CS-1995-20

Generating Levels of Detail for Large-Scale Polygonal Models [1]

Amitabh Varshney[2] Pankaj K. Agarwal[3]
Frederick P. Brooks, Jr.[4] William V. Wright[5]
Hans Weber[6]

Department of Computer Science

Duke University

Durham, North Carolina 27708–0129

[1]Work by the first, third, fourth, and fifth authors was in part supported by NIH, NCRR grant number P41-RR02170, and ARPA/ESTO grant number DABT 63-93-C-0048. Work by the first author has also been supported by National Science Foundation grant CCR-9502239. Work by the second author has been supported by National Science Foundation Grant CCR-93–01259, an NYI award, and by matching funds from Xerox Corp. Equipment support has been provided by NSF/ARPA, Science and Technology Center for Computer Graphics and Scientific Visualization, Agreement number ASC- 8920219.

[2]varshney@cs.sunysb.edu, Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794-4400

[3]pankaj@cs.duke.edu, Department of Computer Science, Duke University, Durham, NC 27708-0129

[4]brooks@cs.unc.edu, Department of Computer Science, UNC–Chapel Hill, Chapel Hill, NC 27599-3175

[5]wright@cs.unc.edu, Department of Computer Science, UNC–Chapel Hill, Chapel Hill, NC 27599-3175

[6]weberh@cs.unc.edu, Department of Computer Science, UNC–Chapel Hill, Chapel Hill, NC 27599-3175

Generating Levels of Detail for Large-Scale Polygonal Models *

Amitabh Varshney† Pankaj K. Agarwal‡ Frederick P. Brooks, Jr.§
William V. Wright¶ Hans Weber‖

Abstract

We present an efficient algorithm for generating various levels-of-detail approximations for a given polygonal model. Our algorithm guarantees that all points of an approximation are within a user-specifiable distance ϵ from the original model and all points of the original model are within a distance ϵ from the approximation. Each approximation attempts to minimize the total number of polygons required to satisfy the previous constraint. We show how the problem of generating levels-of-detail approximations reduces to the classic set partition problem. The various approximations are guaranteed to be topologically consistent with the input polygonal model. The approximations can be constrained by the user to preserve any desired edges of the input model. We also propose a method to compute an estimate of the quality of the approximation generated by our algorithm with respect to the optimal approximation satisfying the same constraints. We have implemented our algorithm and have obtained experimental results of multiresolution hierarchy generation on over a thousand polygonal objects from a CAD model of a notional submarine.

*Work by the first, third, fourth, and fifth authors was in part supported by NIH, NCRR grant number P41-RR02170, and ARPA/ESTO grant number DABT 63-93-C-0048. Work by the first author has also been supported by National Science Foundation grant CCR-9502239. Work by the second author has been supported by National Science Foundation Grant CCR-93–01259, an NYI award, and by matching funds from Xerox Corp. Equipment support has been provided by NSF/ARPA, Science and Technology Center for Computer Graphics and Scientific Visualization, Agreement number ASC- 8920219.

†varshney@cs.sunysb.edu, Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794-4400

‡pankaj@cs.duke.edu, Department of Computer Science, Duke University, Durham, NC 27708-0129

§brooks@cs.unc.edu, Department of Computer Science, UNC–Chapel Hill, Chapel Hill, NC 27599-3175

¶wright@cs.unc.edu, Department of Computer Science, UNC–Chapel Hill, Chapel Hill, NC 27599-3175

‖weberh@cs.unc.edu, Department of Computer Science, UNC–Chapel Hill, Chapel Hill, NC 27599-3175

1 Introduction

We present an efficient algorithm for computing a multiresolution hierarchy a given polygonal model. Multiresolution representations of an object find several uses in computer graphics:
(a) In a level-of-detail based rendering algorithm for providing desired frame update rates [5, 7, 8, 13, 2].
(b) Using low-detail approximations of objects for illumination algorithms, especially radiosity [24].
(c) 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 or no benefits [27, 26, 17, 19, 16].

