Concepts and Algorithms for Polygonal Simplification

Jonathan D. Cohen

Department of Computer Science, The Johns Hopkins University

1. INTRODUCTION

1.1 Motivation

In 3D computer graphics, polygonal models are often used to represent individual objects and entire environments. Planar polygons, especially triangles, are used primarily because they are easy and efficient to render. Their simple geometry has enabled the development of custom graphics hardware, currently capable of rendering millions or even tens of millions of triangles per second. In recent years, such hardware has become available even for personal computers. Due to the availability of such rendering hardware and of software to generate polygonal models, polygons will continue to play an important role in 3D computer graphics for many years to come.

However, the simplicity of the triangle is not only its main advantage, but its main disadvantage as well. It takes many triangles to represent a smooth surface, and environments of tens or hundreds of millions of triangles or more are becoming quite common in the fields of industrial design and scientific visualization. For instance, in 1994, the UNC Department of Computer Science received a model of a notional submarine from the Electric Boat division
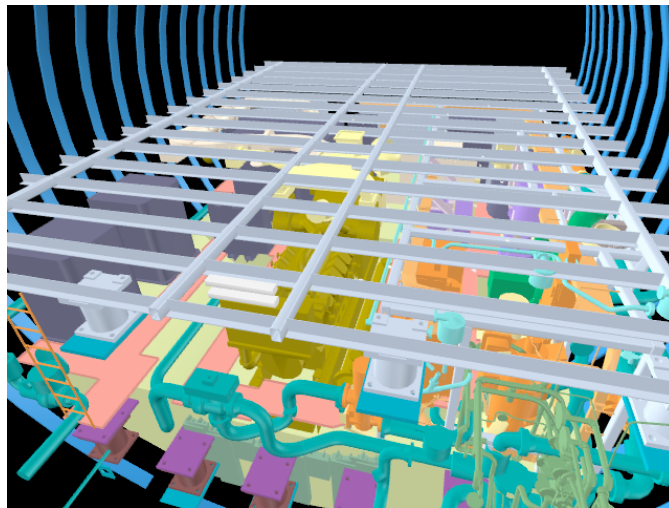

Figure 1: The auxiliary machine room of a notional submarine model: 250,000 triangles

of General Dynamics, including an auxiliary machine room composed of 250,000 triangles (see Figure 1) and a torpedo room composed of 800,000 triangles. In 1997, we received from ABB Engineering a coarsely-tessellated model of an entire coal-fired power plant, composed of over 13,000,000 triangles. It seems that the remarkable performance increases of 3D graphics hardware systems cannot yet match the desire and ability to generate detailed and realistic 3D polygonal models.

1.2 Polygonal Simplification

This imbalance of 3D rendering performance to 3D model size makes it difficult for graphics applications to achieve *interactive* frame rates (10-20 frames per second or more). Interactivity is an important property for applications such as architectural walkthrough, industrial design, scientific visualization, and virtual reality. To achieve this interactivity in spite of the enormity of data, it is often necessary to trade fidelity for speed.

We can enable this speed/fidelity tradeoff by creating a *multi-resolution* representation of our models. Given such a representation, we can render smaller or less important objects in the scene at a lower resolution (i.e. using fewer triangles) than the larger or more important objects, and thus we render fewer triangles overall. Figure 2 shows a widely-used test model: the Stanford bunny. This model was acquired using a laser range-scanning device; it contains over 69,000 triangles. When the 2D image of this model has a fairly large area, this may be a reasonable number of triangles to use for rendering the image. However, if the image is smaller, like Figure 3 or Figure 4, this number of triangles is probably too large. The right-most image in each of these figures shows a bunny with fewer triangles. These complexities are often more appropriate for image of these sizes. Each of these images is typically some small piece of a much larger image of a complex scene.

