HIERARCHICAL GEOMETRIC APPROXIMATIONS

TR-050
1994

Amitabh Varshney

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

This work was supported by NIH National Center for Research Resoures Grant # 5-P41-RR02170.

HIERARCHICAL GEOMETRIC APPROXIMATIONS

by

Amitabh Varshney

A Dissertation submitted to the faculty of The University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill

1994

Approved by:

Frederick P. Brooks, Jr. Advisor

William V. Wright Reader

Dinesh Manocha Reader

Agarwal Reader

AMITABH VARSHNEY. Hierarchical Geometric Approximations (Under the direction of Professor Frederick P. Brooks, Jr.)

Abstract

This dissertation explores some techniques for automatic approximation of geometric objects. My thesis is that using and extending concepts from computational geometry can help us in devising efficient and parallelizable algorithms for automatically constructing useful detail hierarchies for geometric objects. We have demonstrated this by developing new algorithms for two kinds of geometric approximation problems that have been motivated by a single driving problem — the efficient computation and display of smooth solvent-accessible molecular surfaces. The applications of these detail hierarchies are in biochemistry and computer graphics.

The smooth solvent-accessible surface of a molecule is useful in studying the structure and interactions of proteins, in particular for attacking the protein-substrate docking problem. We have developed a parallel linear-time algorithm for computing molecular surfaces. Molecular surfaces are equivalent to the weighted α-hulls. Thus our work is potentially useful in the application areas of α-hulls which include astronomy and surface modeling, besides biochemistry.

We have defined the concept of *interface surfaces* and developed efficient algorithms for computation of surfaces at the interface of two or more molecular units. Interface surfaces are useful for visualizing the inter and intra-molecular interfaces and for characterizing the fit, or complementarity, of molecular interfaces.

We have developed an algorithm for simplification of polygonal meshes. The simplified polygonal mesh has the following properties: (a) every point on it is within a user-specifiable distance ϵ from the input mesh, (b) it is topologically consistent with the input mesh (i.e. both have the same genus), (c) its vertices are a subset of the vertices of the input mesh, and (d) it is within a computable factor in complexity (in terms of number of faces) of the optimal mesh that satisfies (a), (b), and (c) (computing the optimal mesh is known to be NP-hard). We have accomplished this by transforming our problem to the set-partitioning problem.

Acknowledgments

I would like to thank my advisor Dr. Frederick P. Brooks, Jr., for his excellent advice and constant cheering all through my doctoral research. He has not only taught me computer science but has also given me a lifetime philosophy, of working on real-world problems, to pursue. Meeting him every week was an uplifting experience for me – both intellectually and spiritually. His systematic, devoted, and well-organized approach to work has been and shall remain a major source of inspiration to me.

I would like to thank my other committee members for their insightful comments and cheerful encouragement during the course of my research. In particular I should like to thank Pankaj Agarwal for teaching me computational geometry, sharing his bright ideas on research problems in geometry, and for being always available for any advice I needed; Dinesh Manocha for clarifying the numerous doubts I had in algebraic geometry and for several discussions on the molecular surfaces; Jan Prins for his amazingly quick grasp and a ready solution for whatever problem I went to him for (I now wish I had taken the P = NP problem to him, it might have been solved by now) and his unfailing good humor; David Richardson for being exceedingly kind and patient during his several hours of tutoring me biochemistry and for his infectious enthusiasm; and Bill Wright for his calm, sober, and always correct analysis of my ideas and for discussing several versions of the algorithms that appear in this dissertation.

I would like to thank all members of the Walkthrough and GRIP teams for their support during my research. In particular, I would like to thank Hans Weber, Olivier Garamfalvi, Chris Georges, David Luebke, and Brian Upton for spending countless hours on the submarine datasets with me. Their good humor and enthusiasm kept my spirits up all along. I would also like to thank John Airey, for suggesting me a long time ago that simplification of polygonal meshes might be a good research area.

I would also like to thank all the Sittersonites for their good cheer and support. My memories of Sitterson Hall and indeed Chapel Hill shall easily last me through this life.

My parents have taught me the importance of hard work, patience, and perseverance though example. I thank them for everything.

Contents

List of Figures

List of Tables

Chapter I

Overview and Results

1.1 The Hierarchy of Detail

We refer to the hierarchy formed by starting from a relatively simple representation of an object and progressively adding detail to it as a *hierarchy of detail.* Computer Science, as we know it today, has a well-developed concept of hierarchy of detail underlying almost all of its sub-disciplines. For example, in software engineering and programming languages, the concepts of information encapsulation and data abstraction are manifestations of this hierarchy of detail. Similarly in scientific computation when we are modeling the laws of physics for a simulation we find a whole hierarchy of detail awaiting us, namely the modeling accuracy of the laws of physics. Nested within this hierarchy is the hierarchy of mathematical approximations for any given level of physics. The level in this hierarchy that we opt for depends upon the available computing power and the desired response time. Similarly, in computer graphics one can associate a hierarchy of detail with geometric object models. At one end of this hierarchy are low-complexity, low-fidelity approximations of a given object, whereas the other end has high-complexity, high-fidelity approximations.

This dissertation addresses some issues in the automated hierarchical approximations of geometric objects. The space of various detail hierarchies for approximating geometric objects can be organized and described along several orthogonal dimensions. Some of these dimensions are:

- *Type of approximation:* When we say that an object A is an approximation of an object B, what we really mean is that there exists a property of B that is being approximated by A. This property (henceforth called the "approximating property") could be the shape, the Gauss map, the volume, etc. Various kinds of approximations can be defined for a given object based on the several approximating properties of the object.

- *Distance function:* Let us assume that the approximating property of an object spans a multi-dimensional space. We can define various kinds of distance functions that measure the distance between two objects having different values of this approximating property. For example, the distance could be measured in any of the L_p norms, or some other norm. Different distance functions give rise to different approximations for a given object for a given value of the approximating property.

- *Properties of geometric objects:* The kinds of approximation schemes that can be defined depends on the kind of assumptions that one is prepared to make on the properties of the input objects. For example, the input objects could be continuous or discrete, piecewise linear or higher order algebraic functions, open or closed, etc.

As can be seen, the range of possible geometric approximation schemes is enormous – enough to fuel many dissertations.

My thesis is:

> *Using and extending concepts from computational geometry can help us in devising efficient and parallelizable algorithms for automatically constructing useful detail hierarchies for three-dimensional geometric objects.*

I have demonstrated this by developing new algorithms for two kinds of geometric approximation problems that have been motivated by a single driving problem — the efficient computation and display of smooth solvent-accessible molecular surfaces. The applications of these detail hierarchies are in biochemistry and computer graphics.

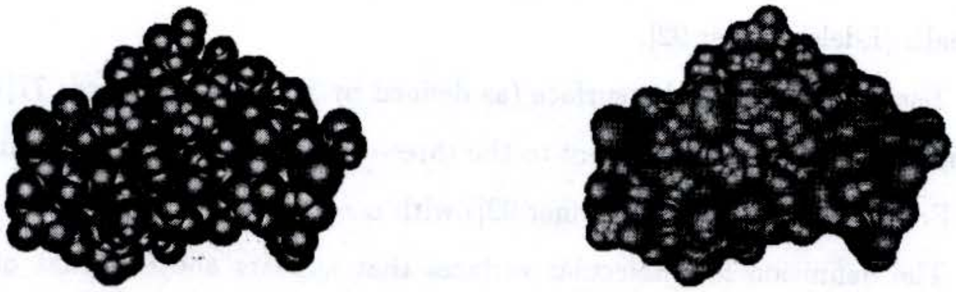
Let a molecule M be represented by a collection of spheres $\{S_1, \ldots, S_n\}$, each sphere S_i corresponding to atom i of the molecule. A molecular surface for M can be considered to be approximated by the surface of the union of these spheres: $\bigcup_{i=1}^{n} S_i$. Approximation of a molecule by its surface has proved to be of immense value in biochemistry in visualizing and understanding the structure and functions of proteins. The first geometric approximation problem that we have addressed in this dissertation is that of approximating a molecule by its molecular surface, which we shall call *α-approximation.* The second geometric approximation problem that we have explored is that of approximating a polygonal representation of the molecular surface by another polygonal surface which is within a distance ϵ from the original surface and has a lower complexity in terms of the number of polygons. We shall use the term *ϵ-approximation* to refer to this second approximation problem.

The α-approximation and the ϵ-approximation are both quite general problems and are useful in areas other than molecular modeling. The α-approximation problem can be used to compute a detail hierarchy of α-hulls and α-shapes [Edelsbrunner & Mücke 94, Edelsbrunner 92, Edelsbrunner *et al* 83] that are useful in such diverse fields as astronomy, biochemistry, statistics, and computer graphics. The ϵ-approximation problem is of immense value in a three-dimensional computer graphics setting in simplifying complex polygonal models under the constraints of topological consistency and bounded error tolerance. We have explored the use of our ϵ-approximation algorithm in this general setting.

This chapter is meant to serve as an extended abstract of the dissertation, outlining the problems, our approaches, and the results in brief.

1.2 Molecular Surfaces — Our Driving Problem

The *smooth surface* of a molecule is the surface which an exterior probe-sphere touches as it is rolled over the (assumed) spherical atoms of that molecule. This definition of a molecular surface was first proposed by Richards [Richards 77]. This surface is useful in studying the structure and interactions of proteins, in particular for attacking the protein-substrate docking problem. In Figure 1.1 (a) crambin is shown as a collection of spheres whose radii are the van der Waal's radii of the corresponding atoms. In Figure 1.1 (b) the molecular surface of crambin is shown for a probe-sphere radius of $1.4\AA$ (the radius of the spherical approximation to the water molecule).



(a) Crambin (396 atoms) (b) Crambin Surface, Probe Radius = 1.4 Å

Figure 1.1: Crambin and its Molecular Surface

Present systems for computing the surfaces of molecules are batch-oriented [Connolly 93]. Our goal has been to compute and display these surfaces at interactive rates, by taking advantage of results from 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 at various levels of detail. 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.

1.3 α-Approximation of Molecules

The *α-hull* has been defined as a generalization of the convex hull of point sets by Edelsbrunner, Kirkpatrick, and Seidel [Edelsbrunner *et al* 83]. Given a set of points P, a ball b of radius α is called an *empty α-ball* if $b \cap P = \phi$. For $0 \leq \alpha \leq \infty$, the *$\alpha$-hull* of P is defined as the surface of the complement of the union of all empty α-balls [Edelsbrunner 92].

The smooth molecular surface (as defined by Richards [Richards 77]) for a probe sphere of radius R is equivalent to the three-dimensional weighted α-hull (as defined by Edelsbrunner in [Edelsbrunner 92]) with $\alpha = R$.

The definition for molecular surfaces that appears above is that of a *complete molecular surface*, i.e. a surface that completely envelopes a molecule. This is useful for visualizing the surface of a single molecule. However, when the objective is to study the interface between two or more molecules (or different sub-units within a single molecule), the complete molecular surface computed independently for each of the molecules is a poor visualization tool. This is due to the occlusion by the complete molecular surfaces which prevents one from studying the contacts that are in the interior of the molecular interfaces. To overcome this, we have defined the concept of *molecular interface surfaces* and devised algorithms for their efficient implementation.

1.3.1 Complete Molecular Surfaces

Complete molecular surfaces are commonly known as smooth molecular surfaces, Richards's molecular surfaces (after their inventor), Connolly surfaces, (after the inventor of the most widely used surface computation algorithm), or three-dimensional weighted α-hulls. In this section we shall first overview our algorithm for efficiently computing these surfaces analytically and then present the results of our implementation.

Algorithm Overview

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.

We construct the solvent-accessible molecular surface for the whole molecule, an atom at a time. For each atom we first determine its neighbors. Two atoms are defined as *neighbors* if it is possible to place a probe-sphere such that it is in simultaneous contact with both these atoms, without considering any hindrance due to other atoms. This determination is done in linear time over the whole molecule by subdividing the molecule using a global grid. Next, we construct a feasible cell (which is very similar to a power-cell [Aurenhammer 87]; for exact definition see Chapter IV) around each atom by using these neighbors. If there are k neighbors for an atom, this stage takes $O(k \log k)$ time. Next the surface is generated for each atom from its power-cell in $O(k)$ time. Thus the overall complexity of our algorithm is $O(nk \log k)$. Since k is a constant that depends on the probe-radius, this algorithm is linear in the number of atoms of a molecule. Since the processing of each feasible cell can be done independently of others the algorithm is parallelizable to degree n. Details of this algorithm and its implementation are given in Chapter IV.

Results

Our implementation has been done on Pixel-Planes 5 [Fuchs *et al* 89], although it is general enough to be easily portable to any other parallel architecture. Table 1.1 shows our timings for computation and display of the molecular surface for various molecules for a probe-radius of $1.4 \mathring{A}$ (the radius of a water molecule). For these results we were using configurations of 8, 16, or 24 Intel i860 processors. The molecules we have studied are crambin, felix, dihydrofolate reductase (DHFR), and superoxide dismutase (SOD). At present, we are representing the molecular surface by triangles, and the column *Tris* in Table 1.1 refers to the complexity of the computed surface in thousands of triangles.

Molecule	Atoms	Times (sec)			k	Tris
		Processors				
		8	16	24		
Crambin	396	0.84	0.43	0.31	44.7	18K
Felix	613	1.42	0.69	0.47	40.7	36K
DHFR	3123	6.11	2.93	1.96	43.8	100K
SOD	4386	8.73	4.16	2.76	46.5	127K

Table 1.1: Molecular Surface Generation for 1.4$\AA$ Probe-Radius

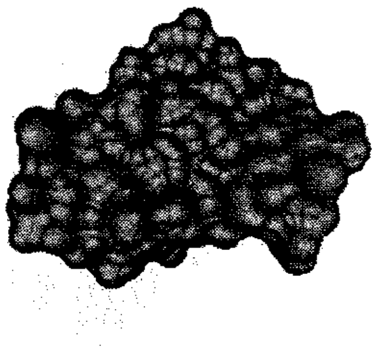
As can be seen, the value of k, the average number of neighbors, is fairly constant for a given probe-radius over different molecules. In fact, using concepts from the theory of packing of spheres and some reasonable assumptions, we will prove in Chapter V that for a probe-radius of 1.4$\AA$, the average number of neighbors in protein molecules can be bounded from above to be in low hundreds. This means that the algorithm we have presented here is more attractive than the widely used $O(n^2)$ algorithm even for medium-sized proteins that have more than a couple of thousand atoms.

Table 1.2 shows the times for the generation of the molecular surface for crambin using 24 processors, and different probe radii varying from 1.0$\AA$ to 10.0$\AA$.

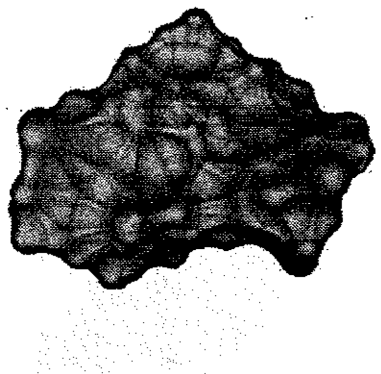
Probe-Radius	1.0$\AA$	1.4$\AA$	2.8$\AA$	5.0$\AA$	10.0$\AA$
Times (sec)	0.29	0.31	0.43	0.70	1.32
k	32.2	44.7	102.8	224.0	384.9
Triangles	22K	18K	14K	13K	13K

Table 1.2: Crambin Molecular Surface Generation (24 Intel i860 Processors)

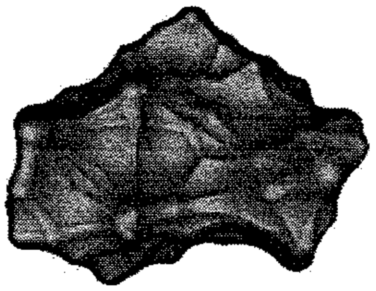
Figure 1.2 shows the smooth molecular surface for crambin with variable probe-sphere radii.



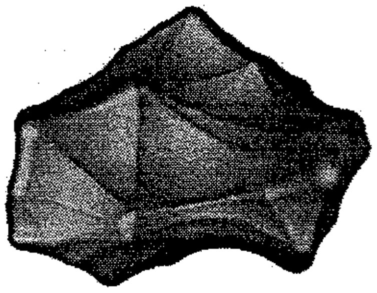
(a) Probe Radius = 1.4$\mathring{A}$



(b) Probe Radius = 2.8$\mathring{A}$



(c) Probe Radius = 5.0$\mathring{A}$



(d) Probe Radius = 10.0$\mathring{A}$

Figure 1.2: Crambin Surfaces for Different Probe-Radii

1.3.2 Molecular Interface Surfaces

To afford a good visualization of the molecular surfaces at the interface of two or more molecules (or sub-units of the same molecule), we have developed the concept of molecular interface surfaces.

Let the complete molecular surfaces defined for a probe-radius α for the molecules A and B be represented by $\mathcal{S}(A, \alpha)$ and $\mathcal{S}(B, \alpha)$, respectively. The *molecular interface surface* $\mathcal{T}(A, B, \alpha, \beta)$ for a probe-radius α and an interface-radius β for the two molecules A and B is defined as the subset of $\mathcal{S}(A, \alpha)$ and $\mathcal{S}(B, \alpha)$ that includes exactly those points of $\mathcal{S}(A, \alpha)$ that are within a distance β from the surface of some atom of B and exactly those points of $\mathcal{S}(B, \alpha)$ that are within a distance β from the surface of some atom of A. Figure 1.3(a) shows the four domains of transthyretin and

(b) shows the molecular interface surface amongst these domains for a probe-sphere radius = $\alpha = 2.4\AA$ and an interface-radius = $\beta = 1.0\AA$.

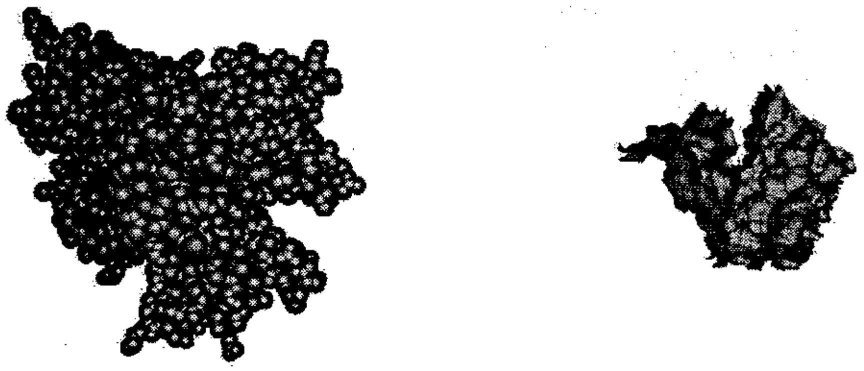

(a) Transthyretin domains (b) Interface surface for $\alpha = 2.4\AA, \beta = 1.0\AA$

Figure 1.3: Transthyretin and its Interface Surface

Besides proving to be a useful visualization tool, the molecular interface surfaces should also provide a means of efficiently characterizing the interactions during a protein-substrate docking. The molecular interface surfaces define a hierarchy of detail parametrized by α and β at the inter-molecular interface. Given a particular value of α, one can define the interface surface by choosing a suitable value for the parameter β based on the computing power available, the desired response time, and the modeling accuracy of the physical interactions. The interface surface thus defined would localize and reduce the set of possible interactions occurring at the interface and could therefore be used as an input to further processing for efficiently characterizing the interactions at the interface.

Algorithm Overview

The algorithm follows from the definition of the molecular interface surfaces. We next describe how to compute the subset of $\mathcal{S}(A, \alpha)$ that belongs to $\mathcal{T}(A, B, \alpha, \beta)$. The subset of $\mathcal{S}(B, \alpha)$ that belongs to $\mathcal{T}(A, B, \alpha, \beta)$ can be computed similarly.

Let us define an atom b of B to be the sphere $\sigma(c_b, r_b)$, where c_b is the center and r_b is the van der Waal's radius of the atom b. Let us define $B(+\beta)$ to be a collection of spheres $\sigma(c_b, r_b + \beta)$ derived from the atoms of B.

Let all the atoms in A that intersect the interior of $\bigcup_{b \in B} \sigma(c_b, r_b + \beta)$ be represented by the set A_T. Determination of A_T is efficiently done by using a cuboidal grid to localize the spheres of A. We next generate the surface patches of $\mathcal{S}(A, \alpha)$ that are contributed by every atom $a \in A_T$. Every such surface patch is clipped by the union of the spheres in $B(+\beta)$. This clipping is efficiently done by using a cuboidal grid to localize the spheres in $B(+\beta)$. All the clipped surface patches that lie within the union of the spheres in $B(+\beta)$ are retained and form that subset of $\mathcal{S}(A, \alpha)$ that belongs to $\mathcal{T}(A, B, \alpha, \beta)$.

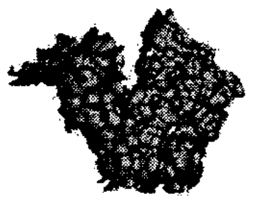
In Chapter VI, we describe the molecular interface surfaces in greater detail including the simple extensions to handle more than two molecular sub-units.

Results

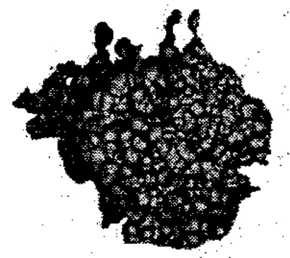
The results of our implementation on computing the interface surfaces at the interface of the four domains of the molecule transthyretin (prealbumin) for different values of α and β are shown in Figure 1.4.