In this paper we discuss an approach for generating minimal approximations to a given polygonal representation of an object that are guaranteed not to deviate from the original by more than a user-specifiable amount. Our approach has several benefits in computer graphics. First, one 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, 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 us to have adaptive approximation over various parts of an object. This could be used in selectively preserving those features of an object that are perceptually important. Third, just the specification of one parameter, the user-specifiable tolerance for approximation, is required to obtain the approximations; fine tweaking of several parameters depending upon the object to be approximated is not required. Thus, this approach is quite useful for automating the process of genus-preserving simplifications of a large number of objects.

The rest of this paper is organized as follows. In Section 2 we review some of the previous work done in the area of approximation of polygonal models. Then in Section 3 we formally state our problem, list assumptions for our algorithm, and give an outline of the algorithm. The next three sections describe the details of our algorithm. In Section 6.5 we discuss some useful features of our approach and in Section 7 we discuss our experimental results.

2 Previous Work

Approximation algorithms for polygonal models can be classified into two categories:
Min-# Approximations. Here 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 away from the input model $\mathcal{I}$.
Min-ϵ Approximations. Here given the number of vertices of the approximation $\mathcal{A}$, 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 [25] and [10] where they adaptively subdivide a series of bicubic patches and polygons over a surface until they fit the data within the tolerance levels.

In the second category, work has been done by Turk, Schroeder *et al*, and Hinker and Hansen [27, 26, 17]. Turk [27] 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* [26] and Hinker and Hansen [17] 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 by Rossignac and Borrel in [23] 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* [18] 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. An interesting and elegant solution to the problem of polygonal simplification by using wavelets has been presented by DeRose *et al* in [11].

In computational geometry, it has been shown that computing the minimal-facet ϵ-approximation is NP-hard for both convex polytopes [9] and polyhedral terrains [1]. 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. An algorithm has been given in [21] for computing an ϵ-approximation of size $O(k_o \log n)$ for convex polytopes. This has recently been improved by Clarkson in [6]; he proposes a randomized algorithm for computing an approximation of size $O(k_o \log k_o)$ in expected time $O(k_o n^{1+\delta})$ for any $\delta > 0$ (the constant of proportionality depends on δ, and tends to $+\infty$ as δ tends to 0). Brönnimann and Goodrich [3] observed that a variant of Clarkson's algorithm yields a polynomial-time deterministic algorithm that computes an approximation of size $O(k_0)$. Working with polyhedral terrains, Agarwal and Suri [1] present a polynomial-time algorithm that computes an ϵ-approximation of size $O(k_o \log k_o)$ to a polyhedral terrain.

To the best of our knowledge, no prior work has been done in genus-preserving ϵ-approximation of polygonal objects, such that the complexity of the approximate solution could be related to the (unknown) optimal solution.

3 Problem Definition and Algorithm Outline

3.1 Problem Definition

Let $\mathcal{I}$ be a three-dimensional compact and orientable object whose polygonal representation $\mathcal{P}(\mathcal{I})$ is given to us. Our objective is to compute a piecewise-linear approximation $\mathcal{A}$ of $\mathcal{P}(\mathcal{I})$. Henceforth, we will refer to $\mathcal{P}(\mathcal{I})$ as $\mathcal{P}$. 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 $\mathcal{P}$

¿From a mathematical and aesthetical point of view, an approximation scheme should be invariant under the similarity transformation, should ensure that the volume of the difference between $\mathcal{A}$ and $\mathcal{P}$ is small, and should be topology- and symmetry-preserving. In most computer graphics models, vertices besides storing position coordinates, also store a lot of additional useful information, such as color, normals, texture coordinates, etc. Therefore it is desirable to have the Vertices($\mathcal{A}$) $\subseteq$Vertices ($\mathcal{P}$). Keeping these issues in mind, we define the problem as follows:

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

3.2 Assumptions on Input

Without loss of generality, we 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, no T-junctions, etc. We also assume 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, then 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 [15] or Phong [22] 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 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 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.