For CAD models, such representations could be created as part of the process of building the original model. Unfortunately, the robust modeling of 3D objects and environments is already a difficult task, so we would like to explore solutions that do not add extra burdens to the original modeling process. Also, we would like to create such representations for models acquired by other means (e.g. laser scanning), models that already exist, and models in the process of being built.
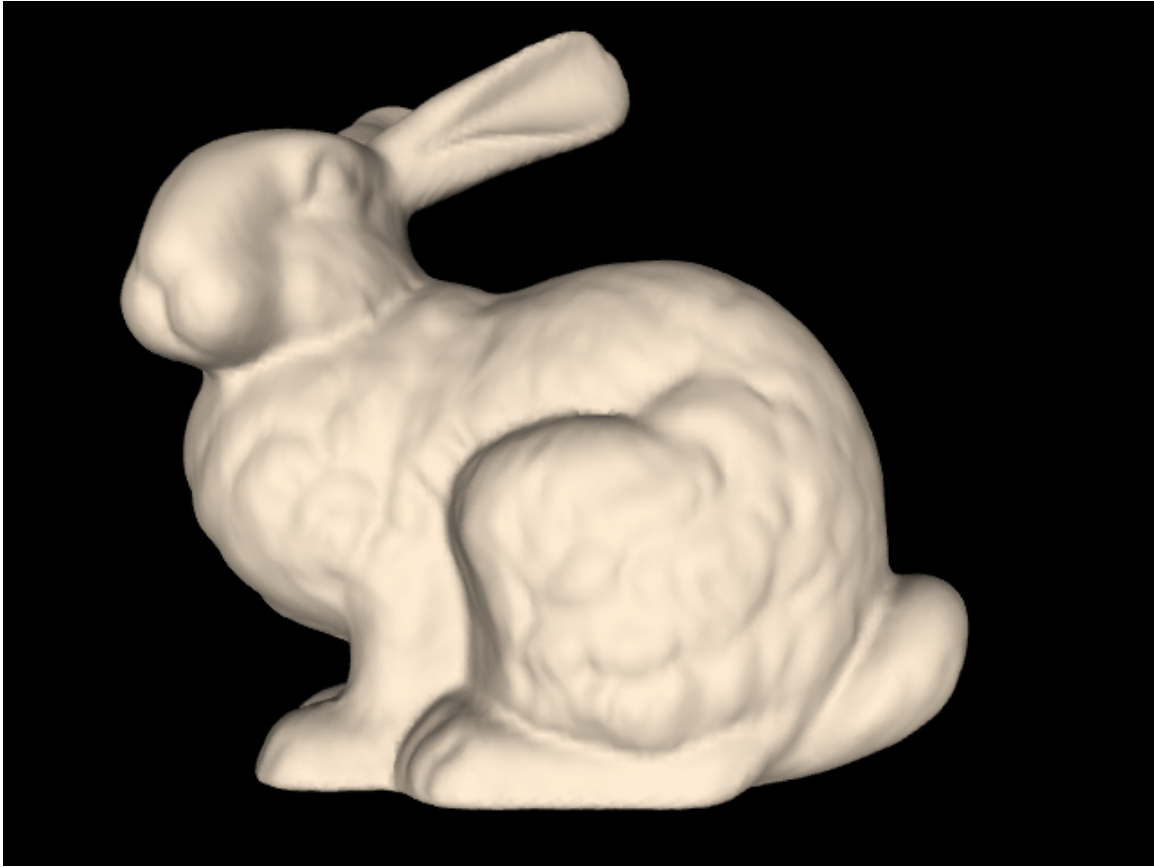
Figure 2: The Stanford bunny model: 69,451 triangles

69,451 triangles   2,204 triangles

Figure 3: Medium-sized bunnies.

69,451 triangles   575 triangles

Figure 4: Small-sized bunnies.