1.4 ϵ-Approximation of Polygonal Models

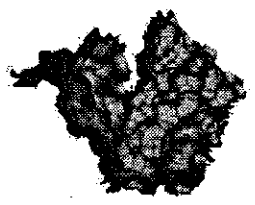
In three-dimensional interactive computer graphics, progressive enrichment of detail is a recurrent theme, more so perhaps than in any other sub-discipline within computer science. The main reason for this is our attempts to simultaneously satisfy the conflicting goals of scene realism and real-time performance. These attempts can be broadly classified into two categories — image-space refinement and object-space refinement. In image-space refinement approaches [Bergman *et al* 86], the scene is first rendered as a low-quality image while the user is constantly changing his viewpoint. Once the user stops and the viewpoint is fixed, a progressively detailed rendering of the scene is then done by the renderer in due time. In object-space refinement approaches, a hierarchy of object descriptions is stored and depending on how important the object is to the scene, an appropriate level-of-detail of the object is used for rendering –



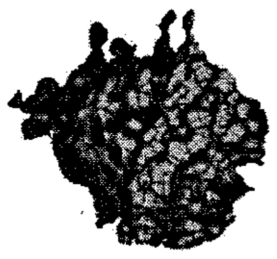
(a) $\alpha = 1.0\AA, \beta = 1.0\AA$



(b) $\alpha = 1.0\AA, \beta = 2.4\AA$



(c) $\alpha = 2.4\AA, \beta = 1.0\AA$



(d) $\alpha = 2.4\AA, \beta = 2.4\AA$

Figure 1.4: Interface Surfaces Amongst Domains in Transthyretin

[Clark 76], [Cosman & Schumacker 81], [Crow 82], and [Funkhouser & Séquin 93].

1.4.1 Modeling in Computer Graphics

At present, the most common form of model representation in computer graphics is a planar polygonal description. There are several reasons for this. First, current graphics workstations can rapidly render polygonal datasets, while most cannot render higher-order algebraic surfaces (without first polygonizing them). Second, any given model can be approximated by a planar polygonal dataset. Examples of such models range from algebraic-surface based models (consisting of Bézier or B-spline patches or CSG solids) to datasets consisting of just scattered points (obtainable from three-dimensional scanners). Third, planar polygonal datasets simplify any form of pre-processing computation (such as visibility computation). For these reasons it is likely that polygonal datasets will continue to play an important role in computer graphics for some time to come. For these reasons, I decided to work on simplification of polygonal datasets instead of any other model representation format.

Simplification of polygonal models by approximation allows one to generate a detail hierarchy for objects. Such a detail hierarchy can be used in several ways:

- In a level-of-detail-based rendering algorithm for providing desired frame update rates.
- Simplifying traditionally over-sampled models such as those generated from volume datasets, laser scanners, and satellites.
- Using low-detail approximations of objects in coarse visibility computations, collision detection, and global illumination algorithms, especially radiosity.

1.4.2 Problem Definition

Before we formally define our problem, let us first define the term *ϵ-approximation.* Given two piecewise linear objects $\mathcal{P}$ and $\mathcal{Q}$, we say that $\mathcal{P}$ and $\mathcal{Q}$ are ϵ - approximations of each other iff every point on $\mathcal{P}$ is within a distance ϵ of some point of $\mathcal{Q}$ and every point on $\mathcal{Q}$ is within a distance ϵ of some point of $\mathcal{P}$.

We define our problem as follows:

Given a polygonal representation $\mathcal{P}(\mathcal{I})$ of an object $\mathcal{I}$ and an approximation parameter ϵ, generate a genus-preserving ϵ-approximation $\mathcal{A}$ with minimal number of polygons such that the vertices of $\mathcal{A}$ are a subset of vertices of $\mathcal{P}(\mathcal{I})$.

Such an approximation scheme has several benefits in computer graphics. First, one can very precisely quantify the maximum amount of approximation error that is tolerable under given circumstances. For instance, one possibility could be to define a tolerable approximation for rendering an object as, say, 2 screen pixels. Using this information in conjunction with the distance of the object from the screen, one can estimate the maximum deviation permissible from the surface of the object. This can then be used to find which precomputed level of detail of that object is most suitable. Second, this approach allows one a fine control over which regions of an object one should approximate more and which ones less. This could be used in selectively preserving those features of an object that are perceptually important.

In our problem definition we have stated that $\mathcal{A}$ should have a minimal number of polygons. It turns out that achieving minimality for $\mathcal{A}$ is NP-hard even for the simple case where $\mathcal{I}$ is a convex polytope [Das & Joseph 90]. Therefore we shall aim to compute an ϵ-approximation $\mathcal{A}$ to $\mathcal{P}(\mathcal{I})$ such that (a) $\mathcal{A}$ has a smaller number of polygons than $\mathcal{P}(\mathcal{I})$ and (b) the number of polygons in $\mathcal{A}$ can be related to the minimal (optimal) number of polygons possible in any ϵ-approximation to $\mathcal{P}(\mathcal{I})$. We shall henceforth refer to the polygonal representation $\mathcal{P}(\mathcal{I})$ of an object $\mathcal{I}$ as $\mathcal{P}$.

1.4.3 Algorithm Overview

The basic outline of the algorithm is as follows:

- Generate two *offset surfaces* to the input model, one on the outside and the other on the inside of the input object.
- Then generate all *candidate triangles* that lie within these two offset surfaces and have their vertices selected from the set of vertices of the input model.
- Find which vertices of the input model are "covered" by which triangles.
- Finally, use a greedy approach for selecting the approximation triangles from the candidate triangles.

Why this approach works and how this can be used to get a quantitative measure of the quality of approximation is described in Chapter VII.

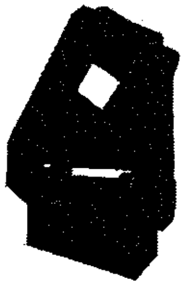
We shall assume that all polygons in $\mathcal{P}$ are triangles and that $\mathcal{P}$ is a well-behaved polygonal model, i.e. every edge has either one or two adjacent triangles, no two triangles interpenetrate, there are no unintentional "cracks" in the model, etc. We will be further assuming that the vertices of $\mathcal{P}$ have possibly several normals (such as at sharp edges in a model) that faithfully represent the normals of the object being modeled.

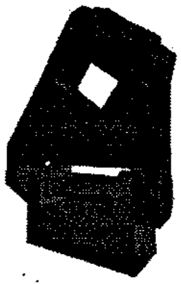
While greedy algorithms are typically sequential in nature, our algorithm can be parallelized in the different stages. Vertex-vertex visibility pairs can be generated completely in parallel. Further, changes made to an existing mesh due to selection of a candidate triangle are local. Such regions that have to be retriangulated are well-defined and can be processed in parallel.

1.4.4 Results

We have been able to achieve roughly 70% reductions with minimal perceptual difference on polygonal datasets that we have attempted to reduce thus far. These datasets have been taken from polygonal models used in real-world problems, such as virtual-reality walkthroughs of proposed submarines.

(a) First Level: 2346 triangles


(b) Second Level: 1180 triangles


(c) Third Level: 676 triangles

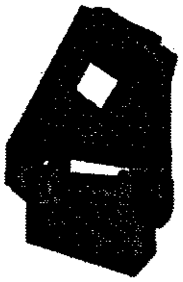

(d) Fourth Level: 514 triangles

Figure 1.5: Four Detail Levels for a Polygonal Model

A typical level-of-detail hierarchy with four levels for a polygonal model of a torpedo roller is shown in Figure 1.5.

The actual level in this hierarchy that is chosen for display depends upon the screen-space area occupied by the polygonal model. Thus, the farther the model from the observer, the coarser the level of detail that is selected. In Figure 1.6 we show a row of rollers used for loading a torpedo into a torpedo tube. In Figure 1.6(a), an appropriately chosen level of detail is used for rendering each of the rollers, whereas in Figure 1.6(b), no level-of-detail-based rendering is used. The frame-update rate in case (a) is 12 frames per second for rendering a total of 22K triangles, whereas the frame-update rate for case (b) is just 7 frames per second for rendering 49K triangles. As can be seen, there is hardly any perceptual difference between the images (a) and (b) in Figure 1.6.

We have simplified a total of 1090 objects of the AMR dataset for testing and validating our algorithm. These polygonal objects are from the Submarine Auxiliary Machine Room (AMR) dataset that was given to us by the Electric Boat Division of the General Dynamics. We have cumulatively reduced these objects as follows:

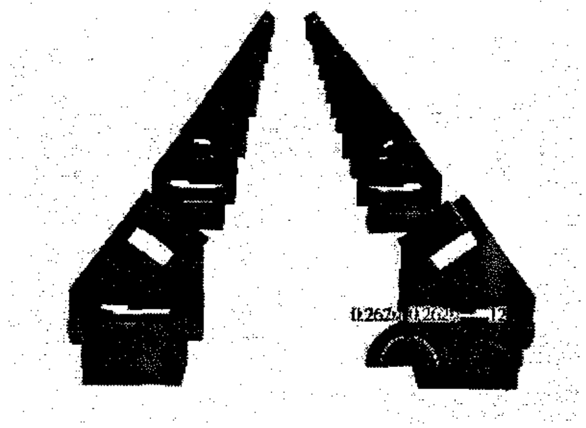
Level-of-Detail	Dataset Complexity	% Reduction
Original dataset	350,023 triangles	0
First level	206,859 triangles	40.90
Second level	141,983 triangles	59.44
Third level	104,874 triangles	70.04

Table 1.3: Polygonal Simplification Results

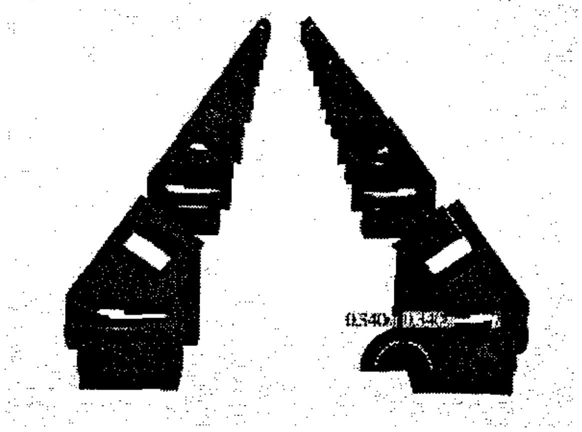
1.5 A Guide to the Chapters

The rest of this dissertation is organized as follows.

In Chapter II, we will give a brief overview of proteins, the protein docking problem, the protein folding problem, and the various molecular surfaces defined in bio-



(a) Using level-of-detail rendering



(b) Not using level-of-detail rendering

Figure 1.6: Rendering With and Without Level-of-detail Models

chemistry.

Chapter III gives an overview of some terminology from computational geometry and a brief review of detail hierarchies for boundaries of points and polygonal curves.

Chapter IV describes our approach for analytically computing the solvent-accessible molecular surfaces in real-time and then overviews a few implementation details, and concludes with our results.

In Chapter V we give a brief introduction to the mathematical theory of packing of spheres and present several techniques to bound the number of unit spheres that can be packed inside a larger sphere of a given radius. We consider two main kinds of techniques – volume-based and surface-based – for deriving these bounds, and work with both – intersecting unit spheres as well as mutually disjoint unit spheres. These techniques have been illustrated by using them to estimate the number of solvent-accessible neighbors for an atom in a protein molecule.

In Chapter VI, we discuss computation of molecular surfaces at the interface of two or more molecular sub-units and give a goodness-of-fit criterion for evaluating molecular interfaces.

In Chapter VII, we describe our approach to compute a level-of-detail object hierarchy for polygonal models and then discuss a few implementation heuristics followed by results.

Finally, we discuss directions for further work in Chapter VIII.

Chapter II

Molecular Surfaces

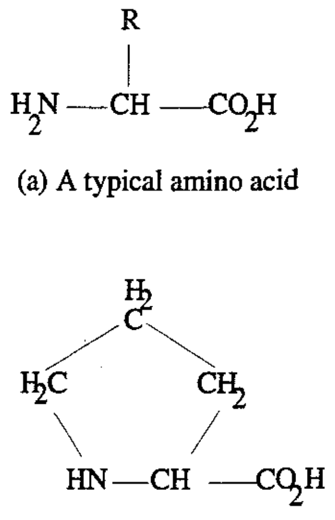
In this chapter we shall first give a brief overview of proteins. Next we shall motivate the need for computing molecular surfaces by considering the protein-substrate docking problem and the protein folding problem. After that we shall overview the two main kinds of molecular surfaces that are defined in biochemistry.

2.1 Proteins

Proteins are long, linear sequences of bonded amino acids. There are 20 naturally occurring amino acids. Of which, 19 amino acids have the same basic structure, as is shown in Figure 2.1(a). The only difference among these 19 amino acids is the difference in the chemical composition of the *sidechain R*. R can be as simple as a single hydrogen atom, or it can be a long chain consisting of carbon, nitrogen, sulfur, oxygen, and hydrogen atoms. R can even consist of aromatic rings. The twentieth amino acid, proline, is special in that it has a bond between the sidechain R and the nitrogen atom as shown in Figure 2.1(b).

For clarity in presentation, we shall ignore proline and assume that the structure shown in Figure 2.1(a) adequately represents the structure of a generic amino acid.

Two amino acids form a bond by releasing a water molecule as shown in Figure 2.2(a). This bond is known as a *peptide bond.* In a protein, many amino acids link together to form a long, linear chain of peptide bonds. This is shown in Fig-



(a) A typical amino acid

(b) Proline

Figure 2.1: Amino Acids

ure 2.2(b). The leftover structure of an amino acid after the formation of peptide bonds with its neighbors (and the consequent loss of a water molecule) is known as the amino acid *residue*. Each amino acid residue consists of the sidechain R and six other atoms as shown in Figure 2.2(c). The group of atoms shown in Figure 2.2(d) behaves as a rigid unit and is known as a *peptide* unit. All the peptide units of a protein are collectively referred to as the *backbone* or the *mainchain* of the protein.

The various amino acids can be characterized to be hydrophilic or hydrophobic based on the interactions of their sidechains with water. Thus all amino acids that have aliphatic hydrocarbon sidechains are hydrophobic, and all amino acids that have polar atoms such as oxygen are hydrophilic.

For a better understanding of the fundamentals of protein structure the interested reader can see the textbook [Dickerson & Geis 69].

2.2 The Protein-Substrate Docking Problem

The *Protein-Substrate Docking Problem* is to identify the position and orientation of the protein molecule with respect to a given substrate (another molecule that may be a protein, a nucleic acid, or a drug molecule) such that the energy of interaction of the

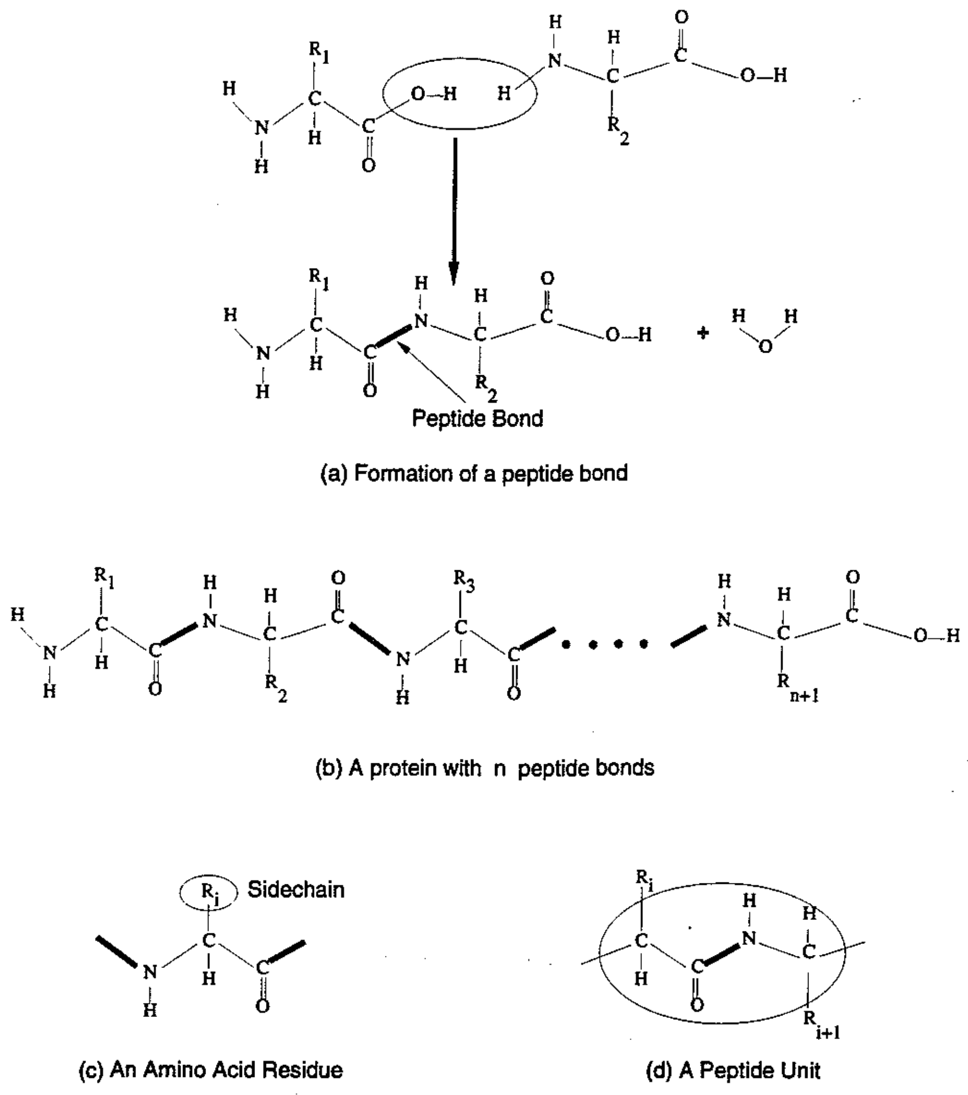

Peptide Bond
(a) Formation of a peptide bond
(b) A protein with n peptide bonds
Sidechain
(c) An Amino Acid Residue
(d) A Peptide Unit

Figure 2.2: Bonding of Amino Acids

two is minimized. This problem is useful in studying enzyme catalysis and antigen-antibody interactions. These interactions have been observed to be very specific in their occurrence. Even slight changes in the structure of a sidechain of one of the participants have been observed to inhibit such interactions.

This docking of the protein with a substrate is characterized by geometric and electrostatic complementarity of the two surfaces and compatible hydrophilicity. Determination of the molecular surfaces of the two molecules thus plays a rather important role in solving this problem.

2.3 The Protein Folding Problem

A protein is initially synthesized in the form of a long, linear chain of amino acid residues by a small cellular body called a ribosome. Once fully synthesized, it rapidly folds into a unique three-dimensional conformation. It has long been believed, and indeed confirmed for relatively small proteins, that the three-dimensional shape of a protein is just a function of its one-dimensional sequence of amino acids. The problem of predicting the three-dimensional structure of a protein based on the one-dimensional sequence of its amino acids has come to be known as the *Protein Folding Problem.*

In general, proteins exist and interact in aqueous media in living things. This solvent is believed to play an important role in protein folding. The interior of most folded proteins has been found to be largely free of water and consisting mostly of hydrophobic amino acid residues. The surface of the folded proteins, in contrast, has mostly hydrophilic residues. The surface area of hydrophobic residues in contact with water provides a good means of estimating the effect of solvent at a given stage of protein folding. Thus, efficient determination of molecular surfaces is of considerable interest in gaining a better understanding of the protein folding.

2.4 Surfaces for Molecules

In 1971 Lee and Richards defined the *solvent-accessible surface* of a molecule as the surface that is traced by the center of a probe sphere representing a solvent molecule as it is rolled over the surface of the molecule [Lee & Richards 71]. The surface of a molecule defined in this fashion has the advantage of simplicity — all patches of the surface are convex spherical. However, it does suffer from some drawbacks. First, as the radius of the probe sphere increases to ∞, the surface area of the molecule as computed by this method also increases to ∞, which is clearly counter-intuitive. Second, the surface of a molecule defined in this fashion is not a useful means for identifying the degree of complementarity between two molecules.

In 1977 Richards gave an alternative solvent-accessible molecular surface definition [Richards 77]. He defined the molecular surface to be the surface which an exterior probe-sphere touches as it is rolled over the spherical atoms of that molecule. This surface is also known as the solvent-excluding surface. This is a more useful definition of the molecular surface as it better approximates the van der Waal's surface of the solvent-accessible atoms and thus gives a more realistic value of a molecule's solvent-accessible surface area. The region where this molecular surface consists of the van der Waal's surface of an atom is called the *contact surface*; other regions are called *reentrant surfaces.*

The surfaces corresponding to the two definitions are shown in Figure 2.3.

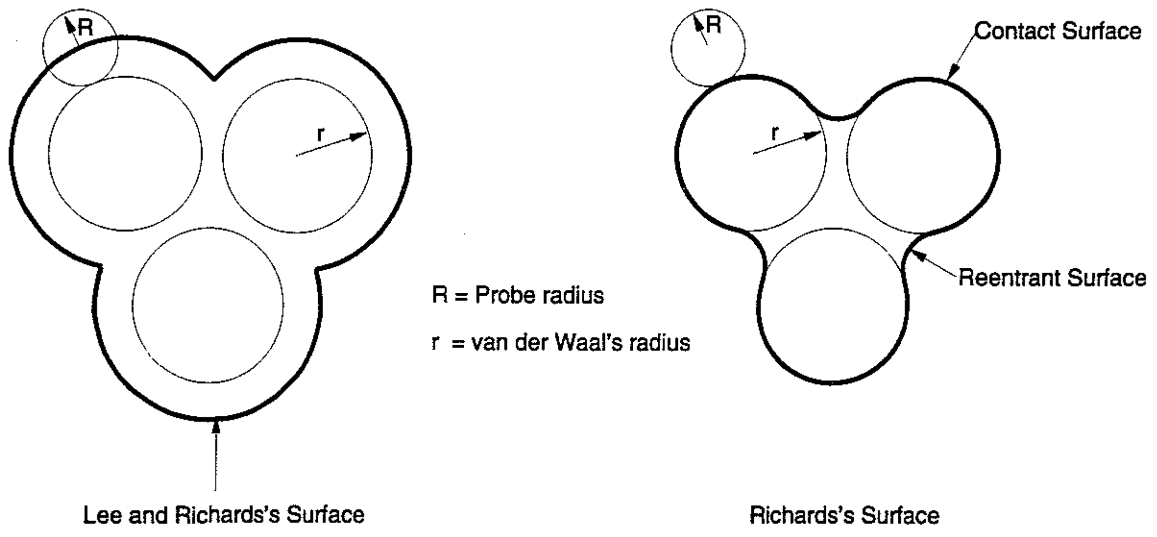


Figure 2.3: Molecular Surfaces

In this dissertation we shall be dealing with the molecular surface defined by Richards in 1977.

Chapter III

Detail Hierarchies in Computational Geometry

3.1 Introduction

In this chapter we shall survey the work that has been done in computational geometry for constructing detail hierarchies of objects. Most of this work can be subdivided into the following three categories:
(i) hierarchy for the boundary of a given set of points,
(ii) hierarchy for a given two-dimensional piecewise linear curve, and
(iii) hierarchy for a given three-dimensional piecewise linear (polygonal) surface.