3.3 Algorithm Outline

Our basic approach is to first generate two *offset surfaces* to the input piecewise linear model $\mathcal{P}$ – one on the outside and the other inside $\mathcal{P}$. This is discussed in Section 4 and illustrated in Figure 6. Next we generate all *candidate triangles* whose vertices are a subset of the vertices of $\mathcal{P}$ and that lie within the two offset surfaces. We then associate the vertices and triangles of P with the candidate triangles that "cover" them. This process is explained in Section 5. Our final step is a greedy approach for selecting the solution triangles that define the approximation $\mathcal{A}$ 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 6.

4 Generation of Offset Surfaces

The ϵ-offset for a parametric surface $\mathbf{f}(s,t) = (f_1(s,t), f_2(s,t), f_3(s,t))$, whose unit normal to $\mathbf{f}$ is $\mathbf{n}(s,t) = (n_1(s,t), n_2(s,t), n_3(s,t))$, 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 the offset surface $\mathcal{P}(+\epsilon)$ (respectively $\mathcal{P}(-\epsilon)$) for an object $\mathcal{I}$ to be the 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. These two offset surfaces are shown for a very simple polygonal object in Figure 6 ($\mathcal{P}(+\epsilon)$ in red, $\mathcal{P}(-\epsilon)$ in green).

In order to preserve the input topology of $\mathcal{P}$, we desire that these offset surfaces do not self-intersect. To meet this criterion we reduce our level of approximation at certain places. In other words, to guarantee that no intersections amongst the offset surfaces occur, we 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 describe a method to generate a particular kind of non-intersecting offset surfaces that lie on an offset of no more than ϵ from $\mathcal{P}$.

4.1 Edge Halfspaces

Let $e = (v_1, v_2)$ be an edge of $\mathcal{P}$. If the normals $\mathbf{n}_1, \mathbf{n}_2$ to $\mathcal{I}$ at both v_1 and v_2, respectively, are identical, then 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. 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^- (see Fig 1(a)). However, in general the normals $\mathbf{n}_1$ and $\mathbf{n}_2$ at the vertices v_1 and v_2 need not be identical, in which case it is not clear how to define the two halfspaces for an edge. 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 β_e. Let the two halfspaces for the edge e in this case be β_e^+ and β_e^-. This is shown in Fig 1(b). A bilinear patch β_e^- is defined over a parametric domain $(u, v), 0 \leq u, v \leq 1$ as: $\mathbf{x}(u, v) = \mathbf{b_{00}}(1-u)(1-v) + \mathbf{b_{01}}(1-u)v + \mathbf{b_{10}}u(1-v) + \mathbf{b_{11}}uv$, where $\mathbf{b_{00}}, \mathbf{b_{01}}, \mathbf{b_{10}}$, and $\mathbf{b_{11}}$ are the four corner points of a bilinear patch. For details, refer [12].

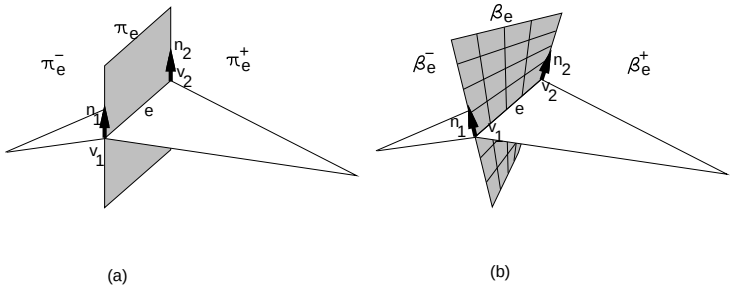

Figure 1: Edge Halfspaces

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 2.