Simplification is the process of automatically reducing the complexity of a given model. By creating one or more simpler representations of the input model (generally called *levels of detail*), we convert it to a multi-resolution form. This problem of automatic simplification is rich enough to provide many interesting and useful avenues of research. There are many issues related to how we represent these multi-resolution models, how we create them, and how we manage them within an interactive graphics application. This dissertation is concerned primarily with the issues of level-of-detail quality and rendering performance. In particular, we explore the question of how to preserve the appearance of the input models to within an intuitive, user-specified tolerance and still achieve a significant increase in rendering performance.

1.3 Topics Covered

This paper reviews some fundamental concepts necessary to understand algorithms for simplification of polygonal models at a high level. These concepts include optimal/near-optimal solutions for the simplification problem, the use of local simplification operations, topology preservation, level-of-detail representations for polygonal models, error measures for surface deviation, and the preservation of appearance attributes. This is not a complete survey of the field of polygonal model simplification, which has grown to be quite large (for more information, several survey papers are available [Erikson 1996, Heckbert and Garland 1997]). In particular, this paper does *not* provide much coverage of algorithms specialized for simplifying polygonal terrains, nor does it cover simplification and compression algorithms geared towards progressive transmission applications.

2. OPTIMALITY

There are two common formulations of the simplification problem, described in [Varshney 1994], to which we may seek optimal solutions:

- **Min-# Problem**: Given some error bound, ε, and an input model, I, compute the minimum complexity approximation, A, such that no point of A is farther than ε distance away from I and vice versa (the complexity of A is measured in terms of number of vertices or faces).

- **Min-ε Problem**: Given some target complexity, *n*, and an input model, *I*, compute the approximation, *A*, with the minimum error, ε, described above.

In computational geometry, it has been shown that computing the min-# problem is NP-hard for both convex polytopes [Das and Joseph 1990] and polyhedral terrains [Agarwal and Suri 1994]. Thus, algorithms to solve these problems have evolved around finding polynomial-time approximations that are *close* to the optimal.

Let k_0 be the size of a min-# approximation. An algorithm has been given in [Mitchell and Suri 1992] for computing an ε-approximation of size $O(k_0 \log n)$ for convex polytopes of initial complexity *n*. This has been improved by Clarkson in [Clarkson 1993]; he proposes a randomized algorithm for computing an approximation of size $O(k_0 \log k_0)$ in expected time $O(k_0 n^{1+\delta})$ for any $\delta > 0$ (the constant of proportionality depends on δ, and tends to $+\infty$ as δ tends to 0). In [Brönnimann and Goodrich 1994] 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, [Agarwal and Suri 1994] present a polynomial-time algorithm that computes an ε-approximation of size $O(k_0 \log k_0)$ to a polyhedral terrain.

Because the surfaces requiring simplification may be quite complex (tens of thousands to millions of triangles), the simplification algorithms used in practice must be $o(n^2)$ (typically $O(n \log n)$) for the running time to be reasonable. Due to the difficulty of computing near-optimal solutions for general polygonal meshes and the required efficiency, most of the algorithms described in the computer graphics literature employ local, greedy heuristics to achieve what appear to be reasonably good simplifications with no guarantees with respect to the optimal solution.

3. LOCAL SIMPLIFICATION OPERATIONS

Simplification is often achieved by performing a series of local operations. Each such operation serves to coarsen the polygonal model by some small amount. A simplification algorithm generally chooses one of these operation types and applies it repeatedly to its input surface until the desired complexity is achieved for the output surface.
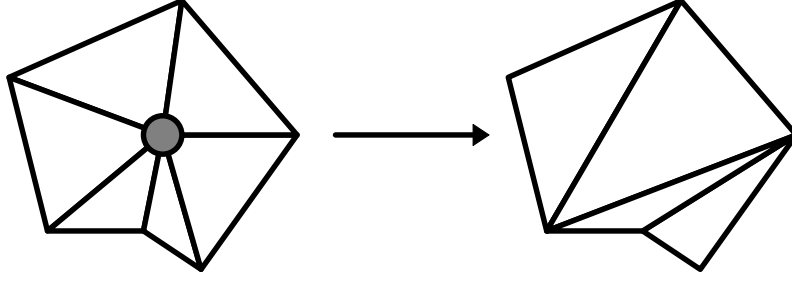
Figure 5: Vertex remove operation

