborhood of Voronoi vertices by checking each group of four neighboring face sample points, which we call a "grid cell" on the bounding cube. In the u-v parametric domain, for each such grid cell, we color-code each of its four corner points based on the corresponding bisector. Within a grid cell, if the four corners have three or more colors, then they correspond to three or more bisectors and thus to a corresponding number of Voronoi faces. Hence, at least one Voronoi vertex must be located within the cell (e.g. where those Voronoi faces intersect).

Fig. 5(a) and Fig. 5(b) show the correspondence between sample points on Voronoi faces in geometric space and the samples on the bounding cube of the base sphere in the u-v domain. The Voronoi cell of the base sphere (the large sphere in the middle) is shown on the left; it has four surrounding unbounded Voronoi faces (red, blue, green, and yellow). Infinite rays are represented by the gray portion in the color map of the bounding cube (Fig. 5(b)). Fig. 5(c) shows one of the u-v domains of the bounding cube of the base sphere. The grid cells indicated contain three colors.

Similar to the Voronoi vertices, Voronoi edges occur where there are at least two different colors out of the four neighboring sample points, which means there is an intersection of two bisectors. The connectivity of the

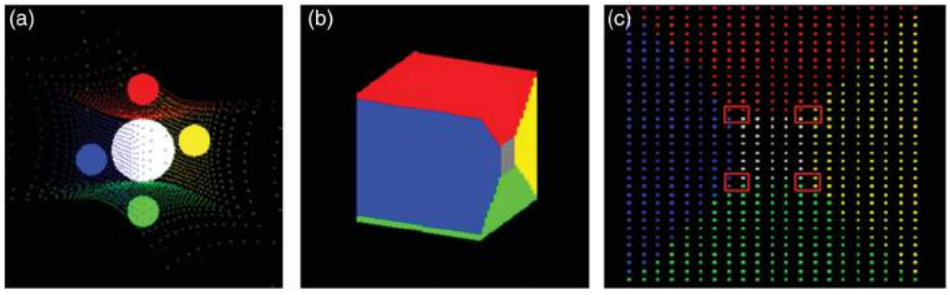

Figure 5. (a) Sample points on Voronoi faces for white base sphere; (b) Corresponding color map of u-v domains on bounding cube; (c) Sample point grid on one face of the bounding cube with 3-color grid cells indicated.

grid cells and the pattern of shared colors of the sample points determine the vertex-edge topology of the Voronoi cell.

5. Calculating Voronoi vertices

For a three-color grid cell, finding the Voronoi vertex is equivalent to solving the three bisector surface equations simultaneously. We use the Newton-Raphson method to find locations of the actual vertices, unless one of the bisectors is the infinity bisector, in which case we already know the vertex is located at infinity. By checking all the grid cells, those grid cells with three or more colors are easily located. To calculate a Voronoi vertex P(x, y, z), we solve for the three unknowns x, y and z. The implicit functions for the corresponding three bisector surfaces are denoted as $f_1(x, y, z)$, $f_2(x, y, z)$, and $f_3(x, y, z)$. The solution for the system of 3 non-linear equations in 3 unknowns is given by:

$$\begin{aligned} f_1(x, y, z) &= 0 \\ f_2(x, y, z) &= 0 \\ f_3(x, y, z) &= 0. \end{aligned} \tag{7}$$

The following vector expression is used to simplify Eq. 7, where **F** denotes the vector representing functions $f_i(P)$ as in Eq. 8, and **0** denotes the zero vector. The solution for the system of implicit functions for the bisector surfaces $f_i(x, y, z)$ can be written using a single vector expression as $\mathbf{F}(\mathrm{p}(\mathrm{x}, \mathrm{y}, \mathrm{z})) = \mathbf{0}$, where

$$\mathbf{F}(\mathrm{P}(\mathrm{x}, \mathrm{y}, \mathrm{z})) = \begin{bmatrix} f_1(x, y, z) \\ f_2(x, y, z) \\ f_3(x, y, z) \end{bmatrix}. \tag{8}$$

If $\mathrm{P} = P_0$ represents the first guess for the solution, successive approximations to the solution are obtained using numerical iteration. From the current k^{th} iteration, iterating from an approximate solution P_k and substituting it in, we obtain point P_{k+1} for the $(k+1)^{th}$ iteration:

$$P_{k+1} = P_k - \mathbf{J}^{-1}(P_k) \cdot \mathbf{F}(P_k) \quad (k = 0, 1, \ldots). \tag{9}$$

At each iteration, in the general case, an improved estimate of the solution is produced (since the bisector surfaces are always planes or hyperboloids), until the new estimate is close enough to the actual solution. A convergence criterion for the solution of this system is defined to be:

$$\max |f_i(P_k)| < \epsilon \tag{10}$$

for the user-defined tolerance ϵ.

