Computing Polygonal Surfaces From Unions of Balls

Roger Tam
rtam@cs.ubc.ca

Wolfgang Heidrich
heidrich@cs.ubc.ca

Imager Computer Graphics Laboratory
Department of Computer Science
University of British Columbia

Abstract

We present a new algorithm for computing a polygonal surface from a union of balls. The method computes and connects the singular points *of a given union of balls in an efficient manner to approximate the boundary. The algorithm uses the* dual shape *of the balls to give the resulting surface the correct topology. Our method is simple and demonstrated to be robust.*

1. Background and Introduction

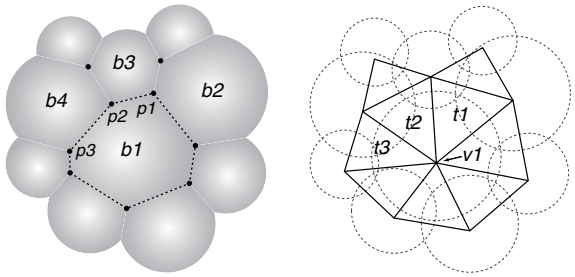
Ball models have been used for numerous applications in graphics and visualization, including shape matching [14], collision detection [9, 11], animation [10], and molecular modelling [13]. It is often useful to generate an approximating polygonal surface from a given union of balls. For example, to display an object with typical graphics hardware it is much faster to do so with polygons than balls.

In a reasonably dense union of balls, there are typically many locations on the surface in which three or more balls meet to form an intersection point, called a *singular point*. Figure 1a shows a union of balls and some of its singular points. Our algorithm connects these points to form a polygonal surface and incorporates a method for dealing with undersampled areas in which the balls do not intersect.

Our algorithm uses the weighted Delaunay triangulation of the union of balls to compute the surface. The subcomplex consisting of the interior Delaunay triangles has been proven to be homotopy equivalent to the union of balls [6]. We take advantage of this property to compute surfaces that are topologically correct.

2. Related Work

Our work is partially inspired by the *power crust* algorithm [1], developed by Amenta *et al.*, which generates a surface of an object from a set of boundary points by first producing two sets of balls, one inside the object and one outside. The power crust then computes the interface between inside and outside balls to make a surface. A drawback of the power crust algorithm is that it produces a large number of faces for the number of balls present. Our method computes a surface from only one set of balls inside the object, and the number of polygons produced is on the same order as the number of balls.



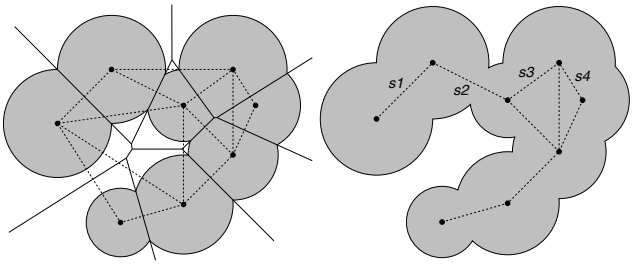
(a) (b)

Figure 1. (a) A union of balls and some of its singular points (b) The dual shape of the union of balls

An algorithm for the *skinning* of unions of balls has been developed by Edelsbrunner *et al.* [7, 8]. The skins are typically composed of parts of the balls connected by hyperboloid and spherical patches to make the surface tangent continuous. The skin is then adaptively triangulated to form a polygonal surface. Kruithof and Vegter [12] extend the method to approximate C^2 surfaces. There are two primary drawbacks to the skinning approach. First, even a small number of balls can generate a complicated skin. Second, there are always concave patches between the balls, sometimes resulting in a bumpy appearance. Our method is considerably more lightweight, and is designed for applications where efficiency is more important than continuity of the surface.

