Geometric Modeling in CAVE

Herbert Edelsbrunner, Ping Fu, and Jiang Qian

Department of Computer Science and National Center for Supercomputing Applications,
University of Illinois at Urbana-Champaign, Illinois, USA.

Abstract

Virtual environments open up new opportunities and challenges for geometric modeling systems. A general approach to geometric modeling suitable for the Cave Automatic Virtual Environment is described. The approach is based on alpha complexes, and some of its capabilities are demonstrated by applying it to the study of biomolecules.

1 Introduction

We describe an approach to geometric modeling in virtual environments. Such environments are well suited for visualizing and exploring 3-dimensional volumetric data. Our goal is to provide state-of-the-art geometric modeling software that allows total immersion in the details of a rendered object. Both inside and outside views are supported. Our software has been developed in the CAVE, short for Cave Automatic Virtual Environment. One of the advantages of the CAVE over other virtual environments is that multiple viewers can observe the same scene at the same time and place. Our modeling software is scalable from high-end virtual environments, such as the CAVE, over mid-range immersive desks, down to low-end graphics workstations.

The implementation of our approach to modeling combines ideas and methods from geometry, topology, algorithms, and graphics. The product is a general modeling software that can be applied to questions studied in the sciences and in engineering. The application areas include computer aided geometric design (CAGD), rapid prototyping, medical imaging, geological modeling, grid generation and analysis, and physical modeling.

For the purpose of specificity, this paper concentrates on modeling macromolecules. These are studied intensely in biology, for the purpose of understanding how life functions on the molecular level, and for the semi-automatic design of drugs that can influence the mechanism of molecular life. Common geometric representations for proteins and other molecules are the *space filling* (SF), the *solvent accessible* (SA), and the *molecular surface* (MS) models. The SF model was introduced by Lee and Richards [10] and represents a protein as the union of possibly overlapping spherical balls with sizes determined by the van der Waals radii of the atoms. The SA model was introduced to study the interaction between a protein and a solvent, itself modeled as a ball [12]. The solvent is deflated to a point and all balls are inflated by the same radius. The MS model is obtained by rolling the solvent ball over the SF model [12]. It is useful in studying the structure of and interaction between proteins.

A representation of the molecule that is less direct than the above sphere models consists of simplices connecting atom centers. The simplices are derived from the Voronoi decomposition of the SF or the SA model [6]. The collection of simplices is referred to as a simplicial complex. The advantage of the complex over the sphere models is primarily computational. For example, with the complex it is possible to identify cavities instantaneously, whereas software working directly on sphere models has difficulties to find the cavities altogether. Furthermore, the complex is instrumental in the robust construction of a surface triangulation for any of the above sphere models [1, 2]. In contrast to previous work, see e.g. [3], the surface triangulation constructed by our software is topologically correct and can be exploited for numerical computations e.g. of the electrostatic potential field, see [14]. We use the surface triangulation to animate continuous deformations between the sphere and complex models of a molecule.

VRST'96 Hong Kong © 1996 ACM 0-89791-825-8/96/0007 3.50

These animations are visually striking and may be the only convincing means to convey the intricacies of the models and their relationships.

The paper is organized as follows. Section 2 introduces the geometric ideas underlying the software. Section 3 describes the geometric kernel of the software. Section 4 presents the hardware and software of the CAVE. Section 5 describes some of the more interesting graphics features of our software, including the above mentioned deformations. Section 6 concludes the paper and mentions sites at which software described in this paper is available.

2 Geometric Foundations

This section introduces some of the geometric concepts that constitute the foundations of our software. These are the Voronoi diagram and the Delaunay complex of a set of points with weights, and the alpha complexes forming a nested sequence of subcomplexes of the Delaunay complex.

Voronoi Diagram. Given a finite set of points, possibly with weights, the *Voronoi diagram* decomposes space into convex cells, see figure 1. Each cell corresponds to a point, z, and contains all locations in space for which z is the closest in the given finite set. In 3D, a Voronoi cell is a convex polyhedron and its boundary consists of faces, edges, and vertices. The Voronoi cells have disjoint interiors and overlap at most along common boundary pieces. Exactly how space is decomposed depends on the notion of distance. We use a weighted distance function that accounts for the interpretation of input points as centers of balls or atoms of a molecule. Consider a point z, which is the center of an atom with van der Waals radius r. Then the *weighted distance* of any arbitrary point x from z is $\pi_z(x) = |x - z|^2 - r^2$. If $\pi_z(x)$ is positive then x lies outside the sphere with center z and radius r and $\pi_z(x)$ is the square-length of the tangent from x to the sphere. The *bisector* of two points, y and z, is the set of points x with equal weighted distance from y and from z. It turns out the bisector is a plane normal to $y - z$. Every Voronoi cell is bounded by pieces of bisectors.