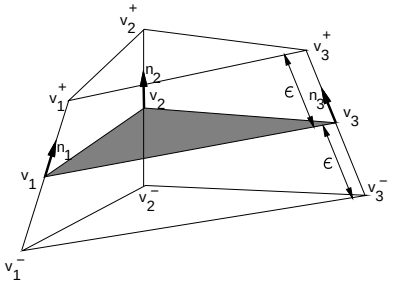

Figure 2: The Fundamental Prism

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 2) and we will henceforth refer to it as the *fundamental prism.*

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 two offset surfaces $\mathcal{P}(+\epsilon)$ and $\mathcal{P}(-\epsilon)$ may self-intersect 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 fundamental 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 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, thus causing self-intersection of the offset surface.

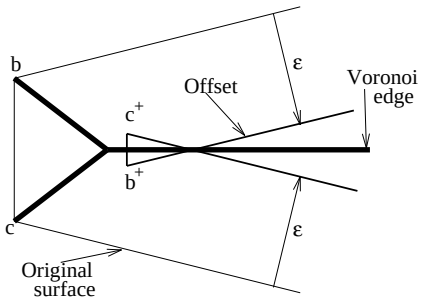

Figure 3: Offset Surfaces

Once we make this observation, the solution to avoid self-intersections becomes quite simple — just do not allow an 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 well in theory, efficient computation of the three-dimensional Voronoi diagram of the fundamental triangles is difficult. To avoid this, we adopt a conservative approach for recomputing the value of ϵ at each vertex. This approach underestimates the values for the positive and negative ϵ. In other words, it guarantees the offset surfaces not to 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 4.

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

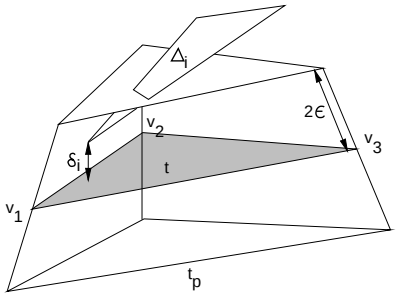

Figure 4: Computation of δ_i

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.

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.

We define two vertices v_i and v_j to be visible to each other iff an observer at v_i (or v_j) can see the vertex v_j (or v_i), with the two offset surfaces providing occlusion. This condition for visibility is equivalent to the condition that the line segment joining v_i and v_j does not intersect $\mathcal{P}(+\epsilon)$ or $\mathcal{P}(-\epsilon)$.

We define an edge e to be visible to a vertex v iff 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 does not intersect $\mathcal{P}(+\epsilon)$ or $\mathcal{P}(-\epsilon)$.

A line segment $e = (v_i, v_j)$ is a *candidate edge* if v_i and v_j are visible to each other. A triangle $\triangle$ is a *candidate triangle* if every edge of $\triangle$ is visible to the vertex not incident to it. Keeping this in mind, we first generate all candidate edges (v_i, v_j). 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 to find which of the vertices of $\mathcal{P}$ are covered by a given candidate triangle. The general idea being that we like to give a greater preference to those candidate triangles that cover more vertices of $\mathcal{P}$. 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 iff 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 $\mathcal{P}$.

6 Composing the Final Solution

Let us take a moment to review the information we have accumulated over the last couple of sections. 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) covers all the vertices of $\mathcal{P}$, (b) does not self intersect, and (c) does 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*.

6.1 Set Cover and Set Partition

Consider a set of integers $P = \{1, 2, \ldots, p\}$ and another set $T = \{t_1, t_2, \ldots, t_m\}$, where $t_j \subseteq P$ for $j \in J = \{1, 2, \ldots, m\}$. A subset $C \subseteq J$ defines a *cover* of P if $\bigcup_{j \in C} t_j = P$. Let $c_j > 0$ be the *cost* 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 $t_i \bigcap t_j = \emptyset$, 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 exclusive and collectively contain all the elements of P. Finding a partition of minimum cost is referred to as the *set partition problem*. We will next show how the problem of finding an ϵ-approximation to a polyhedral object can be transformed to the problem of set partition.