3. Weighted Delaunay Triangulation

To facilitate the discussion, we first provide the definition of the primary geometric construction used in our algorithm. The *power diagram* [3] is a generalized Voronoi diagram that is computed on a set of weighted points. The dual of the power diagram is the *weighted* or *regular* Delaunay triangulation. A 2D example is shown in Figure 2a, which illustrates the power diagram of eight disks; the weighted Delaunay triangulation is shown with dotted lines.



(a) (b)

Figure 2. (a) Power diagram of eight disks (b) Dual shape

The weighted Delaunay triangulation uses the *power distance* to define the adjacencies between ball centres. Given a set of n balls $\{b_1, \ldots, b_n\}$ with centres $\{x_1, \ldots, x_n\}$ and radii $\{r_1, \ldots, r_n\}$, the power distance $\pi(b_i, b_j)$ between any two balls b_i and b_j is defined as

$$\pi(b_i, b_j) = \|x_i - x_j\|^2 - ({r_i}^2 + {r_j}^2).$$

Two balls b_i and b_j are called *orthogonal* if $\pi(b_i, b_j) = 0$. Given a simplex of dimension one or greater in the triangulation, the *orthocentre* of the simplex is the centre of the smallest ball that is orthogonal to each ball of the simplex.

The *dual shape* [6] of a union of balls can be computed from a weighted Delaunay triangulation by discarding all of the simplices that lie outside of the union of balls. The dual shape of the union of disks in Figure 2a is shown in Figure 2b. A very useful property of the dual shape is that it has the same topology as the union of balls [6].

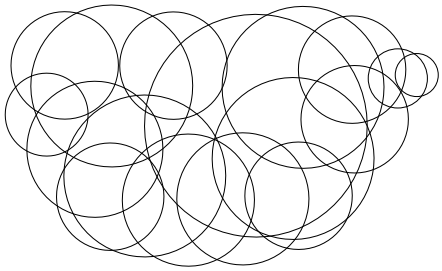
For computing the weighted Delaunay triangulation, we use the CGAL library (`http://www.cgal.org`), which implements a version [5] of the randomized incremental algorithm [4].

4. Algorithm Overview

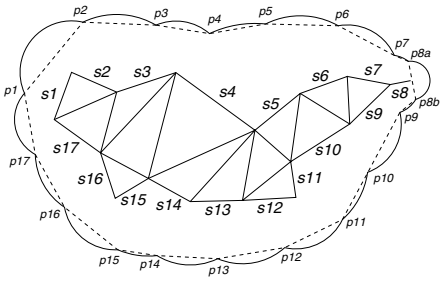
Our algorithm consists of three main steps:

1. Compute the dual shape of the union of balls.
2. For all regular and singular triangles (defined in the next section) in the dual shape, compute the singular points, except for duplicates.
3. Compute surface polygons from the singular points using the connectivity of the dual shape.

The main idea of the method is to traverse the hull of the dual shape in an ordered fashion, connecting the singular points on the way. Figure 3 illustrates how the method works in 2D (*i.e.*, computing a polyline contour from a union of disks). In Figure 3b, the *s*'s are the regular and singular simplices of the dual shape, the *p*'s are the singular points, and the dotted line is the resulting approximating boundary. The idea is that as we traverse the simplices in order along the hull ($s1 \rightarrow s2 \rightarrow s3 \rightarrow s4 \rightarrow \ldots$), we connect the corresponding singular points (*p1* to *p2* to *p3* to *p4*, *etc.*).



(a)



(b)

Figure 3. (a) Union of disks (b) Connecting the singular points to form a contour (dotted line)

5. Singular Point Computation

Each triangle in the dual shape can be classified as one of three types: *interior*, *regular*, or *singular*. This classification is based on the number of tetrahedra to which the triangle belongs. An interior triangle is the interface between

two neighbouring tetrahedra, a regular triangle belongs to only one tetrahedron, and a singular triangle is not part of a tetrahedron. For most data sets, the majority of triangles are faces of tetrahedra, and therefore are either regular or interior. Each type of triangle produces a different number of singular points. An interior triangle produces no singular points, a regular triangle produces one singular point, and a singular triangle produces two singular points. In the 2D analog shown in Figure 2b, *s1* and *s2* are singular simplices, because they do not belong to a triangle, whereas *s3* and *s4* are regular simplices. In Figure 3b, *s8* is the only singular simplex in the dual shape. As a result, *s8* is the only simplex producing two singular points (*p8a* and *p8b*).