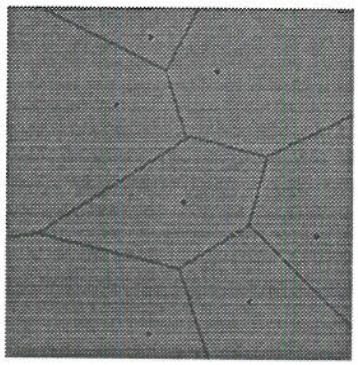


Figure 1: The Voronoi diagram of a set of 7 points in 2D.

Delaunay Complex. Intuitively, the *Delaunay complex* decomposes the convex hull of a set of points into simplices mimicking the neighborhood structure of the Voronoi cells. The following rules specify the collection of simplices. Every Voronoi cell corresponds to a vertex in the complex, and this vertex is located at the data point that generates the cell. The common face of two Voronoi cells is represented by the edge connecting the two corresponding vertices. The common edge of three Voronoi cells is represented by the triangle spanned by the three corresponding vertices. Finally, the common point of four Voronoi cells is represented by the tetrahedron spanned by the four corresponding vertices. Assuming general position of the points in 3D, it is impossible that five or more Voronoi cells share some common boundary. Figure 2 illustrates the duality between the Voronoi diagram and the Delaunay complex in the 2-dimensional case.

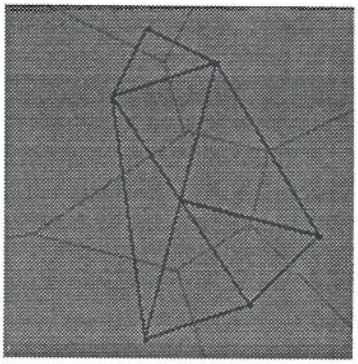


Figure 2: Superposition of the Voronoi diagram in figure 1 and the corresponding Delaunay complex.

The Delaunay complex can be constructed directly from the Voronoi diagram. Indeed, the Delaunay complex contains the same combinatorial information as the Voronoi diagram, only the geometric expression of the information is different. An important consequence of this observation is that algorithms for Delaunay complexes implicitly construct Voronoi diagrams and vice versa. It seems easier to design a robust algorithm for Delaunay complexes. The main reason is that the Delaunay complex comprises no new geometric information, and all edges, triangles, and tetrahedra can be stored combinatorially as pairs, triplets, and quadruplets of vertex indices. In contrast, the Voronoi diagram contains vertices that are not part of the input data.

Alpha Complexes. Each alpha complex is a subcomplex of the Delaunay complex. The family of alpha complexes can be defined by sorting the Delaunay simplices and taking prefixes of the sorted sequence. Consider the following scenario. For each point or atom z, let $r_z(0)$ be its van der Waals radius. Use a parameter α to grow or shrink the balls centered at the points z. For a given real number α^2, the radius of the ball with center z is

$$r_z(\alpha) = \sqrt{r_z^2(0) + \alpha^2}.$$

A most important property of the Voronoi diagram is that it decomposes the union of balls into convex pieces, called *clipped balls*, see figure 3. Indeed, this is true for every real value of α^2 because changing α^2 does not change any bisectors.

To obtain the sorted sequence of Delaunay simplices, imagine α^2 going from minus infinity to plus infinity. At the beginning, every ball has imaginary radius. A vertex z is appended to the emerging sequence at the time its clipped ball becomes non-empty. In typical but not all cases this is when the radius, $r_z(\alpha)$, reaches zero. The balls grow as the value of α^2 increases. At the moment when two clipped balls meet, we append the edge connecting their two centers to the emerging sequence. Similarly, when three clipped balls meet we append the corresponding triangle, and when four clipped balls meet we append the corresponding tetrahedron. See figure 3 for a 2-dimensional illustration of the sorting process.

When α^2 is large enough, the sequence contains all Delaunay simplices. The simplicial complex associated with a given value of α is a subcomplex of the Delaunay complex and is referred to as the *α-complex* [8]; it corresponds to a prefix of the sorted sequence of simplices.

