Invited submission, *IEEE Computer Graphics and Applications*, September, 1994

Linearly Scalable Computation of Smooth Molecular Surfaces

Amitabh Varshney Frederick P. Brooks, Jr. William V. Wright

Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175

Abstract

An algorithm for fast computation of Richards's smooth molecular surface is described. Our algorithm is easily parallelizable and scales linearly with the number of atoms in a molecule.

1 Introduction

The smooth molecular surface of a molecule is defined as the surface which an exterior probe-sphere touches as it is rolled over the spherical atoms of that molecule. This definition of a molecular surface was first proposed by Richards [11]. This surface is useful in studying the structure and interactions of proteins, in particular for attacking the protein-substrate docking problem. For examples of such molecular surfaces, refer to Figures 5 – 10, where these surfaces have been shown for various molecules and with different probe-sphere radii.

Present systems for computing the surfaces of molecules are batch-oriented. They take a few minutes to compute the surface for a couple of thousand atoms. Our goal has been to compute and display these surfaces at interactive rates, by taking advantage of results from the field of computational geometry, making further algorithmic improvements, and parallelizing the computations.

Interactive computation and display of molecular surfaces should benefit biochemists in three important ways. First, the ability to change the probe-radius interactively helps one study the surface. Second, it helps in visualizing the changing surface of a molecule as its atom positions are changed. These changes in atom positions could be due to user-defined forces as the user attempts to modify a molecular model on a computer. Third, it assists in incorporating the effects of the solvent into the overall potential energy computations during the interactive modifications of a molecule on a computer.

The rest of this paper is organized as follows. In Section 2 we briefly review some of the previous work that has been done in computation of smooth molecular surfaces as well as some work that has been done in related areas, particularly computational geometry. Then in Section 3 we outline our approach. In Section 4 we estimate the average number of "neighbors" for an atom in any protein molecule. Section 5 presents our results. Finally, we present our conclusions and some promising areas deserving further research in Section 6.

2 Previous and related work

The analytic computation of the molecular surface was first done by Connolly [2]. Here a molecular surface is represented by a collection of spherical and toroidal patches as follows:

- The surface for a region of a molecule where the probe is in contact with only a single atom is modeled by a convex spherical patch.
- The surface for a region of a molecule where the probe is in simultaneous contact with only two atoms is modeled by a saddle-shaped toroidal patch.
- The surface for a region where the probe is in simultaneous contact with three atoms is modeled by a concave spherical triangular patch.

Only recently have the issues of algorithmic complexity of these algorithms begun to be addressed. Let n be the number of atoms in a molecule and let k be the average number of *neighboring* atoms for an atom in the molecule. By *neighboring* we mean the atoms that are near enough to affect probe placement on a particular atom. Perrot *et al.* [9] present a $O(kn)$ algorithm that generates an approximation to the solvent-accessible surface. In terms of sequential algorithmic complexity this is good, however some points remain unaddressed here. This algorithm is inherently sequential, as it always needs to start from some concave spherical triangular region of the molecule and from there it proceeds by adding an adjacent face at a time. Besides being hard to parallelize, it fails for the cases where the solvent-accessible surface folds back to intersect itself or where the molecule has two or more sub-parts connected by only two overlapping spheres. Also, it cannot generate the interior cavities of a molecule.

In computational geometry, the α-hull has been defined as a generalization of the convex hull of point-sets by Edelsbrunner, Kirkpatrick, and Seidel [5]. For $\alpha > 0$, the α-hull of a set of points P in two dimensions is defined to be the intersection of the closed complements of all discs with radius α whose closed complement contains all points of P. If we generalize this notion of α-hulls over point-sets to the corresponding hulls over spheres of unequal radii in three dimensions, we would get the Richards's smooth molecular surface (along with the surfaces defining the interior cavities of the molecule). It has been shown in [5] that it is possible to compute the α-hulls from the Voronoi diagram of the points of P. For $\alpha = \infty$ the α-hull over the set of points P is the same as their convex hull. Richards [11] had also suggested computing the molecular surface by computing a 3D Voronoi diagram first and then using its faces to determine which nearby atoms to consider.