3.1 Vertex Remove

The vertex remove operation involves removing from the surface mesh a single vertex and all the triangles touching it. This removal process creates a hole that we then fill with a new set of triangles. Given a vertex with *n* adjacent triangles, the removal process creates a hole with *n* sides. The hole filling problem involves a discrete choice from among a finite number of possible retriangulations for the hole. The *n* triangles around the vertex are replaced by this new triangulation with *n*-2 triangles. The Catalan sequence,

$$C(i) = \frac{1}{i+1} * \binom{2i}{i} = \frac{1}{i+1} * \frac{(2i)!}{i!(2i-i)!} = \frac{1}{i+1} * \frac{(2i)!}{i!i!} = \frac{(2i)!}{(i+1)!i!}, \tag{1}$$

describes the number of unique ways to triangulate a convex, planar polygon with *i*+2 sides [Dörrie 1965, Plouffe and Sloan 1995]. This provides an upper bound on the number of non-self-intersecting triangulations of a hole in 3D. For example, holes with 3 sides have only 1 triangulation, and holes with 4, 5, 6, 7, 8, and 9 sides have up to 2, 5, 14, 42, 132, and 429 triangulations, respectively.

Both [Turk 1992] and [Schroeder et al. 1992] apply the vertex remove approach as part of their simplification algorithms. Turk uses point repulsion (weighted according to curvature) to distribute some number of new vertices across the original surface, then applies vertex remove operations to remove most of the original vertices. Holes are retriangulated using a planar projection approach. Schroeder also uses vertex remove operations to reduce mesh complexity, employing a recursive loop splitting algorithm to fill the necessary holes.
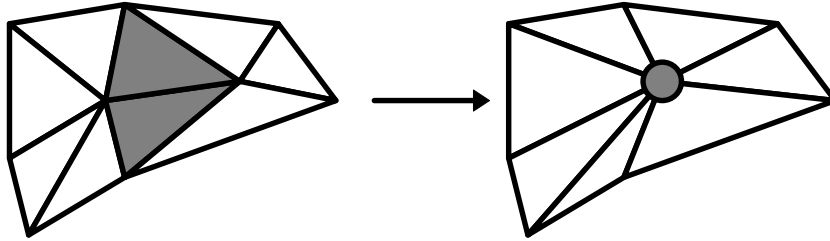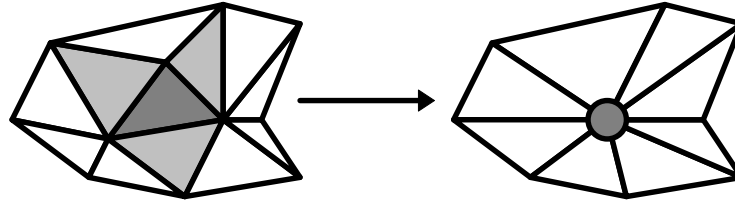
Figure 6: Edge collapse operation

3.2 Edge Collapse

The edge collapse operation has become popular in the graphics community in the last several years. The two vertices of an edge are merged into a single vertex. This process distorts all the neighboring triangles. The triangles that contain both of the vertices (i.e. those that touch the entire edge) degenerate into 1-dimensional edges and are removed from the mesh. This typically reduces the mesh complexity by 2 triangles.

Whereas the vertex remove operation amounts to making a discrete choice of triangulations, the edge collapse operation requires us to choose the coordinates of the new vertex from a continuous domain. Common choices for these new coordinates include the coordinates of one of the two original vertices, the midpoint of the collapsed edge, arbitrary points along the collapsed edge, or arbitrary points in the neighborhood of the collapsed edge.