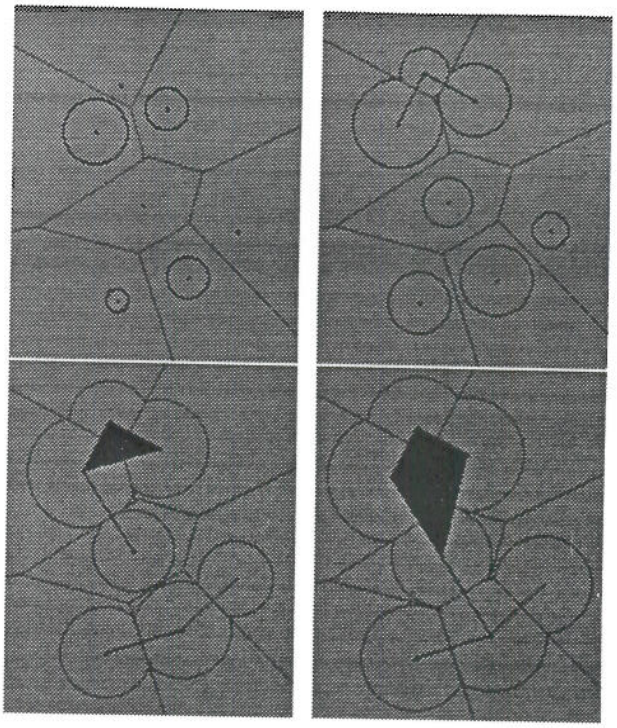


Figure 3: Simplices form when clipped disks appear and when they meet.

3 Geometric Kernel

The geometric kernel consists of two pieces of core software: one for alpha complexes and one for surface triangulations. The alpha complex is used as the underlying data structure from which more involved or more elaborate geometric models are derived. Specifically, we discuss how surface triangulations of the SF, SA, and MS models can be computed from the alpha complex. The geometric integrity of the alpha complex makes it possible to produce topologically correct surface triangulations.

Alpha Shape Library. We have implemented alpha complexes as a software library that provides geometric primitives and can be linked to any application software. The main two data structures are the Delaunay complex and the sequence of Delaunay simplices which contains all alpha complexes as prefixes.

In 3D, we represent a Delaunay complex by a triangle-based pointer structure [5]. Each triangle stores pointers to the 6 neighboring triangles sharing an edge. Following appropriate pointers, each in constant time, it is possible to traverse the triangles around a given edge, or the triangles opposite a given vertex, or all triangles on the convex hull boundary. Details can be found in [11]. An important ingredient in the construction of a Delaunay complex, which is synonymous to constructing its triangle-based pointer structure, is the use of exact arithmetic and symbolic computation. The input coordinates and radii are restricted to integers or fixed-point reals. All geometric tests are performed in exact arithmetic so that degenerate cases can be identified without ambiguity, see [7] for details.

The linear list representation of the simplex sequence contains slightly more information than just the sequence of simplices as they enter the alpha complex. Each simplex occurs up to three times: first when it enters the alpha complex, second when the first simplex containing it as a face enters the alpha complex, and third when it becomes completely surrounded by simplices. After the first occurrence the simplex is *singular*, after the second it is *regular*, and after the third it is *interior*. Quite commonly some of the occurrences coincide or vanish. For example, a simplex on the convex

hull boundary is never interior, and an entering tetrahedron is right away interior. The additional information available through the multiple occurrences is e.g. useful in selecting the simplices needed for a graphical representation. Only the boundary triangles (singular and regular), the singular edges, and the singular vertices need to be drawn.

Surface Triangulation Library. This library is an implementation of the algorithm for molecular surfaces triangulation proposed in [2]. The algorithm uses convex polyhedral approximations of spheres computed [1]. Starting with a random distribution of finitely many points on a sphere, a good approximation is created by an iterative process that moves points in an attempt to maximize the surface area of the convex hull. Approximations of sphere patches, as opposed to spheres, are computed as follows. The patch is bounded by cycles of arcs which are approximated by cycles of edges. A given sphere approximation is modified so it contains the cycles of edges in its boundary. Finally, a depth-first traversal of the modified sphere approximation accepts triangles in the patch and rejects other triangles.

The most time-consuming step in the entire computation is the triangulation of all patches. The performance of this step is improved by using parallel algorithms and hardware. Once we know all the arc cycles bounding patches on a given sphere, these patches are triangulated independently of any other spheres. We distribute computations equally over p processors. In the end, we collect all triangles constructed in approximating the various sphere patches and connect them into a triangle-based data structure that reflects connectivity in a way similar to the quad-edge data structure of Guibas and Stolfi [9]. Local navigation is possible in constant time per step simply by following adjacency pointers to neighboring triangles. These pointers glue triangles along shared edges and patches along shared arcs.