Edelsbrunner and Mücke [6] extend the definition of α-hulls to points in three dimensions. Here an *α-shape* over a set of points P has been defined to be the polytope that approximates the α-hull over P by replacing circular arcs of the α-hull by straight edges and spherical caps by triangles. An α-shape of a set of points P is a subset of the Delaunay triangulation of P. Edelsbrunner in [4], extends the concept of α-shapes to deal with weighted points (i.e. spheres with possibly unequal and non-zero radii) in three dimensions. An α-shape of a set of weighted points P_w is a subset of the regular triangulation of P_w. Since these methods involve computing the entire triangulation first and then culling away the parts that are not required, their complexity is $O(n^2)$ in time. This is worst-case optimal, since an α-shape in three dimensions could have a complexity of $\Omega(n^2)$. We next discuss a different approach that is easy to parallelize and works better for environments where the maximum density of P in a given volume is some constant smaller than n. Molecules are a good example of such environments.

3 Our approach

Our goal has been to formulate a parallel analytical molecular surface algorithm that has expected linear complexity with respect to the total number of atoms of a molecule. For achieving this goal, we have avoided computation of the complete three-dimensional regular triangulation over the entire set of atoms — a process that takes time O(n^2), where n is the number of atoms in the molecule.

Let us consider a molecule as a collection of weighted points (c_i, r_i) in three dimensions, where the coordinates c_i of each point correspond to the center of atom i and the weight r_i is the radius of atom i. Such collections of weighted points representing molecules have two interesting properties: (i) the minimum distance d_{ij} between any two centers c_i and c_j is greater than or equal to a positive constant l_{min} — the smallest bond-length in the molecule and (ii) the set of all the weights can be bounded from above and below by strictly positive values, $0 < r_{min} \leq r_i \leq r_{max}$. We take advantage of the first property to arrive at better running times for our algorithm. Stated simply, the first property says that the number of neighboring atoms within a fixed distance from any atom i, is always bounded from above by a constant k_{max} that depends on the minimum spacing between any two atoms. If the average number of neighbors for an atom is k, then we can just compute an approximation to the power cell (the concept of a power cell is presented in [1] and briefly reviewed in Section 3.1), which we call a *feasible cell* (the definition of a feasible cell appears in Section 3.3), by considering only these neighbors. Each feasible cell can be computed in parallel in time O($k \log k$). For n atoms, this task requires n processors, each processor computing the feasible cell for one atom.

3.1 Formal notation

In this section we will introduce the definitions and notations that we will be using for the rest of the paper. We consider the underlying space to be three-dimensional Euclidean Space $\Re^3$, although these results can be generalized to higher dimensions.

Let $\sigma(c, r)$ be a sphere of center c and radius r. Let x, y be two points. Define $d(x, y)$ to be the Euclidean distance between x and y. The *power of a point* x with respect to a sphere $\sigma(c, r)$ is defined as $p(x, \sigma) = d^2(x, c) - r^2$ Thus, $p(x, \sigma) < 0, = 0, > 0$, depending on whether x lies inside σ, on the boundary of σ, or outside σ, respectively.

Let $M = \{S_1, \ldots, S_n\}$, be a set of spheres, where each sphere, S_i, is expressed as $\sigma(c_i, r_i)$. We shall be assuming that the atom i of a molecule is represented by the sphere S_i and will be using the terms *atom i* and S_i interchangeably. Let the radius of the probe-sphere be R. We define the *extended-radius sphere* for atom i to be $\Psi_i = \sigma(c_i, r_i + R)$. The surface of this extended-radius sphere Ψ_i is the locus of the possible centers of the probe-sphere when it is in contact with atom i.

Define a *chordale* Π_{ij} of the spheres Ψ_i and Ψ_j as $\Pi_{ij} = \{x | p(x, \Psi_i) = p(x, \Psi_j)\} = \{x | 2x(c_j - c_i) = {r_i}^2 - {r_j}^2 - {c_i}^2 + {c_j}^2 - 2R(r_j - r_i)\}$. Thus, Π_{ij} is a plane perpendicular to the line joining c_i and c_j. Define the halfspace H_{ij} as $H_{ij} = \{x | p(x, \Psi_i) < p(x, \Psi_j)\}$. The chordale Π_{ij} divides the whole space into two halfspaces. H_{ij} is that half-space in which all points have a smaller power with respect to Ψ_i than Ψ_j. In other words, all points selected by H_{ij} are closer to c_i than c_j under the distance function defined by the power metric (as defined by the power function above), instead of the conventional Euclidean metric. Thus, whereas $\Pi_{ij} = \Pi_{ji}$, $H_{ij} \neq H_{ji}$.