The third category above had not received much attention till recently. We shall discuss the recent advances for that category in Section 7.2.

Before we proceed further, let us introduce some basic concepts and terminology from computational geometry.

3.2 Some Computational Geometry Concepts

This section explains some terms from computational geometry that we shall be using in the later sections. These have been illustrated in two dimensions, though they can be generalized to three and higher dimensions. In the following definitions, let P be a finite set of points.

We use R^d to denote the d-dimensional real space. E^d is used to denote the d-dimensional real-space in which the distance function is the Euclidean distance or the L_2 norm[1]

Convex Hull

The convex hull of P can be defined as the smallest convex set that contains P. It can also be defined as the intersection of all half-planes that contain P. An example of this is shown in Figure 3.1(b).

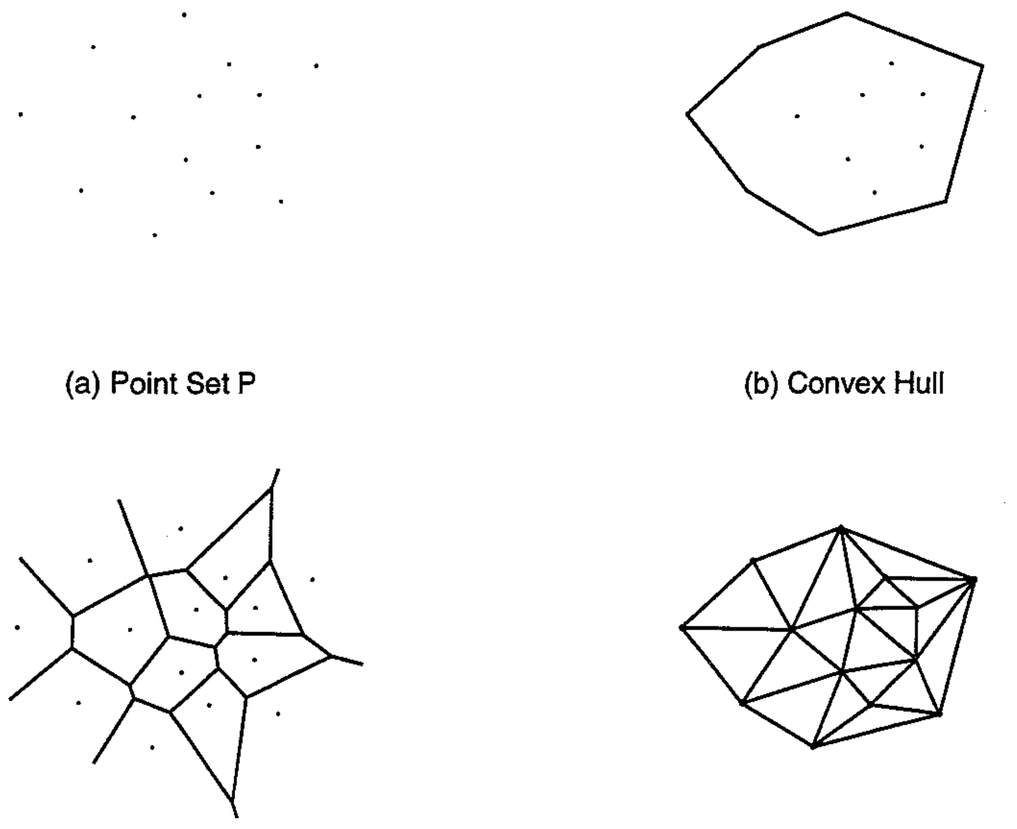


Figure 3.1: Some Basic Computational Geometry Concepts

[1]distance between two points $p(a_1, a_2, \ldots, a_d)$ and $p(b_1, b_2, \ldots, b_d) = (\sum_{i=1}^{d}(a_i - b_i)^2)^{1/2}$

Voronoi Diagram

The Voronoi diagram for a set of points P in the plane is the subdivision of the plane into a set of mutually exclusive, collectively exhaustive regions such that :
(a) each region R_i corresponds to a unique point $i \in P$.
(b) all the points of the plane contained in that region R_i are closer to its point i, than to any other point $j \neq i$ in P. An example of this is shown in Figure 3.1(c). Note that any edge of this diagram between adjacent regions R_i and R_j is the perpendicular bisector of the line joining the points i and j. The vertices of the Voronoi diagram are called the *Voronoi vertices*, the edges *Voronoi edges*, and the regions *Voronoi regions* or *Voronoi cells*. This diagram is named after the Russian mathematician G. M. Voronoi [Voronoi 07].

Delaunay Triangulation

The Delaunay triangulation for a set of points P in the plane is the dual (in the graph-theoretic sense) of their Voronoi diagram. In this triangulation, an edge is created between two points if and only if their Voronoi cells share an edge. As long as there are no degeneracies (2 points coincident, 3 points collinear, or 4 points co-circular) this is guaranteed to produce a valid triangulation. This is shown in Figure 3.1(d). Note that the convex hull is a subset of the Delaunay triangulation. This is named after B. Delaunay who proved that the dual of the Voronoi diagram of P is a triangulation of P [Delaunay 34].

Power Diagrams

Power diagrams are a generalization of the Voronoi diagrams. Voronoi diagrams are defined for simple points (or circles of equal radii) whereas power diagrams are defined for circles of possibly unequal radii. An easy way to understand this is through the following example.

Let us assume that the plane of the points of P represents the map of some geographical region and that the points of P represent the sites of radio transmitters

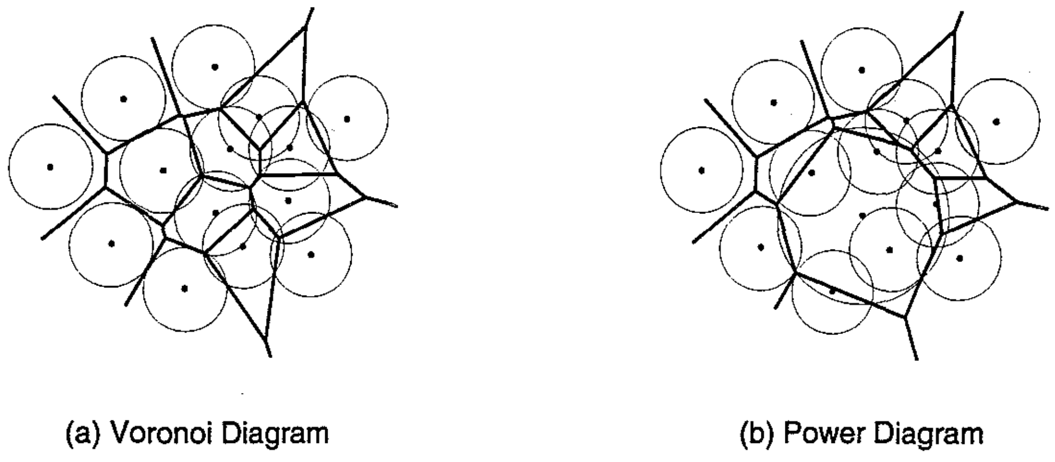


Figure 3.2: Voronoi and Power Diagrams

in that region. The signal power of the radio transmitters is assumed to have a conical distribution function such that the radius of the cone represents the power of the transmitter and is directly proportional to the transmitter's height. Thus the more powerful a transmitter, the greater its height.

Let us define the *dominant region* for a particular transmitter as the region in which its signals are stronger than the signals from any other transmitter. Let us first assume that all transmitters transmit their signals at the same power and are therefore at the same height. The dominant regions of the transmitters for the case where all transmitters have the same power are described by the Voronoi regions in the Voronoi diagram of the transmitter sites. This is shown in Figure 3.2(a). Instead of drawing three-dimensional cones to represent the transmitter-signal's power distribution functions we have simply drawn the circular bases of the cones.

It is easy to see that if we now increase the power of a particular transmitter, its dominant region will increase at the expense of others. The dominant regions for the case were the transmitters do not have the same power are described by power diagrams. This is shown in Figure 3.2(b), where the radius of one circle has been increased. Power cells are defined analogously to the Voronoi cells. Note that a point $i \in P$ in a power diagram may not lie within its power cell; its power cell may not even exist. However, every power cell corresponds to a unique point $i \in P$.

A *regular triangulation* is to a power diagram as a Delaunay triangulation is to a Voronoi diagram. Thus, in a regular triangulation an edge is created between two points i and $j \in P$ if and only if their power cells share an edge.

3.3 Detail Hierarchies for Boundaries of Points

A trivial hierarchy of detail for the boundary of a set of points in R^d can proceed from the minimum enclosing sphere, to the minimum enclosing ellipsoid, and then to the convex hull. This is a hierarchy in two senses: (a) closer and closer approximation to the point-set, and (b) takes more parameters to describe. This progression of detail is shown in Figure 3.3.

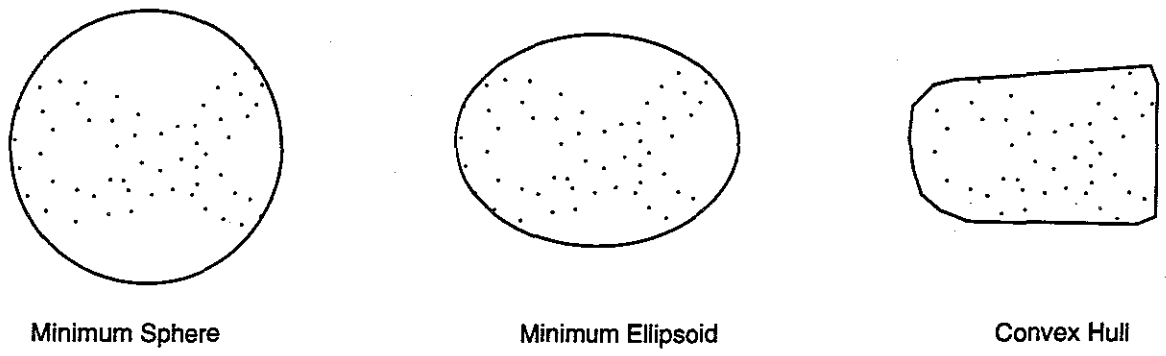


Figure 3.3: A Trivial Hierarchy of Boundary Detail

Higher detail descriptions of boundaries of points were not systematized until the invention of the α-hulls. We shall discuss these in Section 3.3.2.

3.3.1 Spheres, Ellipsoids, and Convex Hulls

It is possible to find the smallest enclosing sphere and ellipsoid for a set of n points in R^d in $O(\delta\delta!n)$ time [Welzl 91], where $\delta = d + 1$ for a sphere and $\delta = (d+3)d/2$ for an ellipsoid. The algorithm given in [Welzl 91] is quite easy to implement and delivers a good performance for low-dimensional spaces. It is also possible to frame the problem of finding the smallest enclosing sphere as a linear programming problem and then use the linear programming algorithms.

The first optimal $O(n \log n)$ algorithm for computing the convex hull of a set of points in the plane was given by Graham [Graham 72]. In terms of output-size-sensitive algorithms for computing the planar convex hull, the asymptotically optimal time of $O(n \log h)$ has been achieved by Kirkpatrick and Seidel [Kirkpatrick & Seidel 86], where h is the number of edges in the convex hull. For higher dimensions, these can be computed in time $O(n^{\lceil d/2 \rceil})$ for odd d [Seidel 81] and at logarithmic cost per face for even $d \geq 4$ [Seidel 86]. For good survey material on convex hulls see [Preparata & Shamos 85, Dobkin & Souvaine 87, Edelsbrunner 87, Graham & Yao 90].

3.3.2 α-Hull

An elegant generalization of the convex hulls for points in a plane is done by Edelsbrunner, Kirkpatrick, and Seidel [Edelsbrunner *et al* 83] by defining the notion of α-hulls. As mentioned before, one could define the convex hull of a set of points P in a plane as the intersection of all half-planes that contain P. Instead of using half-planes, they define the hulls for points by intersecting the interiors of discs or their complements. For $\alpha > 0$, the α-hull of a set of points P in two dimensions is defined to be the complement of the union of all disks of radius α containing no points of P. The notation used for defining the alpha hulls has changed in going from [Edelsbrunner *et al* 83] to [Edelsbrunner 92]. We shall use the more recent notation.

The term *open disc* is used to refer to the points that lie strictly inside a circle and the term *closed disc* is used to refer to the points that lie on or inside a circle. A *generalized disc* is a disc that either selects the points that are in the interior of a circle or the points that are exterior to a circle. In general, a generalized disc with a positive radius is used to refer to the points that are in the interior of a circle, whereas a generalized disc with a negative radius is used to refer to the points that are exterior to a circle. A generalized disc can be open or closed depending on whether it includes the points on its boundary. The complement of a generalized disc with radius $(+\alpha)$ is a generalized disc with radius $(-\alpha)$.

The α-hull of a set of points P is defined as the intersection of all closed generalized discs of radius $(-\alpha)$ that contain all the points of P. Thus, if $\alpha > 0$, we consider higher dimension, the worst-case combinatorial complexity of α-hulls in R^d is the same as that of the convex hulls in R^{d+1}. The α-hulls can be constructed from the Delaunay triangulation and their times are the same as those for convex hulls in one-higher dimension. Thus in a plane, α-hull of a set of n points can be constructed in time $O(n \log n)$ and in three dimensions in time $O(n^2)$.

The α-hulls have been defined for weighted points in [Edelsbrunner 92]. These can be constructed from regular triangulations of the weighted points, which are dual to the power diagrams as defined by Aurenhammer [Aurenhammer 88].

3.4 Detail Hierarchies for Polylines

We shall use the term *polyline* to denote a zero-order continuous (i.e. end-point continuity) chain of line segments none of which crosses another. A function $f : [x^-, x^+] \rightarrow R$ is a *piecewise-linear function* if its graph $y = f(x)$ is a polyline connecting the points $p_1, p_2, \ldots, p_n$, in that order such that $x^- = x(p_1) < x(p_2) < x(p_3) \ldots < x(p_n) = x^+$, where $x(p)$ is the x-coordinate of p. Thus, for a piecewise linear function, we require the polyline $p_1, p_2, \ldots, p_n$ to be strictly x-monotone. A *piecewise linear curve* is a polyline that is not required to be x-monotone. Thus, a piecewise linear curve is an arbitrary polyline on a plane.

Algorithms have been developed for generating a hierarchy of approximations for piecewise linear functions and piecewise linear curves. Two types of approximations have been considered:

- **Min-# Approximations**: Given some error bound $\epsilon > 0$, the objective is to minimize the number of vertices such that no point of the approximation is farther than ϵ distance away from some point on the actual curve.

- **Min-ϵ Approximations**: Given the number of vertices desired in the output approximation curve, minimize the error between the approximation curve and the input curve.

Imai and Iri [Imai & Iri 86], [Imai & Iri 88] have shown that the min-# approxi-

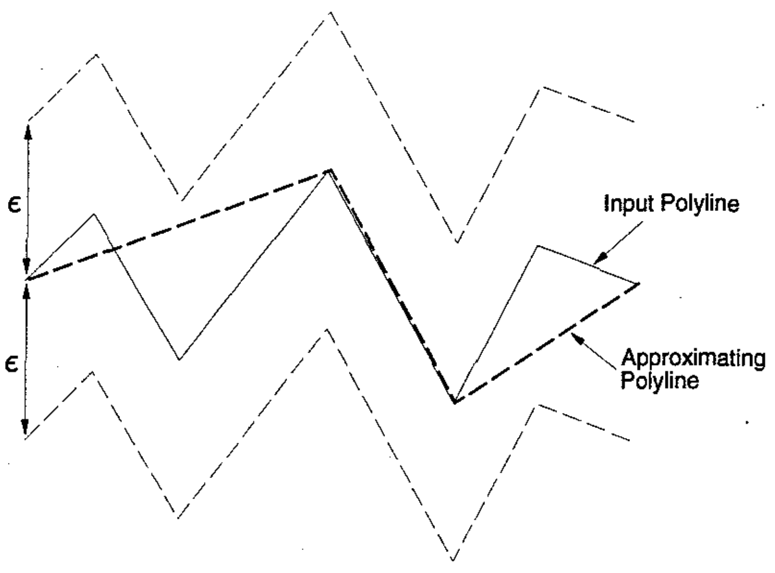


Figure 3.5: A Vertex-Subset Polyline Approximation

mation to a piecewise linear function can be accomplished in an optimal $O(n)$ time. The min-ϵ problem for the piecewise linear functions is much harder to solve and no efficient algorithms are known that solve it optimally for the general case.

The vertices of the approximation of a piecewise linear curve may or may not be a subset of the vertices of the input piecewise linear curve. Here, we shall focus on the problem in which the vertices of the new piecewise linear curve are a subset of the vertices of the input curve. Formally, given a piecewise linear curve C whose vertices are $p_1, p_2, \ldots, p_n$, in this order, an approximate piecewise linear curve for this consists of the vertices $p_{i(1)}, p_{i(2)}, \ldots, p_{i(m)}, 1 = i(1) < i(2) < \ldots < i(m) = n$, which are a subset of $p_1, p_2, \ldots, p_n$. This is shown in Figure 3.5.

Min-ϵ as well as the min-# approximation problems have been solved for this approximation scheme, using different measures of the approximation error criterion [Imai & Iri 88]. Let the error of approximating the curve between vertices $p_i, p_{i+1}, p_{i+2}, \ldots, p_j$ by the line segment p_ip_j be given by the maximum of the distance between the segment p_ip_j and the points $p_k (i \leq k \leq j)$. For this error measure, the min-# problem can be solved in time $O(n^2 \log n)$ [Melkman & O'Rourke 88] while the min-ϵ problem can be solved in time $O(n^2 (\log n)^2)$ [Imai & Iri 88].

Chapter IV

Linearly Scalable Computation of Smooth Molecular Surfaces

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. For examples of such molecular surfaces, refer to Figures 1.1, 1.2, and 4.4, 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.

4.1 Previous and Related Work

The analytic computation of the molecular surface was first done by Connolly [Connolly 83]. 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.* [Perrot *et al* 92] 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 issues remain unaddressed here. Their 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 surfaces for the interior cavities of a molecule.

We have reviewed α-hulls in Section 3.3.2. If we generalize the 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 [Edelsbrunner *et al* 83] 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 [Richards 77] 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 [Edelsbrunner & Mücke 94] 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 [Edelsbrunner 92], 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 is linear in n 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.

4.2 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 [Aurenhammer 87] and briefly reviewed in Section 4.2.1), which we call a *feasible cell* (the definition of a feasible cell appears in Section 4.2.3), 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.

4.2.1 Formal Notation

In this section we will introduce the definitions and notations that we will be using for the rest of the chapter. 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 [Aurenhammer 87].

4.2.2 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, $\forall j \in N_i, p(c_j, \rho_i) \leq 0$. 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 4.1 atoms j_1 and j_2 are neighbors to atom i, but not to each other.

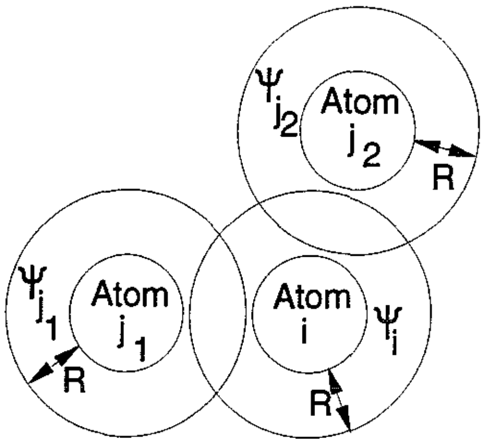


Figure 4.1: Defining Neighbors

4.2.3 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 $F_i = \bigcap_{j \in N_i} H_{ij}$. We will refine this definition of a feasible cell later in this section. Since a power cell PC_i is defined as $PC_i = \bigcap_j H_{ij}$, 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 4.2, 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.

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

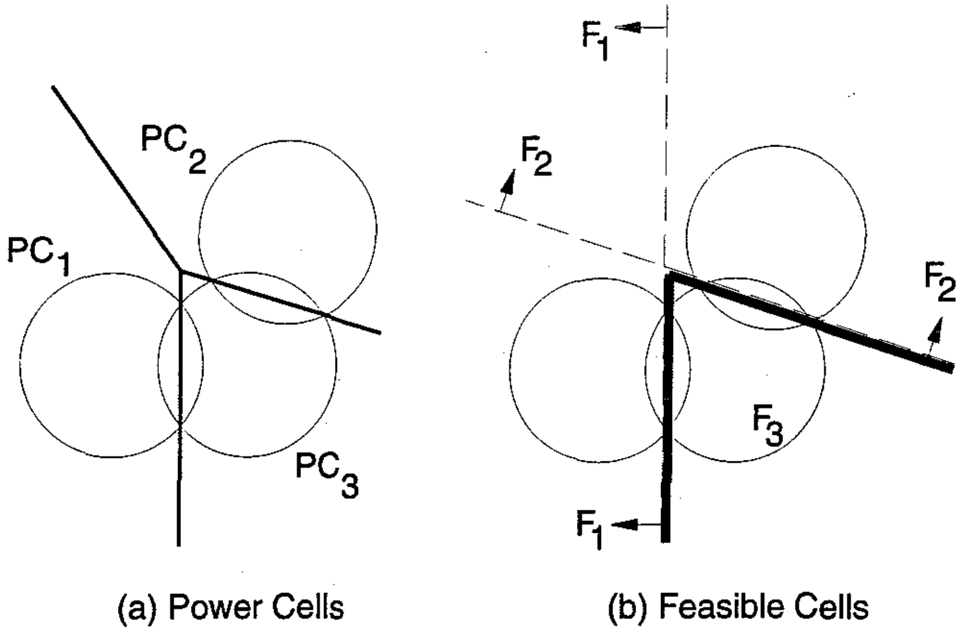


Figure 4.2: Power Cells and Feasible Cells

tessellation of the molecular surface.

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 defines 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, for all i. With this modification we are now ready to give the final definition of F_i as: $F_i = (\bigcap_{j \in N_i} H_{ij}) \bigcap (\bigcap_{f=0}^{3} H_f)$.

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 4.2.2. Then we compute $F_i = (\bigcap_{j \in N_i} H_{ij}) \bigcap (\bigcap_{f=0}^{3} H_f)$. 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 [Seidel 90]. All the atoms for which $F_i \neq \phi$ are classified as candidates for being surface atoms.