The patch computation is done on an SGI power challenge parallel machine, while graphics and other computations are done on a workstation. The required communication between the workstation and the parallel machine uses DTM, short for data transfer mechanism. This is a message passing facility designed to simplify the task of interprocess communication. It provides methods for interconnecting applications at run-time and reliable message passing complete with synchronization and transparent data conversion.

4 Cave Automatic Virtual Environment

The geometric models are constructed in the geometric kernel of the software and rendered in CAVE. The visualization software, `VAlvis`, short for Virtual Alpha Shapes Visualizer, is in many ways similar to the better known desktop version, `Alvis`. This section gives a brief description of the immersive virtual environment architecture of CAVE and of its programming environment.

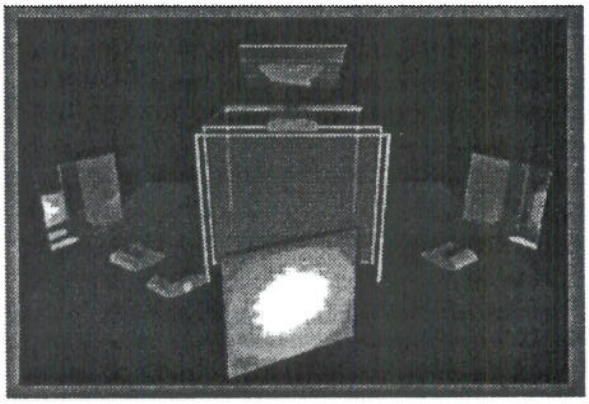


Figure 4: CAVE architecture.

CAVE Architecture. The CAVE [4] is a surround-screen, surround-sound, projection-based virtual reality (VR) system. The illusion of immersion is created by projecting 3D computer graphics pictures into a 10 by 10 by 9 feet cube defined by display screens that completely surround the viewer. Electrohome Marquis 8000 projectors throw full-color workstation fields (1024 by 768, stereo) at 96 Hz onto the screens, giving approximately 2,000 linear pixel resolution to the surrounding composite image. Computer-controlled audio provides a sonification capability to multiple speakers. The theater area of the CAVE can fit inside a 30 by 20 by 13 feet light-tight room, provided the projector optics are redirected with mirrors. A user's head and hand are tracked with Ascension tethered electromagnetic sensors. The viewer explores the virtual world by moving around inside the cube and manipulating objects with a 3D input device, or *wand*.

CAVE users do not need to wear helmets to experience VR. Instead, they put on lightweight stereo glasses and walk around inside the CAVE as they interact with virtual objects. At any given instance, one user is the active viewer, defining the stereo projection reference point, while the other users are passive viewers. Multiple viewers can share the same virtual experience and verbal interaction in the CAVE is easily possible.

CAVE Programming Environment. Application programs using the *CAVE library* run on a multiprocessor SGI Onyx workstation. This machine performs all CAVE tasks. When application programs are executed, they are automatically forked into several concurrent processes. There is an independent rendering process for each projection wall, plus another process for all additional computations. Any data that might change during the execution of the program needs to be put in shared memory to ensure that all graphics and other computations refer to the properly updated data.

For each frame of animation, the stereo image is created by generating two views, one for the left eye and one for the right eye. This implies that each rendering process is called twice, and it is generally advised that expensive computations are separated from all rendering processes.

The CAVE library provides all the functions that are necessary to create a CAVE program. It liberates the developer from dealing with low level tasks, such as the synchronization of CAVE devices, the computation of stereo transformations, etc. Due to the cost and spatial constraints of the CAVE it is impractical to assemble a CAVE for every researcher interested in using it. The *CAVE simulator* bridges the gap and runs CAVE application programs on any SGI workstation that supports SGI's IrisGL library. The simulator makes it possible to develop CAVE applications at remote sites. The *CAVE viewer* is similar to the CAVE simulator except that it provides a set of Motif controls and can dynamically load the application specific code for a CAVE program. Its main purpose is the distribution of CAVE application demos through Mosaic.

5 Graphics Features

This section discusses a few graphics and geometry based features of our modeling software. The first topic is the continuous deformation between different molecular models. The animation of these deformations is visually striking and illuminates relationships between the models. The second topic is the construction of exploded views. Additional topics are the `VAlvis` interface, real-time high quality shading, and fast sphere clipping.