Define the *power cell*, PC_i, for atom i as $PC_i = \cap_j H_{ij}$. Thus PC_i is the set of all the points that are closer to c_i than any other sphere center c_j, assuming that the distance is measured in the power metric. The definition and algorithms for computing power cells have been given by Aurenhammer in [1].

3.2 Determination of neighboring atoms

Determination of neighboring atoms can be done by spatial grid subdivision into cubical voxels, and assigning atoms to the appropriate voxels. We recall that an atom j is considered a *neighbor* to atom i if it is possible to place a probe such that it is in contact with both S_i and S_j (without considering any hindrance due to other atoms). We define the *region of influence*, ρ_i, for atom i to be the sphere $\sigma(c_i, r_i + 2R + \max_{j=1}^{n} r_j)$. Then for computing the list of neighboring atoms, N_i, for atom i, one needs to find all the atoms that are close enough to affect probe placement on atom i. Formally, $N_i = \{j | d(c_i, c_j) \leq r_i + 2R + r_j\}$, or equivalently, $N_i = \{j | \Psi_i \cap \Psi_j \neq \phi\}$. The centers of all atoms whose indices occur in N_i lie inside the sphere ρ_i. Formally, $\forall j \epsilon N_i, p(c_j, \rho_i) \leq 0$. Therefore to compute the list of neighboring atoms for atom i, one needs to look at all the atoms whose centers lie in the voxels that intersect ρ_i. Let the average number of neighboring atoms be k. Note that k grows as R^3 assuming that the atoms are uniformly distributed. In Figure 1 atoms j_1 and j_2 are neighbors to atom i, but not to each other.

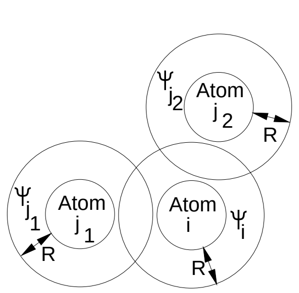

Figure 1: Defining Neighbors

3.3 Determination of surface atoms

Here the aim is to determine the atoms that are buried in the interior of the molecule and would not therefore directly participate in the final definition of the smooth molecular surface. This step is not crucial to the linear time complexity of the overall algorithm but it helps in improving the execution times.

Let us first define a *feasible cell* F_i as $F_i = \cap_{j \epsilon N_i} H_{ij}$. We will refine this definition of a feasible cell later in this section. Since a power cell PC_i is defined as $PC_i = \cap_j H_{ij}$, it is easy to see that $PC_i \subseteq F_i$. This difference between power and feasible cells arises from the fact that for the construction of a feasible cell F_i we use only those halfspaces H_{ij} for which it is true that the extended-radius spheres Ψ_i and Ψ_j intersect. However, for forming the power cells PC_i, we use all the halfspaces H_{ij} regardless of whether Ψ_i and Ψ_j intersect or not.

In Figure 2, we show these differences for power cells and feasible cells defined over circles. The power cell PC_3 contains two edges and one vertex as does the corresponding feasible cell F_3. However, whereas the power cells PC_1 and PC_2 have two edges and one vertex each, the corresponding feasible cells F_1 and F_2 have only one edge each, with no vertices.

