# Simplification Envelopes

Jonathan Cohen* Amitabh Varshney† Dinesh Manocha*
Greg Turk* Hans Weber* Pankaj Agarwal‡ Frederick Brooks*
William Wright*

## Abstract

We propose the idea of *simplification envelopes* for generating a hierarchy of level-of-detail approximations for a given polygonal model. Our approach guarantees that all points of an approximation are within a user-specifiable distance $\epsilon$ from the original model and all points of the original model are within a distance $\epsilon$ from the approximation. Simplification envelopes provide a general framework within which a large collection of existing simplification algorithms can run. We demonstrate this technique in conjunction with two algorithms, one global, the other local. The global algorithm provides a method to compute an estimate of the quality (output size) of its approximations with respect to the optimal approximation satisfying the same constraints. The local algorithm provides a fast method for generating approximations to large input meshes (at least hundreds of thousands of triangles).

Each approximation attempts to minimize the total number of polygons required to satisfy the above $\epsilon$ constraint. The key advantages of our approach are:

- General technique providing guaranteed error bounds for genus-preserving simplification
- Automation of both simplification process and selection of appropriate viewing distances
- Prevention of self-intersection
- Preservation of sharp features
- Allows variation of approximation distance across different portions of a model

**CR Categories and Subject Descriptors:** I.3.3 [**Computer Graphics**]: Picture/Image Generation — Display algorithms; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling — Curve, surface, solid, and object representations.

**Additional Key Words and Phrases:** hierarchical approximation, model simplification, levels-of-detail generation, shape approximation, geometric modeling, offsets.

*Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599-3175. {cohenj,weberh,manocha,turk,brooks,wright}@cs.unc.edu

†Department of Computer Science, State University of New York, Stony Brook, NY 11794-4400. varshney@cs.sunysb.edu

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

# 1 Introduction

We present the framework of *simplification envelopes* for computing various levels of detail of a given polygonal model. These hierarchical 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 [CS81, FS93].
- Simplifying traditionally over-sampled models such as those generated from volume datasets, laser scanners, and satellites for storage and reducing CPU cycles during processing, with relatively few or no disadvantages [Tur92, SZL92, HH93, KT93, GKM93, HHK+95].
- Using low-detail approximations of objects for illumination algorithms, especially radiosity [RPV93].

Simplification envelopes are a generalization of *offset surfaces*. Given a polygonal representation of an object, they allow the generation of minimal approximations that are guaranteed not to deviate from the original by more than a user-specifiable amount while preserving global topology. We surround the original polygonal surface with two envelopes, then perform simplification within this volume. A sample application of the algorithms we describe can be seen in Figure 1.

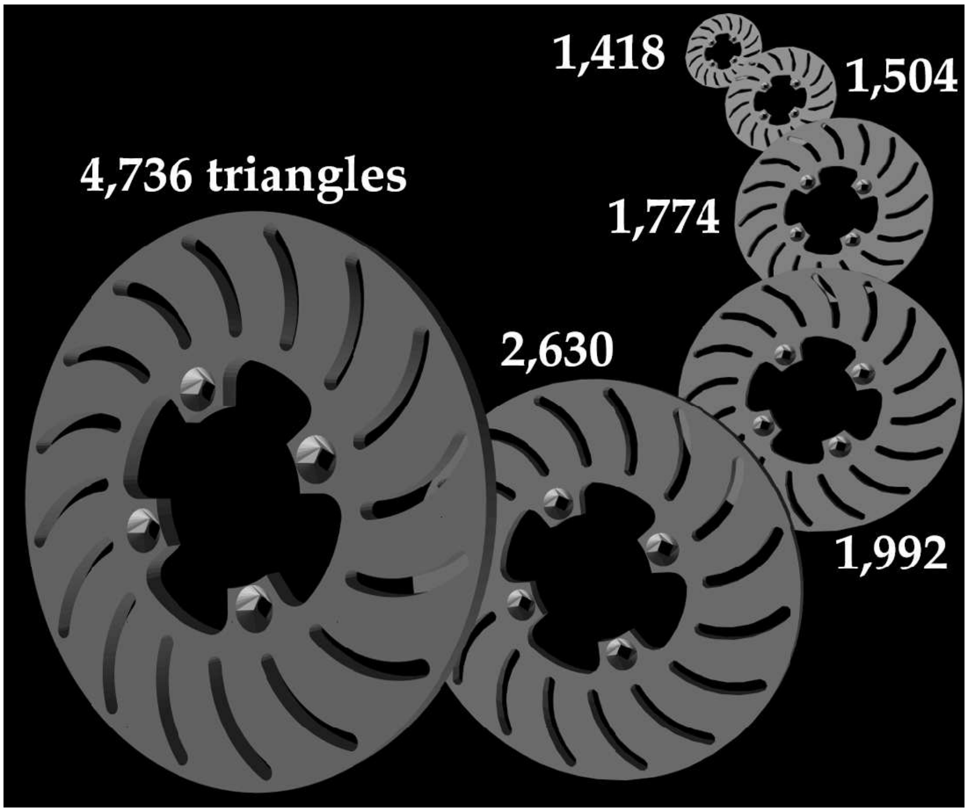



Figure 1: A level-of-detail hierarchy for the rotor from a brake assembly.

Such an approach has several benefits in computer graphics. First, one can very precisely quantify the amount of approximation that is tolerable under given circumstances. Given a user-specified error in number of pixels of deviation of an object's silhouette, it is possible to choose which level of detail to view from a particular distance to maintain that pixel error bound. Second, this approach allows one a fine control over which regions of an object should be approximated more and which ones less. This could be used for selectively preserving those features of an object that are *perceptually* important. Third, the user-specifiable tolerance for approximation is the only parameter 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 the paper is organized in the following manner: we survey the related work in Section 2, explain our assumptions and terminology in Section 3, describe the envelope and approximation computations in Sections 4 and 5, present some useful modifications to the approximation algorithms in Sections 6, 7, and 8, and explain our implementation and results in Section 9.