For a given triangle, we compute its singular point(s) by first locating the orthocentre. We make use of the fact that the two intersection points formed by the three balls are collinear with the orthocentre. To see why this is true, we note that the circle formed by two intersecting balls lies in the plane defined by the power diagram wall between the two balls. This plane is called the *bisector plane* (Figure 4a), and consists of the points that have same power

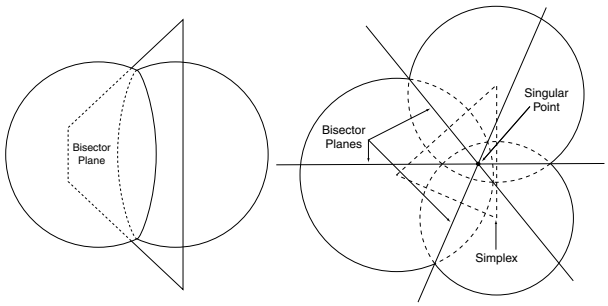


Figure 4. (a) Bisector plane (b) Three bisector planes ("top" view)

distance to both balls. By construction, the three bisector planes formed by three balls are orthogonal to the triangle connecting the ball centres, and the intersection of the three planes results in a line. This line consists of the points that have the same power distance to each of the three balls, and goes through the singular point(s) (Figure 4b). By definition, the orthocentre has the same power distance to to all three balls, and must also lie on this line. Once we find the orthocentre, we only need to compute the correct distance (*d* in Figure 5) to find the singular point(s).

It is important to note that in order to avoid degenerate faces, we need to prevent duplicate singular points from being computed. Multiple identical singular points are often produced when four or more balls intersect at the same point, as is frequently the case when the union of balls

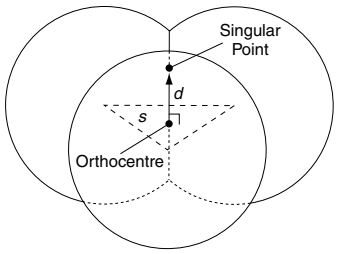


Figure 5. Computing a singular point of the simplex *s*

is constructed by computing the circumscribing balls of the Delaunay triangulation of an object's boundary points (*e.g.*, Amenta and Kolluri [2]). By keeping track of which balls intersect to form which singular points, we can avoid computing duplicate points rather than having to deal with them numerically later on.

6. Polygon Computation

After the singular points have been computed, they must be connected properly so that the resulting surface has the correct topology and well-formed faces. For each vertex in the dual shape, we traverse the incident faces on the hull around the vertex and connect the corresponding singular points in order. Beginning with a triangle on the hull incident to the vertex, we search the neighbouring triangles to find one that is also on the hull and incident to the vertex. The process then continues around the vertex until we encounter the first triangle. Figure 1b shows a projection of the hull of the dual shape of the union of balls in Figure 1a. Traversing around *v1* ($t1 \rightarrow t2 \rightarrow t3 \rightarrow \ldots$) results in connecting *p1* to *p2* to *p3*, *etc.* in Figure 1a. The final result of going around this vertex is the face shown by the dotted line in Figure 1a.

An intuitive way to understand how the method works is to think of the traversal around a vertex as being equivalent to taking the ball centred at the vertex, and using the surface arcs formed by the ball intersecting with its neighbours as "paths" to find the appropriate sequence for the singular points. For example, the path from *p1* to *p2* in Figure 1a is determined by the intersection between the balls *b1* and *b3*. This duality is apparent if we consider the facts that the arcs lie on the Voronoi walls of the power diagram of the balls, and the singular points lie on the intersections between the Voronoi walls.