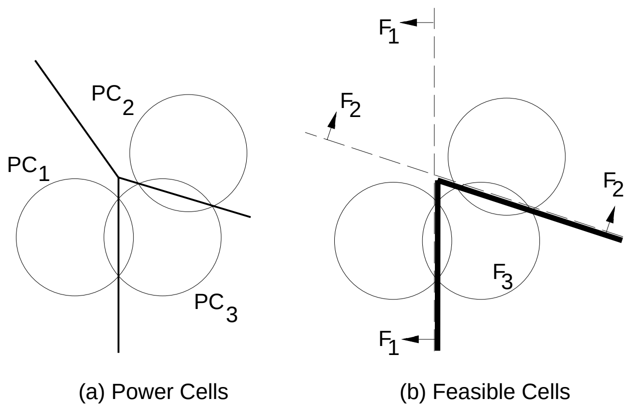

Figure 2: Power Cells and Feasible Cells

As the above example shows, it is possible to get feasible cells that are not bounded. However, it is attractive to have all the F_i closed and bounded. This compactness property of F_i enables one to use the vertices of F_i in computing a tessellation of the molecular surface. We plan to describe the details of this tessellation process elsewhere; it is outside the scope of this paper.

To make all F_i closed and bounded, we first construct a tetrahedron T that encloses the entire molecule. Let each face f of T lie in a plane Π_f. Every such plane Π_f can be considered to be defining two halfspaces, one that includes the molecule and the other that does not. Let $H_f, 0 \leq f \leq 3$ be the four halfspaces, one due to each face f of T, that select the molecule. We include H_f with the set of halfspaces H_{ij} that are used in defining $F_i, \forall i$. With this modification we are now ready to give the final definition of F_i as: $F_i = (\cap_{j \epsilon N_i} H_{ij}) \bigcap (\cap_{f=0}^{3} H_f)$.

With the matter of the definition of F_i having been settled, we can now determine the surface atoms as follows. First, for the entire molecule we compute $H_f, 0 \leq f \leq 3$. Next, for every atom i, we first compute N_i as described in Section 3.2. Then we compute $F_i = (\cap_{j \epsilon N_i} H_{ij}) \bigcap (\cap_{f=0}^{3} H_f)$. If $F_i = \phi$, atom i is totally buried and cannot be a surface atom. This checking for nullity is done by Seidel's randomized linear programming algorithm that has linear expected time and is quite fast in practice [12]. All the atoms for which $F_i \neq \phi$ are classified as candidates for being surface atoms.

3.4 Determination of surface patches

Determination of the vertices defining the convex spherical, concave spherical, and toroidal patches is the most crucial (and time-consuming) part of the whole algorithm.

If one computes a three-dimensional α-shape polytope for the set of atoms in a molecule, with α = probe-radius, then the torii occur along the edges, the concave spherical triangular patches correspond to the faces, and the convex spherical patches correspond to the vertices of this polytope. The method given by Edelsbrunner [4] finds these edges by first computing the entire three-dimensional regular triangulation, an $O(n^2)$ approach. We show here a method for computing the three-dimensional α-hull, for a given value of α, for molecules in parallel time $O(k \log k)$ over n processors

To compute F_i, we compute the convex hull of the points dual to the H_{ij} in the dual-space, as described in [10]. This is an $O(k \log k)$ time process. Next we compute the dual of the convex hull to get the feasible cell F_i, in time $O(k)$. The intersection of the feasible cell F_i with Ψ_i gives rise to a set of components on Ψ_i. Since F_i is convex, every component ∂c_p is closed, simply connected, and does not intersect any other component. Each of these closed components ∂c_p, divides Ψ_i into two connected regions, say R_{p_0} and R_{p_1}. For exactly one of these, say R_{p_m}, it will be true that $R_{p_m} \subset F_i$. We define R_{p_m} to be the *interior* of the closed component ∂c_p. We can determine all these components ∂c_p, by finding the intersections of the edges and faces of F_i with Ψ_i. This can be done in $O(k)$ time.

After a connected component ∂c_p has been determined on Ψ_i we generate the surface patches. It is important to note there the distinction between the component ∂c_p and the surface patches it generates. For each component ∂c_p, there exists a one-to-one mapping, say $\mathcal{F}$, with a convex spherical patch of the Richards's surface together with parts of its adjacent (non-convex) patches.

