Simplification Envelopes

Jonathan Cohen[*] Amitabh Varshney[†] Dinesh Manocha[*] Greg Turk[*] Hans Weber[*]
Pankaj Agarwal[‡] Frederick Brooks[*] William Wright[*]
http://www.cs.unc.edu/˜geom/envelope.html

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 ϵ from the original model and that all points of the original model are within a distance ϵ 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 local, the other global. The local algorithm provides a fast method for generating approximations to large input meshes (at least hundreds of thousands of triangles). The global algorithm provides the opportunity to avoid local "minima" and possibly achieve better simplifications as a result.

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

- General technique providing guaranteed error bounds for genus-preserving simplification
- Automation of both the simplification process and the 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 [4, 9].
- 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 [10, 11, 13, 15, 21, 23].
- Using low-detail approximations of objects for illumination algorithms, especially radiosity [19].

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.

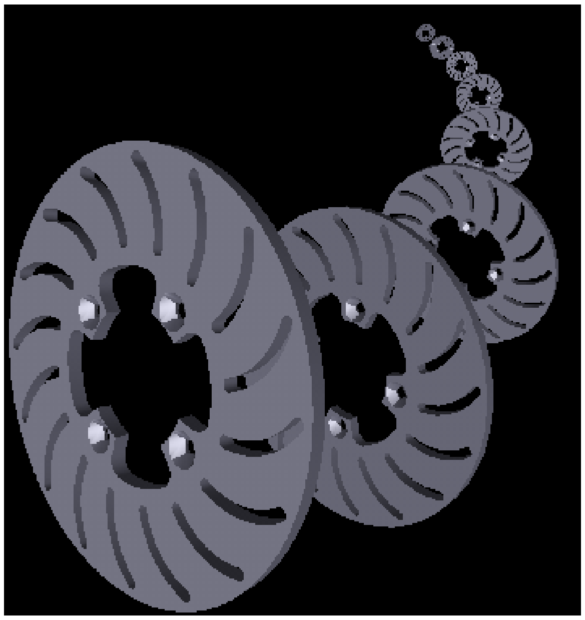


Figure 1: *A level-of-detail hierarchy for the rotor from a brake assembly.*

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 topology-preserving simplifications of a large number of objects. This problem of *scalability* is important for any simplification algorithm. One of our main goals is to create a method for simplification which is not only automatic for large datasets, but tends to preserve the shapes of the original models.

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 extentions to and properties of the approximation algorithms in Section 6, and explain our implementation and results in Section 7.

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 ϵ, the objective is to minimize the number of vertices such that no point of the approximation $\mathcal{A}$ is farther than ϵ distance away from the input model $\mathcal{I}$.

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

Previous work in the area of min-# approximations has been done by [6, 20] 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 [23] 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. [21] and Hinker and Hansen [13] 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 [18] 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. [14] 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 [7, 8] 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 ϵ-approximation is NP-hard for both convex polytopes [5] and polyhedral terrains [1]. 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 [16] for computing an ϵ-approximation of size $O(k_o \log n)$ for convex polytopes. This has recently been improved by Clarkson in [3]; 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). In [2] 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, [1] present a polynomial-time algorithm that computes an ϵ-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, ϵ-approximation of arbitrary polygonal objects. Additionally, we can simplify bordered meshes and meshes with holes. In terms of direct comparison with the global topology preserving approach presented in [7, 8], for a given ϵ our algorithm has been *empirically* able to obtain "reduced" simplifications, which are much smaller in output size (as demonstrated in Section 7). The algorithm in [18] also guarantees a bound in terms of the Hausdorff metric. 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 $\mathcal{P}$ has been given to us. Our objective is to compute a piecewise-linear approximation $\mathcal{A}$ of $\mathcal{P}$. 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 $\mathcal{P}$. Our goal is to outline a method to generate two envelope surfaces surrounding $\mathcal{P}$ and demonstrate how the envelopes can be used to solve the following polygonal approximation problem. Given a polygonal representation $\mathcal{P}$ of an object and an approximation parameter ϵ, generate a genus-preserving ϵ-approximation $\mathcal{A}$ with minimal number of polygons such that the vertices of $\mathcal{A}$ are a subset of vertices of $\mathcal{P}$.