While traversing the triangulation, there are two primary issues that need to be addressed. The first is that if a triangle has more than one neighbour on a given edge, we need a method for determining which neighbour to use in order to stay on the proper side of the hull. The second is that when we encounter a singular triangle, we need to decide which of the two singular points is the proper one to use next.

6.1. Finding the Correct Neighbour

Figure 6 illustrates a scenario in which we need to decide which neighbour of a triangle we should proceed to next. In this case, singular point *p1*, computed from triangle *t1*, has just been added to the current face. We need to determine which of *t2*, *t3*, or *t4* is the correct triangle. Making the wrong choice (either *t3* or *t4*) would cause the traversal to go right through the hull. To find the right neighbour (*t2*), we form a triangle $t1'$ using the given edge and the current singular point (*p1*). The first triangle encountered when rotating in the direction from *t1* to $t1'$ is the correct neighbour. The process then continues by finding the correct neighbour of *t2*.

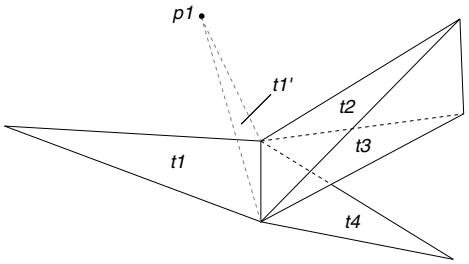


Figure 6. Using the direction from *t1* to $t1'$ to find the right neighbour (*t2*)

6.2. Finding the Correct Singular Point

When we reach a singular triangle, we need to determine which of the two singular points is the correct one to use. By construction, the singular points must be on opposite sides of the triangle. Therefore, choosing the wrong one would cause the constructed face to cut across the interior of the object. In order to find the right singular point, we compute a number of vectors to determine if choosing a particular singular point would cause a flip in orientation to the other side of the hull, relative to the singular point used for the previous triangle.

Figure 7 illustrates the vectors used. The normal vectors (*n1*, *n2*) for the previous (*t1*) and current (*t2*) triangles are computed. In addition, a vector originating from the centroid of the previous triangle pointing to the previous singular point (*p1*) is computed. Similarly, two vectors directed toward the two singular points of the current triangle (*p2a*, *p2b*) are derived. The sign of the dot product between the normal vector and the singular point vector of the previous triangle *t1* gives us a reference to determine which of *p2a* or *p2b* should be used to preserve the current orientation. In this case *p2a* is the correct one.