# 2 Background

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

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

- **Min-$\epsilon$ 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 $\mathcal{I}$.

Previous work in the area of min-# approximations has been done by [SBD86, DZ91] 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 several groups. Turk [Tur92] 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. [SZL92] and Hinker and Hansen [HH93] 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 [RB93] where vertices that are close to each other are clustered and a new vertex is generated to represent them. The mesh is suitably updated to reflect this.

Hoppe et al. [HDD$^+$93] proceed by iteratively optimizing an energy function over a mesh to minimize both the distance of the approximating mesh from the original, as well as the number of approximating vertices. An interesting and elegant solution to the problem of polygonal simplification by using wavelets has been presented in [DLW93, EDD$^+$95] where arbitrary polygonal meshes are first subdivided into patches with *subdivision connectivity* and then multiresolution wavelet analysis is used over each patch. This wavelet approach preserves global topology.

In computational geometry, it has been shown that computing the minimal-facet $\epsilon$-approximation is NP-hard for both convex polytopes [DJ90] and polyhedral terrains [AS94]. Thus, algorithms to solve 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 [MS92] for computing an $\epsilon$-approximation of size $O(k_o \log n)$ for convex polytopes. This has recently been improved by Clarkson in [Cla93]; 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 $\delta$, and tends to $+\infty$ as $\delta$ tends to 0). In [BG94]

Brönnimann and Goodrich 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, [AS94] present a polynomial-time algorithm that computes an $\epsilon$-approximation of size $O(k_o \log k_o)$ to a polyhedral terrain.

Our work is different from the above in that it allows adaptive, genus-preserving, $\epsilon$-approximation of polygonal objects, such that the complexity of the approximate solution can be related to the (unknown) optimal solution. Additionally, we can simplify bordered meshes and meshes with holes. In terms of direct comparison with the global topology preserving approach presented in [DLW93, EDD$^+$95], for a given $\epsilon$ our algorithm has been *empirically* able to obtain "reduced" simplifications, which are closer to the optimal (as demonstrated in Section 9). The algorithm in [RB93] also guarantees a bound in terms of the Hausdorff metric between the vertices of the original model and the simplified model. However, it is not guaranteed to preserve the genus of the original model.

# 3 Terminology and Assumptions

Let us assume 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 $\mathcal{A}$ 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 *$\epsilon$-approximations* of each other iff every point on $\mathcal{P}$ is within a distance $\epsilon$ of some point of $\mathcal{Q}$ and every point on $\mathcal{Q}$ is within a distance $\epsilon$ of some point of $\mathcal{P}$.

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

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 and that linear interpolation is sufficient to reasonably duplicate the normal-based properties of the object. In general, polygonal approximations to curved objects have unique vertex normals to avoid the discontinuities in the normal-based properties of the object. However, if the object being modeled has sharp edges, such as a cylinder, we would like to retain the discontinuity in the normals across the edges. In such cases, a faithful polygonal representation requires that there be multiple normals associated with each vertex — 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 and then show that with a simple and straightforward modification we can handle the case where normal discontinuities are allowed.

# 4 Simplification Envelope Computation

The three-dimensional $\epsilon$-offset surface 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

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

Note that offset surfaces for a polygonal object can self-intersect and may contain non-linear elements. We define a simplification envelope $\mathcal{P}(+\epsilon)$ (respectively $\mathcal{P}(-\epsilon)$) for an object $\mathcal{I}$ to be a *polygonal* surface that lies *within* a distance of $\epsilon$ from every point $p$ on $\mathcal{I}$ in the same (respectively opposite) direction as the normal to $\mathcal{I}$ at $p$. Thus, the simplification envelopes can be thought of as an approximation to offset surfaces. Henceforth we shall refer to simplification envelope by simply envelope.

### 4.1 Non-self-intersecting Envelopes

In order to preserve the input topology of $\mathcal{P}$, we desire that the simplification envelopes 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 envelopes occur, we have to be content at certain places with the distance between $\mathcal{P}$ and the envelope being smaller than $\epsilon$. This is how simplification envelopes differ from offset surfaces.

Let us refer to the triangles of the given polygonal representation $\mathcal{P}$ as the *fundamental triangles*. We construct our envelope such that each of its triangles corresponds to a fundamental triangle. We offset each vertex of the original surface in the direction of its normal vector to transform the fundamental triangles into those of the envelope.

If we offset each vertex $v_i$ by the same amount $\epsilon$, to get the offset vertices $v_i^+$ and $v_i^-$, the resulting envelopes, $\mathcal{P}(+\epsilon)$ and $\mathcal{P}(-\epsilon)$, may contain self-intersections 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 envelopes 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 2 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 envelope.

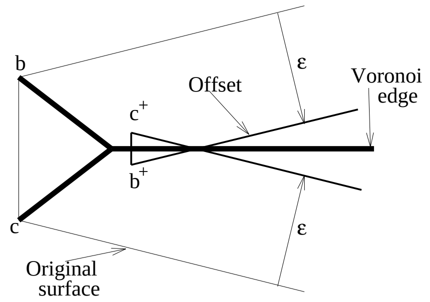



Figure 2: Offset Surfaces

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 $\epsilon$ for each vertex $v_i$ such that the vertices $v_i^+$ and $v_i^-$ determined with this new $\epsilon$ do not lie in the Voronoi regions of the non-adjacent fundamental triangles.

While this works in theory, efficient and robust computation of the three-dimensional Voronoi diagram of the fundamental triangles is non-trivial. We now present two methods for computing the reduced $\epsilon$ for each vertex, the first method analytical, and the second numerical.

### 4.2 Analytical $\epsilon$ Computation

We must first introduce the notions of *edge half-spaces* and the *fundamental prism*. Then, using these concepts, we describe the analytical method for computing the reduced $\epsilon$'s.

### 4.2.1 Edge Half-spaces

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 $\pi_e$ 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 $\pi_e$. Such a plane defines two half-spaces for edge $e$, say $\pi_e^+$ and $\pi_e^-$ (see Fig 3(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 half-spaces for an edge. One choice is 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 $e$ $\beta_e$. Let the two half-spaces for the edge $e$ in this case be $\beta_e^+$ and $\beta_e^-$. This is shown in Fig 3(b).