We 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 each vertex of $\mathcal{P}$ has a single normal vector, which must lie within 90° of the normal of each of its surrounding triangles. For the purpose of rendering, each vertex may have either a single normal or multiple normals. For the purpose of generating envelope surfaces, we shall compute a single vertex normal as a combination of the normals of the surrounding triangles.

The three-dimensional ϵ-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 **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 ϵ 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.

Let us refer to the triangles of the given polygonal representation $\mathcal{P}$ as the *fundamental triangles*. 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 half-spaces for edge e, say π_e^+ and π_e^- (see Fig 2(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 as the *edge half-space* β_e. Let the two half-spaces for the edge e in this case be β_e^+ and β_e^-. This is shown in Fig 2(b).

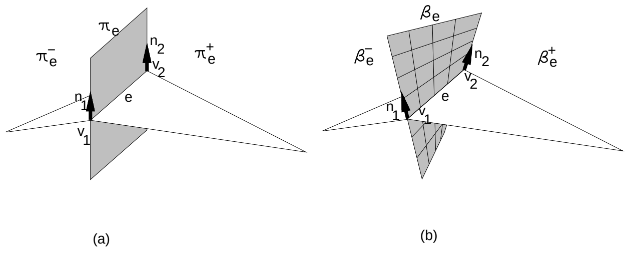


Figure 2: *Edge Half-spaces*

Let the vertices of a fundamental triangle 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 3.

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

4 Envelope Computation

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

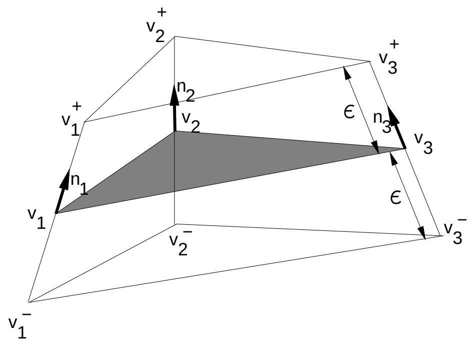


Figure 3: *The Fundamental Prism*

content at certain places with the distance between $\mathcal{P}$ and the envelope being smaller than ϵ. This is how simplification envelopes differ from offset surfaces.

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 ϵ, 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 4 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.

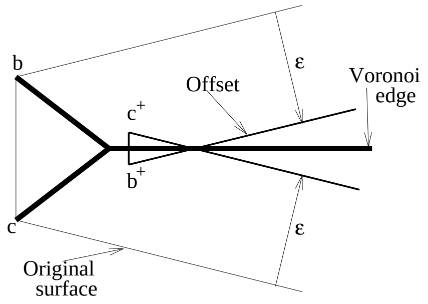


Figure 4: *Offset Surfaces*

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 the vertices v_i^+ and v_i^- determined with this new ϵ do not lie in the Voronoi regions of the non-adjacent fundamental triangles.

While this works in theory, efficient 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 ϵ for each vertex, the first method analytical, and the second numerical.

4.1 Analytical ϵ Computation

We adopt a conservative approach for recomputing the ϵ at each vertex. This approach underestimates the values for the positive and negative ϵ. In other words, it guarantees the envelope 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 envelope 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. 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 5.

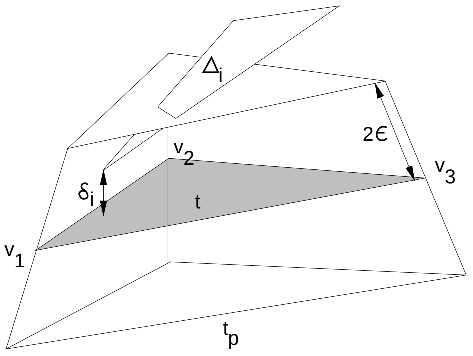


Figure 5: *Computation of* δ_i

Once we have done this for all Δ_i, we compute the new value of the positive ϵ for the triangle t as $\epsilon_{new} = \frac{1}{2}\min_i \delta_i$. If the vertices for this triangle t have this value of positive ϵ, their positive 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 an octree in our implementation to speed up the identification of triangles Δ_i that intersect a given prism.