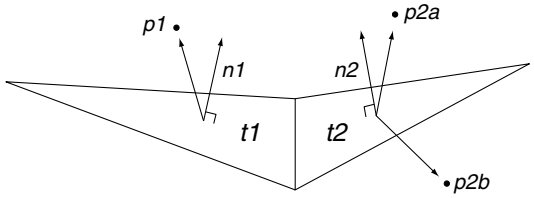


Figure 7. Using the normal vectors to find the right singular point

7. Undersampled Areas

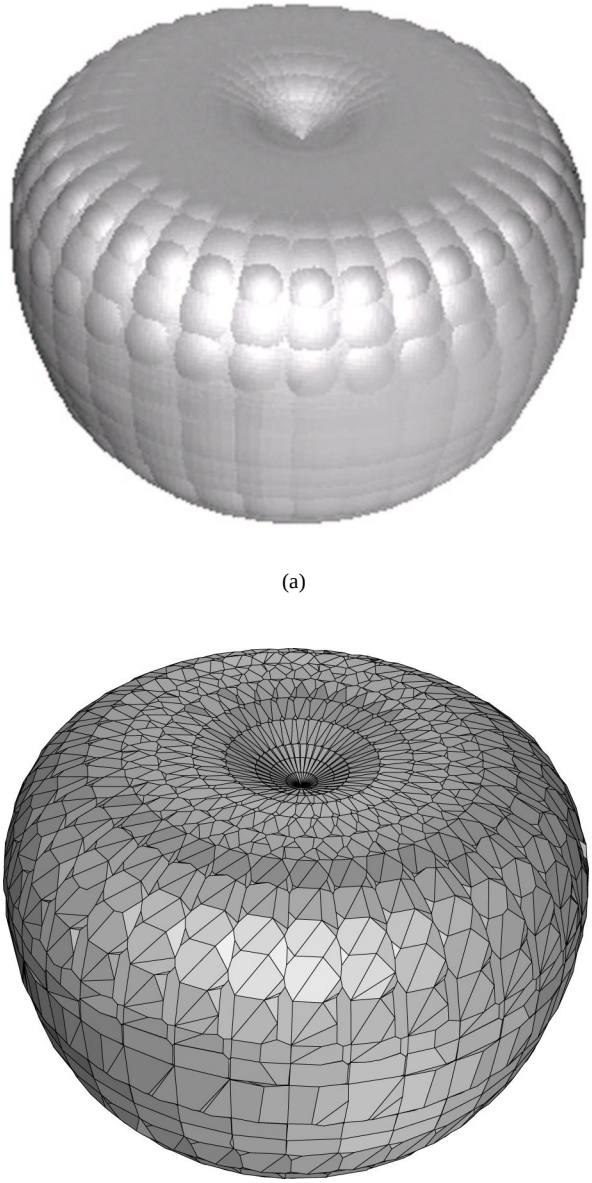
In some data sets, there are areas in which the balls are close enough for the computed Delaunay triangles to be largely inside the shape, but the balls do not actually intersect. This happens most frequently when one ball intersects two other ones, and the other two are close to each other but do not actually intersect. Our method of computing the singular points using the orthocentre is beneficial in such cases, because the orthocentre can be computed without an intersection. This is in contrast to using, for example, great circles for determining intersection points. To compute a "fake" singular point to add to the surface, we simply estimate an appropriate orthogonal offset from the orthocentre by averaging the radii of the three balls connected in the triangle.

8. Results

To demonstrate the effectiveness of our algorithm, we show several examples of varying shape complexity, topology and sampling density in Figures 8 to 11. All of the unions of balls are computed using the method by Amenta and Kolluri [2]. Table 1 shows the processing times to compute the surfaces. The number of balls and faces are also shown for each case. As mentioned in Section 2, the number of faces produced for each model is on the same order as the number of balls. A Pentium 4 processor running at 2.0 Ghz is used for our experiments. Our implementation makes extensive use of the CGAL and LEDA libraries.

Model	Balls	Time (sec.)	Faces
Apple	3095	4.6	4179
Mushroom	3609	5.0	4963
Heart	3405	4.6	3900
Torus	5613	6.2	7154

Table 1. Processing times for surface reconstruction from unions of balls



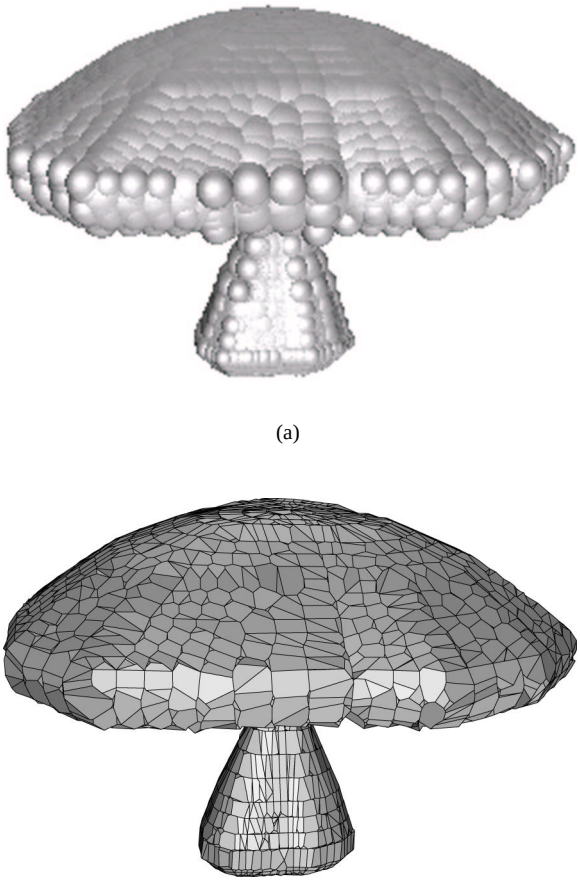
(a)