Let v_i, v_j be a visible pair of vertices (i.e., the segment (v_i, v_j) does not intersect the offset surfaces). We project the segment $\gamma_{ij} = (v_i, v_j)$ on to the surface $\mathcal{P}$ as follows. Since γ_{ij} does not intersect the offset surfaces, it can intersect the fundamental prisms only at the bilinear patches erected on the edges of fundamental triangles. Let $a_1, a_2, \ldots, a_k$ be the sequence of the intersection points of γ_{ij} and the bilinear patches of fundamental prisms, sorted along γ_{ij}. If the intersection point a_l lies on a bilinear patch that was erected on an edge e of a fundamental triangle t, then let a_l^* denote the projection of a_l onto the edge e (if $a_l = (u_{a_l}, v_{a_l})$ and $e = (u, 0), 0 \leq u \leq 1$, in the parametric domain of the bilinear patch as defined in Section 4.1, then $a_l^* = (u_{a_l}, 0)$). By construction, the segment $a_l^* a_{l+1}^*$ lies on a fundamental triangle. Let γ_{ij}^* be the polygonal chain $a_1^* a_2^* ... a_k^*$ lying on the surface $\mathcal{P}$. Let $\Gamma^* = \{\gamma_{ij}^* \mid v_i \text{ and } v_j \text{ is a visible pair of vertices}\}$. Γ^* also includes each edge of $\mathcal{P}$. Γ^* induces a subdivision on $\mathcal{P}$, denoted as $\mathbf{A}(\Gamma^*)$. Every face of $\mathbf{A}(\Gamma^*)$ is contained in a fundamental triangle. Let $f_1, f_2, \ldots, f_p$ be the set of faces in $\mathbf{A}(\Gamma^*)$. Let $\triangle_i$ be a candidate triangle with edges e_{i_1}, e_{i_2}, and e_{i_3} whose projections on $\mathcal{P}$ are the polygonal chains $\gamma_{i_1}, \gamma_{i_2}$, and γ_{i_3}, respectively. These three polygonal chains enclose a set of faces $f_{i_1}, f_{i_2}, \ldots, f_{i_q}$ in $\mathbf{A}(\Gamma^*)$ that lie within the fundamental prisms that $\triangle_i$ intersects. Let us define the projection $\triangle_i^*$ of the triangle $\triangle_i$ on to the surface of $\mathcal{P}$ to be the set $\{f_{i_1}, f_{i_2}, \ldots, f_{i_q}\}$ of these faces.

Let the indices of the faces $f_1, f_2, \ldots, f_p$ in $\mathbf{A}(\Gamma^*)$ be represented by the elements of $P = \{1, 2, \ldots, p\}$, and let the projections $\triangle_j^*$ of the candidate triangles $\triangle_j$ be the elements t_j of the set T, with $i \in t_j$ iff the face f_i of $\mathbf{A}(\Gamma^*)$ is contained in $\triangle_j^*$. With this equivalence, the problem of finding the smallest number of corresponding candidate triangles $\triangle_i$ that ϵ-approximate a given polygonal object $\mathcal{P}$ can be transformed to the problem of determining the smallest set partition of the set P by the elements t_i of the set T. The above transformation, although theoretically sound, has a large running time. If the number of vertices in $\mathcal{P}$ is n, $|\Gamma^*| = O(n^2)$ and therefore $|P| = |\mathbf{A}(\Gamma^*)| = O(n^4)$. Also, $|T| = O(n^3)$. Therefore the binary occupancy matrix that relates which elements of P belong to which elements of T has a size $O(n^4) \times O(n^3) = O(n^7)$, a trivial lower bound for any set-partitioning algorithm that attempts to solve this problem.