We describe the mapping $\mathcal{F}$ next. Let the component ∂c_p be composed of r arcs, $a_{p_0}, a_{p_1}, \ldots, a_{p_{r-1}}$, and r vertices, $v_{p_0}, v_{p_1}, \ldots, v_{p_{r-1}}$. The arcs $a_{p_q}, 0 \leq q < r$ determine the locus of the center of the probe while it is in contact with two atoms. These arcs a_{p_q} therefore are used to generate the toroidal patches. The vertices v_{p_q} of this component ∂c_p, where two arcs intersect, define the positions of the center of the probe while it is in contact with three atoms. These vertices v_{p_q} are used to generate the concave spherical triangular patches. The interior of the component ∂c_p corresponds to the positions of the center of the probe while it is tangent to only atom i. This is used to generate a convex spherical patch.

In Figure 3 a component defined by three chordales Π_{ij}'s intersecting Ψ_i has been shown with its interior unshaded.

3.5 Parallelization

Our approach to computing the smooth molecular surface can be parallelized over all the atoms of the molecule. Each of the steps as described above can be carried out independently for each atom. The most expensive of these steps is the construction of a feasible cell which takes time

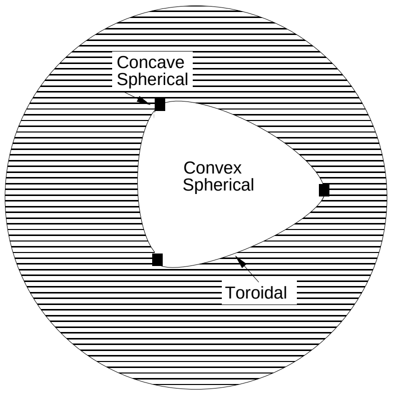

Figure 3: Determination of Molecular Surface Patches

$O(k \log k)$, for k neighbors. Therefore the complexity of our algorithm over n processors would be $O(k \log k)$. If the number of available processors $p < n$, we can allocate $\frac{n}{p}$ atoms per processor to get a time complexity of $O(\frac{nk \log k}{p})$. These bounds hold in a CREW (concurrent-read exclusive-write) PRAM model of parallel computation.

3.6 Robustness

In the algorithms for computing the convex hull of a set of points, it is assumed that the points are in a general position, ie. no more than d points lie on the same $d-1$ dimensional hyperplane. In reality this assumption often fails to hold, leading to problems. For example, planar benzene rings occur often in proteins, causing six carbon and six hydrogen atoms to be all coplanar.

One of the recent approaches to solving this problem has been to perturb the input point set slightly to avoid these degeneracies. We are using a version of the deterministic perturbation scheme proposed by Emiris and Canny [7], which perturbs the j^{th} dimension of the i^{th} point as:

$$p_{i,j}(\epsilon) = p_{i,j} + \epsilon(i^j \bmod q) 1 \le i \le n, 1 \le j \le d \quad (1)$$

where ϵ is a symbolic infinitesimal and q is the smallest prime greater than n.

4 Estimating the Number of Neighbors in Proteins

In this section we shall outline a method for estimating the average number of neighbors k in proteins – the molecules for which the molecular surfaces are most often computed. The value of k that we have observed in practice is around 45 for a probe-radius of $1.4\mathring{A}$. The purpose of this section is to assure that the average value of k in any protein cannot not be too much larger than the values of k that we have observed. In this section, we shall first state our assumptions for protein molecules and then outline a method for estimating k. This and some other methods for estimating k, are described in [13].

4.1 Assumptions

Let us start with the basic structure of a protein. A protein is an arbitrarily long chain of bonded amino acid residues. Each amino acid residue has an identical *backbone* or main-chain part and a side chain of one of 20 types. For a good introduction to the basic structure of proteins the interested reader can refer to the book by Dickerson and Geis [3].

Let us consider a graph G representing the covalent bond structure of a protein by representing each atom of the protein by a vertex and each covalent bond by an edge. G will be largely acyclic, with a few exceptions. These exceptions are — (a) the three aromatic amino acid residues (phenylalanine, tyrosine, and tryptophan) – which each have either one or two cycles in the side chain (b) proline – which forms a cycle through a bond between its side chain and main chain, and (c) disulphide bonds. To a first approximation we can ignore these cycles and simply consider the graph G to be a tree.