The 3×3 Jacobian of the system (Eq.9) is:

$$\mathbf{J}(P_k) = \frac{\partial \mathrm{F}(\mathrm{x}, \mathrm{y}, \mathrm{z})}{\partial \mathrm{P}(\mathrm{x}, \mathrm{y}, \mathrm{z})} = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial z} \end{bmatrix}. \tag{11}$$

When the Jacobian matrix $\mathbf{J}(P_k)$ is singular, it is not invertible. This occurs when the three bisector equations are not linearly independent, which will occur, for example, where the three bisectors intersect at one common edge instead of at a vertex (Fig. 6). A singularity can also occur when the three bisectors actually have no intersection (Fig. 7). A third sort of singularity occurs if the start point for the iteration happens to fall at a local minimum. We distinguish the third case from the other two by subdividing the grid cell; we expect all of the sub-cells with the same colors of bisectors as the parent to also have singular Jacobian in the first two cases. In the third case, we should have a non-singular Jacobian in the sub-cell with the same color bisectors as the parent and can now successfully use iteration to solve for the location of the Voronoi vertex.

For the start point for the Newton-Raphson method, we intersect a ray that is the average of the four corner

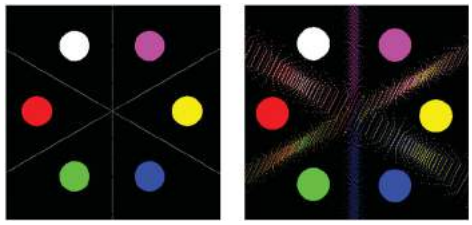

Figure 6. Special case of Jacobian singularity for atoms in a hexagonal configuration, the bisectors all intersect along an edge.

points of the grid cell with the corresponding bisector surfaces. Occasionally this leads to converging to a solution at the intersection of an incorrect sheet of a hyperbolic surface (Fig. 10(a)). We identify such cases by confirming that the solution is truly equidistant to the spheres defining the corresponding bisectors, and if not, reinitialize the start point with the ray intersection with a different bisector. In practice we found that the iteration converges within 10 steps except in cases such as shown in Fig. 8, where the iteration oscillates between two intersections of the bisector surfaces close together. When we identify such non-convergence, we subdivide the grid cell.

5.1. Subdivision

We uniformly subdivide the original grid cell into four sub grid cells. By taking the midpoints of the edges of the original grid cell, we have five new rays. The original parent grid cell is decomposed into four new child cells. Taking the lower envelope of each new ray's intersections with bisectors gives us five new sample points. We identify if any of the sub-cells have the same three colors as their parent. If so, we take the average of the rays of such cells as a new start point for the iteration. Otherwise, when all of the four children share only one or two colors with the parent, there is no vertex to output.

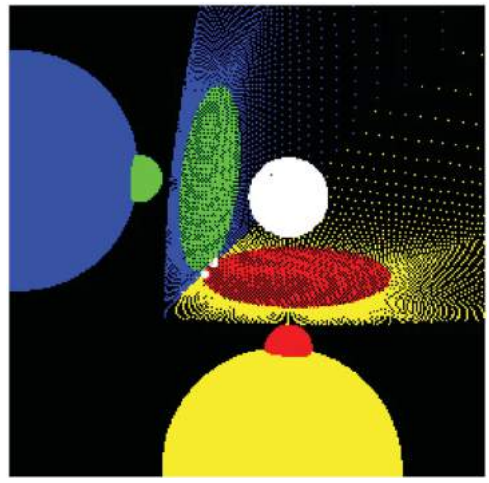

Figure 8. Case of a non-converging grid cell with two intersections. The two Voronoi vertices (white) are at the intersection of the same four bisectors (blue, green, red, and yellow).

When there is a new color identified during the subdivision, there is a new Voronoi face that was missed in the original sampling. If the new color lies on an edge of the old grid cell, we also subdivide the neighboring grid cell sharing the edge.