4.2 Numerical ϵ Computation

To compute an envelope surface numerically, we take an iterative approach. Our envelope surface is initially identical to the input model surface. In each iteration, we sequentially attempt to move each envelope vertex a fraction of the ϵ distance in the direction of its normal vector (or the opposite direction, for the inner envelope). This effectively stretches or contracts all the triangles adjacent to the vertex. We test each of these adjacent triangles for intersection with each other and the rest of the model. If no such intersections are found, we accept the step, leaving the vertex in this new position. Otherwise we reject it, moving the vertex back to its previous position. The iteration terminates when all vertices have either moved ϵ or can no longer move.

In an attempt to guarantee that each vertex gets to move a reasonable amount of its potential distance, we use an adaptive step size. We encourage a vertex to move at least K (an arbitrary constant which is scaled with respect to ϵ and the size of the object) steps by allowing it to reduce its step size. If a vertex has moved less than K steps and its move is been rejected, it divides its step size in half and tries again (with some maximum number of divides allowed on any particular step). Notice that if a vertex moves i steps and is rejected on the $(i + 1)$st step, we know it has moved at least $i/(i + 1)$ % of its potential distance, so $K/(K + 1)$ which is a lower bound of sorts. It is possible, though rare, for a vertex to move less than K steps, if its current position is already quite close to another triangle.

Each vertex also has its own initial step size. We first choose a global, maximum step size based on a global property: either some small percentage of the object's bounding box diagonal length or ϵ/K, whichever is smaller. Now for each vertex, we calculate a local step size. This local step size is some percentage of the vertex's shortest incident edge (only those edges within 90^o of the offset direction are considered). We set the vertex's step size to the minimum of the global step size and its local step size. This makes it likely that each vertex's step size is appropriate for a step given the initial mesh configuration.

This approach to computing an envelope surface is robust, simple to implement (if difficult to explain), and fair to all the vertices. It tends to maximize the minimum offset distance amongst the envelope vertices. It works fairly well in practice, though there may still be some room for improvement in generating maximal offsets for thin objects. Figure 6 shows internal and external envelopes computed for three values of ϵ using this approach.

As in the analytical approach, a simple octree data structure makes these intersection tests reasonably efficient, especially for models with evenly sized triangles.

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 first create a hole by removing some connected set of triangles from the surface mesh. Then we fill the hole with a smaller set of triangles, resulting in some reduction of the mesh complexity.

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 makes only local choices, creating relatively small holes, while the second algorithm uses global information about the surface to create maximally-sized 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 and a description of their relative merits.

5.1 Validity Test

A *candidate triangle* is one which we are considering for inclusion in an approximation to the input surface. Valid candidate triangles must lie between the two envelopes. Because we construct candidate triangles from the vertices of the original model, we know its vertices lie between the two envelopes. Therefore, it is sufficient to test the candidate triangle for intersections with the two envelope