Deforming SA to Complex. The deformation is governed by a time parameter, $t \in [0,1]$. $t = 0$ corresponds to the SA model, and $t = 1$ corresponds to the appropriate alpha complex. The surface of the SA model consists of sphere patches, circular arcs, and corner points. As t goes from 0 to 1, each spherical patch shrinks towards a vertex of the alpha complex. Each circular arc shrinks and widens to form a cylindrical section that eventually becomes an edge of the alpha complex. Each corner point of the SA model grows towards a triangle of the alpha complex. Recall that the SA surface is triangulated, so each patch is approximated by a collection of triangles, each arc is approximated by a sequence of edges, and each corner point is a vertex.

The deformation can be specified by prescribing the motion of the vertices in the SA triangulation. We use a straightforward linear motion determined by the starting and ending positions. Denote the vertices of the SA triangulation by S_i and the vertices of the alpha complex by A_j. Each motion follows the formula

$$(1-t)\cdot S_i + t\cdot A_j.$$

In the case of a triangle on an SA patch the three vertices, S_1, S_2, S_3, move linearly towards a vertex, A_1, of the alpha complex. The deformation of the triangle is implied by the motion of its vertices. In the case of an edge on an SA arc, the two vertices, S_1 and S_2, move simultaneously and linearly towards two vertices of the alpha complex, A_1 and A_2. For any t strictly between 0 and 1 we have 4 points forming a rectangle, which is part of the shrinking and widening cylinder that eventually collapses to an edge of the alpha complex. A corner point, S_1, moves simultaneously towards three vertices of the alpha complex, A_1, A_2, and A_3. Figure 5 shows snap-shots of the deformation from the SA model to the alpha complex of the Gramicidin A protein.

Deforming SA to MS. The MS model is smoothed by rolling a solvent ball over the SF model. Its surface consists of sphere patches, torus patches, and inverse sphere patches. As the time parameter t goes from 0 to 1, each sphere patch of the SA surface shrinks towards a smaller but otherwise identical sphere patch of the MS surface. Each arc of the SA surface shrinks and widens to a torus patch. Each corner point of the SA surface grows towards an inverse sphere patch.

Again we use a straightforward linear motion that deforms the SA triangulation into a triangulation of the MS model. The motion of each vertex is determined by its starting and ending positions. Denote the vertices of the MS triangulation by M_j. Each motion follows the formula

$$(1-t)\cdot S_i + t\cdot M_j.$$

Figure 6 shows snap shots of the deformation from the SA model to the MS model of the Gramicidin A protein.

Exploded View. An exploded view is obtained by moving the pieces of a geometric shape or complex apart. Such a view can be very informative and reveal a

great deal about how the pieces fit together. We define a continuous exploding motion so the user can animate an explosion or implosion with growing or shrinking gaps. Such an animation is especially impressive in the CAVE where it enhances the feeling of immersion.

The exploding motion is defined by another time parameter $e \in [0, +\infty)$. $e = 0$ corresponds to the unexploded state. To specify the motion choose a center, C_i, for each piece, P_i. The motion moves the center away from the origin, and with the center it moves the piece. More formally, the piece at time e is

$$P_i + e \cdot C_i.$$

Figure 7 illustrates the explosion for a small collection of carbon atoms arranged in a diamond lattice. At the left, the surface of the overlapping balls is shown in the unexploded state, and in the middle the sphere patches of the surface are moved apart.

It is important to choose the centers so that collisions between different moving pieces are avoided. In the case of a simplicial complex this is achieved by choosing C_i equal to the barycenter of the simplex P_i. To see that the resulting explosion avoids collisions consider the deformation

$$(1-t) \cdot P_i + t \cdot C_i,$$

for $t \in [0, 1]$, which contracts P_i to the point $C_i \in P_i$. There are no collisions if simplices are contracted simultaneously, each towards its own center. The exploding motion is obtained from the contraction by scaling the entire space:

$$\frac{(1-t) \cdot P_i + t \cdot C_i}{1-t} = P_i + \frac{t}{1-t} \cdot C_i.$$

It follows the explosion is also free of collisions. We use the same barycenters for motions that explode SA, MS, and partially deformed models. There are three cases.

(i) A vertex of the alpha complex corresponds to sphere patches of the SA model, of the MS model, and of partially deformed models. The centers of all these spheres coincide with the vertex.

(ii) An edge of the alpha complex corresponds to torus patches of the MS and SA-to-MS deformed models and to cylinder patches of SA-to-complex deformed models. The barycenter is the midpoint of the edge.