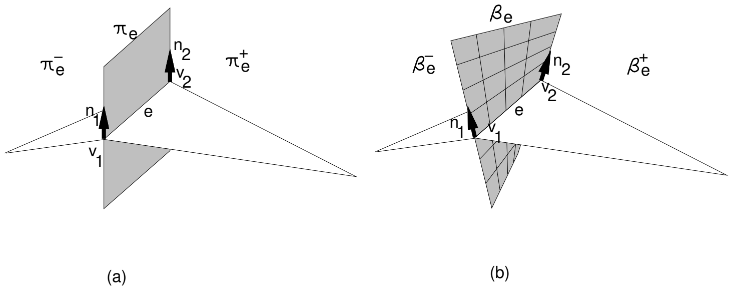



Figure 3: Edge Half-spaces

### 4.2.2 The Fundamental Prism

Consider one fundamental 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. The coordinates and the normal of a $(+\epsilon)$-offset vertex $v_i^+$ for a vertex $v_i$ are: $\mathbf{c}(v_i^+) = \mathbf{c}(v_i) + \epsilon\mathbf{n}(v_i)$, and $\mathbf{n}(v_i^+) = \mathbf{n}(v_i)$. The $(-\epsilon)$-offset vertex can be similarly defined in the opposite direction. These offset vertices for a fundamental triangle are shown in Figure 4.

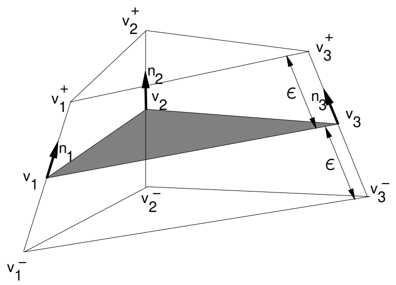


Figure 4: The Fundamental Prism

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 half-spaces. This object contains the fundamental triangle (shown shaded in Figure 4) and we will henceforth refer to it as the *fundamental prism*.

### 4.2.3 Analytical Approach

Now we adopt a conservative approach for recomputing the $\epsilon$ at each vertex. This approach underestimates the values for the positive and negative $\epsilon$. In other words, it guarantees the envelope surfaces not to intersect, but it does not guarantee that the $\epsilon$ at each vertex is the largest permissible $\epsilon$. We next discuss this approach for the case of computing the positive $\epsilon$ for each vertex. Computation of negative $\epsilon$ 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 $2\epsilon$ instead of $\epsilon$ for defining the envelope vertices. Next, consider all triangles $\Delta_i$ that do not share a vertex with $t$. If $\Delta_i$ 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 $\Delta_i$ that lies within $t_p$ and is closest to $t$. This point would be either a vertex of $\Delta_i$, 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 $\delta_i$ of this point from $t$. This is shown in Figure 5.

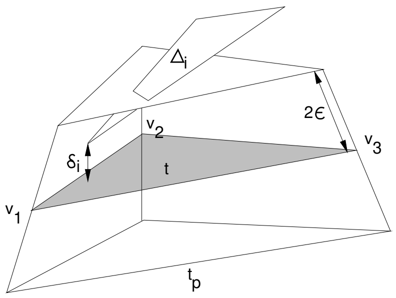



Figure 5: Computation of $\delta_i$

Once we have done this for all $\Delta_i$, we compute the new value of the positive $\epsilon$ 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 $\epsilon$, their positive envelope 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.

We use a dynamic octree in our implementation to speed up the identification of triangles $\Delta_i$ that intersect a given prism.

### 4.3 Numerical Computation

The above method sometimes causes certain envelope vertices to lie very close to the original surface even when they do not necessarily have to. This is because the offsets for the envelope vertices are computed one vertex at a time causing some vertices to have a very large offset at the expense of others. In addition, the analytical method requires intersection calculations involving bilinear patches, which are relatively hard to perform robustly.

Instead, we may chose to adopt an iterative approach that tends to maximize the minimum offset amongst the envelope vertices. Our main criteria in constructing such an approach are robustness, fairness to vertices, efficiency, and simplicity. In each iteration, we move each envelope vertex a fraction of the $\epsilon$ distance, checking to be sure this does not cause the envelope surfaces to intersect. Moves which cause no such intersections are accepted. We attempt to guarantee that each vertex gets to move at least $K$ steps.

The step size for each vertex is independently adapted to nearby geometry. Initially, all vertices are assigned a step size based on a global property: either some small percentage of the object's bounding box diagonal length or $\epsilon/K$, whichever is smaller. These initial step sizes may then be individually reduced to some fraction of a vertex's shortest incident edge (only those edges within $90^o$ of the offset direction are considered) if necessary. This makes it likely that each vertex's step size is appropriate for a step given the initial mesh configuration.

Now at each iteration, we attempt to move each vertex. If the move creates an intersection and the vertex has taken less than $K$ steps, we divide the step size for that vertex in half and try again, up to some maximum number of tries. This technique works fairly well in practice, though there may still be some room for improvement in generating maximal $\epsilon$'s for thin objects. As in the analytical approach, a simple octree data structure makes these intersection tests reasonably efficient, especially for models with evenly sized triangles. Figure 6 shows internal and external envelopes computed for three values of $\epsilon$ using this approach.