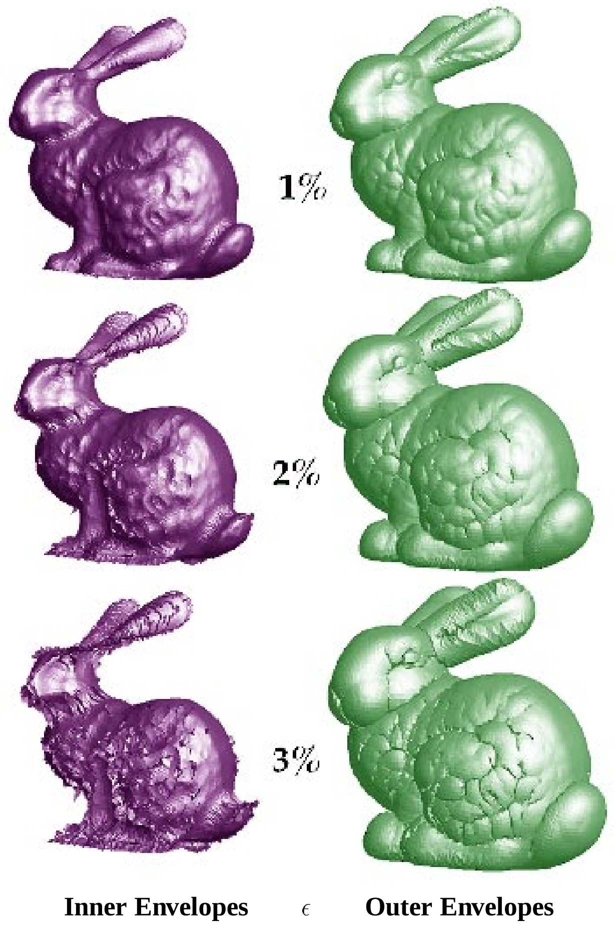


Figure 6: *Simplification envelopes for various* ϵ

surfaces. We can perform such tests efficiently using a space-partitioning data structure such as an octree.

A valid candidate triangle must also not cause a self-intersection in our surface, Therefore, it must not intersect any triangle of the current approximation surface.

5.2 Local Algorithm

To handle large models efficiently within the framework of simplification envelopes we construct a vertex-removal-based local algorithm. This algorithm draws heavily on the work of [21], [23], and [14]. 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 the vertex's 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 terminates when 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 7). If a hole cannot be filled at any level of the recursion, the entire hole filling attempt is considered a failure. Note that this is a single-pass hole filling strategy; we do not backtrack or undo selection of a triangle chosen for filling a hole. Thus, this approach does not guarantee that if a triangulation of a hole exists we will find it. However, it is quite fast and works very well in practice.

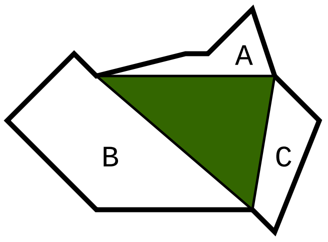


Figure 7: *Hole filling: adding a triangle into a hole creates up to three smaller holes*

We have compared the above approach with an exhaustive approach in which we tried all possible hole-filling triangulations. For simplifications resulting in the removal of hundreds of vertices (like highly oversampled laser-scanned models), the exhaustive pass yielded only a small improvement over the single-pass heuristic. This sort of confirmation reassures us that the single-pass heuristic works well in practice.

5.3 Global Algorithm

This algorithm extends the algorithm presented in [3] 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 present only a sketch of the algorithm here ; see [24] for the full details.

We begin by generating all possible candidate triangles for our approximation surface. These triangles are all 3-tuples of the input vertices which do not intersect either of the offset surfaces. Next we determine how many vertices each triangle *covers*. We rank the candidate triangles in order of decreasing covering.

We then choose from this list of candidate triangles in a greedy fashion. For each triangle we choose, we create a large hole in the current approximation surface, removing all triangles which *overlap* this candidate triangle. Now we begin the recursive hole-filling process by placing this candidate triangle into the hole and filling all the subholes with other triangles, if possible. One further restriction in this process is that the candidate triangle we are testing should not overlap any of the candidate triangles we have previously accepted.

5.4 Algorithm Comparison

The local simplification algorithm is fast and robust enough to be applied to large models. The global strategy is theoretically elegant. While the global algorithm works well for small models, its complexity rises at least quadratically,

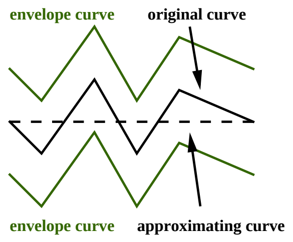


Figure 8: *Curve at local minimum of approximation*

making it prohibitive for larger models. We can think of the simplification problem as an optimization problem as well. A purely local algorithm may get trapped in a local "minimum" of simplification, while an ideal global algorithm will avoid all such minima.

Figure 8 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. Any of the interior vertices we remove would cause a new edge to penetrate an envelope curve. But if we remove all of the interior vertices, the resulting edge is perfectly acceptable.

This observation motivates a wide range of algorithms of which our local and global examples are the two extremes. We can easily imagine an algorithm that chooses nearby groups of vertices to remove simultaneously rather than sequentially. This could potentially lead to increased speed and simplification performance. However, choosing such sets of vertices remains a challenging problem.