Note that this method does not reject atoms that are completely buried but adjacent to an internal cavity of the molecule. For such atoms, F_i will be non-null and would contribute surface patches. Thus, our algorithm correctly computes the surfaces for the internal cavities of the molecule.

4.2.4 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 [Edelsbrunner 92] 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 [Preparata & Shamos 85]. 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, 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 here 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 4.3 a component defined by three chordales Π_{ij}'s intersecting Ψ_i has been shown with its interior unshaded.

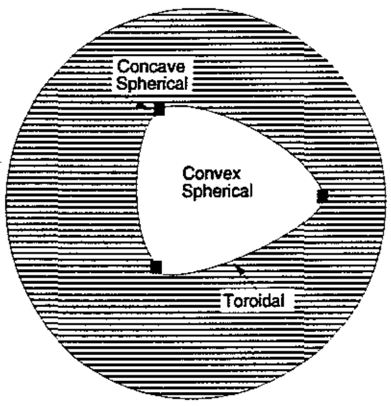


Figure 4.3: Determination of Molecular Surface Patches

4.2.5 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 $O(k \log k)$, 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 (parallel random-access machine) model of parallel computation.

In the parallel computation of molecular surface, it is important to ensure that two adjacent surface patches that have been generated on two different processors do not have any cracks between them. In other words, the tessellation of the two adjacent patches should share all the vertices along the common boundary edges. This is easy to ensure amongst the surface patches for a single surface atom that are generated at the same processor, as all the information about the patches is locally available. However, ensuring that no cracks arise in the toroidal and concave spherical patches that are typically shared across two or three processors, respectively, is more interesting. We solve this problem by having each processor generate sub-patches (half of a toroidal patch or a third of a concave spherical triangular patch). Tessellation at the boundary of the sub-patches is done based on the length of the shared sub-patch edges and a global maximum-triangle-edge-length parameter t. Thus, if a sub-patch edge has length l units, we generate $(\lceil l/t \rceil - 1)$ additional, equispaced vertices along the sub-patch edge independently at the two processors sharing that edge. This ensures a continuous tessellation of the molecular surface with hardly any cracks. Some cracks do arise due to precision problems when l is almost equal to a multiple of t. In such cases, even slight differences in the value of l evaluated on different processors (from different parameters) cause the introduction of an extra vertex in the shared boundary edge. However, such cases are reasonably rare in practice.

4.2.6 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 a 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 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 generic perturbation scheme proposed by Emiris and Canny [Emiris & Canny 92], which perturbs the j^{th} dimension of the i^{th} point as:

$$p_{i,j}(\epsilon) = p_{i,j} + \epsilon(i^j \bmod q) 1 \leq i \leq n, 1 \leq j \leq d \qquad \text{(IV.1)}$$

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

4.3 Results

Our implementation has been done on Pixel-Planes 5 [Fuchs *et al* 89], although it is general enough to be easily portable to any other parallel architecture. Table 4.1 shows our timings for computation and display of the molecular surface for various molecules for a probe-radius of 1.4$\AA$. 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 4.1. 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 4.1 refers to the complexity of the computed surface in thousands of triangles.

As can be seen, the value of k, the average number of neighbors, is fairly constant for a given probe-radius over different molecules. Using concepts from the mathematical theory of packing of spheres and some reasonable assumptions, we shall show in Chapter V that for protein molecules, k is expected to be less than 140.

Table 4.2 shows the times for the generation of the molecular surface for crambin using 24 processors, and different probe-radii varying from 1.0$\AA$ to 10.0$\AA$.

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 4.1: Molecular Surface Generation for $1.4\AA$ Probe-Radius.

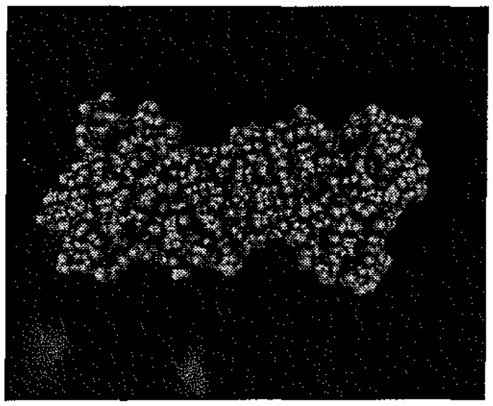
Probe-Radius	$1.0\AA$	$1.4\AA$	$2.8\AA$	$5.0\AA$	$10.0\AA$
Times (sec)	0.23	0.24	0.32	0.53	0.95
k	29.9	41.3	91.8	191.5	318.3
Triangles	16K	14K	12K	11K	11K

Table 4.2: Crambin Molecular Surface Generation (24 Intel i860 Processors)

The smooth molecular surfaces for crambin with probe-sphere radii of $1.4\AA$, $2.8\AA$, $5.0\AA$, and $10.0\AA$ are shown in Figure 1.2(a), (b), (c), and (d), respectively. The smooth molecular surfaces for dihydrofolate reductase and superoxide dismutase for a probe-sphere radius of $1.4\AA$ are shown in Figure 4.4 (a) and (b), respectively.

4.4 Conclusions

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.


(a) Dihydrofolate reductase) (2980 atoms)

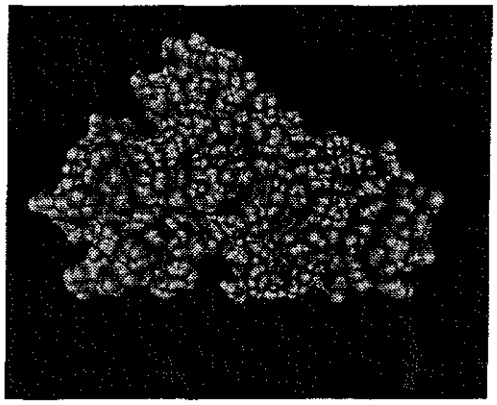

(b) Superoxide dismutase (4392 atoms)

Figure 4.4: DHFR and SOD Molecular Surfaces, 1.4$\AA$ Probe-Radius

We would like to point out here that for large globular proteins one can expect $O(n^{2/3})$ atoms to lie on the surface. The other $O(n^{1/3})$ interior atoms will not contribute to the molecular surface (assuming that there are no internal molecular cavities). We can check for empty feasible cells by linear programming in $O(k)$ time per atom. If the feasible cells for the buried atoms are empty, we are required to spend $O(k \log k)$ time for generating the molecular surface patches for just the $O(n^{2/3})$ surface atoms. For such cases, the sequential complexity of our algorithm is $O(nk + n^{2/3}k \log k)$.

Chapter V

Estimating the Number of Unit Spheres Inside a Larger Sphere

This chapter outlines a set of techniques that can be used for efficiently estimating the number of unit spheres that can be placed within a given sphere of larger radius. For the case of packing of mutually disjoint spheres these techniques provide upper bounds, whereas for the case of packing of intersecting spheres these techniques provide good estimates. These techniques are directly applicable to the problem of estimating the number of neighbors under the commonly used Euclidean distance function, the L_2-norm.

Thus, for instance, these techniques can be used to estimate:

- the number of atoms that "effectively" interact via Lennard-Jones attractive/repulsive energy function (which becomes infinitesimal beyond $6 - 7\mathring{A}$), as mentioned in [Surles 92].

- the number of points that need to be considered in a relaxation procedure where the force of repulsion due to a point falls off linearly with the Euclidean distance from it (and thus becomes zero at a fixed radius), as used in [Turk 92].

- the number of atoms that could potentially define a solvent-accessible molecular surface due to a given atom, as described in [Varshney *et al* 94a, Varshney & Brooks 93].

Although we shall be primarily working in three-dimensional Euclidean space, the techniques presented in this chapter can be generalized to higher dimensions.

The rest of this chapter is organized as follows. In Section 5.1 we review some of the concepts from the mathematical theory of packings of spheres. In Section 5.2 we briefly outline the problem of computing the solvent-accessible molecular surface for proteins which we shall use to illustrate the different techniques. Sections 5.3 and 5.4 outline the various techniques. Finally, in Section 5.5 we present our conclusions.

5.1 A Review of the Theory of Sphere Packings

In this section we shall briefly review some of the relevant results from the theory of packing of spheres.

5.1.1 The Sphere Packing Problem

The classical sphere-packing problem is to find out how densely can one pack mutually disjoint spheres of equal radii in three-dimensional Euclidean space. The general sphere-packing problem is to find the densest packing of disjoint equal-radius spheres in n-dimensional space.

Let us define the maximum density $\delta(n)$ of a packing of spheres in n-dimensions to be the proportion of the space that is occupied by spheres in their tightest packing.

The value of $\delta(2)$ was proved to be $0.9069\ldots$ in 1892 by A. Thue. The classical sphere-packing problem for three dimensions is still an open problem, perhaps one of the most famous open problems in mathematics. For several years, the best known upper bound for the densest packing of spheres was $\delta(3) \leq 0.7796\ldots$, proved by Rogers [Rogers 58]. This was improved by Lindsey [Lindsey 86] to $0.7784\ldots$ in 1986. A packing of density $\pi/\sqrt{18} = 0.7405\ldots$, can actually be achieved by arranging spheres in the form of a face-centered cubic lattice. Thus, at present, we know that $0.7405 \leq \delta(3) \leq 0.7784$, though Rogers [Rogers 58] remarks — "many mathematicians believe" that the correct answer is 0.7405. For the values of $\delta(n), n > 3$, the interested reader can refer to [Conway & Sloane 88].

5.1.2 Sphere Packings in Spherical Space

A problem that is closely related to the problem of packing spheres in the Euclidean space is that of packing $(n-1)$-dimensional spheres (spherical caps) of angular diameter ϕ on the surface of an n-dimensional unit sphere.

Let us define $A(n, \phi)$ to be the maximal number of mutually disjoint spherical caps of angular diameter ϕ that can be placed on the surface of a n-dimensional unit sphere. Rankin [Rankin 55] has found the exact values for $A(n, \phi)$ for $\phi \geq \pi/2$:

$$A(n, \phi) = 1, \pi < \phi \leq 2\pi,$$

$$A(n, \phi) = \lfloor 1 - \sec(\phi) \rfloor, \sec^{-1}(-n) \leq \phi \leq \pi,$$

$$A(n, \phi) = n + 1, \pi/2 < \phi < \sec^{-1}(-n),$$

$$A(n, \pi/2) = 2n.$$

5.1.3 Multiple Packings of Spheres

Till now, we have been assuming that the spheres that are used in the packing are mutually disjoint. What happens if we allow them to intersect? This concept has been studied under the notion of *multiple packings*. A set of spheres is said to form a *k-fold packing* if each point of the space belongs to at most k spheres. Let the maximum density of a k-fold spherical packing in n-dimensions be denoted by $\delta_k(n)$. Fejes Tóth [Fejes Tóth 79] has shown that $\delta_2(3) \leq 1.826$.

The results reproduced above are those most relevant to our research. They form but a tiny fraction of the results from the deeply exciting mathematical theory of packing and covering. The interested reader can further study this subject, starting perhaps with the classical book by Rogers [Rogers 64]. For more recent results in this field see the survey article by Fejes Tóth [Fejes Tóth 83] and the book by Conway and Sloane [Conway & Sloane 88].

5.2 Solvent-Accessible Protein Surfaces

In Section 5.2.1, we shall quickly review the terminology of Section 4.2.1 and the concept of neighborhood as described in Section 4.2.2. After that, we shall look at some relevant properties of proteins, the molecules for which the solvent-accessible surfaces are most often computed, and then list our assumptions for this problem.

5.2.1 Terminology

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 σ 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.

We shall be assuming that the atom i of a molecule is represented as a sphere $S_i = \sigma(c_i, r_i)$, where c_i and r_i are the center and radius, respectively, of atom i. Let the radius of the probe sphere be r_{probe}. We define the *extended-radius sphere* for atom i to be $\Psi_i = \sigma(c_i, r_i + r_{probe})$. 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.

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 $\sigma(c_i, r_i + 2r_{probe} + \max_{j=1}^{n} r_j)$. 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, $N_i = \{j | d(c_i, c_j) < r_i + 2r_{probe} + r_j\}$, 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, $\forall j \in N_i, p(c_j, \rho_i) \leq 0$. In Figure 5.1 atoms j_1 and j_3 are neighbors to atom i, but atom j_2 is not.

Let us define the *primary region* P_i for atom i to be the interior of the sphere $P_i = \sigma(c_i, R_{primary})$, where $R_{primary}$ is the smallest radius such that it completely encloses all spheres whose centers lie in the region of influence ρ_i. Formally, $S_j \subset P_i, \forall j \in N_i$. This has been shown in Figure 5.2.

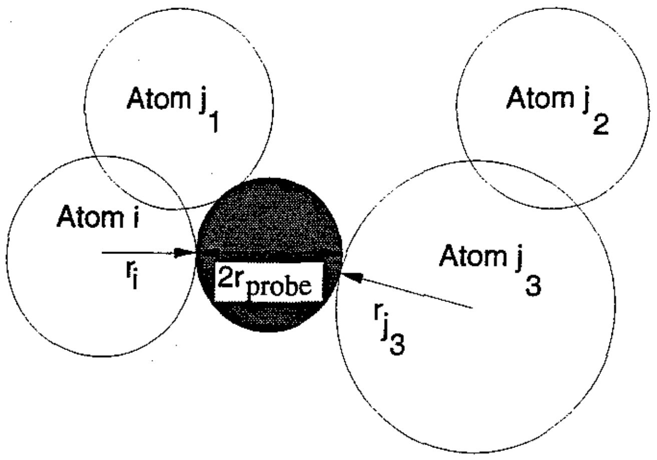


Figure 5.1: Determination of Neighboring Atoms

Let k be the average number of neighboring atoms and let k_{max} be an upper bound on k.

5.2.2 Proteins

We recall from Section 2.1 that 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.

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 except for a few exceptions. These exceptions are — (a) the three aromatic amino acid residues (phenylalanine, tyrosine, and tryptophan), which have either one or two cycles each in the side chain (b) proline – which forms a cycle through a bond between its side chain and main chain, (c) histidine – which has one cycle in its side chain, and (d) 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

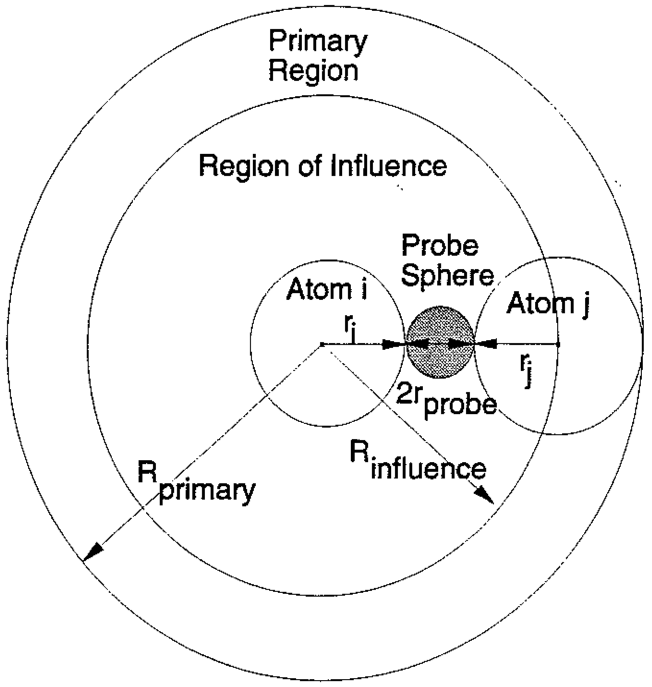


Figure 5.2: Primary and Influence Regions

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.

5.2.3 Assumptions

Since it is extremely difficult to derive bounds for k_{max} 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 an equal radii $r_a = 1.75Å$. For comparison, the radii of various commonly occurring atoms in proteins are indeed close to each other: Carbon – 1.9$\AA$, Nitrogen – 1.7$\AA$, Oxygen – 1.5$\AA$, Sulfur – 2.00$\AA$, Phosphorous – 2.10$\AA$ [Richardson 94, Weiner *et al* 84]. These values assume implicit hydrogens.

C: The average distance between the centers of any two bonded atoms is $l = 1.5\AA$ – the bond length of a single C-C bond [Weiner *et al* 84].

D: The radius of the probe-sphere, which determines the radius of the region of influence, is $r_{probe} = 1.4\AA$.

With these assumptions, the radius of the region of influence is $R_{influence} = 2 \times r_a + 2 \times r_{probe} = 6.3\AA$.

5.3 Volume-based Techniques

This section explores some techniques that use volume-based arguments to estimate the number of unit spheres that can be placed within a sphere of larger radius. These techniques have been explored for the cases where the unit spheres are mutually disjoint or intersecting.

5.3.1 Mutually Disjoint Spheres

If all the unit spheres are mutually disjoint, then a trivial upper bound for k_{max} can be given by the ratio of the respective volumes. Thus,

$$k_{max} \leq \frac{4\pi}{3} R^3_{primary} / (\frac{4\pi}{3}) = R^3_{primary}$$

From the theory of packing of spheres we know that the maximum density of packing in three dimensions is given by $\delta(3) = 0.7784$. Thus, we can obtain an improved bound:

$$k_{max} \leq \delta(3) R^3_{primary} \tag{V.1}$$

For our example of solvent-accessible surfaces, we can use (V.1) by considering each atom to be represented by a sphere with radius half the average bond-length $= l/2 = 0.75\AA$. The radius of the primary region would be $R_{primary} = (R_{influence} + 0.75)/0.75 = 9.4$. Here we divide by 0.75 to normalize $R_{primary}$ so that the radius of each sphere representing an atom is unity. Using (V.1) we get $k_{max} \leq 0.7784 \times 9.4^3 = 646$.

5.3.2 Intersecting Spheres

If the unit spheres are allowed to overlap each other completely, it seems best to use results from the multiple packings of spheres. Thus, if a k-fold packing of unit spheres is permitted, one can essentially use (V.1) above with $\delta(3)$ replaced by $\delta_k(3)$.

However, the more common case is one where the spheres are allowed to overlap each other only to a limited extent. Thus, the centers of two spheres are not allowed to get arbitrarily close to each other. In such cases one can proceed by first finding the *smallest volume per center*, which is the volume within which the center of no other sphere is allowed. In certain cases one can get a better (larger) value of the smallest volume per center by dividing the smallest volume for a collection of m centers with m. We can then compute k by dividing the total volume with the smallest volume per center. We illustrate this method for the case of solvent-accessible surface problem.

As shown in Figure 5.2, we are interested in computing the number of spheres whose centers lie within the region of influence. From Section 5.2.3, we have the radius of the region of influence $= 6.3\AA$, where the radius of each atom is $1.75\AA$. Normalizing the former, so that we have unit radius spheres, we have $r_{influence} = 6.3/1.75 = 3.6$.

In computing the solvent-accessible molecular surface for proteins we can assume that each atom is covalently bonded to two other atoms on an average, as stated in Section 5.2.2. This in conjunction with assumption C in Section 5.2.3, implies that the maximum number of atom centers that can lie within a sphere of radius $l = 1.5\AA$ is 3. Normalizing l for the coordinate system of unit spheres, we have $l = 1.5/1.75 = 0.857$. This is shown in Figure 5.3.

Volume per center $\geq \frac{1}{3}(\frac{4\pi}{3}(0.857)^3) = \frac{4\pi}{3}(0.2099)$ and total volume $= \frac{4\pi}{3}r^3_{influence} = \frac{4\pi}{3}(46.65)$.

Therefore, $k \leq \frac{\frac{4\pi}{3}(46.65)}{\frac{4\pi}{3}(0.2099)} = 222$.

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

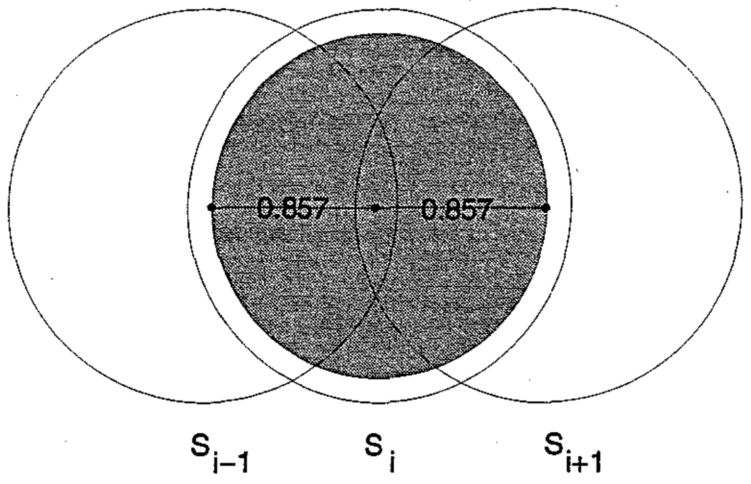


Figure 5.3: Smallest Volume per Three Centers

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

With the above assumption we have the volume per center $\geq \frac{1}{3}(\frac{4\pi}{3}(1)^3) = \frac{4\pi}{3}(0.3333)$.

Therefore, $k \leq \frac{\frac{4\pi}{3}(46.65)}{\frac{4\pi}{3}(0.3333)} = 139$.

5.4 A Surface-based Technique

In this section we will explore a surface-based technique to bound the number of unit spheres that can be placed around a unit sphere centered at the origin, such that all of them lie inside a larger sphere of a given radius centered at the origin. First, we will explain the intuition behind the problem by considering a simpler version of the problem. Next, we will generalize the solution technique to work under general conditions. Finally, we will demonstrate the use of this technique to estimate the number of solvent-accessible neighbors in proteins.

5.4.1 Special Case

We recall from Section 5.2.1 that we denote a sphere of radius r centered at c by $\sigma(c, r)$. Let us first consider the following simpler problem: "Bound the number of unit spheres in $\sigma(0,r) \setminus \sigma(0,1)$, $1 \leq r < 4$". Here, $\sigma(0,r) \setminus \sigma(0,1)$ denotes the region that lies outside a unit radius sphere centered at the origin, but on or inside the sphere of radius r centered at the origin. Let us further assume for now that all the unit spheres are mutually disjoint.