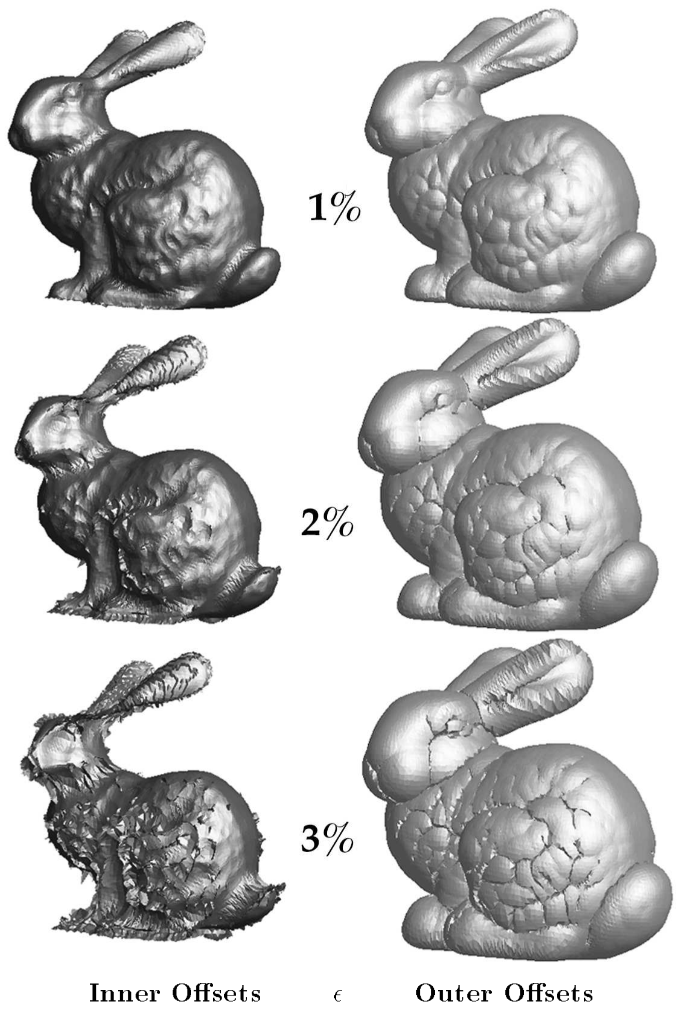



Figure 6: Simplification envelopes for various $\epsilon$

# 5 Generation of Approximation

Generating a surface approximation typically involves starting with the input surface and iteratively making modifications to ultimately reduce its complexity. This process may be broken into two main stages: *hole creation*, and *hole filling*.

We demonstrate the generality of the simplification envelope approach by designing two algorithms. The hole filling stages of these algorithms are quite similar, but their hole creation stages are quite different. The first algorithm creates maximally sized holes by considering global information about the surface, while the second algorithm makes only local choices, creating relatively small holes. These design choices produce algorithms with very different properties.

We begin by describing the envelope validity test used to determine whether a candidate triangle is valid for inclusion in the approximation surface. We then proceed to the two example simplification algorithms.

### 5.1 Validity Test

Valid candidate triangles must lie between the two envelopes. Because we use only vertices from the original model, any approximation triangle lies at least partially within the envelope volume. So a validity test involves testing a candidate triangle for intersections with the envelopes. We can perform such tests efficiently using a space-partitioning data structure such as an octree.

### 5.2 Global Algorithm

This algorithm extends the algorithm presented in [Cla93] to non-convex surfaces. Our major contribution is the use of simplification envelopes to bound the error on a non-convex polygonal surface and the use of fundamental prisms to provide a generalized projection mechanism for testing for regions of multiple covering (overlaps).

We first generate all tuples, $(v_1,\ v_2)$, of vertex pairs. Let any such tuple define an edge $e$ and let the set of all such edges be $S_e$. We eliminate all such edges which intersect either of the two envelopes. 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 envelopes and discard those that intersect either of the two envelopes.

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. We 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 envelope 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 $\mathcal{P}$.

At this stage we have available to us all candidate triangles that lie between the two non self-intersecting envelopes and information about which candidate triangles cover which vertices. 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.

We can show that the above problem reduces to that of *set partitioning* with one additional triangle disjointness constraint. However, the set partition problem is known to be NP-complete [GJ79], 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 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.

The greedy heuristic in our setting translates to selecting the candidate triangle that covers the largest number of vertices of $\mathcal{P}$ in its interior. 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 $t_j$ and the new triangles do not *overlap* each other and the triangles in $M_{i-1} - T_j$. The process for checking for overlaps is outlined in Section 5.4. If such a valid 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 $M_{i-1}$. This is repeated until all valid candidate triangles have been tried.

It has been shown in the literature [Lov75] that the greedy heuristic yields a solution 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 $t_j$, 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 $\alpha$ of the largest cost, then the worst case ratio guarantee of the approximate solution to that of the optimal becomes $\alpha(1+\lg d)$. 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 $\alpha$ in the above expression. Of course, if $\alpha > d/(1 + \lg d)$, the whole exercise is debatable, since the maximum compression we can ever hope to achieve is no more than $d$.

Figure 7 shows a two-dimensional example of a curve for which a local vertex removal algorithm might fail, but an algorithm that globally searches the solution space will succeed in finding a valid approximation.

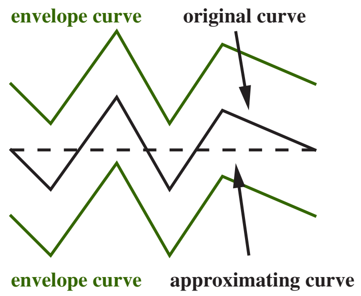



Figure 7: Curve at local minimum of approximation

### 5.3 Local Algorithm

The global strategy outlined above is theoretically elegant, allows comparison of the quality of the solution with respect to the optimal, and works well for small models. However, its complexity rises at least quadratically, making it prohibitive for large models.

To handle large models within the framework of simplification envelopes we propose a vertex-removal-based local algorithm. This algorithm draws heavily on the work of [SZL92], [Tur92], and [HDD+93]. Its main contributions are the use of envelopes to provide global error bounds as well as topology preservation and non-self-intersection. We have also explored the use of a more exhaustive hole-filling approach than any previous work we have seen.

The local algorithm begins by placing all vertices in a queue for removal processing. For each vertex in the queue, we attempt to remove it by creating a hole (removing its adjacent triangles) and attempting to fill it. If we can successfully fill the hole, the mesh modification is accepted, the vertex is removed from the queue, and its neighbors are placed back in the queue. If not, the vertex is removed from the queue and the mesh remains unchanged. This process is convergent because the global error bounds eventually prevent the removal of any more vertices. Once the vertex queue is empty we have our simplified mesh.