If we consider covering vertices, instead of covering the faces of $\mathbf{A}(\Gamma^*)$, we can achieve better run times. Let the indices of the vertices of $\mathcal{P}$ define the set P, and the candidate triangles $\triangle_i$ define the set T, with $i \in t_j$ iff the candidate triangle t_j covers a vertex v_i (i.e., the segment $v_i^+ v_i^-$ intersects t_j). Now 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 the inverse of its cardinality, i.e., the inverse of the number of vertices covered by it, then the solution of this set partitioning problem will yield the smallest number of triangles that cover all the vertices of the input $\mathcal{P}$. With this approach we get $|P| = n$ instead of $|P| = O(n^4)$. We have implemented this latter formulation.

6.2 The Greedy Heuristic

The set cover and the set partition problems are both known to be NP-complete [14], so we cannot hope to have a polynomial-time algorithm for computing an optimal solution to our polygonal approximation problem. However, there exist several heuristics that compute approximate solutions to these problems. These heuristics generate solutions that have been found, in general, 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 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 that 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 until all points have been covered.

We next explain the process of checking whether two given triangles overlap each other or not. 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 5 illustrates the overlapping projections on a fundamental triangle.

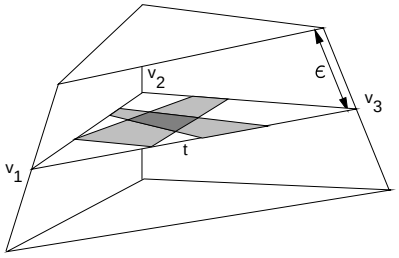

Figure 5: Checking for Overlaps

It should be noted that this greedy heuristic may not preserve an object's symmetry.

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 use the following method.

We maintain a complete mesh at every iteration of the algorithm. Let the mesh at the i^{th} iteration 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}.

6.4 Estimating Solution Quality

It has been shown in the literature [20, 4] that the greedy heuristic yields a solution to the set cover problem that is guaranteed to be within a factor of $(1 + \lg d)$ of the optimal, where d is the maximum number of elements in any set t_j. It is easy to prove that if in the greedy heuristic at each stage we select the set that 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 $\alpha(1 + \lg d)$.

Due to the set-disjointness constraint, our approach does not give any performance guarantees, such as the above, on the quality of the solution obtained with respect to the optimal. However, we can compute a worst-case estimate of how far away our solution is from the optimal in the following way. At each iteration of the greedy approach, we compute the ratio of the largest number of uncovered points in any candidate triangle (that may or may not overlap with the triangles already determined to be in $\mathcal{A}$) to the largest number of points in a non-overlapping candidate triangle. Let α be the maximum of these ratios computed over all iterations of the greedy algorithm. Then, $|\mathcal{A}_o| \geq |\mathcal{A}|/(\alpha(1 + \lg d))$ where $\mathcal{A}_o$ is the optimal solution. This gives a worst-case estimate of the quality of our solution $\mathcal{A}$ with respect to the optimal $\mathcal{A}_o$.

6.5 Some Useful Properties

Interpolation. Smooth interpolation between various levels of detail is useful in avoiding 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, to incorporate interpolation with this approach we can move each vertex along the direction of its normal till it reaches its replacement triangle.

Preserving user-specified edges. One of the important properties in any approximation scheme is the way it preserves normal discontinuities or sharp edges present in the input model. Our approach can preserve any user-specifiable edge e. We note that an edge e can disappear from the output $\mathcal{A}$ iff there is a triangle in $\mathcal{A}$ that intersects the bilinear patch β_e or the plane π_e that defines the two edge halfspaces for e. We can invalidate all such candidate triangles, thereby retaining e in the solution, by making β_e (or π_e) "opaque" in the visibility computations.

Adaptive approximation. Consider the case where certain features of an object are crucial to human-perception and are not to be approximated beyond a certain level. In our approach since all the candidate triangles are constrained to lie within the two offset surfaces, manipulation of these

offset surfaces provides an easy way to smoothly control the local level of approximation. Thus, a user may vary the ϵ associated with a region to adaptively change the level of local approximation.