(iii) A triangle of the alpha complex corresponds to inverse sphere patches of the MS and SA-to-MS deformed models and to triangles of SA-to-complex models. The barycenter of the triangle with vertices A_1, A_2, A_3 is $\frac{1}{3}(A_1 + A_2 + A_3)$.

The reason why these choices of explosion centers avoid collisions have to do with properties of Voronoi cells and arguments are omitted.

User Interface. An event driven Motif-like interface is incorporated into the software to give the user flexibility and control over the system. The widgets are drawn on the front wall of the CAVE, and the user can select different options by pointing and clicking with the wand.

A position on the front wall is specified by pointing with the wand, and the position is reconstructed from the location and the direction vector available from the CAVE library. The appropriate callbacks can be implemented based on this information. In addition to interacting with `VAlvis` through the interface, we use the three buttons on the wand and its orientation, angle, and direction to control `VAlvis`.

Shading and Clipping. Real-time texture mapping capabilities of high-end graphics workstations can be exploited to compute Phong shading in real-time [13]. The normals of the vertices are computed the same way as for Gouraud shading, but rather than to constructing an interpolating surface, the normals are used to automatically generate texture coordinates. In this context, the texture map is the image of a sphere perfectly rendered with traditional Phong shading. Consider a triangle of the surface. Its three vertices define three normals, and the corresponding three texture coordinates identify a portion of the texture map used to give the triangle the appropriate shading. The result is a map from the surface of the geometric model to a perfectly shaded sphere that creates the illusion of a Phong shaded surface in real-time.

A unique feature of our software is the generation of uncluttered inside views of SA and MS models. Indeed, there are other software packages that generate high quality renderings of views from the outside, but uncluttered views from the inside require that all spheres and tori are appropriately clipped. Voids and tunnels can be detected by walking into the model and viewing the surface from the inside. In CAVE, the user feels the complete immersion into the model. Figure 8 shows views from the inside of an SA model, an MS model, and the voids of an MS model.

6 Discussion

There is an ongoing debate on whether the high cost of virtual environments can be justified in the face of very affordable high quality graphics interfaces. We wish to contribute a piece of anecdotal evidence for the claim

that the immersive experience indeed generates new insights and discoveries. After presenting this evidence, we summarize the contributions of this paper and give pointers to distribution sites for the software.

Self-intersections of the molecular surface. Consider the MS model of a molecule. It is well known that this surface can have self-intersections, but these are often discarded as "degeneracies". It was only in the CAVE that we noticed the seriousness and frequency of such self-intersections. The easily imagined first type of self-intersection involves pieces that are fairly far apart along the surface: the solvent ball peeking inside-out and outside-in through a narrow window. Example of such self-intersections can be seen in figure 6.

There is another and apparently more frequent second type of self-intersection resulting from torus-sweeping motions of the solvent ball. If the radius of the sweeping ball is larger than the radius of the circle swept by its center then the torus intersects itself. In most cases, only a small patch of such a torus is part of the surface. Locally, the surface sharply folds backward and again forward.

Contributions of this paper. A general approach to geometric modeling in the CAVE is described. The approach is based on alpha complexes and applications specific to macromolecules are used to demonstrate its capabilities. All algorithm mentioned in this paper are implemented, with an option to run time-consuming tasks on a parallel architecture.

The visualizer, `VAlvis`, is used to display all geometric concepts and structures in the CAVE. It is based on two geometric software packages, the Alpha shape library and the surface triangulation library. The CAVE library is responsible for rendering and tracking in the CAVE. The Alpha shape library can be obtained via ftp at `ftp.ncsa.uiuc.edu` from directory `Visualization/Alpha-shape`. The CAVE library is available via ftp at `evl.eecs.uic.edu` from directory `pub/CAVE`. Copies of `VAlvis` and the surfaces triangulation library are not available via ftp and can be requested by sending email to `alpha@ncsa.uiuc.edu`.

Acknowledgments

We thank Ernst Mücke and Michael Facello for their work on the Alpha shape library, and Nataraj Akkiraju for his work on the surface triangulation library. We thank NCSA for providing access to the CAVE.

References

[1] N. Akkiraju. Approximating spheres and sphere patches. Manuscript, 1994.

[2] N. Akkiraju and H. Edelsbrunner. Triangulating the surface of a molecule. *Discrete Appl. Math.*, to appear.

[3] M. L. Connolly. Molecular surface triangulation. *J. Appl. Cryst.* **18** (1985), 499–505.

[4] C. Cruz-Neira, D.J. Sandin, T.A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the CAVE. *Proceedings of SIGGRAPH'93* (1993), 135–142.