After some thought, we can convince ourselves that all unit spheres which lie in $\sigma(0,r) \setminus \sigma(0,1)$ intersect $\sigma(0,\sqrt{r})$, $1 \leq r < 4$. This is shown in Figure 5.4. We shall henceforth use the term *shell* to refer to the surface of a sphere on which we are computing the intersections of the unit spheres. Thus, in this case $\sigma(0,\sqrt{r})$ is the

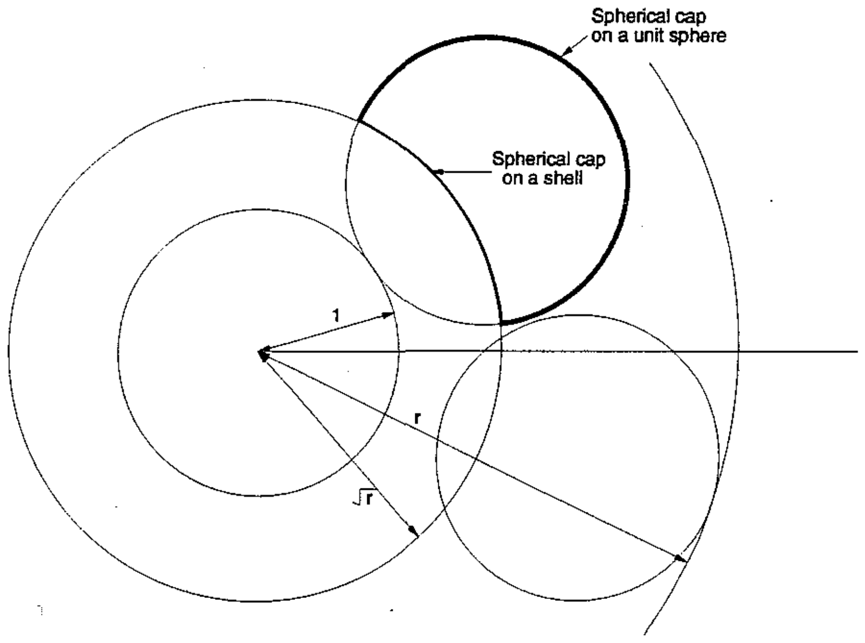


Figure 5.4: Intersections With a Single Shell

As can be seen in Figure 5.4, a unit sphere intersects the shell forming two spherical caps – one on the shell and the other on the unit sphere. Let the minimum angular radius of the spherical caps on the unit spheres be ϕ_{min}. G. Fejes Tóth [Fejes Tóth 79] has claimed that for this case $\phi_{min} \geq \arcsin\left(\sqrt{(4-r)}/2\right)$, and indeed it is easy enough to verify this.

Now, to bound the number of unit spheres, we just need to bound the number of spherical caps on the shell $\sigma(0,\sqrt{r})$. Using the terminology introduced in Section 5.1.2, we observe that $k_{max} \leq A(3, 2\phi_{min})$. For $\phi_{min} \geq \pi/4$, we can use the formulas from Rankin [Rankin 55] reproduced in Section 5.1.2 and that will give us a bound on k_{max}.

However, the exact values of $A(3, \phi)$, for all values of ϕ are not yet known. For values of ϕ_{min} for which $A(3, 2\phi_{min})$ is not known, we can simply divide the total surface area of the shell by the surface area of the smallest spherical cap on it, to get an approximate upper bound for k_{max}.

5.4.2 General Case

There are two different ways in which the problem as defined in the previous section, can be generalized. First, we should allow r to assume values larger than 4. Second, we should lift the restriction of mutually disjoint unit spheres and allow them to intersect.

Before looking at the completely general case, let us first consider the case where $1 \leq r < 6$, and any sphere can intersect at most n other spheres. Thus, we are interested in a $(n+1)$-fold packing of unit spheres.

Consider, $\sigma(0, 1)$, the unit sphere centered at the origin. Since no more than n spheres can overlap its center, the total number of its neighbors whose centers lie inside $\sigma(0, 1)$ is at most n. We also note that any unit sphere that lies completely inside $\sigma(0, r)$ will have its center inside $\sigma(0, r-1)$. Thus, we are interested in placing a bound on the number of spheres whose centers lie in $\sigma(0, r-1) \setminus \sigma(0, 1)$, while they form a $(n+1)$-fold packing. Let $N(i, j)$, $i \leq j$ denote the number of spheres forming a $(n+1)$-packing such that their centers lie in $\sigma(0, j) \setminus \sigma(0, i)$.

We are interested in computing $N(0, r) = N(0, 1) + N(1, r-1) + N(r-1, r)$. For us, $N(0, 1) \leq n$ and $N(r-1, r) = 0$. Thus, $N(0, r) \leq n + N(1, r-1) + 0$ and we shall now attempt to bound $N(1, r-1)$.

Theorem 1 *If $1 \leq r < 6$, all unit spheres whose centers lie in $\sigma(0, r-1) \setminus \sigma(0, 1)$ will intersect either $\sigma(0, \sqrt{(r+2)/2})$ or $\sigma(0, \sqrt{(r^2-r+2)/2})$ in a minimum angular radius of $\phi_{min} = \arcsin\sqrt{(6-r)(r+2)}/4$.*

Proof: Let $H(\sigma_1, \sigma_2)$ denote the plane passing through the intersection of $\sigma_1(c_1, r_1)$ and $\sigma_2(c_2, r_2)$, and let $e(\sigma_1, \sigma_2)$ denote the distance of this plane from the origin. In Figure 5.5(a), the intersection of σ_1 and σ_2 is shown. Figure 5.5(b) shows a magnified view of the same and labels $e(\sigma_1, \sigma_2)$, ϕ, etc.

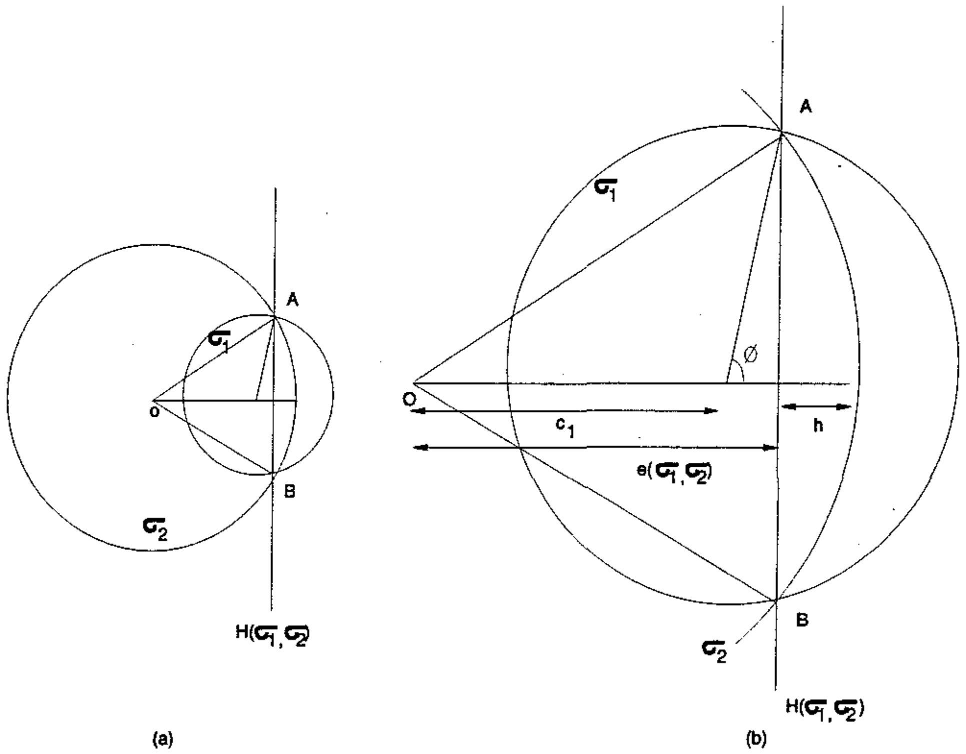


Figure 5.5: Computing ϕ_{min}

We will be using the polar coordinates (r, Θ, Φ) to specify the centers of the spheres. Without loss of generality, let us consider the centers to lie along $\Phi = 0, \Theta = 0$.

We will first prove that if the center of a unit sphere lies between $(1,0,0)$ and $(r/2,0,0)$, it will intersect the sphere $\sigma(0, \sqrt{(r+2)/2})$ with a minimum angular radius of $\arcsin\sqrt{(6-r)(r+2)}/4$. Figure 5.6 shows this.

It is easy to see from Figure 5.5 that to minimize ϕ, the center of σ_1 should be as far away from the boundary of σ_2 as possible. Given that the center has to lie between $(1,0,0)$ and $(r/2,0,0)$, the two possibilities for minimizing ϕ are: $(1,0,0)$ or $(r/2,0,0)$.

Let in Figure 5.5, σ_1 be a unit sphere centered at $(1,0,0)$: $\sigma((1,0,0),1)$, and let

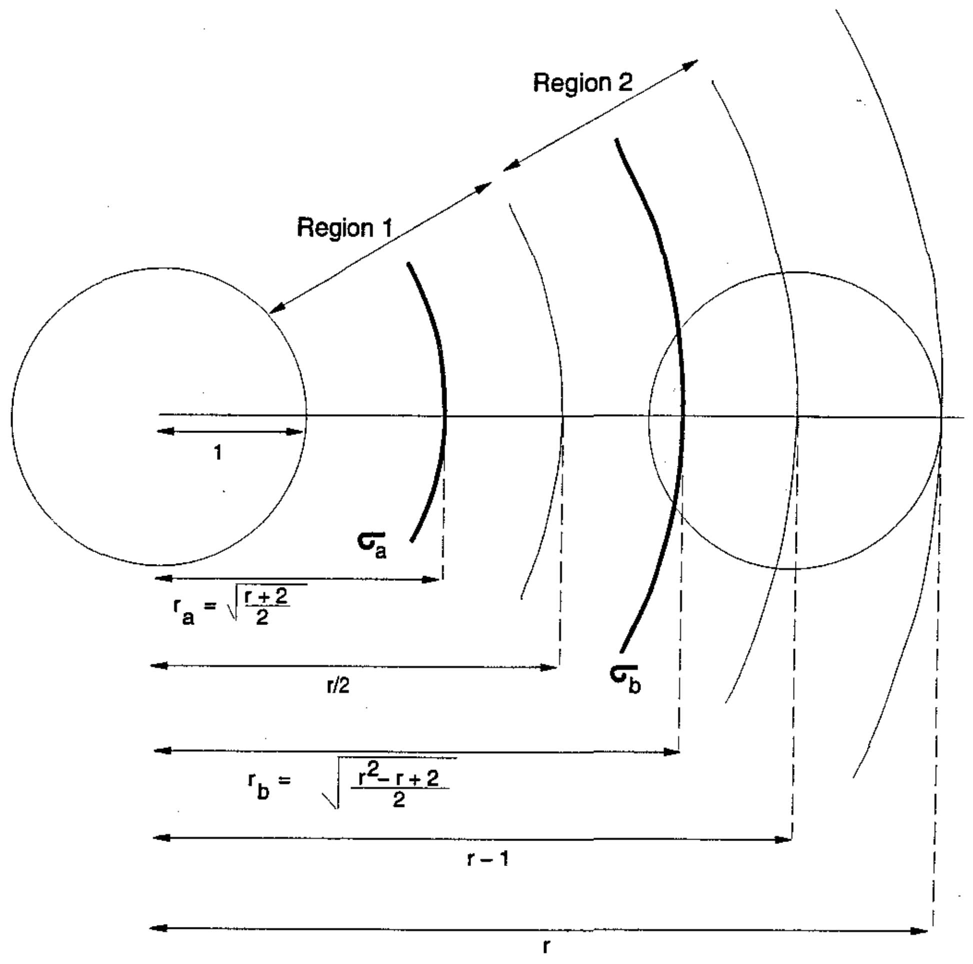

Region 2
Region 1
1
σ_a
$r_a = \sqrt{\frac{r+2}{2}}$
r/2
σ_b
$r_b = \sqrt{\frac{r^2-r+2}{2}}$
r – 1
r

Figure 5.6: Two Nested Shells σ_a and σ_b

$\sigma_2 = \sigma((0,0,0), \sqrt{(r+2)/2})$. The distance of their radical plane (passing through their intersection), from the origin is: $d(\sigma_1, \sigma_2) = (r+2)/4$. Thus, as can be seen from Figure 5.5(b), $\cos\phi = e(\sigma_1, \sigma_2) - c_1 = (r-2)/4$ and $\sin\phi = \sqrt{1-\cos^2\phi} = \sqrt{(6-r)(r+2)}/4$.

Similarly, if we consider $\sigma_1 = \sigma((r/2,0,0),1)$ and $\sigma_2 = \sigma((0,0,0), \sqrt{(r+2)/2})$, we will again get $\sin\phi = \sqrt{(6-r)(r+2)}/4$. Therefore, in *Region 1*, $\phi_{min} \geq \arcsin\sqrt{(6-r)(r+2)}/4$.

The proof that if the center of the unit sphere lies between $(r/2,0,0)$ and $(r-1,0,0)$, it will intersect the sphere $\sigma(0, \sqrt{(r^2-r+2)/2})$ in at least an angular radius of $\arcsin\sqrt{(6-r)(r+2)}/4$, is quite similar and can be proved along the same lines. $\square$

To bound $N(1, r-1)$ we would like to bound the number of circles with a minimum angular radius of ϕ_{min} that can occur in a $(n+1)$-fold packing on the surfaces of $\sigma_a = \sigma(0, \sqrt{(r+2)/2})$ and $\sigma_b = \sigma(0, \sqrt{(r^2-r+2)/2})$. We do this by dividing the surface area of the whole sphere (σ_a or σ_b) with the surface area of one such spherical cap of minimum angular radius.

The value of h (as shown in Figure 5.5(b)) for the smaller shell σ_a (shown in Figure 5.6) can be computed to be $h_a = \sqrt{(r+2)/2} - (r+2)/4$ and for the larger sphere σ_b to be $h_b = \sqrt{(r^2-r+2)/2} - (3r-2)/4$.

The surface area of a spherical cap of height h on the surface of a sphere of radius r is given by $2\pi rh$.

Therefore:

$$N(1, r-1) \leq (n+1)(\frac{4\pi r_a^2}{2\pi r_a h_a} + \frac{4\pi r_b^2}{2\pi r_b h_b}) = (n+1)(\frac{2r_a}{h_a} + \frac{2r_b}{h_b}) \tag{V.2}$$

In general, for s shells, let the radii defining the various regions be given by $x_0, x_1, \ldots, x_s$ (first region is $\sigma(0, x_1) \setminus \sigma(0, x_0)$, second region is $\sigma(0, x_2) \setminus \sigma(0, x_1)$, etc.), and the radii of the shells be given by $r_1, r_2, \ldots, r_s$ ($x_0 < r_1 < x_1, x_1 < r_2 < x_2, \ldots, x_{s-1} < r_s < x_s$). By imposing the constraint that all the unit spheres in $\sigma(0, r) \setminus \sigma(0, 1)$, intersect at least one of the s shells in the same minimum angular radius, we have derived the following equations:

$$x_0 = 1 \tag{V.3}$$

$$x_s = r - 1 \tag{V.4}$$

$$r_i^2 = x_i x_{i-1} + 1, 1 \leq i \leq n \tag{V.5}$$

$$r_i^2 + r_{i-1}^2 = 2(x_{i-1}^2 + 1), 2 \leq i \leq n \tag{V.6}$$

For a given value of r and s, one can solve the above system of simultaneous equations to get the value of the various r_i and x_i. The value of ϕ_{min} can thus be computed for each shell and can then be used to compute $N(0, r)$.

For the problem of computing the number of solvent-accessible neighbors, we have: radius of an atom $r_a = 1.75\mathring{A}$ and the probe radius $r_p = 1.4\mathring{A}$. Then $R_f =$

$2 \times r_a + 2 \times r_p + r_a = 8.05$. Normalizing it so that the original atom has unit radius, we have $r = R_f/r_a = 4.6$. For $r = 4.6$ and $n = 2$, we get: $r_a = 1.8165, h_a = 0.166, r_b = 3.046, h_b = 0.0963$. Substituting these values in (V.2), we get: $N(0,r) \leq n + N(1, r-1) = 2 + 252 = 254$.

Note that due to our assumption that on an average no point in space is covered by more than three spheres, the number of solvent-accessible neighbors we have found is an estimate and not an actual upper bound. However, the technique itself is general and can be used to derive upper bounds for the number of unit spheres that can be placed inside a larger sphere, given an upper bound on the number of spheres that can cover any point within the larger radius sphere.

5.5 Conclusions

We have described several techniques to bound or estimate the number of unit spheres that can be accommodated inside a larger sphere of a given radius. Depending on the application, one of these techniques could be used to derive a good estimate on the number of neighbors. For our application of computing the solvent-accessible neighbors in a protein molecule, the best estimate that we achieved was by using the volume-based technique of Section 5.3.2, which yielded 139 to be the maximum average number of neighbors one could expect for a probe radius of $1.4\AA$.

Chapter VI

Molecular Interface Surfaces

6.1 Surfaces at Molecular Interfaces

One of the important factors that influences the position and orientation of the protein with respect to the substrate in protein-substrate docking is the geometric fit or surface complementarity between them. It is quite difficult to visualize the molecular surface at the interface of the protein and substrate. Traditionally, the interface has been studied by using a clipping plane that is moved along the z-axis in the screen-space [Richardson 92]. This enables one to step-through the interface studying its cross-sections in a manner similar to that a physician employs while studying the various CT-scans of a patient one at a time. This essentially means studying a three-dimensional molecular interface in a two-dimensional manner, which is quite tedious and hard to understand.

Things get even harder to visualize by the clipping-plane method when one is studying an interface that does not lie in a plane or when one is interested in a simultaneous study of pairwise-interfaces across three or more molecular sub-units. Such cases do occur in practice, for example, in studying the packing of α-helices in crystalline protein structures.

We have developed an approach that allows biochemists to visualize the inter- and intra-molecular interfaces in three dimensions. The clipping of the molecular surfaces is defined by a piecewise polygonal surface derived from the power-diagrams defined over the participating molecular units. This provides biochemists with a powerful tool to study the surface complementarity across molecular interfaces in a natural three-dimensional manner.

6.2 Computation of Molecular Interface Surfaces

Since we are most interested in visualizing the interface between two molecular units, let us first characterize it.

Construct a single power-diagram of the spheres/atoms from the two molecules A and B consisting of n and m atoms, respectively. Let these two sets of atoms be represented as $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_m\}$. Each face of this power-diagram would be defined by two atoms. If the two atoms defining a face are $a_i, 1 \leq i \leq n$ and $b_j, 1 \leq j \leq m$, i.e. they come from two different molecules, then let us label such a face as an *interface-face.* Interface-faces for two molecules are shown in Figure 6.1 in bold. Let us define an *interface-cell* of the power diagram to be a cell that has at least one *interface-face.* Thus, the interface-cells would occur one deep on either side of the inter-molecular interface as shown in Figure 6.1. Let us define *interface-atoms* to be those atoms whose cells are interface-cells.

We note that the molecular interface between molecules A and B is completely defined by the piecewise planar surface formed by the interface faces. A two-dimensional version of this problem is shown in Figure 6.1 in which the interface is is represented as a bold polyline.

This approach is easily extensible to handle cases where more than two molecules form an interface. We note that every face of the power diagram is defined by exactly two atoms, regardless of how many molecules participate at the interface. We simply label a face as an interface-face if the two atoms defining it come from two different molecules. Interface-cells and interface-atoms are analogously determined.

We next define the interface surfaces between the two molecules A and B. Let the complete molecular surfaces defined for a probe-radius α for the molecules A and B be represented by $\mathcal{S}(A, \alpha)$ and $\mathcal{S}(B, \alpha)$, respectively. The *molecular interface surface*

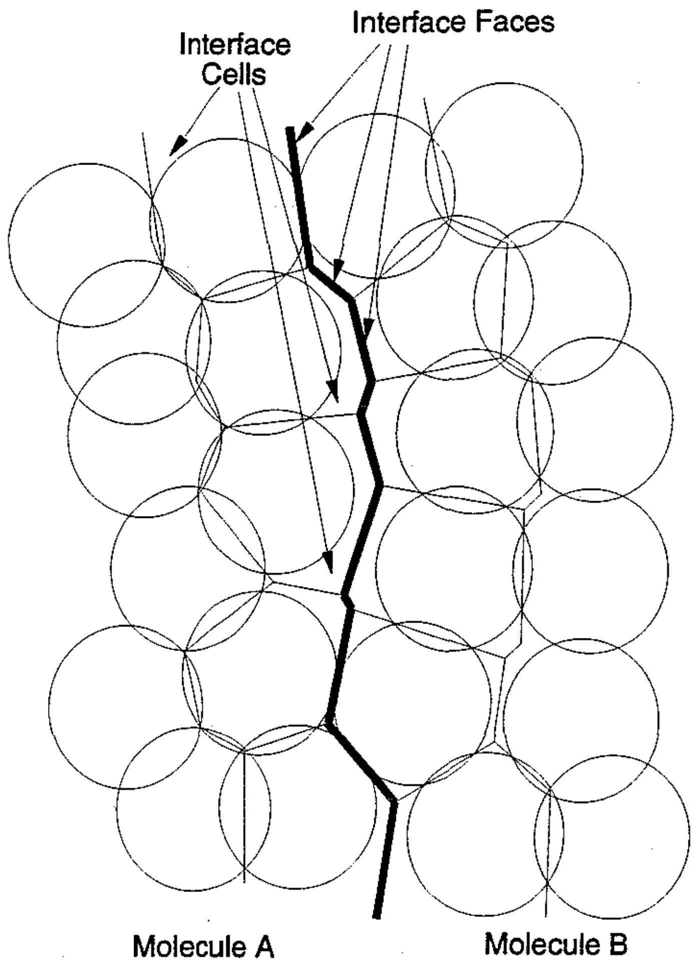


Figure 6.1: Interface Cells and Interface Faces

$\mathcal{T}(A, B, \alpha, \beta)$ for a probe-radius α and an interface-radius β for the two molecules A and B is defined as the subset of $\mathcal{S}(A, \alpha)$ and $\mathcal{S}(B, \alpha)$ that includes exactly those points of $\mathcal{S}(A, \alpha)$ that are within a distance β from the surface of some atom of B and exactly those points of $\mathcal{S}(B, \alpha)$ that are within a distance β from the surface of some atom of A.