For a given vertex, we first create a hole by removing all adjacent triangles. We begin the hole-filling process by generating all possible triangles formed by combinations of the vertices on the hole boundary. This is not strictly necessary, but it allows us to use a greedy strategy to favor triangles with nice aspect ratios. We fill the hole by choosing a triangle, testing its validity, and recursively filling the three (or fewer) smaller holes created by adding that triangle into the hole (see figure 8). Our validity test includes not only the envelope intersection test, but also a surface intersection test to ensure that the new triangle doesn't intersect any of the other triangles in the proposed surface mesh. If a hole cannot be filled at any level of the recursion, the entire hole filling attempt is considered a failure.

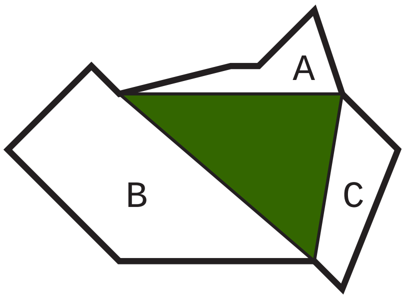



Figure 8: Hole filling: adding a triangle into a hole creates up to three smaller holes

This sort of hole-filling heuristic is employed by most simplification algorithms. Such heuristics are justified, because hole filling in three dimensions is NP-Hard [EBD95]. The number of possible ways to fill a three-dimensional hole is bounded above by the Catalan sequence of numbers ($C_n = (1/(n+1)) * (2n!/(n!n!))$), which represent the number of ways to triangulate a convex polygon in two dimensions.

We have experimented with a more exhaustive hole-filling technique which explores multiple paths down the recursion tree of possible hole tesselations. If we allow the hole-filling algorithm to traverse up to a few hundred paths, it will have the opportunity to exhaustively search for solutions to holes of size eight or nine. It can be shown that the average hole size for surfaces of genus $G$ is bounded above by $2G+6$, so most holes will be searched exhaustively for objects with one or zero topological holes.

To make this exhaustive hole-filling approach more practical, we first perform simplification using the typical one-path heuristic. Then we re-enqueue all the remaining vertices and perform another pass using the more exhaustive approach.

For simplifications resulting in the removal of hundreds of vertices (like highly oversampled laser-scanned models), the exhaustive pass yielded only about a 5% improvement over the one-path heuristic alone. This sort of confirmation reassures us that the typical heuristic works well. However, for more difficult simplifications, which may result in the removal of fewer vertices, every vertex removed is significant, and an exhaustive pass naturally provides an important improvement over the typical one-path heuristic.

### 5.4 Triangle Overlap Checking

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 onto the fundamental triangle. Third, we check if the projections of the two triangles on the fundamental triangle overlap using two-dimensional linear programming. Figure 9 illustrates the overlapping projections on a fundamental triangle. This process is important in avoiding unnecessary *creases* in the output approximation surface where none existed in the input.

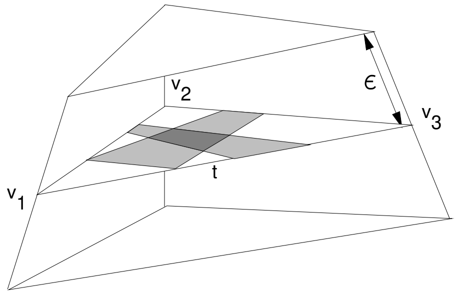



Figure 9: Checking for overlaps

# 6 Bordered Surfaces and Holes

The above algorithm will guarantee our global error bound for closed triangle meshes, but not for meshes with borders. While the envelopes constrain our error with respect to the normal direction of the surface, bordered surfaces require some additional constraints to hold the approximation border close to the original border. Without such constraints, the border of the approximation surface may "creep in," possibly shrinking the surface out of existence.

We construct a set of border tubes to constrain this error. Each border is a cyclic polyline. Intuitively, a border tube is a smooth, non-self-intersecting surface around one of these polylines. Removing a border vertex causes a pair of border edges to be replaced by a single border edge. If this new border edge lies within a border tube, we may safely attempt to remove the border vertex.

To construct a tube we define a plane passing through each vertex of the polyline. We choose a coordinate system on this plane and use that to define a circular set of vertices. We connect these vertices for consecutive planes to construct our tube. Our initial tubes are narrow surfaces, which we then expand using the same technique we used previously to construct our simplification envelopes. Choosing the coordinate systems at each vertex to encourage non-self-intersecting tubes is non-trivial, but we have a method based on projecting coordinate systems from vertex to vertex that works well.

# 7 Preserving Sharp Edges

One of the important properties in any approximation scheme is the way it preserves any sharp edges or normal discontinuities present in the input model. Simplification envelopes deal gracefully with sharp edges (those with a small angle between their adjacent faces). When the $\epsilon$ tolerance is small, there is not enough room to simplify across these sharp edges, so they are automatically preserved. As the tolerance is increased, it will eventually be possible to simplify across the edges (which should no longer be visible from the appropriate distance). Notice that it is not necessary to explicitly recognize these sharp edges.

Across certain edges, some models may contain artificial normal discontinuities which are not inherent to the local geometry. Both the global and local simplification algorithms may be modified to preserve such discontinuities constraining them to maintain these edges throughout the simplification process.

# 8 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 more detail near the observer and less detail further away. A possible solution could be to subdivide the model into various spatial cells and use a different $\epsilon$-approximation for each cell. However, problems would arise at the boundaries of such cells where the $\epsilon$-approximation for one cell, say at a value $\epsilon_1$ need not necessarily be continuous with the $\epsilon$-approximation for the neighboring cell, say at a different value $\epsilon_2$.

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

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 $\epsilon$ associated with a region to increase or decrease the level of approximation.

# 9 Implementation and Results

We have implemented both algorithms and tried out the local algorithm on several thousand objects. We will first discuss some of the implementataion issues and then present some results.