[5] D. P. Dobkin and M. J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. *Algorithmica* **4** (1989), 3–32.

[6] H. Edelsbrunner. The union of balls and its dual shape. *Discrete Comput. Geom.* **13** (1995), 415–440.

[7] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graphics* **9** (1990), 66–104.

[8] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graphics* **13** (1994), 43–72.

[9] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graphics* **4**, (1985). 74–123.

[10] B. Lee and F. M. Richards. The interpretation of protein structures: estimation of static accessibility. *J. Mol. Biol.* **55** (1971), 379–400.

[11] E. P. Mücke. *Shapes and Implementations in Three-dimensional Geometry.* Ph. D. thesis, Dept. Comput. Sci., Univ. Illinois, Urbana, 1993.

[12] F. M. Richards. Areas, volumes, packing, and protein structure. *Ann. Rev. Biophys. Bioeng.* **6** (1977), 151–176.

[13] M. Teschner, C. Henn, H. Volhardt, S. Reiling, and J. Brickmann. Texture mapping: A new tool for molecular graphics. *J. Mol. Graphics* **12** (1994), 98.

[14] R. J. Zauhar and R. S. Morgan. Computing the electric potential of biomolecules: application of a new method of molecular surface triangulation. *J. Comput. Chemistry* **11** (1990), 603–622.

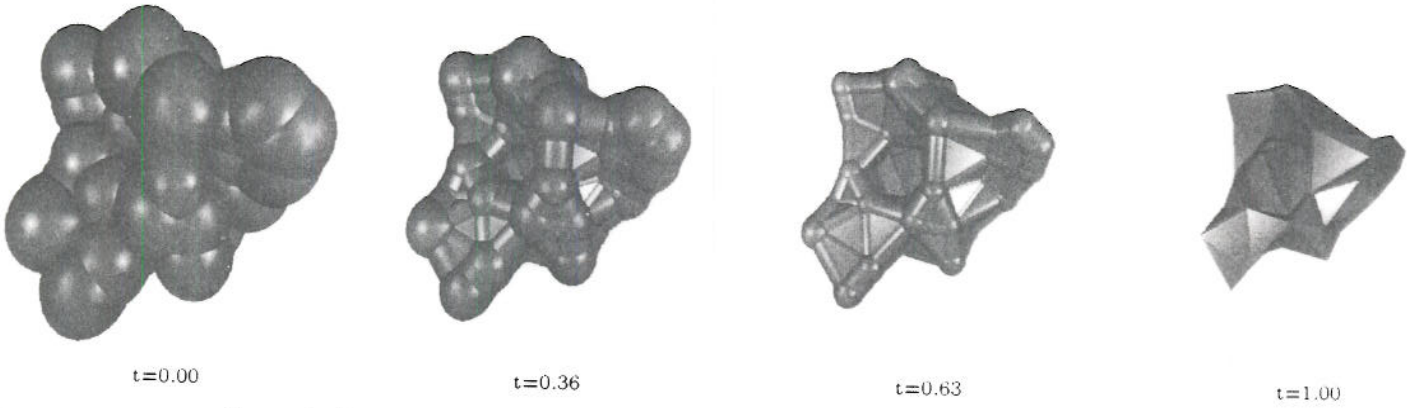


Figure 5: Four snap-shots during the deformation of the SA model into the alpha complex.