To efficiently compute the surfaces at the molecular interface, we start outwards from the interface atoms. First, the entire layer of interface atoms of a given molecule, say A, are used to generate the smooth molecular surface for A. This surface is generated in a manner similar to that described in Section 4.2.

Let us define an atom b of B to be the sphere $\sigma(c_b, r_b)$, where c_b is the center and r_b is the van der Waal's radius of the atom b. Let us define $B(+\beta)$ to be a collection of spheres $\sigma(c_b, r_b + \beta)$ derived from the atoms of B. The surface patches for each atom a of A are clipped by the union of those spheres from $B(+\beta)$ that overlap it. If any of the neighbors of a overlap with spheres of $B(+\beta)$, we include them in the list of atoms of A that have to be processed for defining the surface at the interface; otherwise, we continue on to the next interface atom of A. At the end of this step, all the interface atoms, that are one layer deep from the interface (as shown in Figure 6.1) would have been correctly processed and those atoms from the next layer that should be processed would have been added to the list of atoms of A to be processed. We keep iterating in a similar manner with the next layer of interface atoms of A, till we get to a stage where none of the unprocessed atoms of A intersect the union of $B(+\beta)$. In this manner, we construct the interface surfaces for all the molecules at the interface.

Since construction of a three-dimensional power diagram could get expensive, we again adopt a feasible-cell approach that approximates the power diagram well enough for our purposes and is linear in the total number of atoms. This has been described in Section 4.2.

For results of our implementation, refer to Figure 1.3 where we have shown interface surfaces for various values of α and β for the four domains of transthyretin.

6.3 Goodness-of-fit for Molecular Interfaces

This section outlines a possible criterion for measuring the geometric goodness-of-fit between two molecules or between two sub-units of the same molecule. The motivation behind this is to quantify the surface complementarity between two molecules. This could be used in applications like docking to measure how well two given molecules fit.

We note that for a good fit, the two molecules should be close to each other. Volume between the two molecules provides one goodness-of-fit measure albeit a poor one, since it does not take into account the surface area of the interface. Similarly, the area of the interface surface, gives another criterion for measuring the surface complementarity, but it fails to give a sense of intervening volume between the two molecules. However, if we blend these two measures, we can quantify the surface complementarity in a better manner.

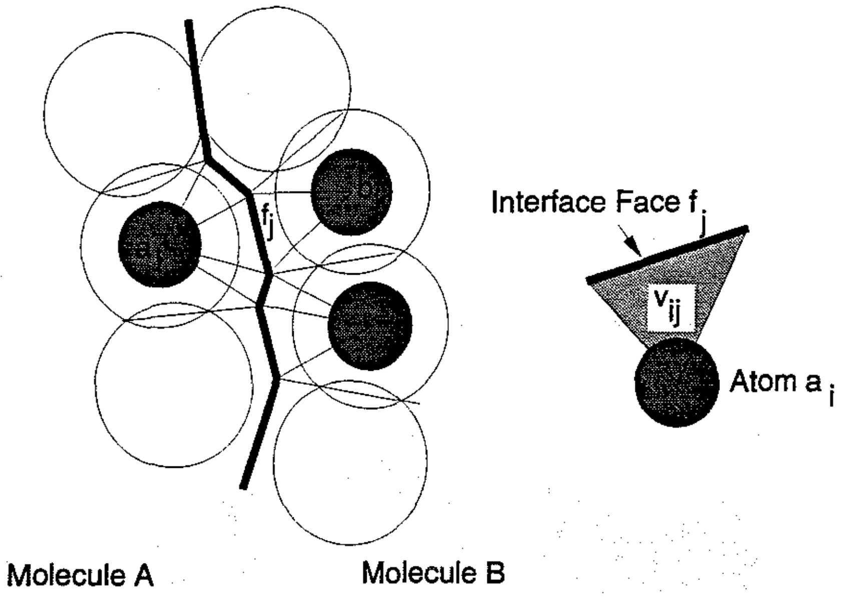


Figure 6.2: Computing Goodness-of-Fit

One such useful measure could be the average intervening volume per unit area of interface-faces.

Let the set of all interface-faces be given by I_F, and the set of all interface-atoms be given by I_A. Consider an interface-face $f_j \in I_F$ as shown in Figure 6.2. Let the atom on the side of molecule A defining this face be a_i. In general, the face f_j would be a convex polygon. Consider the pyramid formed by connecting the vertices of f_j to the center of atom a_i. Let the volume in this pyramid lying outside the atom a_i be v_{ij}, as shown in Figure 6.2. Similarly, let the volume that lies outside the atom b_k but within the pyramid defined by f_j and the center of atom b_k be v_{kj}. Let us attribute to face f_j a volume $\mathcal{V}_j = v_{ij} + v_{kj}$. Let the area of the interface-face f_j be $\mathcal{A}_j$. Then one could measure the fit between two molecules as:

$$F = (\Sigma_{f_j \in I_F} \mathcal{V}_j)^2 / (\Sigma_{f_j \in I_F} \mathcal{A}_j)^3 \qquad \text{(VI.1)}$$

In the expression for F we have raised the volume and area terms to powers of 2 and 3, respectively, to make F a scale-invariant dimensionless quantity.

The smaller F, the better the fit and vice-versa.

An additional issue that remains to be addressed, before the above measure can be used, is to have a good definition of the *region of interface.* For this, we need to find a criterion to define a suitable subset of the interface-faces as belonging to the region of interface. We note that such a criterion cannot be simply based on local properties such as distances of the interface-faces from their defining atoms. This criterion has to be based on a global property so that in Figure 6.3, the interface between the molecules A and B is quantitatively evaluated to be a better fit than the interface between the molecules C and D.

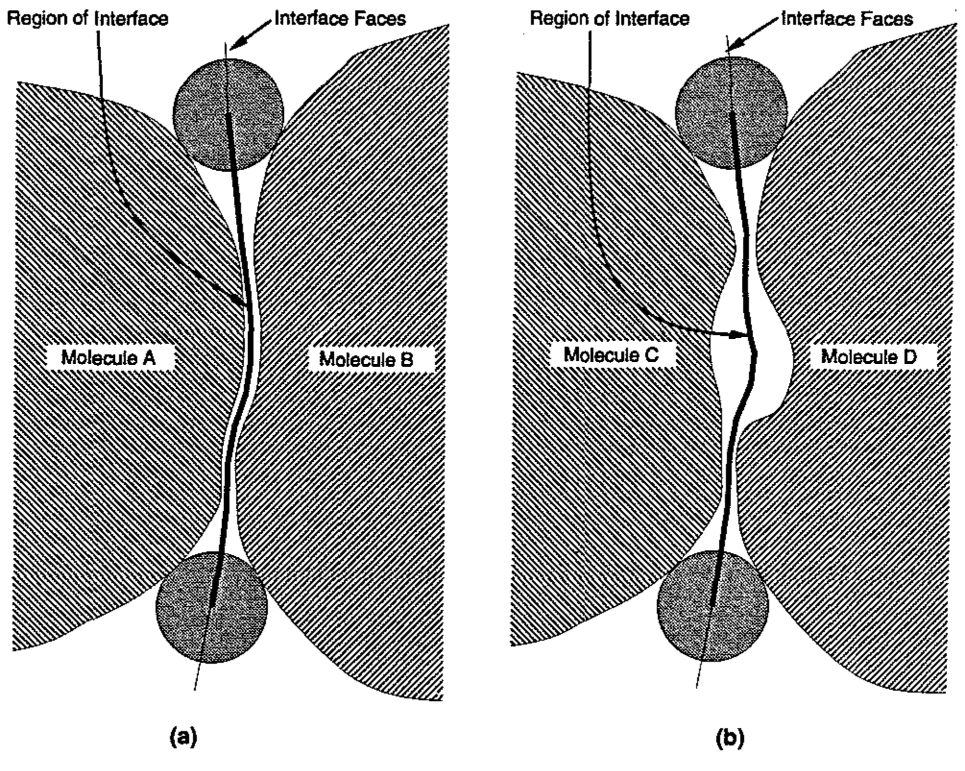


Figure 6.3: Defining the Region of Interface

We propose that the region of interface be determined by rolling an exterior sphere of an appropriate radius (depending upon the properties of the interface being studied) over the interface.

In Figure 6.3, the interface-faces as defined above are shown by the curve between two pairs of molecules (a) A and B, and (b) C and D. For both (a) and (b) we would like to specify the region of interface as the subset of the interface-faces. By rolling an exterior sphere as shown in Figure 6.3 we can now define the region of interface, represented as a bold polyline in that figure for a two-dimensional version of the problem. With this definition of the region of interface, if the above goodness-of-fit measure is evaluated over both the cases (a) and (b), it would turn out to be better for (a) than for (b).

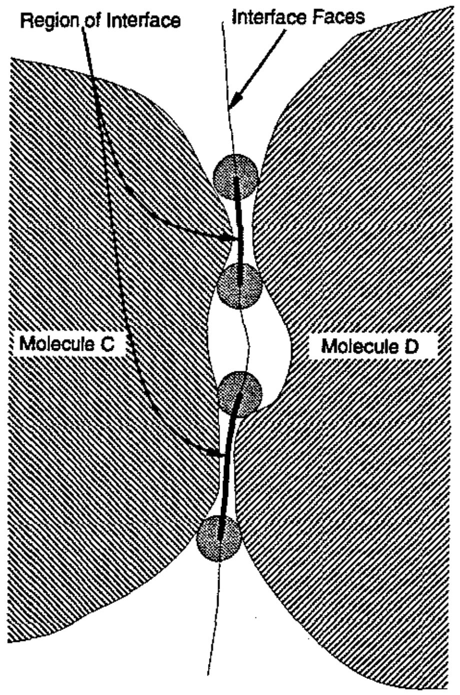


Figure 6.4: Redefining Region of Interface for Molecules C and D

For certain cases, biochemists expect the interface to be subdivided into several components. Our method of rolling an exterior sphere extends gracefully to allow the biochemist to incorporate his or her knowledge in reasonably defining the region of interface. Thus, for the interface of molecules C and D, the biochemist can choose a smaller radius of the exterior sphere, if that is more appropriate, to define a smaller, two component region of interface that yields a better goodness-of-fit between the molecules C and D. This is shown in Figure 6.4.

In general, we expect the biochemists to arrive at a reasonable value for the radius of the exterior sphere, for defining the region of interface between a given set of molecules, through their knowledge of the molecular interface characteristics and by visualizing the interface using the molecular interface surfaces defined in this chapter.

Chapter VII

Level-of-Detail Generation for Polygonal Models

7.1 Motivation

Advantages of using simplified models of an object for efficient scene rendering have been well-documented in the literature – [Clark 76], [Cosman & Schumacker 81], [Crow 82], and [Funkhouser & Séquin 93]. The basic idea is to use simplified models for objects that are perceptually less important and detailed models for objects that are more important. Perceptual importance is in general difficult to define precisely. However, heuristics such as the percentage of screen area covered, distance from the viewer, distance from the center of the screen, etc. have been found to work well. A prerequisite to this approach for rendering complex datasets is a method to simplify object models. Manual simplification of models is possible and has been done in the past [Cosman & Schumacker 81]. However, such simplification is time-consuming and may not even be feasible for large datasets whose complexity is in the order of millions of polygons.

In this chapter, we present an algorithm for computing various levels of detail of a given polygonal model. Different levels of detail representations of an object can be used in several ways in computer graphics. Some of these are:

- Use in a level-of-detail based rendering algorithm for providing desired frame update rates.
- Using low-detail approximations of objects for illumination algorithms, especially radiosity.
- Simplifying traditionally over-sampled models such as those generated from volume datasets, laser scanners, and satellites. Storing them in their original form as opposed to storing their approximations, amounts to wasting memory for storage as well as CPU cycles during processing, with disproportionately few benefits.

In this chapter we discuss an approach for generating lower-complexity approximations to a given polygonal representation of an object that are guaranteed to deviate from the original by no more than a user-specifiable amount. Such an approach has several benefits in computer graphics. First, we can very precisely quantify the amount of approximation that is tolerable under given circumstances. For instance, one possibility could be to define a tolerable approximation for rendering an object as, say, 2 screen pixels. Using this information in conjunction with the distance of the object from the screen, we can estimate the maximum deviation permissible from the surface of the object. This can then be used to find which precomputed level of detail of that object is most suitable. Second, this approach allows us a fine control over which regions of an object we should approximate more and which ones less. This could be used in selectively preserving those features of an object that are perceptually important.

In Section 7.2 we shall review some of the previous work done in the area of approximation of polygonal models. Then in Section 7.3 we shall formally state our problem and list assumptions for our algorithm. Our basic approach is to first generate two *offset surfaces* to the input model, one on the outside and the other on the inside of the input object. The definition and generation of offset surfaces is presented in Section 7.4. Then we generate all *candidate triangles* that lie within these two offset surfaces and have their vertices selected from the set of vertices of the input model. We then associate the vertices and triangles of the input model with the candidate triangles. Methods for these steps are described in Section 7.5. Our final step is a greedy approach for selecting the approximation triangles from the candidate triangles. Why this approach works and how this can be used to get a quantitative measure of the quality of approximation is described in Section 7.6. In Section 7.7 we discuss various features of this approach. Finally in Section 7.9 we present our results

7.2 Previous Work

We have seen in Chapter III some of the previous work that has been done in approximation of two-dimensional piecewise linear curves. In this section we shall consider the work done in approximation of three-dimensional objects.

Let us define a *polygonal object* to be an object with planar faces. This is a computer graphics terminology. In computational geometry polygonal objects are referred to as *piecewise linear objects.* We shall be using these two terms interchangeably.

Let us next define the term *ϵ-approximation.* Given two piecewise linear objects $\mathcal{P}$ and $\mathcal{Q}$, we say that $\mathcal{P}$ and $\mathcal{Q}$ are ϵ-approximations of each other iff every point on $\mathcal{P}$ is within a distance ϵ of some point of $\mathcal{Q}$ and every point on $\mathcal{Q}$ is within a distance ϵ of some point of $\mathcal{P}$. This is also called the Hausdorff distance, $H(\mathcal{P}, \mathcal{Q}) \leq \epsilon$.

Approximation algorithms for three-dimensional polygonal models can be classified into two broad categories:

- **Min-# Approximations**: For this version of the approximation problem, given some error bound ϵ, the objective is to minimize the number of vertices such that no point of the approximation $\mathcal{A}$ is farther than ϵ distance away from the input model $\mathcal{I}$.

- **Min-ϵ Approximations**: Here we are given the number of vertices of the approximation $\mathcal{A}$ and the objective is to minimize the error, or the difference, between $\mathcal{A}$ and $\mathcal{I}$.

In computer graphics, work in the area of min-# approximations has been done by [Schmitt *et al* 86] and [DeHaemer, Jr. & Zyda 91] where they adaptively subdivide a series of bicubic patches and polygons over a surface until they fit the data within the tolerance levels.

[Turk 92, Schroeder *et al* 92, Hinker & Hansen 93] are a good representative collection in the second category. Turk first distributes a given number of vertices over the surface depending on the curvature and then re-triangulates them to obtain the final mesh. Schroeder et al., and Hinker and Hansen, operate on a set of local rules – such as deleting edges or vertices from almost coplanar adjacent faces, followed by local re-triangulation. These rules are applied iteratively till they are no longer applicable. A somewhat different local approach is taken in [Rossignac & Borrel 92] where vertices that are close to each other are clustered and a new vertex generated to represent them. The mesh is suitably updated to reflect this.

Hoppe et al. proceed by iteratively optimizing an energy function over a mesh to minimize both the distance of the approximating mesh from the original, as well as the number of approximating vertices [Hoppe *et al* 93]. An interesting and elegant solution to the problem of polygonal simplification by using wavelets has been presented in [DeRose *et al* 93].

In computational geometry literature it has been shown that computing the minimal-facet ϵ-approximation is NP-hard for convex polytopes [Das & Joseph 90] as well as polyhedral terrains [Agarwal & Suri 94]. Thus, algorithms to these problems have evolved around finding polynomial-time approximations that are *close* to the optimal.

Let k_o be the size of a min-# approximation. [Mitchell & Suri 92] present an algorithm for computing an ϵ-approximation of size $O(k_o \log n)$ for convex polytopes. This has recently been improved in [Clarkson 93] where Clarkson proposes a randomized algorithm for computing an approximation of size $O(k_o \log k_o)$ in expected time $O(k_o n^{1+\delta})$, where δ can be an arbitrarily small positive number. Working with polyhedral terrains, [Agarwal & Suri 94] present a polynomial-time algorithm that computes an ϵ-approximation of size $O(k_o \log k_o)$ to a polyhedral terrain. Similar results have been obtained by Mitchell [Mitchell 93].

7.3 Problem Definition

In this section, we shall first identify the desirable features of an approximation scheme. Then we shall define our approximation problem. Finally, we shall state our assumptions for the rest of this paper. We will be assuming that $\mathcal{I}$ is a three-dimensional compact and orientable object whose polygonal representation $\mathcal{P}(\mathcal{I})$ has been given to us. Our objective is to compute a piecewise linear approximation $\mathcal{A}$ to $\mathcal{P}(\mathcal{I})$. Henceforth we will be referring to $\mathcal{P}(\mathcal{I})$ as simply $\mathcal{P}$.

7.3.1 Desiderata of a Good Approximation Scheme

Let us consider the desiderata for any "good" approximation scheme $\mathcal{F}$ that maps an input object $\mathcal{P}$ to its approximation $\mathcal{A}$, where $\mathcal{F}$ is denoted as $\mathcal{F} : \mathcal{P} \rightarrow \mathcal{A}$. It seems reasonable to expect the following of $\mathcal{F}$ from a mathematical and aesthetical point of view:

- $\mathcal{F}$ should be invariant under translation and rotation.
- $\mathcal{F}$ should ensure that the volume of the difference between $\mathcal{A}$ and $\mathcal{P}$ is small and bounded.
- $\mathcal{F}$ should be genus-preserving.
- $\mathcal{F}$ should be symmetry-preserving.
- $\mathcal{F}$ should allow adaptive approximation of different parts of an object to different user-specifiable degrees.
- $\mathcal{F}$ should be amenable to a parallel implementation.

In most computer graphics models, there is a lot of useful information stored at the vertices such as color, normals, texture coordinates, etc. It would therefore be desirable if in addition to the above, we also have:

Vertices($\mathcal{A}$) $\subseteq$Vertices ($\mathcal{P}$).

7.3.2 Problem Statement

Keeping in mind the desiderata outlined in the previous subsection, we define our problem as follows:

Given a polygonal representation $\mathcal{P}$ of an object $\mathcal{I}$ and an approximation parameter ϵ, generate a genus-preserving ϵ-approximation $\mathcal{A}$ with minimal number of polygons such that the vertices of $\mathcal{A}$ are a subset of vertices of $\mathcal{P}$.

We have already seen in Section 7.2 that it is NP-hard to find a minimal ϵ-approximation for even convex objects or polyhedral terrains. Thus, our objective will be to compute the ϵ-approximation $\mathcal{A}$ that has a smaller number of polygons than $\mathcal{P}$ and whose number of polygons can be related to the smallest possible number of polygons in any ϵ-approximation of $\mathcal{P}$.

7.3.3 Assumptions on Input

Without loss of generality, we shall assume that all polygons in $\mathcal{P}$ are triangles and that $\mathcal{P}$ is a well-behaved polygonal model, i.e. every edge has two adjacent triangles, no two triangles interpenetrate, there are no unintentional "cracks" in the model, no T-junctions, etc.

We will be further assuming that the vertices of $\mathcal{P}$ have normals that faithfully represent the normals of the object being modeled. By this we mean that it should not be possible for an observer to distinguish (to a reasonable degree) between the polygonal representation of an object and the object itself, by just examining those properties that depend on the object normals (for instance shading). Thus, if the object is a sphere and its polygonal representation is an octahedron, for a faithful representation of the former, the latter should have unique normals at each vertex and edge that are equal to the normal of the sphere at those points. With this representation, shading models such as those of Gouraud or Phong will give an approximately sphere-like shading to the octahedral approximation. In general, polygonal approximations to curved objects have unique vertex and edge normals to avoid the discontinuities in the normal-based properties of the object.

However, if the object being modeled has sharp edges, such as an octahedron, we would like to retain the discontinuity in the normals across the faces. In such cases, a faithful polygonal representation requires that there be multiple normals associated with each vertex and edge – one associated with every adjacent face. An arbitrary object could have both kinds of vertices, with or without unique normals. We will first present our algorithm with the assumption that the vertex normals are unique (i.e. there are no normal discontinuities, or sharp edges, in the model), and then show that with a very simple and straightforward modification, we can handle the case where normal discontinuities are allowed. We shall further assume, as is done in most computer graphics, that bilinear interpolation of the vertex normals in the polygonal representation of an object is sufficient to reasonably duplicate the normal-based properties of the object.

7.4 Generation of Offset Surfaces

Let the x, y, z coordinates of a three-dimensional surface parametrized by s and t be given as: $x = f_1(s,t)$, $y = f_2(s,t)$, and $z = f_3(s,t)$. Using vector notation, we can say that the three-dimensional parametric surface $\mathbf{f}$ is given by: $\mathbf{f}(s,t) = (f_1(s,t), f_2(s,t), f_3(s,t))$. Let the unit normal to $\mathbf{f}$ be: $\mathbf{n}(s,t) = (n_1(s,t), n_2(s,t), n_3(s,t))$. Then, the ϵ-offset for $\mathbf{f}$ is defined as

$$\mathbf{f}^\epsilon(s,t) = (f_1^\epsilon(s,t), f_2^\epsilon(s,t), f_3^\epsilon(s,t))$$

where,

$$f_i^\epsilon(s,t) = f_i(s,t) + \epsilon n_i(s,t).$$

For our purposes, let us simply define an offset surface $\mathcal{P}(+\epsilon)$ (respectively $\mathcal{P}(-\epsilon)$) for an object $\mathcal{I}$ to be a surface that lies within a distance of ϵ from every point p on $\mathcal{I}$ in the same (respectively opposite) direction as the normal to $\mathcal{I}$ at p.

Since we would be generating all triangles that lie within these two offset surfaces, in the interests of preserving the genus of $\mathcal{P}$, we desire that these offset surfaces not intersect each other or themselves.

To meet this criterion we might have to reduce our level of approximation at certain places. In other words, to guarantee no intersections amongst the offset surfaces, we will have to be content at certain places with the distance between $\mathcal{P}$ and an offset surface being smaller than ϵ.