## 9.1 Implementation Issues

The first important implementation issue is what sort of input model to accept. We chose to accept only manifold triangle meshes (or bordered manifolds). This means that each edge is adjacent to two (one in the case of a border) triangles and that each vertex is surrounded by a single ring of triangles.

In addition, we currently accept only *clean* meshes. Many mesh degeneracies are not apparent on casual inspection, so we have implemented an automatic degeneracy detection program. This program detects non-manifold vertices, non-manifold edges, sliver triangles, coincident triangles, T-junctions, and intersecting triangles in a proposed input mesh. Note that correcting these degeneracies is more difficult than detecting them.

Robustness issues are important for implementations of any geometric algorithms. For instance, the analytical method for envelope computation involves the use of bilinear patches and the computation of intersection points. The computation of intersection points, even for linear elements, is difficult to perform robustly. The numerical method for envelope computation is much more robust because it involves only linear elements. Furthermore, it requires an intersection test but not the calculation of intersection points. We perform all such intersection tests in a conservative manner, using fuzzy intersection tests that may report intersections even for some close but non-intersecting elements.

Another important issue is the use of a space partitioning scheme to speed up intersection tests. We chose to use an octree because of its simplicity. Our current octree implementation deals only with the bounding boxes of the elements stored. This works well for models with triangles that are evenly sized and shaped. For CAD models, which may contain long, skinny, non-axis-aligned triangles, a simple octree does not always provide enough of a speed-up, and it may be necessary to choose a more appropriate space partitioning scheme.

## 9.2 Results

We have simplified a total of 2633 objects from the auxiliary machine room (AMR) of a notional submarine dataset for testing and validating our algorithm. We reproduce the timings and simplifications achieved by our implementation for the AMR and a few other models in Table 1. All simplifications were performed on a Hewlett-Packard 735/125 with 80 MB of main memory. Images of these simplifications appear in Figures 10 and 11. It is interesting to compare the results on the bunny and phone models with those of [DLW93, EDD+95]. For the same error bound, we are able to obtain improved solutions which are closer to the optimal solution.

| Bunny | | | Phone | | | Rotor | | | AMR | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\epsilon$ % | # Polys | Time | $\epsilon$ % | # Polys | Time | $\epsilon$ % | # Polys | Time | $\epsilon$ % | # Polys | Time |
| 0 | 69,451 | N/A | 0 | 165,936 | N/A | 0 | 4,736 | N/A | 0 | 428,554 | N/A |
| 1/16 | 10,793 | 641.2s | 1/32 | 12,364 | 2,088.9s | 1/4 | 1,992 | 168.3s | 1 | 217,058 | 9,217.2s |
| 1/4 | 2,204 | 671.5s | 1/16 | 4,891 | 2,308.3s | 3/4 | 1,774 | 152.5s | 3 | 171,382 | 6,057.4s |
| 1 | 575 | 659.3s | 1 | 412 | 1,811.2s | 3 3/4 | 1,418 | 242.6s | 15 | 120,174 | 9,526.8s |

Table 1: Simplification $\epsilon$'s and run times

We have automated the process which sets the $\epsilon$ value for each object by assigning it to be a percentage of the diagonal of its bounding box. We obtained the reductions presented in Table 1 and Figures 10 and 11 by using this heuristic.

For the rotor and AMR models in the above results, the $i^{th}$ level of detail was obtained by simplifying the $i-1^{th}$ level of detail. This causes to total $\epsilon$ to be the sum of all previous $\epsilon$'s, so choosing $\epsilon's$ of 1, 2, 4, and 8 percent results in total $\epsilon's$ of 1, 3, 7, and 15 percent. There are two advantages to this scheme:
(a) It allows one to proceed incrementally, taking advantage of the work done in previous simplifications.
(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.

One of the advantages of the setting $\epsilon$ to a percent of the object size is that it provides an a way to automate the selection of switching points used to transition between the various representations. In an attempt to eliminate visual artifacts, we derive the maximum percentage of the screen that should be covered by the bounding volume of a given level-of-detail, and switch to a more faithful representation when this percentage is exceeded. This maximum percentage is a function of the $\epsilon$ for that approximation, the output display resolution, and the corresponding maximum tolerable visible error in pixels.

# 10 Future Work

## 10.1 Generating Correspondences

A true geometric hierarchy should contain not only representations of an object at various levels of detail, but also some correspondence information about the relationship between adjacent levels. These relationships are necessary for propogating local information from one level to the next. For instance, this information would be helpful for using the hierarchical geometric representation to perform radiosity calculations. It is also necessary for performing geometric interpolation between the models when using the levels of detail for rendering. Note that the envelope technique preserves silhouettes when rendering, so it is also a good candidate for alpha blending rather than geometric interpolation to smooth out transitions between levels of detail.

We can determine which elements of a higher level of detail surface are covered by an element of a lower level of detail representation by noting which fundamental prisms this element intersects. This is non-trivial only because of the bilinear patches that are the sides of a fundamental prism. We can approximate these patches by two or more triangles and then tetrahedralize each prism. Given this tetrahedralization of the envelope volume, it is possible to stab each edge of the lower level-of-detail model through the tetrahedrons to determine which ones they intersect, and thus which triangles are covered by each lower level-of-detail triangle.

### 10.2 Moving Vertices

The output mesh generated by either of the algorithms we have presented has the property that its set of vertices is a subset of the set of vertices of the original mesh. If we can afford to relax this constraint somewhat, we may be able to reduce the output size even further. If we allow the vertices to slide along their normal vectors, we should be able to simplify parts of the surface that might otherwise be impossible to simplify for some choices of epsilon. We are currently working on a goal-based approach to moving vertices within the envelope volume. For each vertex we want to remove, we slide its neighboring vertices along their normals to make them lie as closely as possible to a tangent plane of the original vertex. Intuitively, this should increase the likelihood of successfully removing the vertex. During this whole process, we must ensure that none of the neighboring triangles ever violates the envelopes. This approach should make it possible to simplify surfaces using smaller epsilons than previously possible. In fact, it may even enable us to use the original surface and a single envelope as our constraint surfaces rather than two envelopes. This is important for objects with areas of high maximal curvature, like thin cylinders.