Not only is the choice of new vertex coordinates for the edge collapse a continuous problem, but the actual edge collapse operation may be performed continuously in time. We can linearly interpolate the two vertices from their original positions to the final position of the new vertex. This allows us to create smooth transitions as we change the mesh complexity. As described in [Hoppe 1996], we can even perform *geomorphs*, which smoothly transition between versions of the model with widely varying complexity by performing many of these interpolations simultaneously.

In terms of the ability to create identical simplifications, the vertex removal and edge collapse operations are not equivalent. If we collapse an edge to one of its original vertices, we can create n of the triangulations possible with the vertex remove, but there are still $C(n+2)$-n triangulations that the edge collapse cannot create. Of course, if we allow the edge collapse to choose arbitrary coordinates for its new vertex, it can create infinitely many simplifications that the vertex remove operation cannot create. For a given input model and

Figure 7: Face collapse operation

desired output complexity, it is not clear which type of operation can achieve a closer approximation to the input model.

The edge collapse was used by [Hoppe et al. 1993] as part of a mesh optimization process that employed the vertex remove and edge swap operations as well (the edge swap is a discrete operation that takes two triangles sharing an edge and swaps which pair of opposite vertices are connected by the edge). In [Hoppe 1996], the vertex remove and edge swaps are discarded, and the edge collapse alone is chosen as the simplification operation, allowing a simpler system that can take advantage of the features of the edge collapse. Although systems employing multiple simplification operations might possibly result in better simplifications, they are generally more complex and cannot typically take advantage of the inherent features of any one operation.

3.3 Face Collapse

The face collapse operation is similar to the edge collapse operation, except that it is more coarse-grained. All three vertices of a triangular face are merged into a single vertex. This causes the original face to degenerate into a point and three adjacent faces to degenerate into line segments, removing a total of four triangles from the model. The coarser granularity of this operation may allow the simplification process to proceed more quickly, at the expense of the fine-grained local control of the edge collapse operation. Thus, the error is likely to accumulate more quickly for a comparable reduction in complexity. [Hamann 1994, Gieng et al. 1997] use the face collapse operation in their simplification systems. The new vertex coordinates are chosen to lie on a local quadratic approximation to the mesh. Naturally, it is possibly to further generalize these collapse operations to collapse even larger connected portions of the input model. It may even be possible to reduce storage requirements by grouping nearby collapse operations with similar error bounds into larger collapse operations.
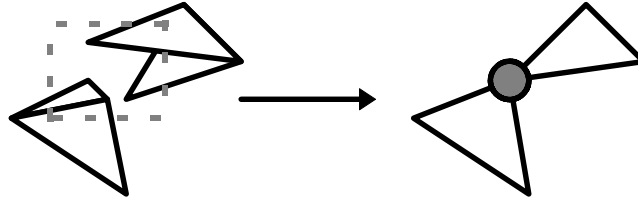
Figure 8: Vertex Cluster operation

Thus, the fine-grained control may be traded for reduced storage and other overhead requirements in certain regions of the model.
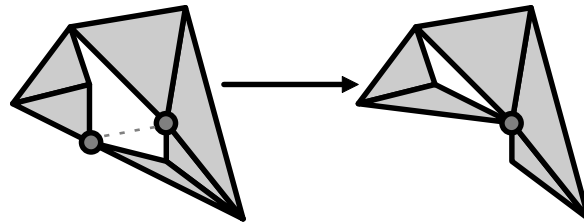
3.4 Vertex Cluster

Unlike the preceding simplification operations, the vertex cluster operation relies solely on the geometry of the input (i.e. the vertex coordinates) rather than the topology (i.e. the adjacency information) to reduce the complexity. Like the edge and face collapses, several vertices are merged into a single vertex. However, rather than merging a set of topologically adjacent vertices, a set of "nearby" vertices are merged [Rossignac and Borrel 1992]. For instance, one possibility is to merge all vertices that lie within a particular 3D axis-aligned box. The new, merged vertex may be one of the original vertices that "best represents" the entire set, or it may be placed arbitrarily to minimize some error bound. An important property of this operation is that it can be robustly applied to arbitrary sets of triangles, whereas all the preceding operations assume that the triangles form a connected, manifold mesh.