By using subdivision in our algorithm, we can increase the local sample density when the numerical iteration fails to converge or leads to a singular Jacobian in finding the Voronoi vertex.

5.2. Four-color special cases

Subdivision is also required to distinguish the special cases where all four corners of the grid cell are different colors. There are two possibilities: 1) the four corresponding bisector surfaces intersect at one vertex (Fig. 9(a); or 2) there is more than one Voronoi vertex in the neighborhood and we increase the sampling density for this grid cell to identify the other Voronoi vertices (Fig.

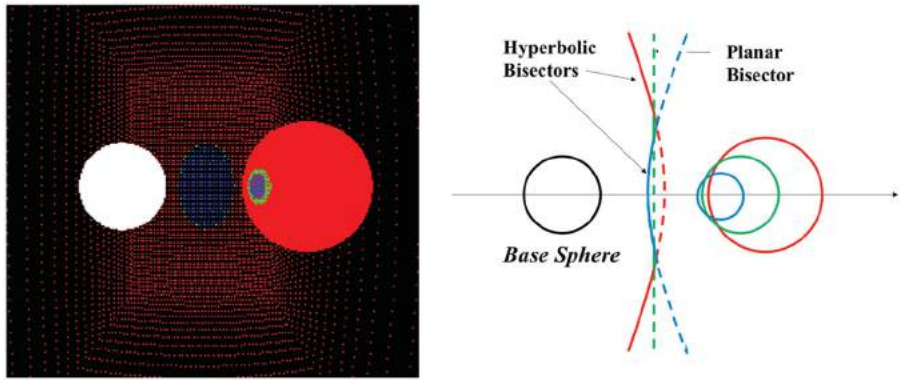

Figure 7. Special case of Jacobian singularity when bisectors do not actually intersect (Input consists of 3 intersecting colored spheres and a white base sphere; a cross section is also shown).

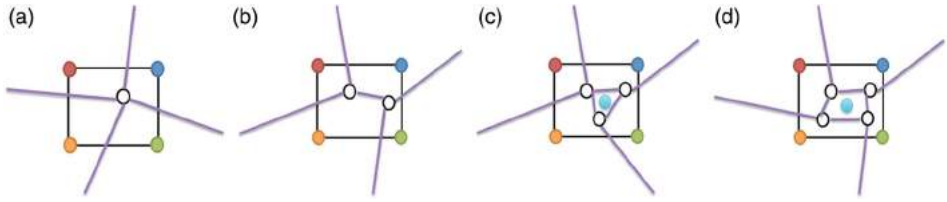

Figure 9. Special cases with four colors within one grid.

9(b-d)). To identify the vertex the four bisectors intersect at, we check if choosing any three of the four corner points as input for Newton's method gives us the same calculated vertex location.

If there is more than one vertex within this grid cell, we increase the samples in this grid cell by subdividing the original grid cell (Section 5.1). The next level of subdivision may reveal either two vertices within the grid cell (Fig. 9(b)), three vertices within the grid cell (Fig. 9(c)), or four vertices within the grid cell (Fig. 9(d)). For Fig. 9(c) and Fig. 9(d), there is a fifth color inside the grid cell but it has yet to be found without increasing the sampling density of the grid cell.

5.3. Replacement of vertices on incorrect sheets

We use implicit functions of corresponding bisectors in the Newton-Raphson method. From Fig. 3, we know that points on incorrect sheets of the hyperboloids (as well as points on the actual bisector surfaces) can make the implicit functions equal to zero. Although we only preserve the correct bisectors in the ray shooting process, the Newton-Raphson method may converge to vertices that are at an intersection involving one or more incorrect sheets of the hyperboloids. Fig. 10(a) shows the problem in a 2D case, where the iteration start point is farther from the real Voronoi vertex than from the indicated point that is at an intersection with an incorrect sheet, and therefore Newton-Raphson converges to this incorrect point. This situation typically arises when two spheres have close (but not identical) radii, which leads to the correct and incorrect sheets of the hyperboloid surfaces being very close to each other.