7 Results

We have implemented our algorithm and tried it out on several thousand objects, most of which are part of a model of a notional submarine. On average we were able to achieve simplifications 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. The results in this section are for a version of our implementation that removes vertices instead of triangles.

We simplified a total of 1090 objects from the auxilliary machine room (AMR) of the submarine dataset for testing our algorithm. These objects have been cumulatively reduced as follows:

Level-of-Detail	Dataset Complexity	% Reduction
Original dataset	350,023 triangles	0.0
First level	206,859 triangles	40.9
Second level	141,983 triangles	59.4
Third level	104,874 triangles	70.0

Table 1: Polygonal Simplification Results

The value of ϵ that was chosen to approximate the polygonal objects varied from one object to the other, as it should. For the set of results described above, we manually assigned ϵ values. Recently we have automated the process by using a simple heuristic which sets the ϵ value for each object to be a percentage of the diagonal of its bounding box. When performing simplifications on the AMR model described above using this heuristic, we obtained the reductions presented in Table 2.

Level-of-Detail	% of Diagonal	% Reduction
Original dataset	0	0.0
First level	1	37.8
Second level	2	54.6
Third level	4	63.7
Fourth level	8	68.7

Table 2: Polygonal Simplification Results for ϵ Set to a Percentage of the Object's Bounding Box

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. First, 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. Second, 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 the 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.

Figures 7 and 8 show simplifications of a roller structure from the torpedo room portion of the model, while Figure 9 shows 3 levels-of-detail of a segment of the AMR dataset.

References

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

[2] J. Bloomenthal. Polygonalization of implicit references. *Computer and Geometric Design*, 5:341–355, 1988.

[3] H. Brönnimann and M. Goodrich. Almost optimal set covers in finite VC-dimension. In *Proceedings Tenth ACM Symposium on Computational Geometry*, pages 293–302, 1994.

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

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

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

[7] 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.

[8] 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.

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

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

[11] 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.

[12] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press, San Diego, California, third edition, 1993.

[13] 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.

[14] 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.

[15] H. Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, C-20(6):623–629, June 1971.

[16] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. In *Computer Graphics: Proceedings of SIGGRAPH 1993*, pages 231–238. ACM SIGGRAPH, 1993.

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

[18] 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.

[19] A. D. Kalvin and R. H. Taylor. Superfaces: Polyhedral approximation with bounded error. Technical Report RC 19135 (#82286), IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10958, 1993.

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

[21] 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.

[22] Bui-T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, June 1975.

[23] 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.

[24] H. E. Rushmeier, C. Patterson, and A. Veerasamy. Geometric simplification for indirect illumination calculations. In *Proceedings Graphics Interface '93*, pages 227–236, 1993.

[25] 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.

[26] 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.

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

Figure 6: The blue surface is the original surface, the red and green are the outer and inner shells, and the beige is the simplification.

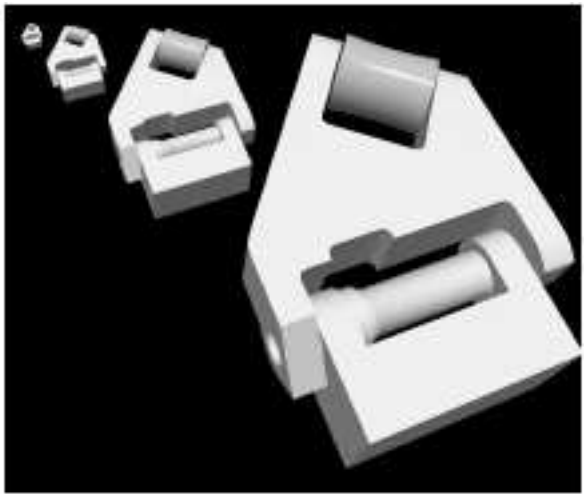

Figure 7: Decreasing level-of-detail roller structures at increasing distances from the viewer.