Next, we introduce the notions of *edge halfspaces* and the *fundamental prism.* Then using these concepts, we will discuss a method to generate a particular kind of non-intersecting offset surfaces that lie at an offset of no more than ϵ from $\mathcal{P}$.

7.4.1 Edge Halfspaces

In Section 7.3.3 we had made the assumption that the normals to the vertices in $\mathcal{P}$ faithfully represent the object $\mathcal{I}$ and that bilinear interpolation of the normals is sufficient across any triangle of $\mathcal{P}$ for computing the normal-based properties.

If for every edge $e = (v_1, v_2)$, we have three constraints — coordinates of the vertices v_1 and v_2 and the same normal to $\mathcal{I}$ at both the vertices v_1 and v_2, we can construct a plane π_e that passes through the edge e and has a normal that is perpendicular to that of v_1 and v_2. Thus v_1, v_2 and their normals all lie along π_e. Such a plane defines two halfspaces for edge e, say π_e^+ and π_e^-. This is shown in Figure 7.1(a).

However, in general the normals $\mathbf{n}_1$ and $\mathbf{n}_2$ to the vertices v_1 and v_2 defining an edge e need not be identical. How can we reasonably define the two halfspaces for an edge in such a case? One choice could be to use a bilinear patch that passes through v_1 and v_2 and has a tangent $\mathbf{n}_1$ at v_1 and $\mathbf{n}_2$ at v_2. Let us call such a bilinear patch for e as β_e. Let the two halfspaces for the edge e in this case be β_e^+ and β_e^-. This is shown in Figure 7.1(b). We shall refer to a halfspace for an edge as an *edge halfspace.*

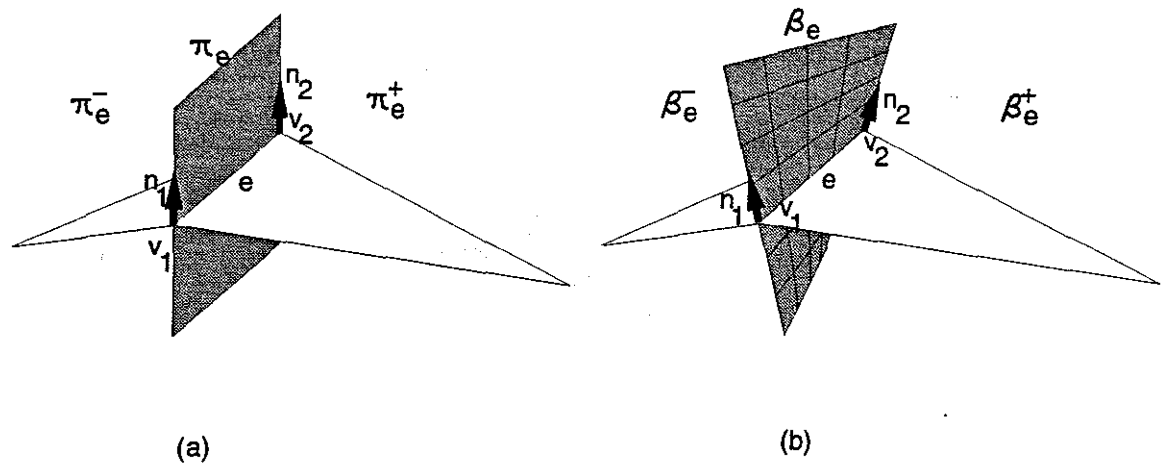


Figure 7.1: Edge Halfspaces

7.4.2 The Fundamental Prism

Let us refer to the triangles of the given polygonal representation $\mathcal{P}$ as the *fundamental triangles.* Consider one such triangle. Let its vertices be v_1, v_2, and v_3. Let the coordinates and the normal of each vertex v be represented as $\mathbf{c}(v)$ and $\mathbf{n}(v)$, respectively. We next define the coordinates and the normal of a $(+\epsilon)$-offset vertex v_i^+ for a vertex v_i as: $\mathbf{c}(v_i^+) = \mathbf{c}(v_i) + \epsilon\mathbf{n}(v_i)$, and $\mathbf{n}(v_i^+) = \mathbf{n}(v_i)$. Essentially, we translate each vertex in the direction of its normal by an amount ϵ to obtain its $(+\epsilon)$-offset vertex. The $(-\epsilon)$-offset vertex can be similarly defined in the opposite direction. These offset vertices for a fundamental triangle are shown in Figure 7.2.

Now consider the closed object defined by v_i^+ and v_i^-, $i = 1, 2, 3$. It is defined by two triangles, at the top and bottom, and three edge halfspaces. This object contains the fundamental triangle (shown shaded in Figure 7.2) and we will henceforth refer to it as the *fundamental prism.*

7.4.3 Non-intersecting Offset Computation

If we offset each vertex v_i by the same amount ϵ, to get the offset vertices v_i^+ and v_i^-, the reason why we can get the two offset surfaces $\mathcal{P}(+\epsilon)$ and $\mathcal{P}(-\epsilon)$ to respectively self-intersect is because one or more offset vertices are closer to some non-adjacent fundamental triangle. In other words, if we define a Voronoi diagram over the funda-

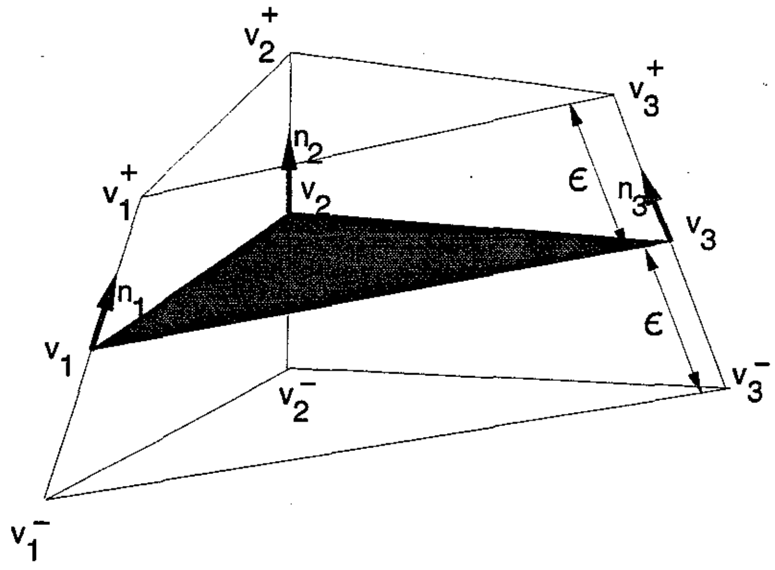


Figure 7.2: The Fundamental Prism

mental triangles of the model, the condition for the offset surfaces to intersect is that there be at least one offset vertex lying in the Voronoi region of some non-adjacent fundamental triangle. This is shown in Figure 7.3 by means of a two-dimensional example. In the figure, the offset vertices b' and c' are in the Voronoi regions of edges other than their own, causing self-intersection of the offset surface.

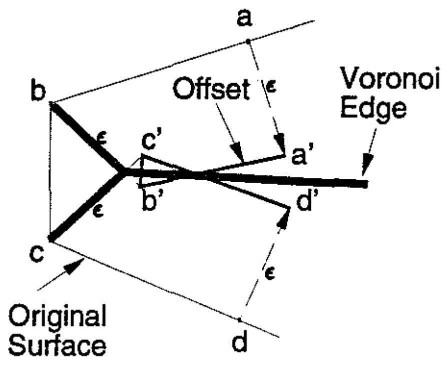


Figure 7.3: Offset Surfaces

Once we make this observation, the solution to avoid self-intersections becomes quite simple – just do not allow any offset-vertex to go beyond the Voronoi regions of its adjacent fundamental triangles. In other words, determine the positive and negative ϵ for each vertex v_i such that its offset vertices v_i^+ and v_i^- determined with this new ϵ do not lie in the Voronoi regions of the non-adjacent fundamental triangles.

While this works in theory, efficient computation of the three-dimensional Voronoi diagrams of the fundamental triangles is difficult. To avoid this, we adopt a conservative approach for recomputing the ϵ at each vertex. This approach underestimates the values for the positive and negative ϵ. In other words, it guarantees that the offset surfaces do not intersect, but it does not guarantee that the ϵ at each vertex is the largest permissible ϵ. We next discuss this approach for the case of computing the positive ϵ for each vertex. Computation of negative ϵ follows similarly.

Consider a fundamental triangle t. We define a prism t_p for t, which is conceptually the same as its fundamental prism, but uses a value of 2ϵ instead of ϵ for defining the offset vertices. Next consider all triangles Δ_i that do not share a vertex with t. If Δ_i intersects t_p above t (the directions above and below t are determined by the direction of the normal to t, above is in the same direction as the normal to t), we find the point on Δ_i that lies within t_p and is closest to t. Since we are dealing with convex objects, this point would be either a vertex of Δ_i, or the intersection point of one of its edges with the three sides of the prism t_p. Once we find the point of closest approach, we compute the distance δ_i of this point from t. This is shown in Figure 7.4.

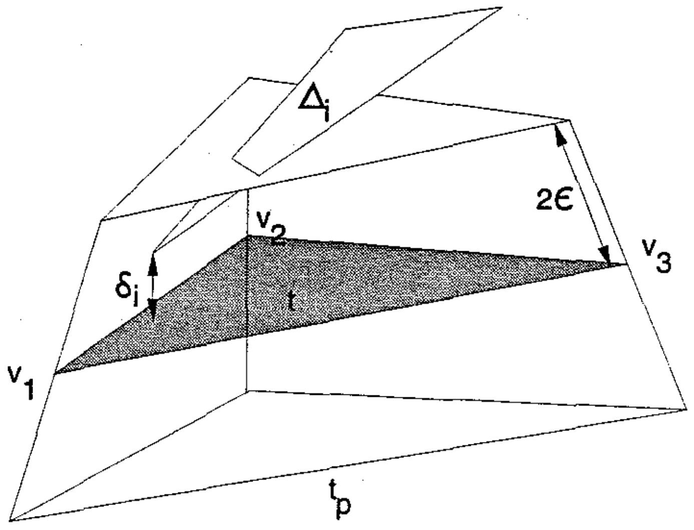


Figure 7.4: Computation of δ_i

Once we have done this for all Δ_i we compute the new value of the positive ϵ for the triangle t as $\epsilon_{new} = \frac{1}{2}\min_i \delta_i$. If the vertices for this triangle t have this value of positive ϵ, their positive offset surface will not self-intersect. Once the $\epsilon_{new}(t)$ values for all the triangles t have been computed, the $\epsilon_{new}(v)$ for each vertex v is set to be the minimum of the $\epsilon_{new}(t)$ values for all its adjacent triangles.

The offset surfaces are then computed with these modified values of ϵ at each vertex v. Connectivity of the offset surfaces mirrors that of the given polygonal model $\mathcal{P}$. We use an octree in our implementation to speed up the identification of triangles Δ_i that intersect a given prism.

7.5 Generation of Candidate Triangles

Generation of candidate triangles for the approximation involves computing visibilities between vertices and edges, with occlusion being provided by the offset surfaces.

Vertex-Vertex Visibility: We define two vertices v_1 and v_2 to be visible to each other if and only if an observer at v_1 (or v_2) can see the vertex v_2 (or v_1), with the two offset surfaces providing occlusion. This condition for visibility is equivalent to the condition that the line segment joining v_1 and v_2 (i.e. the convex combination of v_1 and v_2) does not intersect $\mathcal{P}(+\epsilon)$ or $\mathcal{P}(-\epsilon)$.

Vertex-Edge Visibility: We define an edge e to be visible to a vertex v if and only if an observer at v can see the entire edge e, with the two offset surfaces providing occlusion. This condition for visibility is equivalent to the condition that the triangle formed by v and e (i.e. the convex combination of v and e) does not intersect $\mathcal{P}(+\epsilon)$ or $\mathcal{P}(-\epsilon)$.

A valid candidate triangle is one in which every edge is visible to the vertex opposite it. Keeping this in mind, we first generate all tuples (v_1, v_2) where vertices v_1 and v_2 are visible to each other. Let any such tuple define an edge e and let the set of all such edges be S_e. Clearly, the set of all candidate triangles will have edges drawn from S_e. To generate the exact set of candidate triangles, we can intersect all triangles with edges in S_e with the two offset surfaces and discard those that intersect either of the two offset surfaces.

Having generated the candidate triangles, the next step is finding which of the vertices of $\mathcal{P}$ are covered by each candidate triangle. The general idea is that we would like to give a greater preference to those candidate triangles that cover more vertices of $\mathcal{P}$ over those that cover fewer vertices. The implementation of this step is quite simple. For each vertex v_i, consider the line segment formed by its offset vertices (v_i^+, v_i^-). We say that a vertex v_i is covered by a candidate triangle if and only if the line segment (v_i^+, v_i^-) intersects the candidate triangle. Using this approach, we find the vertices of $\mathcal{P}$ that are covered by a given candidate triangle and the candidate triangles that cover a given vertex of $\mathcal{P}$.

7.6 Composing the Final Solution

At this stage we have available to us all candidate triangles that lie between the two non-self-intersecting offset surfaces and information about which candidate triangle covers which vertex. Our goal is to find a subset of those candidate triangles that (a) cover all the vertices of $\mathcal{P}$, (b) do not mutually intersect, and (c) do not leave any holes in the mesh where there were none before.

Before we go any further, let us introduce the problems of *set cover* and *set partition.*

7.6.1 Set Cover and Set Partition

Consider a set of integers $P = \{1, 2, \ldots, n\}$ and another set $T = \{t_1, t_2, \ldots, m\}$, where $t_j \subseteq P$ for $j \in J = \{1, 2, \ldots, m\}$. A subset $C \subseteq J$ defines a *cover* of P if $\bigcup_{j \in C} t_j = P$. Intuitively, a cover of P is a collection of those sets of T that collectively contain all the elements of P. Let a cost $c_j > 0$ be associated with each element t_j of T. The total cost of a cover C is defined as $\sum_{j \in C} c_j$. The *set covering problem* is to find a cover of the minimum cost.

If we impose the restriction that for all distinct elements $i, j \in C$ we must have that $t_i \bigcap t_j = \phi$, we then say that C defines a partition of P. In other words, a partition of P is a collection of those sets of T that are mutually disjoint and collectively contain all the elements of P. Finding a partition of minimum cost is referred to as the *set partition problem.*

If we let the indices of the vertices of $\mathcal{P}$ define the set P, and the candidate triangles define the set T, with $i \in t_j$ if and only if the candidate triangle t_j covers a vertex v_i, we can see that our problem reduces to that of set partitioning with one additional triangle disjointness constraint. The set partitioning solution guarantees that there are no common vertices of $\mathcal{P}$ in the interior of any pair of triangles of the final solution. However this does not prevent two triangles from overlapping each other if the overlapping region does not have any common vertices. Further, if we define the cost c_j associated with each triangle t_j to be its cardinality, i.e. the number of vertices covered by it, solution of this set partitioning problem will yield the smallest number of triangles that cover all the vertices of the input model $\mathcal{P}$.

7.6.2 The Greedy Heuristic

The set cover and the set partition problems are both known to be NP-complete [Garey & Johnson 79]. Therefore any hopes of computing an optimal solution to our polygonal approximation problem are remote. However, there exist several heuristics which compute approximate solutions to these problems. These heuristics generate solutions that have been in general found to be reasonably close to the optimal. A user could implement any of the heuristics for solving the polygon approximation problem. We are using the greedy heuristic, because of its simplicity and because it facilitates an easy analysis of the quality of the solution. We will discuss the issue of estimating quality in Section 7.6.4.

A greedy heuristic for solving the set cover problem proceeds as follows. We start with an empty cover and at every step, we add that set which covers the largest number of thus far uncovered points. In our setting, this translates to selecting the triangle which covers the largest number of vertices. However, since we have an additional disjointness constraint to observe, we proceed slightly differently.

At each step, out of all the candidate triangles we pick the triangle that covers the largest number of vertices of $\mathcal{P}$ in its interior. Then we check to see if it overlaps any other triangles of the solution generated thus far. If it does, we discard it, otherwise it becomes a part of the solution. This is done till all candidate triangles have been seen.

We next explain the algorithm for detecting whether two triangles overlap. First, for each triangle we find the fundamental prisms that it intersects. Second, for every fundamental prism that both triangles intersect we compute the intersection of each triangle with the fundamental prism and project it on to the fundamental triangle. Third, we check if the projections of the two triangles on the fundamental triangle overlap. Since the projection of each triangle is convex, we determine whether two triangles overlap or not by straightforward two-dimensional linear programming. Figure 7.5 illustrates the overlapping projections on a fundamental triangle.

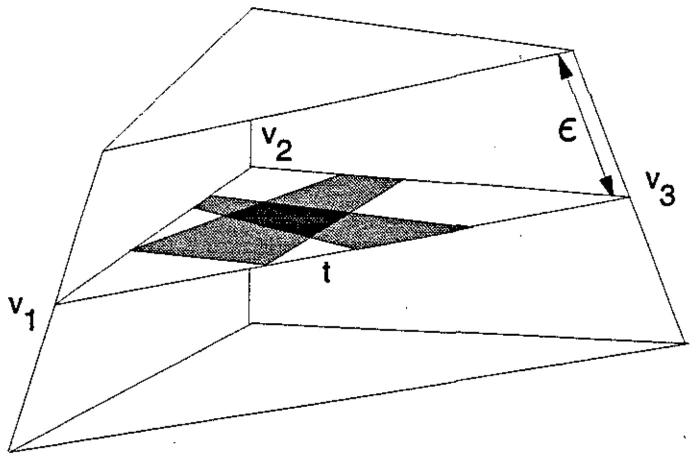


Figure 7.5: Checking for Overlaps

7.6.3 Modification to the Greedy Heuristic

The greedy heuristic as we have presented above has a few drawbacks. First, its implementation suffers from serious numerical degeneracy problems. One of the most common problems with this is that cracks are left in the final mesh due to inconsistencies in determining whether the long and thin, sliver, triangles overlap the partially constructed approximation mesh. To overcome this we follow the following method.

We maintain a complete mesh at every iteration of the algorithm. Let the mesh at the i^{th} iteration of be M_i; M_0 is the input mesh. Let the triangle that covers the largest number of vertices of M_{i-1} be t_j. Let the set of all the triangles of M_{i-1} that are covered by t_j be denoted by T_j. We find the hole in M_{i-1} that is formed if we delete all the triangles in T_j. Our objective is to determine if we can triangulate this hole such that one of the triangles is t_j. If such a triangulation is possible, let the set of triangles in this triangulation be represented by T'_j. By construction, $t_j \in T'_j$. We delete T_j from M_{i-1} and add the new triangles T'_j to get M_i. In other words,

$$M_i = M_{i-1} - T_j + T'_j$$

. If we cannot find such a triangulation T'_j, we try the same with the next triangle that covers the maximum number of triangles of M_{i-1}.

The advantage of this approach over the basic greedy heuristic is that we have a completely valid mesh at every stage of our algorithm and so our solution does not have the visible artifacts arising from numerical precision problems.

7.6.4 Estimating Solution Quality

It has been shown in the literature [Lovász 75] and [Chvátal 79] that the greedy heuristic yields a solution which is guaranteed to be within a factor of $(1 + \ln d)$ of the optimal, where d is the maximum number of elements in any set t_j, i.e. the maximum number of vertices covered by a candidate triangle.

We can prove that if in the greedy heuristic at each stage we select the set that does not necessarily have the largest cost, but has a cost within a factor α of the largest cost, then the worst case ratio guarantee of the approximate solution to that of the optimal becomes $\alpha(1 + \ln d)$. This suggests the following scheme for estimating the quality of the solution. At each step, we compute the ratio of the number of uncovered points of the largest overlapping triangle with the number of points in the largest non-overlapping triangle. The maximum of all these ratios (if it is greater than 1), can be used as α in the above expression. Of course, if $\alpha > d/(1 + \ln d)$, the whole exercise is debatable, since the maximum compression we can ever hope to achieve is no more than d.

7.7 Algorithm Features

7.7.1 Desiderata of a Good Approximation Scheme

Let us see how well our algorithm measures up to the desiderata of a good approximation scheme that we had stated in Section 7.3.1:

- **Invariance under translation and rotation:** The approximation produced by our algorithm is invariant under translation and rotation, i.e. it does not change if the input object is translated or rotated.
- **Volume of difference between $\mathcal{A}$ and $\mathcal{P}$:** This depends upon the user-specifiable parameter ϵ and can therefore be made arbitrarily small.
- **Genus-preservation:** The non-intersecting offset surface property guarantees genus preservation.
- **Symmetry-preservation:** Our algorithm does not guarantee preservation of symmetry. However, since it is a greedy algorithm, it is likely to select triangles with similar areas (and shapes, if that is incorporated into the selection criterion at each iteration). Thus, it is likely although not guaranteed, that the approximation produced by our algorithm will preserve the symmetry of the input object.
- **Adaptive Approximation:** This is possible by specifying ϵ as a function of location on the object. See Section 7.7.4.
- **Parallelizability:** Our algorithm is based on a greedy heuristic and is therefore sequential. However, as long as the regions covered by the selected candidate triangles do not overlap, we can process them in parallel. This is an important and desirable property, especially for large polygonal objects that have curved geometries.
- **Vertex Subset Criterion:** This is accomplished by definition. Only those candidate triangles whose vertices belong to the existing mesh are considered.

7.7.2 Interpolation

Smooth interpolation between various levels of detail is useful to avoid any jerkiness of abrupt changes during switching from one level of detail to the other. Our approach lends itself naturally to this interpolation since we always generate a subset of the vertices of the original model. Further, during the course of the algorithm, we associate each vertex with the triangle that it will be replaced by. Therefore, for incorporating interpolation with this approach, all that we need to do is to move each vertex along its normal direction (or reverse normal direction), till it reaches the triangle that it is to be replaced with.

7.7.3 Preserving Sharp Edges

One of the important properties in any approximation scheme is the way it preserves any normal discontinuities or sharp edges present in the input model. We next outline how our method easily incorporates this feature.

Consider any edge e that a user wishes to preserve in the output approximation. This edge will be adjacent to two fundamental prisms corresponding to its two adjacent fundamental triangles. This edge can disappear from the output if and only if there is a candidate triangle that intersects the bilinear patch β_e or the plane π_e that defines the two edge halfspaces for this edge. Therefore a simple solution to retain this edge is to make β_e (or π_e) "opaque" in the visibility computations. This will ensure that no candidate triangle will cross it and the edge will be retained in the output.