To determine if the point to which the iteration converges is a correct Voronoi vertex, we check its distance to the corresponding spheres. If the point is equidistant from the spheres, it is a Voronoi vertex. Otherwise, we use additional rounds of Newton-Raphson iteration with different start points, determined as follows. From the base sphere, we shoot a new ray through the previous start point. The intersection of the new ray and the bisectors between the base sphere and the other spheres are the new iteration start points (Fig. 10(b)). After Newton-Raphson iteration with these new start points, we check the equidistant property of their new converged points. (In practice, one of them always leads to the correct intersection, within tolerance.)

5.4. Sorting of the Voronoi vertices

For each vertex, there are four or more spheres from which it is equidistant, including the base sphere (in general position, it is exactly four). When combining the individual Voronoi cells into the Voronoi diagram, we need to find each vertex in the cells of all of its contributing spheres. To determine if we have done so, we sort the vertices by the lexicographic order of the indices of the contributing spheres. We consider vertices identified

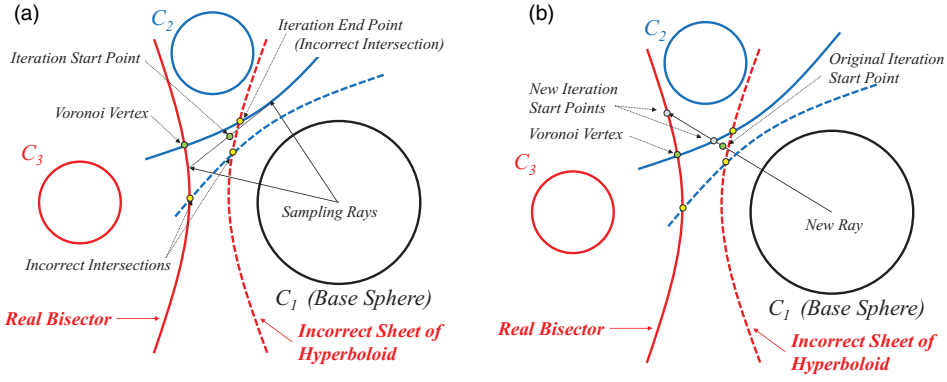

Figure 10. (a) Converging to an incorrect intersection with an unnecessary sheet of the bisectors. (b) Finding new start points for Newton-Raphson method.

in different Voronoi cells to be the same Voronoi vertex if they have the same contributing spheres and their locations are within the user-defined tolerance.

5.5. Incompletely matched vertices

In the sorting process, we may discover that a Voronoi vertex is not found in the cells of all of its contributing spheres. This occurs when the sampling is not dense enough to detect those vertices from all the base spheres (typically due to a Voronoi face with small area that none of the sampling rays hit). For such incompletely matched vertices, we perform a second targeted search from each of the corresponding base spheres where we have not found the specific vertex.

To do so, we shoot a new ray from the center of the base sphere in the direction of the undiscovered vertex, whose exact location we now know. We then use the corresponding u-v coordinate on the parametric bounding cube to determine the existing grid cell that the ray intersects. Centered around this u-v coordinate point, we construct a much smaller grid cell that is a user-defined fraction of the size of the original grid cell (Fig. 11). If this new grid cell's corners are three or more colors matching those of the other contributing spheres, then the same Voronoi vertex has now been found for this base sphere and we can add this appearance to the sorting list. Otherwise, we recursively create even smaller grid cells using the same size reduction ratio, until we find the 3-color grid cell or we reach a maximum depth of recursion.

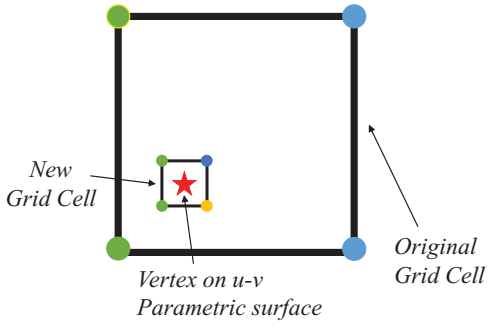

Figure 11. Construction of the new grid cell.

After processing all incompletely matched vertices, we sort the list again, and treat vertices found for all the contributing spheres as Voronoi vertices. For any point that still cannot be found for all of its contributing spheres, we test if it is truly a Voronoi vertex as follows. We can always find the sphere centered at this point and cotangent to all its contributing spheres. If this tangent sphere intersects or contains any other input sphere except its contributing spheres, the point is not a real Voronoi vertex and we remove it from the sorting list. (These spurious Voronoi vertices arise due to sampling errors.) For the full Voronoi diagram, we output each distinct Voronoi vertex once.