6 Additional Features

Envelope surfaces used in conjunction with simplification algorithms are powerful, general-purpose tools. As we will now describe, they implicitly preserve sharp edges and can be extended to deal with bordered surfaces and perform adaptive approximations.

6.1 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 ϵ 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.

6.2 Bordered Surfaces

A bordered surface is one containing points that are homeomorphic to a half-disc. For polygonal models, this corresponds to edges that are adjacent to a single face rather than two faces. Depending on the context, we may naturally think of this as the boundary of some plane-like piece of a surface, or a hole in an otherwise closed surface.

The algorithms described in 5 are sufficient for closed triangle meshes, but they will not guarantee our global error bound 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.

In many cases, the complexity of a surface's border curves may become a limiting factor in how much we can simplify the surface, so it is unacceptable to forgo simplifying these borders.

We construct a set of border tubes to constrain the error in deviation of the border curves. Each border is actually a cyclic polyline. Intuitively speaking, 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 does not intersect the relevant 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 have a very narrow radius to minimize the likelihood of self-intersections. We then expand these narrow tubes using the same technique we used previously to construct our simplification envelopes.

The difficult task is to define a coordinate system at each polyline vertex which encourages smooth, non-self-intersecting tubes. The most obvious approach might be to use the idea of Frenet frames from differential geometry to define a set of coordinate systems for the polyline vertices. However, Frenet frames are meant for smooth curves. For a jagged polyline, a tube so constructed often has many self-intersections.

Instead, we use a projection method to minimize the twist between consecutive frames. Like the Frenet frame method, we place the plane at each vertex so that the normal to the plane approximates the tangent to the polyline. This is called the *normal plane*.

At the first vertex, we choose an arbitrary orthogonal pair of axes for our coordinate system in the normal plane. For subsequent vertices, we project the coordinate system from the previous normal plane onto the current normal frame. We then orthogonalize this projected coordinate system in the plane. For the normal plane of the final polyline vertex, we average the projected coordinate systems of the previous normal plane and the initial normal plane to minimize any twist in the final tube segment.

6.3 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 ϵ-approximation for each cell. However, problems would arise at the boundaries of such cells where the ϵ-approximation for one cell, say at a value ϵ_1 need not necessarily be continuous with the ϵ-approximation for the neighboring cell, say at a different value ϵ_2.

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

As another possibility, consider the case where certain features of a model are very important and are not to be approximated beyond a certain level. Such features might have human perception as a basis for their definition or they might have mathematical descriptions such as regions of high curvature. In either case, a user can vary the ϵ associated with a region to increase or decrease the level of approximation. The bunny in Figure 9 illustrates such an approximation.

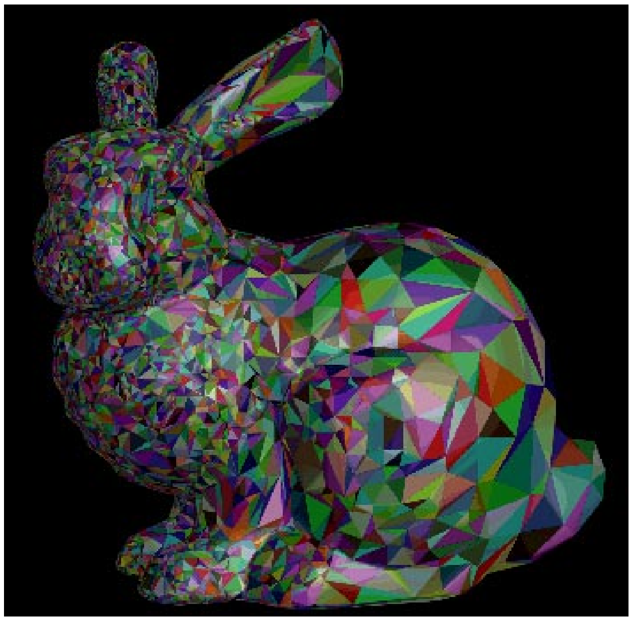


Figure 9: *An adaptive simplification for the bunny model. ϵ varies from 1/64% at the nose to 1% at the tail.*

7 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 implementation issues and then present some results.

7.1 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.

We also do not accept other forms of degenerate 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.