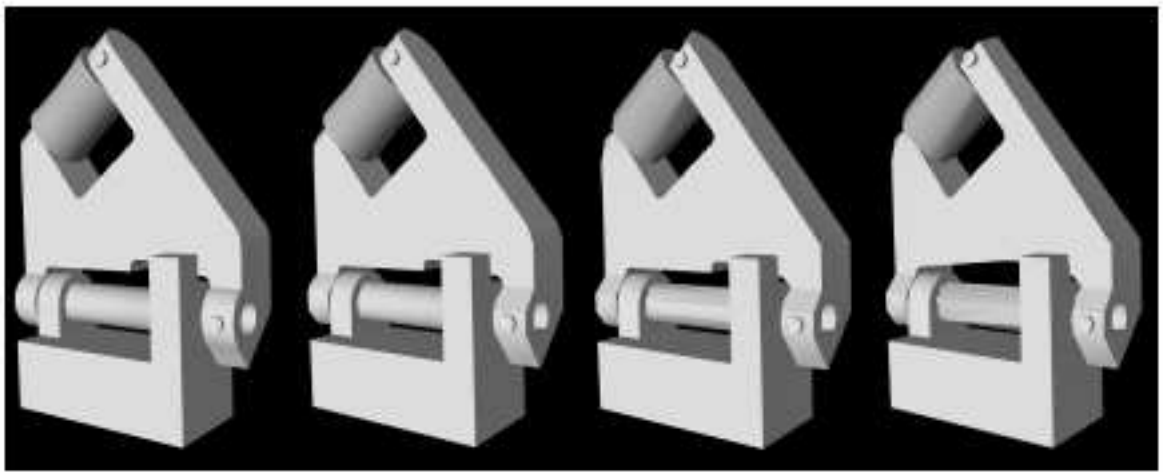

Figure 8: Side-by-side comparison of 4 levels-of-detail of a torpedo room roller. From left to right: 2,346 triangles, 1,180 triangles, 676 triangles, and 514 triangles.

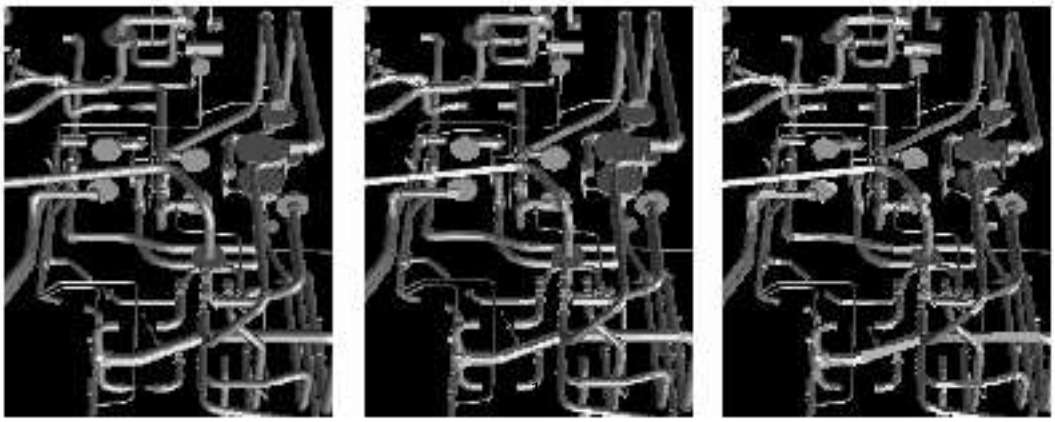

Figure 9: Side-by-side comparison of 3 levels-of-detail of a 625 object segment of the auxilliary machine room dataset. From left to right: 86,232 triangles, 31,486 triangles, and 12,788 triangles.