6. Design of GPU framework

The computation of intersections and lower envelopes as well as the calculation of Voronoi vertices are more time consuming when the input data size increases, which informs the direction of parallelism. Our algorithm is well suited to GPU programming because it has high-density sampling, making for arithmetically intensive operations that can be parallelized, and at each sample the calculation is relatively independent. The calculation of intersections with bisectors and the minimum distances depends only on the ray and the bisector, independent of neighbor rays or other bisectors. The problem is then reduced to a set of identical and independent distance equations to be solved for high-density sample points. Similarly, the numerical iteration is performed identically and independently for each of the Voronoi vertices.

Fig. 12 diagrams the framework of our algorithm. Serial operations done on the CPU are: reading input spheres, sorting Voronoi vertices, processing incompletely matched vertices, and plotting sample points and vertices in OpenGL. Most of the implementation is done on the GPU. There are seven kernel operations on the GPU in total:

1. Transform Base Spheres.
2. Sample Rays.
3. Calculate Bisectors.
4. Lower Envelope.
5. Calculate Vertices.
6. Replace Vertices on Incorrect Sheets.
7. Transform Back.

Transform Base Sphere: This kernel function transforms spheres according to the corresponding base sphere; each base sphere has its own object space where it is located at the origin. Let each base sphere be taken care of by one thread.

Sample Rays: This kernel function samples the rays for all spheres. Let each ray be taken care of by one thread. Denote n as the number of input spheres, and S^2 as the number of samples per u-v domain. For each Voronoi cell, there are $6S^2$ rays. For the whole Voronoi diagram, there are $(6S^2 \cdot n)$ rays and such that there are $(6S^2 \cdot n)$ threads on the GPU (see Section 3.1).

Calculate Bisectors: This kernel function calculates all the bisector surface functions between each base sphere and all other spheres. For each base sphere, there are $(n-1)$ valid bisectors. There are $n \times (n-1)$ bisectors in total (see Section 3.2). Let each base sphere be taken care of by one thread.

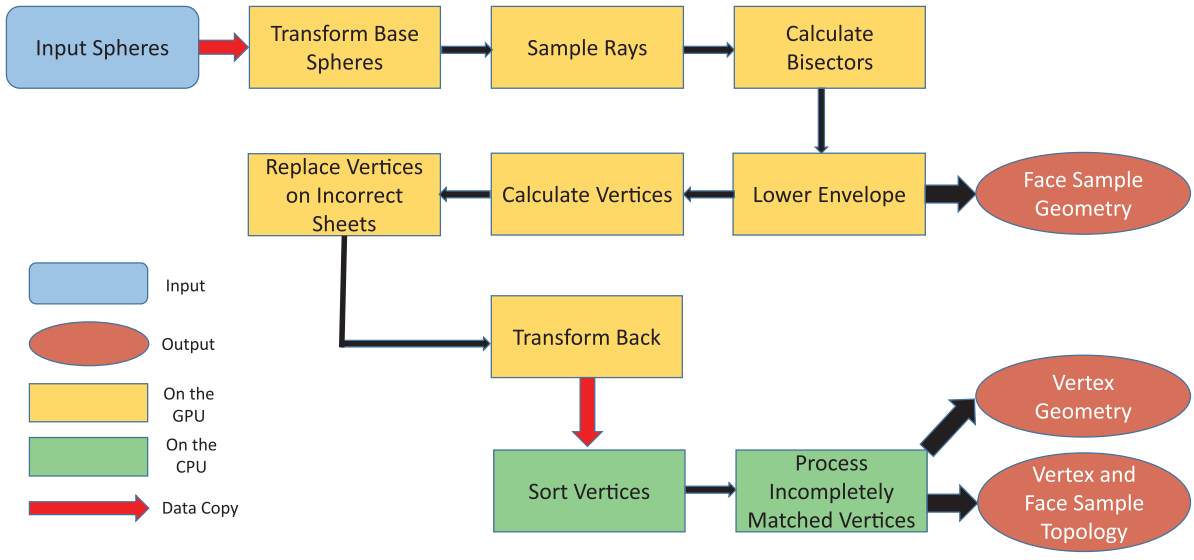

Figure 12. GPU framework.