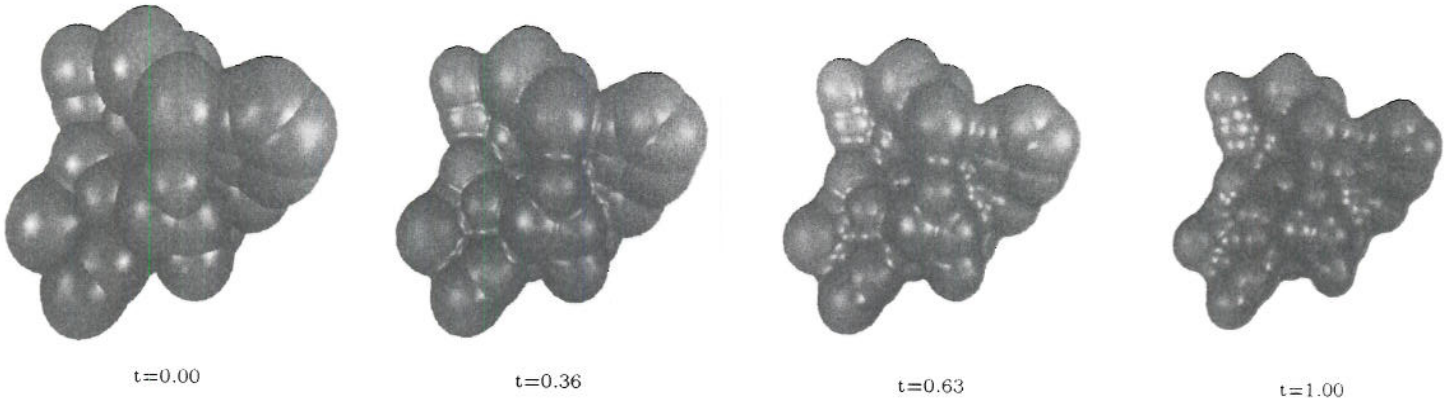
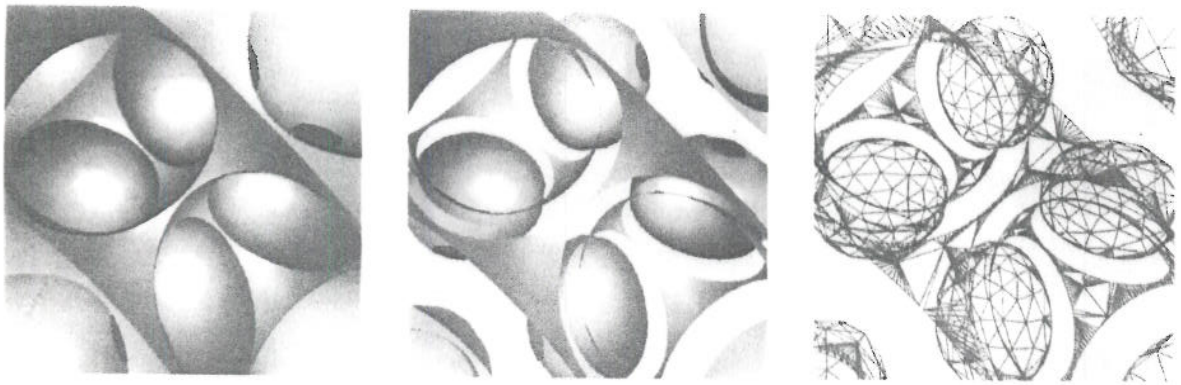


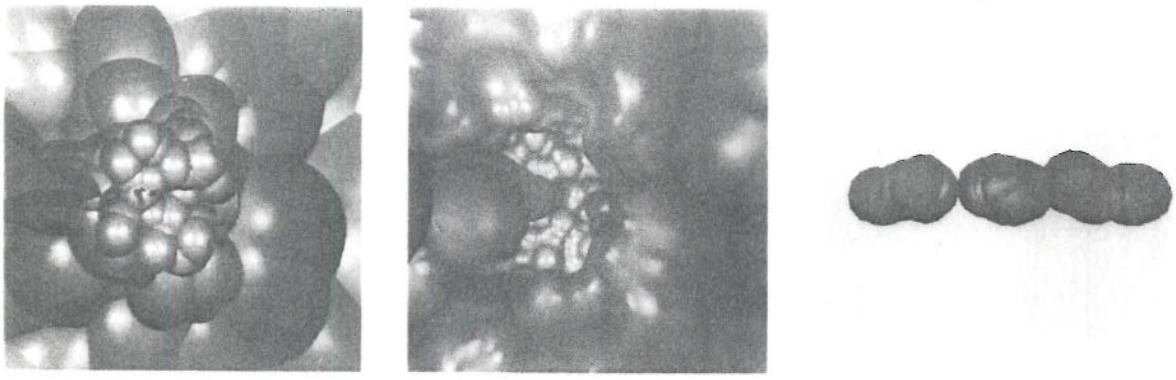
Figure 6: Four snap-shots during the deformation of the SA model into the MS model. Observe that the spheres shrink from left to right. There seems to be one exception to this trend, namely the growing bulge roughly in the middle of the picture. The reason for this anomaly will be discussed in section 6.

Colour Plates for Edelsbrunner, Fu, and Qian (page 35)



a) b) c)

Figure 7: a) Inside view of the union of atoms arranged in a diamond lattice. b) Exploded view of the surface. c) Wireframe of the exploded triangulation.



SA model MS model voids in the MS model

Figure 8: Inside views of the Gramicidin A protein.

Colour Plates for Edelsbrunner, Fu, and Qian (page 35), continued