The effects of this vertex cluster are similar to those of the collapse operations. Some triangles are distorted, whereas others degenerate to a line segment or a point. In addition, there may be coincident triangles, line segments, and points originating from non-coincident geometry. One may choose to render the degenerate triangles as line segments and points, or one may simply not render them at all. Depending on the particular graphics engine, rendering a line or a point may not be much faster than rendering a triangle. This is an important consideration, because achieving a speed-up is one of the primary motivations for simplification.

There is no point in rendering several coincident primitives, so multiple copies are filtered down to a single copy. However, the question of how to render coincident geometry is complicated by the existence of other surface attributes, such as normals and colors. For

instance, suppose two triangles of wildly different colors become coincident. No matter what color we render the triangle, it may be noticeably incorrect.

[Rossignac and Borrel 1992] use the vertex clustering operation in their simplification system to perform very fast simplification on arbitrary polygonal models. They partition the model space with a uniform grid, and vertices are collapsed within each grid cell. [Luebke and Erikson 1997] build an octree hierarchy rather than a grid at a single resolution. They dynamically collapse and split the vertices within an octree cell depending on the current size of the cell in screen space as well as silhouette criteria.

Figure 9: Generalized edge collapse operation

3.5 Generalized Edge Collapse

The generalized edge collapse (or vertex pair) operation combines the fine-grained control of the edge collapse operation with the generality of the vertex cluster operation. Like the edge collapse operation, it involves the merging of two vertices and the removal of degenerate triangles. However, like the vertex cluster operation, it does not require that the merged vertices be topologically connected (by a topological edge), nor does it require that topological edges be manifold.

[Garland and Heckbert 1997] apply the generalized edge collapse in conjunction with error quadrics to achieve simplification that gives preference to the collapse of topological edges, but also allows the collapse of virtual edges (arbitrary pairs of vertices). These virtual edges are chosen somewhat heuristically, based on proximity relationships in the original mesh.
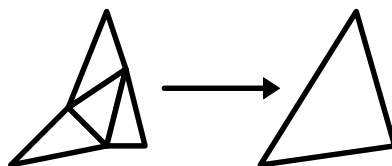
Figure 10: Unsubdivide operation

10

3.6 Unsubdivide

Subdivision surface representations have also been proposed as a solution to the multi-resolution problem. In the context of simplification operations, we can think of the "unsubdivide" operation (the inverse of a subdivision refinement) as our simplification operation. A common form of subdivision refinement is to split one triangle into four triangles. Thus the unsubdivide operation merges four triangles of a particular configuration into a single triangle, reducing the triangle count by three triangles.

[DeRose et al. 1993] shows how to represent a subdivision surface at some finite resolution as a sequence of wavelet coefficients. The sequence of coefficients is ordered from lower to higher frequency content, so truncating the sequence at a particular point determines a particular mesh resolution. [Eck et al. 1995] presents an algorithm to turn an arbitrary topology mesh into one with the necessary subdivision connectivity. They construct a base mesh of minimal resolution and guide its refinement to come within some tolerance of the original mesh. This new refined subdivision mesh is used in place of the original mesh, and its resolution is controlled according to the wavelet formulation.

4. TOPOLOGICAL CONSIDERATIONS

4.1 Manifold vs. Non-manifold Meshes

Polygonal simplification algorithms may be distinguished according to the type of input they accept. Some algorithms require the input to be a *manifold* triangle mesh, while others accept more general triangle sets. In the continuous domain, a manifold surface is one that is everywhere homeomorphic to an open disc. In the discrete domain of triangle meshes, such a surface has two topological properties. First, every vertex is adjacent to a set of triangles that form a single, complete cycle around the vertex. Second, each edge is adjacent to exactly two triangles. For a manifold mesh with *borders*, these restrictions are slightly relaxed. A border is simply a chain of edges with adjacent triangles only to one side. In a manifold mesh with borders, a vertex may be surrounded by a single, incomplete cycle (i.e. the beginning need not meet the end). Also, an edge may be adjacent to either one or two triangles.