Lower Envelope: This kernel function includes the calculation of finding intersection as described in Section 3.3 and then takes the minimum distance on each thread. Each ray finds intersections with other bisector functions on its thread.

Calculate Vertices: This kernel function finds the grid cells containing the Voronoi vertices and then calculates the vertices as described in Section 5. Let each grid cell of the parametric surface be taken care of by one thread.

Replace Vertices on Incorrect Sheets: This kernel function identifies the vertices on the incorrect sheets, and rectifies them by additional rounds of iteration in their corresponding grid cells with different start points (see Section 5.3). Let each vertex calculated from the previous step be taken care of by one thread.

Transform Back: This kernel function transforms back the points on Voronoi faces and Voronoi vertices to the regular Euclidean space according to the current base sphere. Let each grid cell be taken care of by one thread.

There is overhead associated with the GPU, such as allocating and deleting memory, and communication between the CPU and the GPU. In our implementation, there are allocation and deletion of rays and bisectors, sample points, and vertices on the GPU, copying of input spheres from the CPU to the GPU, and copying of sample points and Voronoi vertices from the GPU to the CPU.

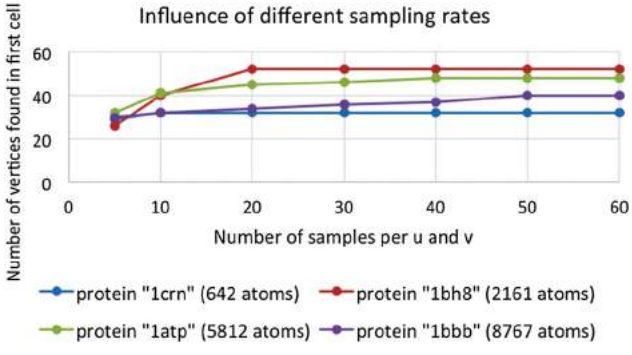

Figure 13. Sampling rate vs. the number of vertices calculated.

7. Results

Our algorithm to compute the Voronoi cells was run on a PC with an Intel(R) Core(TM) 2 Quad CPU Q9400 @2.66 GHz with 4.0 GB RAM and an NVIDIA Quadro 6000 graphics card. In our tests, we use a relative percentage as our tolerance ε for convergence (Eq. 10), specifically, 0.001% of the length of the diagonal of the bounding cube of all spheres for each input.

Our algorithm is robust in handling special cases of Voronoi vertices that arise with spheres not in general position, such as arise with symmetrical input, as in the cell in Fig. 1(a) with a degree-4 vertex. In addition to such synthetic test cases (e.g. Fig. 1(a) and 1(b)), we tested our implementation with several examples of each of two types of protein structures from the protein data bank [14]: PDB format and PQR format. In PDB format, atom spheres usually have different radii even for the same element, whereas the PQR format includes hydrogen atoms that all have the same radii. The hydrogen atoms greatly increase the presence of local symmetrical patterns, which can lead to non-general position Voronoi vertices.

Fig. 13 compares the number of Voronoi vertices for the first Voronoi cell found by our implementation when varying the sampling density for four different input proteins in PQR format. For smaller protein molecules, a 30 or 40 sampling rate may be sufficient to discover its

Table 1. Total computation time under different sampling rates. Running time in seconds.

Sampling	1al1-PQR (217 atoms)	1crn-PDB (327 atoms)	1crn-PQR (642 atoms)	1bh8-PQR (2161 atoms)	1JD0-PDB (4195 atoms)
30*30	1.30	2.02	3.85	23.3	50.2
60*60	1.92	3.06	5.83	35.4	81.4
80*80	2.29	3.85	7.05	41.2	93.6
100*100	2.45	4.21	7.82	45.9	102

Voronoi vertices, but for proteins consisting of a large number of atoms, the threshold is usually higher.

To test our ability to effectively build Voronoi diagrams for inputs of hundreds or thousands of spheres and handle Voronoi vertices not in general position, we selected protein "1JD0" under PDB format, protein "1al1" and "1bh8" under PQR format, and protein "1crn" under both formats. Table 1 shows the total running time for computing the Voronoi cells for these proteins as sampling density increases. Running times increase roughly linearly with the number of input spheres (atoms), and sublinearly with the number of sample rays, which are processed in parallel on the GPU.