7.2 Results

We have simplified a total of 2636 objects from the auxiliary machine room (AMR) of a submarine dataset, pictured in Figure 10 to test and validate 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 11 and 12. It is interesting to compare the results on the bunny and phone models with those of [7, 8]. For the same error bound, we are able to obtain much improved solutions.

We have automated the process which sets the ϵ 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 for the AMR and Figures 11 and 12 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 ϵ to be the sum of all previous ϵ'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 ϵ 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. To eliminate visual artifacts, we switch to a more faithful representation of an object when ϵ projects to more than some user-specified number of pixels on the screen. This is a function of the ϵ for that approximation, the output display resolution, and the corresponding maximum tolerable visible error in pixels.

8 Future Work

There are still several areas to be explored in this research. We believe the most important of these to be the generation of correspondences between levels of detail and the moving of vertices within the envelope volume.

Bunny			Phone			Rotor			AMR		
ϵ %	# Polys	Time	ϵ %	# Polys	Time	ϵ %	# Polys	Time	ϵ %	# Polys	Time
0	69,451	N/A	0	165,936	N/A	0	4,735	N/A	0	436,402	N/A
1/64	44,621	9	1/64	43,537	31	1/8	2,146	3	1	195,446	171
1/32	23,581	10	1/32	12,364	35	1/4	1,514	2	3	143,728	61
1/16	10,793	11	1/16	4,891	38	3/4	1,266	2	7	110,090	61
1/8	4,838	11	1/8	2,201	32	1 3/4	850	1	15	87,476	68
1/4	2,204	11	1/4	1,032	35	3 3/4	716	1	31	75,434	84
1/2	1,004	11	1/2	544	33	7 3/4	688	1			
1	575	11	1	412	30	15 3/4	674	1			

Table 1: *Simplification ϵ's and run times in minutes*

8.1 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 propagating 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.

8.2 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.

9 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.

10 Acknowledgements

Thanks to Greg Angelini, 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, ARO MURI grant DAAH04-96-1-0013, NSF Grant CCR-9319957, NSF Grant CCR-9301259, NSF Career Award CCR-9502239, ONR Contract N00014-94-1-0738, ARPA Contract DABT63-93-C-0048, NSF/ARPA Center for Computer Graphics and Scientific Visualization, NIH Grant RR02170, an NYI award with matching funds from Xerox Corp, and a U.S.-Israeli Binational Science Foundation grant.

References

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

[2] 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.

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

[4] 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.

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

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

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

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

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

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

[20] 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.

[21] 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.

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

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

[24] A. Varshney. Hierarchical geometric approximations. Ph.D. Thesis TR-050-1994, Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599-3175, 1994.

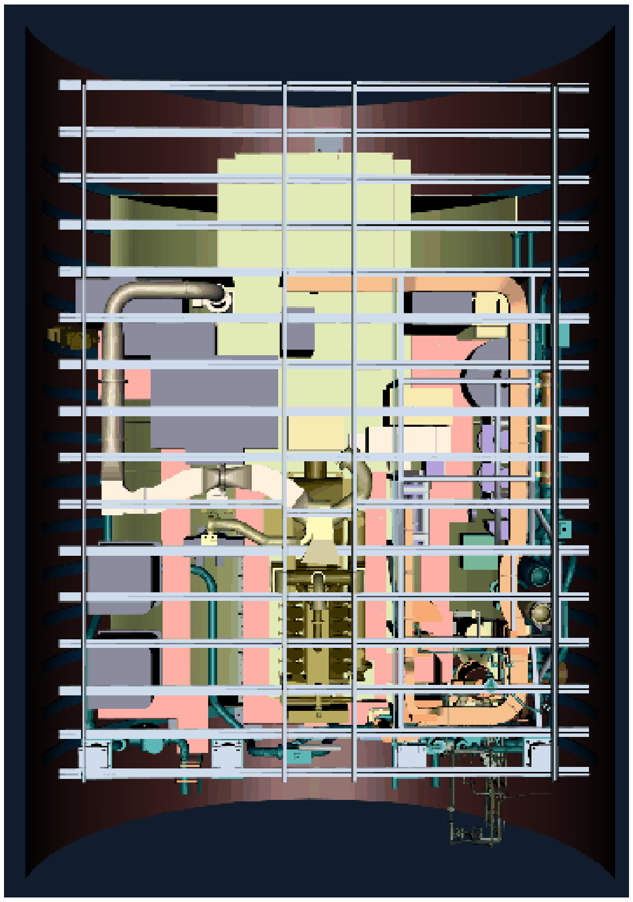


Figure 10: *Looking down into the auxiliary machine room (AMR) of a submarine model. This model contains nearly 3,000 objects, for a total of over half a million triangles. We have simplified over 2,600 of these objects, for a total of over 430,000 triangles.*