We recall that the degree of a vertex in a graph is defined as the number of edges incident at that vertex. From this it follows that in any graph, including a tree, the sum of the degrees of the vertices equals twice the number of edges. Now, if a tree has n vertices, it will have $n-1$ edges, and therefore the sum of degrees will be $(2n-2)$. This sum will increase by one for every cycle in the protein. Therefore, to a first approximation we can assume that the average degree per vertex in G is 2. In other words, to a first approximation, the average number of atoms covalently bonded to an atom in a protein molecule is 2.

Since it is extremely difficult to derive bounds for k for the general case where the radii of the atoms are different and the shape of the molecule is arbitrary, we shall make the following assumptions:

A: The boundary effects of the molecule will be ignored. This means that for any atom, we will be assuming that its entire region of influence is completely filled with other atoms, even though it is clear that for atoms on the boundary of the molecule this will not be true. Although this assumption is not always true, it can only lead us to overestimate the average number of neighbors.

B: For our purposes of finding the average number of neighbors, we shall consider all atoms to have equal radii $r_a = 1.75\mathring{A}$. For comparison, the radii of various commonly occurring atoms in proteins are indeed close to each other: C – $1.85\mathring{A}$, N – $1.75\mathring{A}$, O – $1.6\mathring{A}$, H – $1.00\mathring{A}$, S – $2.00\mathring{A}$, P – $2.10\mathring{A}$ [14].

C: The average distance between the centers of any two atoms is $l = 1.5\mathring{A}$ – the bond length of a single C-C bond [14].

D: The radius of the probe-sphere, which determines the radius of the region of influence, is $R = 1.4\mathring{A}$, approximately the radius of a water molecule.

With these assumptions, the radius of the region of influence is $R_{influence} = 2 \times r_a + 2 \times R = 6.3\mathring{A}$.

4.2 Volume-based Estimation of k

We are interested in estimating the number of spheres whose centers lie within the region of influence. In Section 3 we had stated that the minimum distance d_{ij} between any two atom centers c_i and c_j is greater than or equal to a positive constant l_{min} — the smallest bond-length in the molecule. Using this property with the assumption C stated above we can assume that the average bond length $l = 1.5\mathring{A}$. Further, since we are dealing with proteins, we can assume to a first approximation that on an average each atom is covalently bonded to two other atoms. From these we can conclude that the the average number of atom centers that can lie within a sphere of radius $l = 1.5\mathring{A}$ is no more than 3, as shown in Figure 4.

Therefore, volume per atom center $\ge \frac{1}{3}(\frac{4\pi}{3}(1.5)^3) = \frac{4\pi}{3}(1.125)$ and total volume $= \frac{4\pi}{3}R^3_{influence} = \frac{4\pi}{3}(250.047)$.

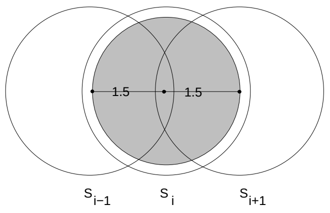

Figure 4: Volume per Three Atom Centers

Thus, $k \leq \frac{\frac{4\pi}{3}(250.047)}{\frac{4\pi}{3}(1.125)} = 222$.

We can improve the above bound, if we are prepared to make the following assumption:

Two atoms i and j are bonded iff $c_i \epsilon \sigma(c_j, r_j)$ and $c_j \epsilon \sigma(c_i, r_i)$, that is the center of atom i lies inside the sphere representing atom j and vice-versa.

With the above assumption, the average number of atom centers that can lie within a sphere of radius $l = r_a = 1.75\mathring{A}$ is no more than 3. Therefore, we have:

volume per atom center $\geq \frac{1}{3}(\frac{4\pi}{3}(1.75)^3) = \frac{4\pi}{3}(1.786)$.

Thus, $k \leq \frac{\frac{4\pi}{3}(250.047)}{\frac{4\pi}{3}(1.786)} = 139$.