A mesh that does not have the above properties is said to be *non-manifold*. Such meshes may occur in practice by accident or by design. Accidents are possible, for example, during either the creation of the mesh or during conversions between representation, such as the conversion from a solid to a boundary representation. The correction of such accidents is a subject of much interest [Barequet and Kumar 1997, Murali and Funkhouser 1997]. They may occur by design because such a mesh may require fewer triangles to render than a visually-comparable manifold mesh or because such a mesh may be easier to create in some situations. If the non-manifold portions of a mesh are few and far between, we may refer to the mesh as *mostly manifold*.

At the extreme, some data sets take the form of a set of triangles, with no connectivity information whatsoever (sometimes referred to as a "triangle soup"). Such data might turn out to be manifold or non-manifold if we were to attempt to reconstruct the connectivity information. In general, if any conversion has been performed on the original data, it's safe to assume that a naïve reconstruction will result in at least some non-manifold regions.

The most robust algorithms, based on vertex clusters, operate as easily on a triangle soup as on a perfectly manifold mesh [Rossignac and Borrel 1992], [Luebke and Erikson 1997]. This advantage cannot be stressed enough and is extremely important in the case where the simplification user has no control over the data. The ability to view an large, unfamiliar data set interactively is invaluable in the process of learning its ins and outs, and these algorithms allow one to get up and running quickly.

However, these very general algorithms do not typically create simplifications that look as attractive as those produced by algorithms that operate on manifold meshes. These algorithms, which rely on operations such as the vertex remove or edge collapse, respect the topology of the original mesh and avoid catastrophic changes to the surface and its appearance. The manifold input criterion does limit the applicability of these algorithms to some real-world models, but many of these algorithms may be modified to handle mostly manifold meshes by avoiding simplification of the non-manifold regions. This can be an effective strategy until the non-manifold regions begin to dominate the surface complexity.

The vertex pair and edge collapse operations can both operate on non-manifold meshes as well as manifold ones. Vertex-pair algorithms must deal with the non-manifold meshes they are bound to create by merging non-adjacent vertices. Edge collapse algorithms can operate on non-manifold meshes, but it may be difficult to adapt the most rigorous error metrics for manifold meshes to use on non-manifold meshes.

4.2 Topology Preservation

The topological structure of a polygonal surface typically refers to features such as its *genus* (number of topological holes, e.g. 0 for a sphere, 1 for a torus or coffee mug) and the number and arrangement of its borders. These features are fully determined by the adjacency graph of the vertices, edges, and faces of a polygonal mesh. For manifold meshes with no borders (i.e. closed surfaces), the Euler equation holds:

$$F - E + V = 2 - G, \tag{2}$$

where F is the number of faces, E is the number of edges, V is the number of vertices, and G is the genus.

In addition to this combinatorial description of the topological structure, the embedding of the surface in 3-space impacts its perceived topology in 3D renderings. Generally, we expect the faces of a surface to intersect only at their shared edges and vertices.

Most of the simplification operations described in section 3 (all except the vertex cluster and the generalized edge collapse) preserve the connectivity structure of the mesh. If a simplification algorithm uses such an operation and also prevents local self-intersections (intersections within the adjacent neighborhood of the operation), we say the algorithm *preserves local topology*. If the algorithm prevents any self-intersections in the entire mesh, we say it *preserves global topology*.

If the simplified surface is to be used for purposes other than rendering (e.g. finite element computations), topology preservation may be essential. For rendering applications, however, it is not always necessary. In fact, it is often possible to construct simplifications with fewer polygons for a given error bound if topological modifications are allowed.