## 11 Conclusion

We have outlined the notion of simplification envelopes and how they can be used for generation of multiresolution hierarchies for polygonal objects. Our approach guarantees non-self-intersecting approximations and allows the user to do adaptive approximations by simply editing the simplification envelopes (either manually or automatically) in the regions of interest. It allows for a global error tolerance, preservation of the input genus of the object, and preservation of sharp edges. Our approach requires only one user-specifiable parameter, allowing it to work on large collections of objects with no manual intervention if so desired. It is rotationally and translationally invariant, and can elegantly handle holes and bordered surfaces through the use of cylindrical tubes. Simplification envelopes are general enough to permit both simplification algorithms with good theoretical properties such as our global algorithm, as well as fast, practical, and robust implementations like our local algorithm. Additionally, envelopes permit easy generation of correspondences across several levels of detail.

## 12 Acknowledgements

Thanks to Greg Angeline, Jim Boudreaux, and Ken Fast at Electric Boat for the submarine model, Rich Riesenfeld and Elaine Cohen of the Alpha_1 group at the University of Utah for the rotor model, and the Stanford Computer Graphics Laboratory for the bunny and telephone models.Thanks to Carl Mueller, Marc Olano, and Bill Yakowenko for many useful suggestions, and to the rest of the UNC Simplification Group (Rui Bastos, Carl Erikson, Merlin Hughes, and David Luebke) for providing a great forum for discussing ideas. The funding for this work was provide by a Link Foundation Fellowship, Alfred P. Sloan Foundation Fellowship, ARO Contract P-34982-MA, NSF Grant CCR-9319957, ONR Contract N00014-94-1-0738, ARPA Contract DABT63-93-C-0048 and NSF/ARPA Center for Computer Graphics and Scientific Visualization, NIH Grant RR02170, and NSF Career Award CCR-9502239.

## References

[AS94] P. Agarwal and S. Suri. Surface approximation and geometric partitions. In *Proceedings Fifth Symposium on Discrete Algorithms*, pages 24–33, 1994.

[BG94] 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.

[Cla93] K. L. Clarkson. Algorithms for polytope covering and approximation. In *Proc. 3rd Workshop Algorithms Data Struct.*, Lecture Notes in Computer Science, 1993.

[CS81] 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.

[DJ90] G. Das and D. Joseph. The complexity of minimum convex nested polyhedra. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 296–301, 1990.

[DLW93] 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.

[DZ91] M. J. DeHaemer, Jr. and M. J. Zyda. Simplification of objects rendered by polygonal approximations. *Computers & Graphics*, 15(2):175–184, 1991.

[EBD95] D. Eppstein, G. Barequet, and M. Dickerson. On triangulating three-dimensional polygons. manuscript, 1995.

[EDD+95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. *Computer Graphics: Proceedings of SIGGRAPH'95*, pages 173–182, 1995.

[FS93] 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.

[GJ79] 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.

[GKM93] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. In *Computer Graphics: Proceedings of SIGGRAPH 1993*, pages 231–238. ACM SIGGRAPH, 1993.

[HDD+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.

[HG94] P. Heckbert and M. Garland. Multiresolution modeling for fast rendering. *Proceedings of Graphics Interface*, 1994.

[HH93] P. Hinker and C. Hansen. Geometric optimization. In Gregory M. Nielson and Dan Bergeron, editors, *Proceedings Visualization '93*, pages 189–195, October 1993.

[HHK+95] T. He, L. Hong, A. Kaufman, A. Varshney, and S. Wang. Voxel-based object simplification. In G. M. Nielson and D. Silver, editors, *IEEE Visualization '95 Proceedings*, pages 296–303, 1995.

[KT93] 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.

[Lov75] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[MS92] 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.

[RB93] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering. In *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, June–July 1993.

[RO95] Kevin J. Renze and J. H. Oliver. Generalized surface and volume decimation for unstructured tesselated domains. In *Proceedings of SIVE'95*, 1995.

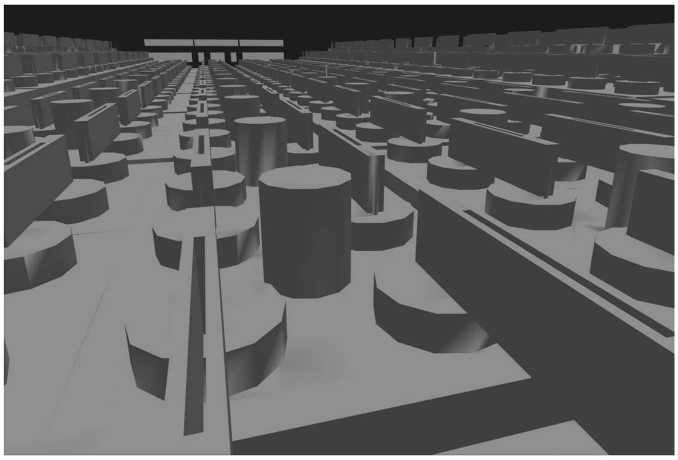
[RPV93] H. E. Rushmeier, C. Patterson, and A. Veerasamy. Geometric simplification for indirect illumination calculations. In *Proceedings Graphics Interface '93*, pages 227–236, 1993.

[SBD86] 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.

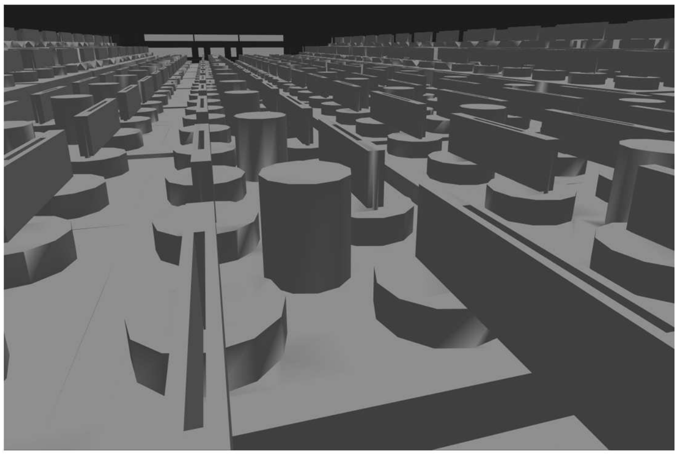
[SZL92] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.