(b)

Figure 8. (a) Union of balls of an apple (3095 balls) (b) Surface reconstructed from the union of balls of an apple (4179 faces)

9. Summary

In this paper, we have presented an efficient method for constructing an approximating polygonal boundary of a union of balls. We have shown the results of applying our algorithm to several unions of balls of varying shape complexity, topology and sampling density.



(a)

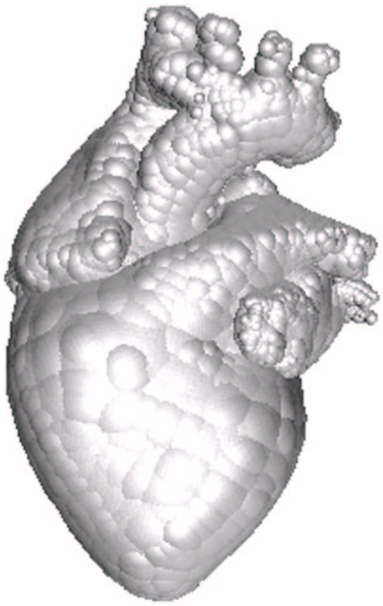
(b)

Figure 9. (a) Union of balls of a mushroom (3609 balls) (b) Surface reconstructed from the union of balls of a mushroom (4963 faces)

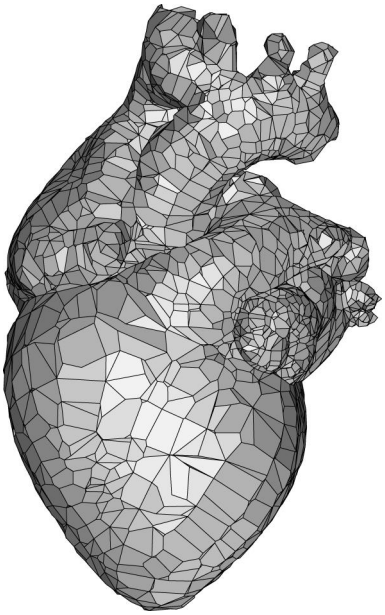
10. Future Work

We intend to pursue a number of directions to enhance our algorithm:

- While our algorithm produces surfaces that the average viewer would say are faithful reconstructions, geometrically speaking they are only approximations. A derivation of the error bounds would be informative. In addition, while we conjecture that the reconstructed surface converges to the envelop of the union of balls as the sampling density tends to infinity, this should be rigorously proven.
- The polygons produced by our algorithm are non-planar in general. Although a number of available software packages can retessellate non-planar faces into



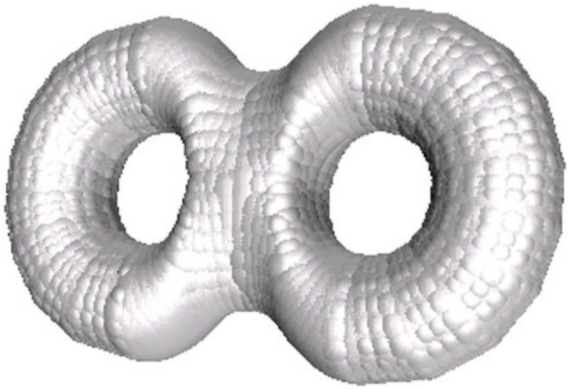
(a)