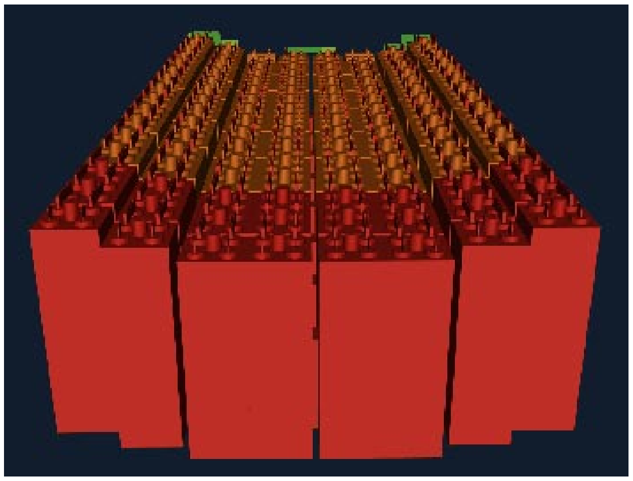
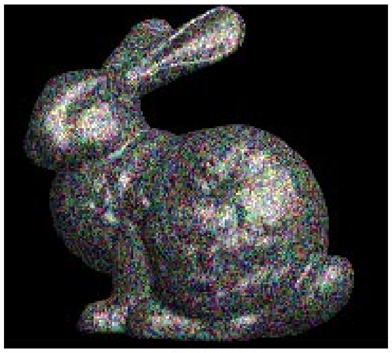
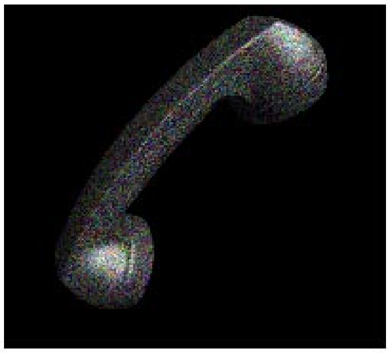
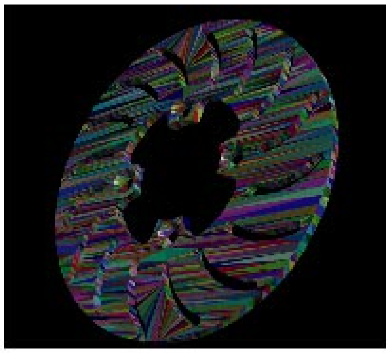
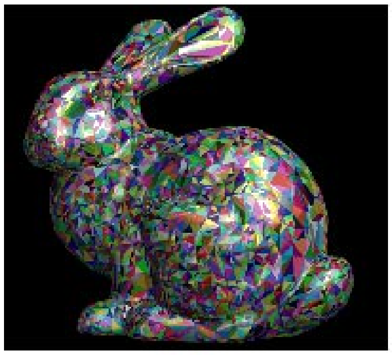


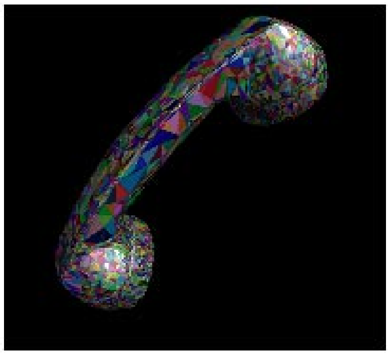
Figure 11: *An array of batteries from the AMR. All parts but the red are simplified representations. At full resolution, this array requires 87,000 triangles. At this distance, allowing 4 pixels of error in screen space, we have reduced it to 45,000 triangles.*