5 Results

Our implementation has been done on Pixel-Planes 5 [8], although it is general enough to be easily portable to any other parallel architecture. Table 1 shows our timings for computation and display of the molecular surface for various molecules for a probe-radius of $1.4\mathring{A}$. For these results we were using configurations of 8, 16, or 24 Intel i860 processors. Our configuration of p processors consists of one master processor and $p-1$ slave processors. The master processor is responsible for distributing the work amongst the slave processors that perform the actual surface computations. This explains the superlinear times observed in Table 1. The molecules for which we have made these studies are crambin, felix, dihydrofolate reductase (DHFR), and superoxide dismutase (SOD). The Brookhaven Protein Data Bank files that we have used for these molecules are pdb1crn.ent, pdb1flx.ent, pdb2dhf.ent, and pdb2sod.ent, respectively. We have removed all the extra water molecules that were at the end of pdb2dhf.ent as they are not a part of the DHFR molecule per se. At present, we are representing the molecular surface by triangles, and the column *Tris* in Table 1 refers to the complexity of the computed surface in terms of number of triangles (rounded to the nearest thousand).

As can be seen, the value of k, the average number of neighbors, is fairly constant for a given probe-radius over different molecules.

Molecule	Atoms	Times (sec)			k	Tris
		Processors				
		8	16	24		
Crambin	327	0.66	0.32	0.24	41.3	14K
Felix	613	1.34	0.66	0.42	40.7	31K
DHFR	2980	5.62	2.70	1.79	44.8	92K
SOD	4392	8.36	3.99	2.65	46.6	127K

Table 1: Smooth Molecular Surface Generation Times for $1.4\mathring{A}$ probe-radius.

Table 2 shows the times for the generation of the molecular surface for crambin using 24 processors, and different probe-radii varying from $1.0\mathring{A}$ to $10.0\mathring{A}$.

Probe-Radius	$1.0\mathring{A}$	$1.4\mathring{A}$	$2.8\mathring{A}$	$5.0\mathring{A}$	$10.0\mathring{A}$
Times (sec)	0.23	0.24	0.32	0.53	0.95
k	29.9	41.3	91.8	191.5	318.3
Triangles	16K	14K	12K	11K	11K

Table 2: Smooth Molecular Surface Generation Times for Crambin using 24 Intel i860 Processors.

The smooth molecular surfaces for crambin with probe-sphere radii of $1.4\mathring{A}$, $2.8\mathring{A}$, $5.0\mathring{A}$, and $10.0\mathring{A}$ are shown in Figures 5, 6, 7, and 8 respectively. The smooth molecular surfaces for dihydrofolate reductase and superoxide dismutase for a probe-sphere radius of $1.4\mathring{A}$ are shown in Figures 9 and 10 respectively.

6 Conclusions and future work

We have presented a parallel algorithm for computing the molecular surfaces in parallel time $O(k \log k)$ over n processors. This is sufficiently general enough to be used for computation of α-hulls and α-shapes for a given value of α as long as no two points are arbitrarily close (i.e. the ratio of the distance between the closest pair of points to the diameter of the set of points is bounded from below by a strictly positive number). Our algorithm would give an order of magnitude improvement over the previous best known algorithms for molecules with moderately large values of n, on the order of a few thousands or more, in both sequential and parallel implementations.

At present we are not using any incremental temporal information in constructing these surfaces. Thus, if the atoms move slightly from their positions, the whole surface has to be recomputed from the beginning. Assuming the atoms of the molecule move along continuous trajectories, it should be possible to compute such surfaces (and indeed α-hulls and α-shapes) incrementally and efficiently by using the information from previous time steps.

Sometimes the molecular surface self-intersects due to overlap from probes that come from opposite sides of a surface. Traditionally, such overlapping surfaces are clipped away to form cusps in the molecular surface. At present, we correctly handle only those cases where the cusps are either minor or can be easily determined by limited local checks. A general approach to this problem needs to be developed.

Acknowledgments

We would like to acknowledge the valuable discussions we had with Pankaj Agarwal, Dinesh Manocha, Jan Prins, and David Richardson during various stages of this work. We would like to thank the editors and the anonymous referees for their suggestions which have led to improvements in the presentation of this paper. We would also like to thank Nelson Max for pointing out an error in the expression of Π_{ij} in the conference version of this paper. This work was supported by NIH National Center for Research Resources grant number 5-P41-RR02170. Equipment support provided by NSF/ARPA Science and Technology Center for Computer Graphics and Scientific Visualization, NSF Prime Contract Number 8920219.

References

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

[2] M. L. Connolly. Analytical molecular surface calculation. *Journal of Applied Crystallography*, 16:548–558, 1983.