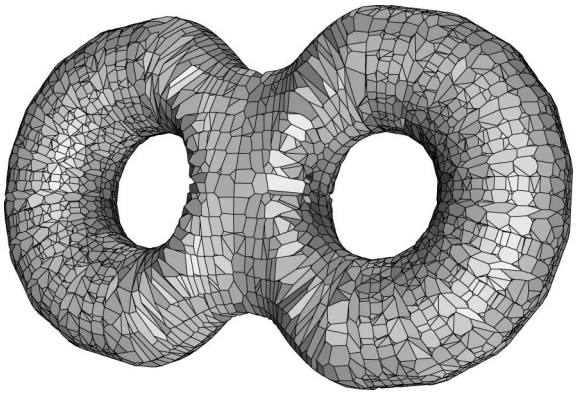
(b)

Figure 10. (a) Union of balls of a heart (3405 balls) (b) Surface reconstructed from the union of balls of a heart (3900 faces)

planar ones, incorporating such a conversion step into our method would be useful.



(a)



(b)

Figure 11. (a) Union of balls of a two-holed torus (5613 balls) (b) Surface reconstructed from the union of balls of a two-holed torus (7154 faces)

- We would like to do further work on the detection and handling of undersampled areas, as well as investigate the effects of such degeneracies on the topology of the reconstructed surface.

11. Acknowledgments

We gratefully acknowledge the reviewers' comments, which have significantly improved this paper. We would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the British Columbia Advanced Systems Institute (ASI) for their continuing support.

References

[1] N. Amenta, S. Choi, and R. Kolluri. The power crust. In *Proceedings of the ACM Symposium on Solid Modeling and Applications*, pages 249–260, Ann Arbor, Michigan, June 2001.

[2] N. Amenta and R. Kolluri. Accurate and efficient unions of balls. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 119–128, Hong Kong, June 2000.

[3] F. Aurenhammer. Power diagrams: properties, algorithms, and applications. *SIAM Journal on Computing*, 16(1):78–96, 1987.

[4] K. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Computational Geometry*, 3(4):185, Sept. 1993.

[5] O. Devillers. Improved incremental randomized Delaunay triangulation. In *Proc. Symposium on Computational Geometry*, pages 106–115, 1998.

[6] H. Edelsbrunner. The union of balls and its dual shape. In *Proc. 9th Annual ACM Symposium on Computational Geometry*, pages 218–231, 1993.

[7] H. Edelsbrunner. Deformable smooth surface design. *Discrete & Computational Geometry*, 21:87–115, 1999.

[8] H. Edelsbrunner and A. Üngör. Relaxed scheduling in dynamic skin triangulation. In *Proc. Japan Conference on Discrete and Computational Geometry*, Tokyo, Japan, Dec. 2002. Springer-Verlag.

[9] N. Gagvani and D. Silver. Shape-based volumetric collision detection. In *Proc. IEEE Volume Visualization and Graphics Symposium*, Salt Lake City, Utah, Oct. 2000.

[10] N. Gagvani and D. Silver. Animating volumetric models. *Graphical Models*, Mar. 2002.

[11] P. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996.

[12] N. Kruithof and G. Vegter. Approximation by skin surfaces. In *Proc. International Conference on Shape Modelling and Applications*, pages 86–95, Seattle, June 2003.

[13] J. Liang, H. Edelsbrunner, P. Fu, P. Sudharkar, and S. Subramaniam. Analytic shape computation of macromolecules I: molecular area and volume through alpha shape. *Proteins: Structure, Function, and Genetics*, 33:1–17, 1998.

[14] V. Ranjan and A. Fournier. Volume models for volumetric data. *IEEE Computer, Special Issue on Volume Visualization*, 27(7):28–36, 1994.