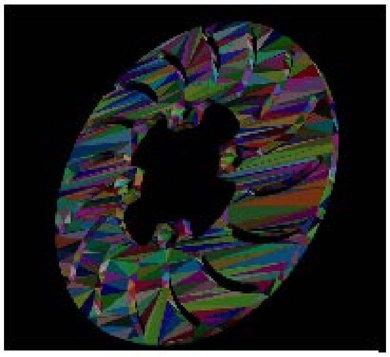

(a) bunny model: 69,451 triangles

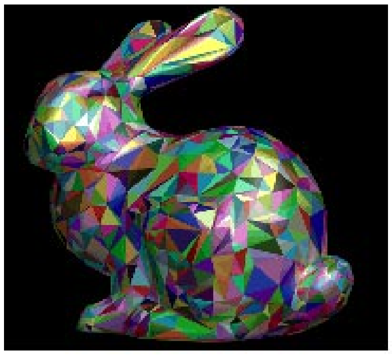

(b) phone model: 165,936 triangles

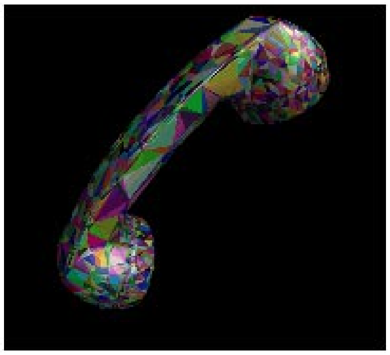

(c) rotor model: 4,736 triangles

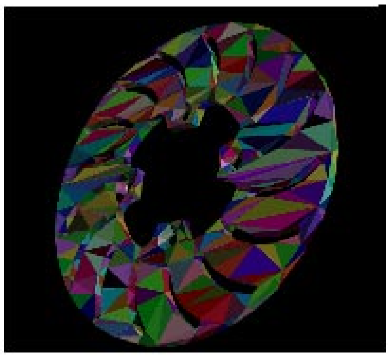

(a) $\epsilon = 1/16\%$, $10,793$ triangles

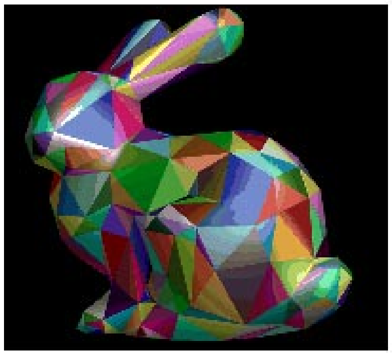

(b) $\epsilon = 1/32\%$, $12,364$ triangles

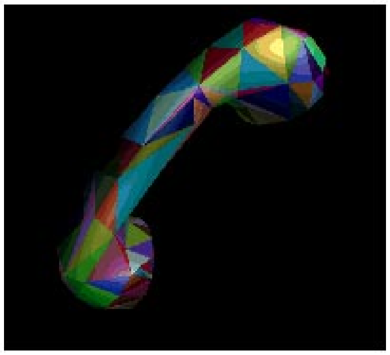

(c) $\epsilon = 1/8\%$, $2,146$ triangles


(a) $\epsilon = 1/4\%$, $2,204$ triangles


(b) $\epsilon = 1/16\%$, $4,891$ triangles


(c) $\epsilon = 3/4\%$, $1,266$ triangles


(a) $\epsilon = 1\%$, 575 triangles


(b) $\epsilon = 1\%$, 412 triangles

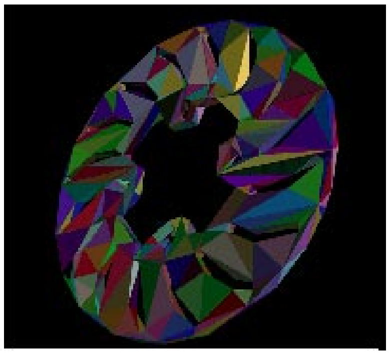

(c) $\epsilon = 3\ 3/4\%$, 716 triangles

Figure 12: *Level-of-detail hierarchies for three models. The approximation distance, ϵ, is taken as a percentage of the bounding box diagonal.*