[Tau95] G. Taubin. A signal processing approach to fair surface design. In *Proc. of ACM Siggraph*, pages 351–358, 1995.

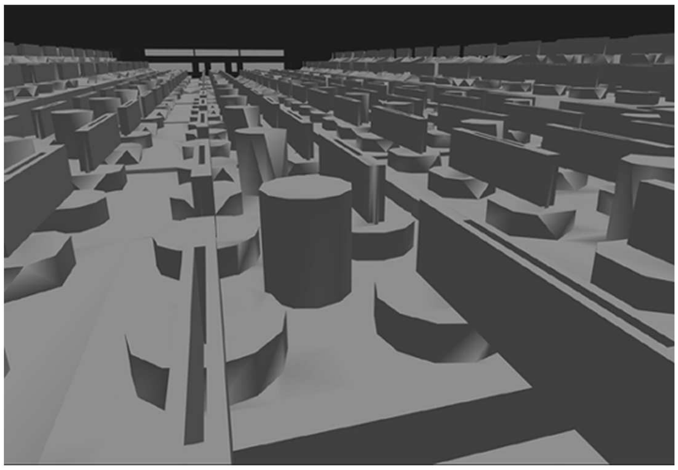
[Tur92] G. Turk. Re-tiling polygonal surfaces. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 55–64, July 1992.



**a)**



**b)**



**c)**

Figure 10: An array of batteries from a submarine's auxilliary machine room (AMR). As we increase the approximation error tolerated, the number of triangles in the entire AMR is correspondingly reduced from a) 428,554 to b) 217,058 to c) 120,174.

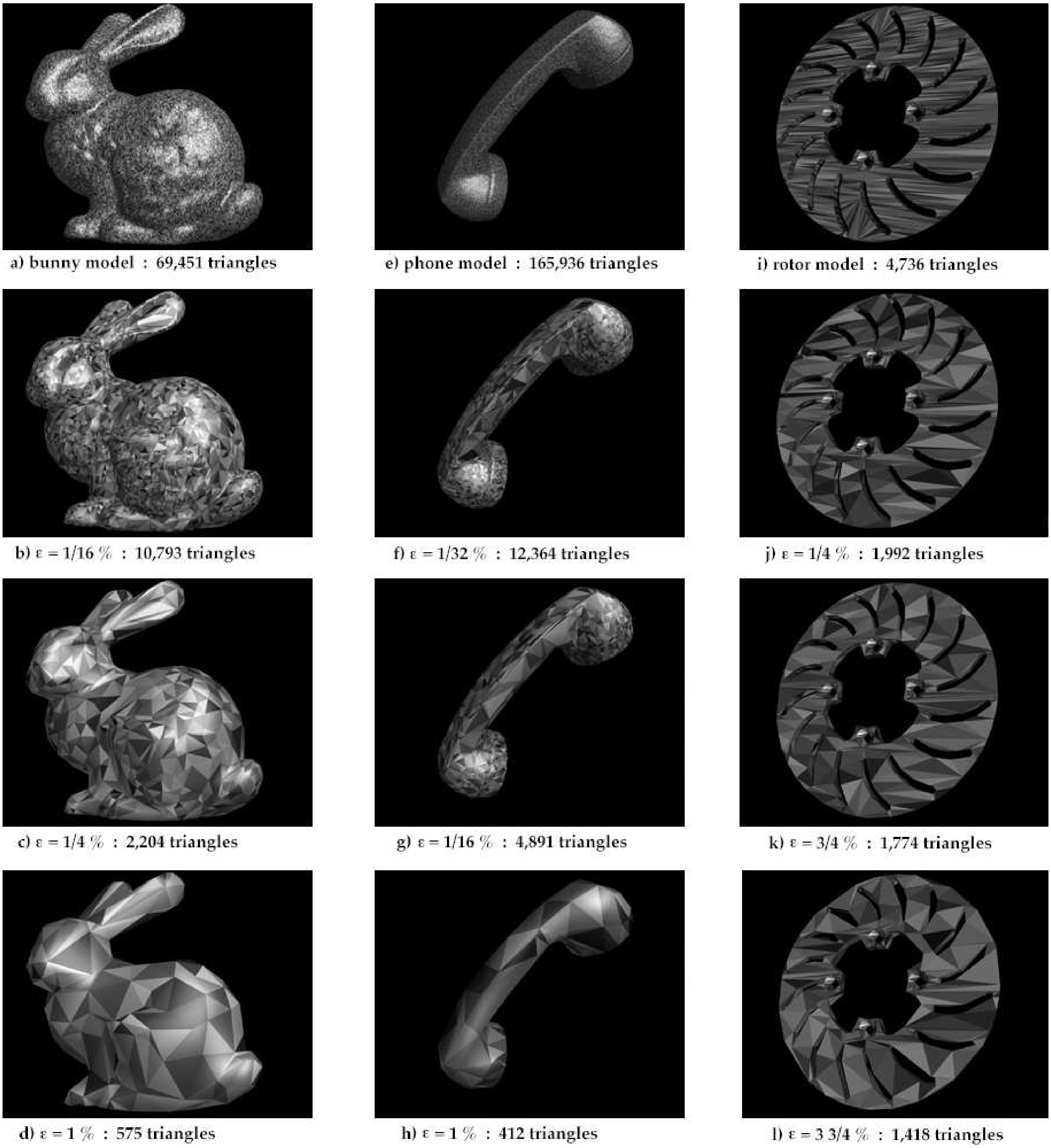



Figure 11: Level-of-detail hierarchies for three models. The spproximation distance, $\epsilon$, is taken as a percentage of the bounding box diagonal.