8. Conclusion

We have presented a new approach to calculating Voronoi cells of spheres that combines a sample-based, lower-envelope approach with numerical iteration. The numerical iteration allows us to calculate the geometry of the vertices to a much greater accuracy than the ray sampling density. The lower-envelope calculations for sample rays are able to exploit the parallelism of the GPU, and are robust for non-general position input.

Acknowledgements

We gratefully acknowledge support from National Science Foundation grant 1331352.

ORCID

Zhongyin Hu http://orcid.org/0000-0003-2652-1325
Xiang Li http://orcid.org/0000-0001-5936-8457
Adarsh Krishnamurthy http://orcid.org/0000-0002-5900-1863
Iddo Hanniel http://orcid.org/0000-0001-9429-7973
Sara McMains http://orcid.org/0000-0002-7152-9409

References

[1] Anton, F.; Mioc, D.; Santos, M.: Exact computation of the topology and geometric invariants of the Voronoi diagram of spheres in 3D, Journal of Computer Science and Technology, 28(2), 2013, 255–266. http://dx.doi.org/10.1007/s11390-013-1327-3

[2] Berberich, E.; Fogel, E.; Halperin, D.; Kerber, M.; Setter, O.: Arrangements on parametric surfaces II: Concretizations and applications, Mathematics in Computer Science, 4, 2010, 67–91. http://dx.doi.org/10.1007/s11786-010-0043-4

[3] Chovancova, E.; Pavelka, A.; Benes, P.; Strnad, O.; Brezovsky, J.; Kozlikova, B.; Gora, A.; Sustr, V.; Klvana, M.; Medek, P.; Biedermannova, L.; Sochor, J.; Damborsky, J.: CAVER 3.0: a tool for the analysis of transport pathways in dynamic protein structures, PLoS Computational Biology, 8(10), 2012, e1002708. http://dx.doi.org/10.1371/journal.pcbi.1002708

[4] Edelsbrunner, H.; Seidel, R.: Voronoi diagrams and arrangements, in: Proceedings of the First Annual Symposium on Computational Geometry, SCG '85, ACM, New York, NY, USA, 1985, 251–262. http://dx.doi.org/10.1145/323233.323266

[5] Elber, G.; Kim, M.-S: Computing rational bisectors, IEEE Computer Graphics and Applications, 19(6), 1999, 76–81. http://dx.doi.org/10.1109/38.799747

[6] Hanniel, I.; Elber, G.: Computing the Voronoi cells of planes, spheres and cylinders in R^3, Computer Aided Geometric Design, 26(6), 2009, 695–710. http://dx.doi.org/10.1016/j.cagd.2008.09.010

[7] Hanniel, I.; Krishnamurthy, A.; McMains, S.: Computing the Hausdor distance between NURBS surfaces using numerical iteration on the GPU, Graphical Models, 74(4), 2012, 255–264. http://dx.doi.org/10.1016/j.gmod.2012.05.002

[8] Hanniel, I.; Muthuganapathy, R.; Elber, G.; Kim, M.-S.: Precise Voronoi cell extraction of free-form rational planar closed curves, in: Proceedings of the 2005 ACM symposium on Solid and physical modeling, ACM, 2005, 51–59, http://dx.doi.org/10.1145/1060244.1060251

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

[10] Kim, D.-S; Cho, Y.; Kim, D: Euclidean Voronoi diagram of 3D balls and its computation via tracing edges, Computer-Aided Design, 37(13), 2005, 1412–1424. http://dx.doi.org/10.1016/j.cad.2005.02.013

[11] Petrek, M.; Kosinova, P.; Koca, J; Otyepka, M.: Mole: a Voronoi diagram based explorer of molecular channels, pores, and tunnels, Structure, 15(11), 2007, 1357–1363. http://dx.doi.org/10.1016/j.str.2007.10.007

[12] Random spheres data set from Voronoi Diagram Research Center, http://voronoi.hanyang.ac.kr/qtdb/testdataset/random_sphereset.htm, accessed: 2015-04-05.

[13] Seong, J.-K.; Cohen, E.; Elber, G.: Voronoi diagram computations for planar NURBS curves, in: Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling, SPM '08, ACM, New York, NY, USA, 2008, 67–77, http://dx.doi.org/10.1145/1364901.1364913

[14] The RSCB Protein Data Bank, http://www.rcsb.org/pdb/home.home.do, accessed: 2016-02-25.