[3] R. E. Dickerson and I. Geis. *The Structure and Action of Proteins*. Harper & Row, New York, NY, 1969.

[4] H. Edelsbrunner. Weighted alpha shapes. Technical Report UIUCDCS-R-92-1740, Department of Computer Science, University of Illinois at Urbana-Champaign, 1992.

[5] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, IT-29(4):551–559, 1983.

[6] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1), 1994.

[7] I. Emiris and J. Canny. An efficient approach to removing geometric degeneracies. In *Eighth Annual Symposium on Computational Geometry*, pages 74–82, Berlin, Germany, June 1992. ACM Press.

[8] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel. Pixel-planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. In *Computer Graphics: Proceedings of SIGGRAPH'89*, volume 23, No. 3, pages 79–88. ACM SIGGRAPH, 1989.

[9] G. Perrot, B. Cheng, K. D. Gibson, J. Vila, K. A. Palmer, A. Nayeem, B. Maigret, and H. A. Scheraga. MSEED: A program for the rapid analytical determination of accessible surface areas and their derivatives. *Journal of Computational Chemistry*, 13(1):1–11, 1992.

[10] F. P. Preparata and M. I. Shamos. *Computational Geometry - an Introduction*. Springer-Verlag, 1985.

[11] F. M. Richards. Areas, volumes, packing and protein structure. *Ann. Rev. Biophys. Bioengg.*, 6:151–176, 1977.

[12] R. Seidel. Linear programming and convex hulls made easy. In *Sixth Annual ACM Symposium on Computational Geometry*, pages 211–215, Berkeley, California, June 1990. ACM Press.

[13] A. Varshney, W. V. Wright, and F. P. Brooks, Jr. Bounding the number of unit spheres inside a larger sphere. Technical Report UNC-CS-TR-93-039, Department of Computer Science, University of North Carolina at Chapel Hill, 1993.

[14] S. J. Weiner, P. A. Kollman, D. A. Case, U. C. Singh, C. Ghio, G. Alagona, S. Profeta Jr., and P. Weiner. A new force field for molecular mechanical simulation of nucleic acids and proteins. *Journal of the American Chemical Society*, 106(3):765–784, 1984.

Biographies

Amitabh Varshney is a doctoral student at the University of North Carolina at Chapel Hill. His research interests include molecular graphics, computational geometry, object simplification, global illumination, and texturing. He received his B.Tech. in Computer Science from the Indian Institute of Technology Delhi, and his M.S. in Computer Science from the University of North Carolina at Chapel Hill.

Frederick P. Brooks, Jr. is Kenan Professor of Computer Science at the University of North Carolina at Chapel Hill. He was an architect of the IBM Stretch and Harvest computers. He was Corporate Project Manager for the System/360, including development of the System/360 computer family hardware, and the Operating System/360 software. He founded the Department of Computer Science in 1964 and chaired it for 20 years. His research there has been in computer architecture, software engineering, and interactive 3-D computer graphics ("virtual reality"). Dr. Brooks has received National Medal of Technology and the John von Neumann Medal of the IEEE.

William V. Wright is a research professor of computer science at the University of North Carolina at Chapel Hill and director of the GRIP project, a Research Resource funded by the National Institutes of Health developing computer graphics systems for studying molecular graphics, visualization of scientific data, and computer architecture and implementation. He received his Ph.D. in computer science from the University of North Carolina at Chapel Hill in 1972, is a member of ACM Siggraph, and is a senior member of the IEEE.

Figure 5: Molecular Surface for Crambin, Probe Radius = 1.4$\mathring{A}$

Figure 6: Molecular Surface for Crambin, Probe Radius = 2.8$\mathring{A}$

Figure 7: Molecular Surface for Crambin, Probe Radius = 5.0$\mathring{A}$

Figure 8: Molecular Surface for Crambin, Probe Radius = 10.0$\mathring{A}$

Figure 9: Molecular Surface for Dihydrofolate reductase, Probe Radius = 1.4$\mathring{A}$

Figure 10: Molecular Surface for Superoxide dismutase, Probe Radius = 1.4$\mathring{A}$