7.7.4 Adaptive Approximation

For certain classes of objects it is desirable to perform an adaptive approximation. For instance, consider large terrain datasets, models of spaceships, or submarines. One would like to have a higher detail near the observer and a lower detail further away. A possible solution could be to subdivide the model into various spatial cells and use a different ϵ-approximation for each cell. However, problems would arise at the boundaries of such cells where the ϵ-approximation for one cell, say at a value ϵ_1 need not necessarily be continuous with the ϵ-approximation for the neighboring cell, say at a different value ϵ_2.

Since all candidate triangles generated are constrained to lie within the two offset surfaces, manipulation of these offset surfaces provides one way to smoothly control the level of approximation. Thus, one could specify the ϵ at a given vertex to be a function of its distance from the observer — the larger the distance, the greater is the ϵ.

As another possibility, consider the case where certain features of a model are very important and are not to be approximated beyond a certain level. Such features might have human-perception as a basis for their definition or they might have mathematical descriptions, such as regions of high curvature. In either case, a user can vary the ϵ associated with a region to increase or decrease the level of approximation.

7.8 Degeneracies

The greedy approach of selecting the triangles from all possible triangles that approximate a surface leads to the following kinds of degeneracy problems:

- Extremely thin triangles, also known as *slivers*. Such triangles are characterized by having one or two very small angles. A better measure for detecting such triangles is to use the ratio of the radius of their circumcircle to the radius of their incircle. The bigger is this ratio for a triangle, the more slivery it is. Slivers are a source of several problems later in the computer graphics pipeline. First, any ray-casting-based processing (such as ray-tracing or ray-casting-based radiosity) is very likely to miss slivers, leading to cracks in the processed model. Second, slivers tend to produce strong aliasing effects, which are very distracting.

- Floating-point problems due to various geometric degeneracies. The geometric degeneracies could be either of the type where four or more points are coplanar, or they could be due to limited-accuracy intersection computations.

We shall next consider some heuristics to handle these degeneracies.

7.8.1 Slivers

Almost all the slivers that have been observed arising from the surface approximation algorithm are of the form shown in Figure 7.6, where a sliver triangle (*abd*) is adjacent to a better shaped triangle (*bcd*).

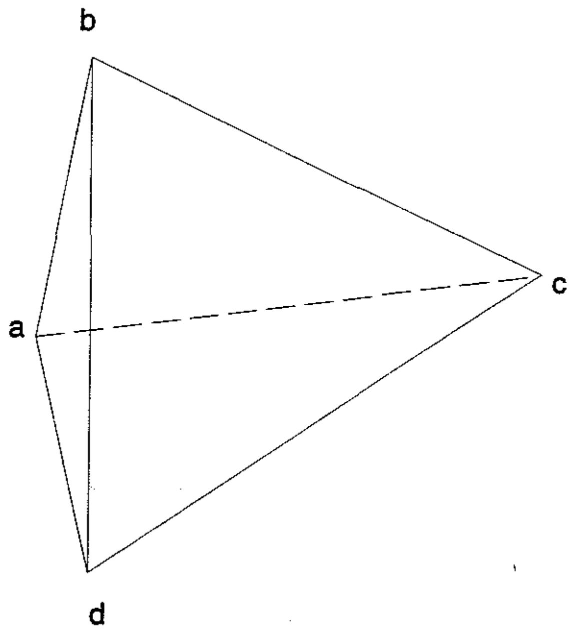


Figure 7.6: Getting Rid of Slivers

One way to handle this kind of degeneracy is to simply flip the diagonal of the quadrilateral *abcd* from *bd* to *ac*, so that we get the two triangles *abc* and *acd*. Although, in the worst case this can still yield slivers, in practice this is a good heuristic for getting rid of slivers. This step of edge-flips can be thought of as a post-processing step to the actual approximation algorithm [Turk 94].

7.8.2 Floating-point Problems

These are the problems that arise from limited-precision floating-point arithmetic. Although one could in principle switch all computation to exact arithmetic and avoid such problems, the resulting much higher execution times might be too big a penalty to pay. In practice, we are using the following heuristics to help us:

- Check for common vertex, edge, and triangle identification labels as far as possible. Thus for instance consider the intersection of two candidate triangles *abc* and *ade* as shown in Figure 7.7.

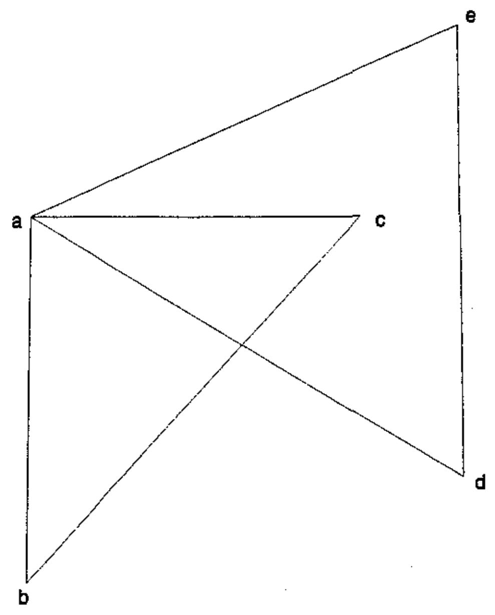


Figure 7.7: An Intersection Degeneracy

It is better to check the labels of the vertices to determine whether the edge *ad* of the triangle *ade* intersects the edges *ab* and *ac* of the triangle *abc*, instead of actually performing the intersection using floating-point arithmetic. Floating-point arithmetic might yield the result that the edge *ad* intersects either only *ab*, or only *ac*, or neither. Similar ideas are used for detecting shared edges and avoiding intersection computations with them.

- Using fuzzy coplanarity tests. The previous scheme of checking for common vertices and edges, although preferable, doesn't always work. For such cases, explicit checking for collinearity or coplanarity needs to be done. This checking is done with a tolerance. This amounts to checking for nearly collinear or nearly coplanar degeneracies instead of the exact ones.

7.9 Results

We have implemented our algorithm and tried it out on more than a thousand polygonal objects. These polygonal objects are from the Submarine Auxiliary Machine Room (AMR) dataset given to us by the Electric Boat Division of the General Dynamics.

On an average we were able to achieve simplifications of the order of roughly 70% with minimal perceptual differences. Higher levels of simplifications should be possible with a less conservative offset-surface computation program. Our present implementation is somewhat overcautious in preventing offset-surface intersections.

We simplified a total of 1090 objects of the AMR dataset for testing and validating our algorithm. These objects have been cumulatively reduced as follows:

Level-of-Detail	Dataset Complexity	% Reduction
Original dataset	350,023 triangles	0
First level	206,859 triangles	40.90
Second level	141,983 triangles	59.44
Third level	104,874 triangles	70.04

Table 7.1: Polygonal Simplification Results

For an example object, see the four levels of details for a torpedo roller in Figure 1.5.

The value of ϵ that was chosen to approximate the polygonal objects varied from one object to the other, as it should. For this set of results, ϵ values were chosen manually. However, in future, it might be a reasonable idea to automate the process by assigning the value of ϵ for approximating an object as some fraction, say 0.5%, of its extents.

For the above results, the i^{th} level of detail was obtained by simplifying the $i-1^{th}$ level of detail. There are two advantages to this scheme:

(a) It allows one to proceed incrementally, taking advantage of the work done in previous simplifications. As a result the time taken to simplify reduces with every new level-of-detail.

(b) It builds a hierarchy of detail in which the vertices at the i^{th} level of detail are a subset of the vertices at the $i-1^{th}$ level of detail. This hierarchy allows these levels of details to be useful in not only efficient rendering but also in a wide variety of accuracy-guided simulation of physical processes, such as radiosity.

Chapter VIII

Future Work

The scope for future work from this dissertation can be categorized into two broad classes of research topics:
(1) Topics that are extensions of the areas directly addressed by this dissertation.
(2) Topics that are related but not direct extensions of this dissertation. These are research areas which have assumed a greater importance to me as a direct result of this dissertation.
In this chapter I shall outline these possible research topics.

8.1 Real-Time Area and Volume Computation

We have seen in Chapter IV how one can compute the molecular surfaces in real time. The computation of such surfaces is just the first stage towards the incorporation of the effects of solvent in the potential energy computations of the molecule. Accurate and fast computation of the molecular surface area and the molecular volumes should be attempted next.

To efficiently compute the surface areas, one would have to start working from the higher-level spherical and toroidal patches (instead of triangles that are currently being generated). These patches are currently generated implicitly and then triangulated for display.

A good starting place for efficiently computing the molecular volumes would be [Edelsbrunner 93], where Edelsbrunner has described how to perform volume computations for a union of spheres. Our problem is somewhat different and thus remains interesting enough.

Another approach to computing the areas and volumes could be to try a Monte-Carlo-based approximation to the actual areas and volumes. The basic idea here being that we can test a small number of points to determine whether they are inside the molecular surface, on the surface, or outside it and based on it we can estimate the value of the area and volume of the molecule as well as the bound on the error. Depending on how much error one is willing to tolerate, this approach might be faster than the exact analytical approach to compute the molecular areas and volumes. A good starting place for this would be [Spirakis 84, Spirakis 83] where Spirakis examines the probablistic approaches to area and volume computations of a union of circles and spheres.

8.2 Determination of Molecular Interface Surfaces, Areas, and Volumes

We have just opened up this new area of research and it certainly deserves much more attention than was possible to give in this dissertation. All of the approaches that have been outlined for computing the general areas and volumes of molecules in Section 8.1 remain equally valid here.

The pioneering idea of boolean textures [Lorensen 93] can be also used to efficiently visualize these molecular interfaces. Every vertex of the molecular surface of a given molecule A could be given a texture value based on its distance from the molecular surface of another molecule B. To visualize the molecular surface of A that is within a distance β of B, one could simply set the texture for A to be transparent above a value of β.

In some sense, the problem of the determination of interfaces between molecules is the inverse of the registration problem in medical imaging. It remains to be seen if any transfer of technology from that area can be brought to bear upon this problem.

8.3 Use of Temporal Information

At present we are not using any incremental temporal information in constructing the molecular surfaces. Thus, if the atoms move even 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 molecular surfaces (and indeed α-hulls and α-shapes) incrementally and efficiently by using the information from previous time steps. Some work has been done in the incremental computation of Voronoi diagrams of moving points in a plane [Aurenhammer 91], but to the best of my knowledge no work has been done in three dimensions for spheres of unequal radii. Research into efficient algorithms that exploit the temporal coherence would also benefit interactive docking applications. To get an idea of how molecular surfaces behave with changing atom positions, see [Varshney *et al* 94b].

8.4 Molecular Surface Cusps

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. This problem was first addressed in [Connolly 85]. Work on this problem has also been reported by Sanner [Sanner 92, Sanner 94].

8.5 Polygonal Simplification

We have presented an approach that is global and deterministic. Other approaches to this problem have been local and deterministic. Scope for further work remains in developing randomized algorithms for solving this problem, as well as developing hybrid approaches that fall between completely global and completely local approaches.

Our approach to polygonal simplification always preserves the genus of the object. In some cases, it might be worthwhile to simplify the genus itself and thus generate non-topology preserving simplifications that still look reasonably good.

8.6 Automatic Cleaning of Polygonal Datasets

Most of the computer graphics pipeline assumes accurate and well-behaved polygonal models, whereas in reality most polygonal models do not satisfy this criterion. Computer graphics practitioners have been struggling against this wall of difference between theory and practice. Problems with polygonal datasets lead to robustness problems in all visibility computations, shading problems in radiosity, and topological-consistency problems in polygonal simplification algorithms.

The time seems about right for someone to develop a generalized approach to this problem that deals with degeneracies in large polygonal datasets in a unified manner, instead of dealing with each case in a special way. After giving some thought to this problem, I think that it might be fruitful to categorize these degeneracies based on their dimension and then to develop a general approach that works in the general d-dimensions. Zero-dimensional degeneracies would then include the problem of coincident points, one-dimensional degeneracies would cover the coincident edges (T-vertices are a special case of these), and two-dimensional degeneracies would cover the coincident polygons. It might be educational to study how the computational geometers solved the somewhat similar problems of geometric degeneracies of point sets for general d-dimensions [Edelsbrunner & Mücke 88, Emiris & Canny 92].

Bibliography

[Agarwal & Suri 94] P. Agarwal and S. Suri. Surface approximation and geometric partitions. In *Proceedings Fifth Symposium on Discrete Algorithms*, pages 24–33, 1994.

[Aurenhammer 87] F. Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM Journal of Computing*, 16(1):78–96, February 1987.

[Aurenhammer 88] F. Aurenhammer. Improved algorithms for discs and balls using power diagrams. *J. Algorithms*, 9:151–161, 1988.

[Aurenhammer 91] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[Bergman *et al* 86] L. Bergman, H. Fuchs, E. Grant, and S. Spach. Image rendering by adaptive refinement. In *Computer Graphics: Proceedings of SIGGRAPH'86*, volume 20, No. 4, pages 29–37. ACM SIGGRAPH, 1986.

[Chvátal 79] V. Chvátal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4:233–235, 1979.

[Clark 76] J. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[Clarkson 93] K. L. Clarkson. Algorithms for polytope covering and approximation. In *Proc. 3rd Workshop Algorithms Data Struct.*, Lecture Notes in Computer Science, 1993.

[Connolly 83] M. L. Connolly. Analytical molecular surface calculation. *Journal of Applied Crystallography*, 16:548–558, 1983.

[Connolly 85] M. L. Connolly. Molecular surface triangulation. *Journal of Applied Crystallography*, 18:499–505, 1985.

[Connolly 93] M. L. Connolly. The molecular surface package. *Journal of Molecular Graphics*, 11:139–141, June 1993.

[Conway & Sloane 88] J. H. Conway and N. J. A. Sloane. *Sphere Packing, Lattices, and Groups.* Springer-Verlag, New York, NY, 1988.

[Cosman & Schumacker 81] M. Cosman and R. Schumacker. System strategies to optimize CIG image content. In *Proceedings of the Image II Conference*, Scottsdale, Arizona, June 10–12 1981.

[Crow 82] F. C. Crow. A more flexible image generation environment. In *Computer Graphics: Proceedings of SIGGRAPH'82*, volume 16, No. 3, pages 9–18. ACM SIGGRAPH, 1982.

[Das & Joseph 90] G. Das and D. Joseph. The complexity of minimum convex nested polyhedra. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 296–301, 1990.

[DeHaemer, Jr. & Zyda 91] M. J. DeHaemer, Jr. and M. J. Zyda. Simplification of objects rendered by polygonal approximations. *Computers & Graphics*, 15(2):175–184, 1991.

[Delaunay 34] B. Delaunay. Sur la sphère vide. *Bull. Acad. Sci. USSR: Class. Sci. Math. Nat.*, 7:793–800, 1934.

[DeRose *et al* 93] T. D. DeRose, M. Lounsbery, and J. Warren. Multiresolution analysis for surface of arbitrary topological type. Report 93-10-05, Department of Computer Science, University of Washington, Seattle, WA, 1993.

[Dickerson & Geis 69] R. E. Dickerson and I. Geis. *The Structure and Action of Proteins.* Harper & Row, New York, NY, 1969.

[Dobkin & Souvaine 87] D. P. Dobkin and D. L. Souvaine. Computational geometry: a user's guide. In J. T. Schwartz and C.-K. Yap, editors, *Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics*, pages 43–93. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[Edelsbrunner & Mücke 88] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 118–133, 1988.

[Edelsbrunner & Mücke 94] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, January 1994.

[Edelsbrunner 87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, Heidelberg, West Germany, 1987.

[Edelsbrunner 92] H. Edelsbrunner. Weighted alpha shapes. Technical Report UIUCDCS-R-92-1760, Department of Computer Science, University of Illinois at Urbana-Champaign, 1992.

[Edelsbrunner 93] H. Edelsbrunner. The union of balls and its dual shape. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 218–231, 1993.

[Edelsbrunner *et al* 83] 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, July 1983.

[Emiris & Canny 92] 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.

[Fejes Tóth 79] G. Fejes Tóth. Multiple packing and covering of spheres. *Acta Mathematica Academiae Scientiarum Hungarica*, 34:165–176, 1979.

[Fejes Tóth 83] G. Fejes Tóth. New results in the theory of packing in covering. In P. M. Gruber and J. M. Wills, editors, *Convexity and Its Applications*, pages 318–359. Birkhäuser Verlag, 1983.

[Fuchs *et al* 89] 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, July 1989.

[Funkhouser & Séquin 93] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 247–254, August 1993.

[Garey & Johnson 79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.

[Graham & Yao 90] R. Graham and F. Yao. A whirlwind tour of computational geometry. *Amer. Math. Monthly*, 97(8):687–701, 1990.

[Graham 72] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, 1:132–133, 1972.

[Hinker & Hansen 93] P. Hinker and C. Hansen. Geometric optimization. In Gregory M. Nielson and Dan Bergeron, editors, *Proceedings Visualization '93*, pages 189–195, October 1993.

[Hoppe *et al* 93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 19–26, August 1993.

[Imai & Iri 86] H. Imai and M. Iri. An optimal algorithm for approximating a piecewise linear function. *Journal of Information Processing*, 9(3):159–162, 1986.

[Imai & Iri 88] H. Imai and M. Iri. Polygonal approximations of a curve – Formulations and Algorithms. In G. T. Toussaint, editor, *Computational Morphology*, pages 71–86. North-Holland, Amsterdam, Netherlands, 1988.

[Kirkpatrick & Seidel 86] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15:287–299, 1986.

[Lee & Richards 71] B. Lee and F. M. Richards. The interpretation of protein structures: Estimation of static accessibility. *Journal of Molecular Biology*, 55:379–400, 1971.

[Lindsey 86] J. H. Lindsey, II. Sphere-packing in R^3. *Mathematika*, 33:137–147, 1986.

[Lorensen 93] W. E. Lorensen. Geometric clipping using boolean textures. In G. M. Nielson and D. Bergeron, editors, *IEEE Visualization '93 Proceedings*, pages 268–274, October 1993.

[Lovász 75] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[Melkman & O'Rourke 88] A. Melkman and J. O'Rourke. On polygonal chain approximation. In G. T. Toussaint, editor, *Computational Morphology*, pages 87–95. North-Holland, Amsterdam, Netherlands, 1988.

[Mitchell & Suri 92] J. Mitchell and S. Suri. Separation and approximation of polyhedral surfaces. In *Proceedings of 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 296–306, 1992.

[Mitchell 93] J. S. B. Mitchell, 1993. Approximation Algorithms for Geometric Separation Problems, Unpublished Manuscript.

[Perrot *et al* 92] 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.

[Preparata & Shamos 85] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

[Rankin 55] R. A. Rankin. The closest packing of spherical caps in n dimensions. *Proceedings of the Glasgow Mathematical Association*, 2:139–144, 1955.

[Richards 77] F. M. Richards. Areas, volumes, packing and protein structure. *Ann. Rev. Biophys. Bioengg.*, 6:151–176, 1977.

[Richardson 92] D. C. Richardson, 1992. Private Communication.

[Richardson 94] D. C. Richardson, 1994. Private Communication.

[Rogers 58] C. A. Rogers. The packing of equal spheres. *Proceedings of the London Mathematical Society*, 8:609–620, 1958.

[Rogers 64] C. A. Rogers. *Packing and Covering.* Cambridge University Press, London, UK, 1964.

[Rossignac & Borrel 92] J. R. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering complex scenes. Technical Report RC 17697 (#77951), IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10958, 1992.

[Sanner 92] M. F. Sanner. *Modeling and applications of molecular surfaces.* PhD thesis, Universite de Haute-Alsace, France, 1992.

[Sanner 94] M. F. Sanner. Reduced surface, an efficient way to compute solvent excluded surfaces, 1994. To be published.

[Schmitt *et al* 86] F. J. Schmitt, B. A. Barsky, and W. Du. An adaptive subdivision method for surface-fitting from sampled data. *Computer Graphics (SIGGRAPH '86 Proceedings)*, 20(4):179–188, 1986.

[Schroeder *et al* 92] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.

[Seidel 81] R. Seidel. A convex hull algorithm optimal for point sets in even dimensions. M.Sc. Thesis and Report 81/14, Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, 1981.

[Seidel 86] R. Seidel. Constructing higher-dimensional convex hulls at logarithmic cost by face. In *Proc. 18th Annu. ACM Sympos. Theory Comput.*, pages 404–413, 1986.

[Seidel 90] 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.

[Spirakis 83] P. G. Spirakis. Very fast algorithms for the area of the union of many circles. Report 98, Dept. Comput. Sci., New York Univ., New York, NY, 1983.

[Spirakis 84] P. G. Spirakis. The volume of the union of many spheres and point inclusion problems. Report TR 133, Courant Inst. Math. Sci., New York Univ., New York, NY, 1984.

[Surles 92] M. C. Surles. An algorithm with linear complexity for interactive, physically-based modeling of large proteins. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 221–230, July 1992.

[Turk 92] G. Turk. Re-tiling polygonal surfaces. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 55–64, July 1992.

[Turk 94] G. Turk, 1994. Private Communication.

[Varshney & Brooks 93] A. Varshney and F. P. Brooks, Jr. Fast analytical computation of Richards's smooth molecular surface. In G. M. Nielson and D. Bergeron, editors, *IEEE Visualization '93 Proceedings*, pages 300–307, October 1993.

[Varshney *et al* 94a] A. Varshney, Jr. F. P. Brooks, and W. V. Wright. Computing smooth molecular surface. *IEEE Computer Graphics & Applications*, September 1994.

[Varshney *et al* 94b] A. Varshney, Jr. F. P. Brooks, and W. V. Wright. Interactive visualization of weighted three-dimensional alpha hulls. In *Video Review, Proceedings of the Tenth Annual Symposium on Computational Geometry*, pages 395–396. ACM Press, June 1994.

[Voronoi 07] G. M. Voronoi. Nouvelles applications des paramètres continus à la théorie d es formes quadratiques. premier Mémoire: Sur quelques propriétés des formes quadratiques positives parfaites. *J. Reine Angew. Math.*, 133:97–178, 1907.

[Weiner *et al* 84] 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.

[Welzl 91] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *LNCS 555 (New Results and New Trends in Computer Science)*, pages 359–370. Springer-Verlag, 1991.