However, some types of topological modifications may have a dramatic impact on the appearance of the surface. For instance, many meshes are the surfaces of solid objects. For example, consider the surface of a thin, hollow cylinder. When the surface is modified by more than the thickness of the cylinder wall, the interior surface will intersect the outer surface. This can cause artifacts that cover a large area on the screen. Problems also occur when polygons with different color attributes become coincident.

Certain types of topological changes are clearly beneficial in reducing complexity, and have a smaller impact on the rendered image. These include the removal of topological holes and thin features (such as the antenna of a car). Topological modifications are encouraged in [Rossignac and Borrel 1992], [Luebke and Erikson 1997], [Garland and Heckbert 1997] and [Erikson and Manocha 1998] and controlled modifications are performed in [He et al. 1996] and [El-Sana and Varshney 1997].

5. LEVEL-OF-DETAIL REPRESENTATIONS

We can classify the possible representations for level-of-detail models into two broad categories: *static* and *dynamic*. Static levels of details are computed totally off-line. They are fully determined as a pre-process to the visualization program. Dynamic levels of detail are typically computed partially off-line and partially on-line within the visualization program. We now discuss these representations in more detail.

5.1 Static Levels of Detail

The most straightforward level-of-detail representation for an object is a set of independent meshes, where each mesh has a different number of triangles. A common heuristic for the generation of these meshes is that the complexity of each mesh should be reduced by a factor of two from the previous mesh. Such a heuristic generates a reasonable range of complexities, and requires only twice as much total memory as the original representation.

It is common to organize the objects in a virtual environment into a hierarchical *scene graph* [van Dam 1988, Rohlf and Helman 1994]. Such a scene graph may have a special type of node for representing an object with levels of detail. When the graph is traversed, this

level-of-detail node is evaluated to determine which child branch to traverse (each branch represents one of the levels of detail). In most static level-of-detail schemes, the children of the level-of-detail nodes are the leaves of the graph. [Erikson and Manocha 1998] presents a scheme for generating *hierarchical levels of detail*. This scheme generates level-of-detail nodes throughout the hierarchy rather than just at the leaves. Each such interior level-of-detail node involves the merging of objects to generate even simpler geometric representations. This overcomes one of the previous limitations of static levels of detail — the necessity for choosing a single scale at which objects are identified and simplified.

The transitions between these levels of detail are typically handled in one of three ways: discrete, blended, or morphed. The discrete transitions are instantaneous switches; one level of detail is rendered during one frame, and a different level of detail is rendered during the following frame. The frame at which this transition occurs is typically determined based on the distance from the object to the viewpoint. This technique is the most efficient of the three transition types, but also results in the most noticeable artifacts.

Blended transitions employ alpha-blending to fade between the two levels of detail in question. For several frames, both levels of detail are rendered (increasing the rendering cost during these frames), and their colors are blended. The blending coefficients change gradually to fade from one level of detail to the other. It is possible to blend over a fixed number of frames when the object reaches a particular distance from the viewpoint, or to fade over a fixed range of distances [Rohlf and Helman 1994]. If the footprints of the objects on the screen are not identical, blending artifacts may still occur at the silhouettes.

Morphed transitions involve gradually changing the shape of the surface as the transition occurs. This requires the use of some correspondence between the two levels of detail. Only one representation must be rendered for each frame of the transition, but the vertices require some interpolation each frame. For instance, [Hoppe 1996] describes the *geomorph* transition for levels of detail created by a sequence of edge collapses. The simpler level of detail was originally generated by collapsing some number of vertices, and we can create a transition by simultaneously interpolating these vertices from their positions on one level of detail to their positions on the other level of detail. Thus the number of triangles we render during the

transition is equal to the maximum of the numbers of triangles in the two levels of detail. It is also possible to morph using a mutual tessellation of the two levels of detail, as in [Turk 1992], but this requires the rendering of more triangles during the transition frames.

5.2 Dynamic Levels of Detail

Dynamic levels of detail provide representations that are more carefully tuned to the viewing parameters of each particular rendered frame. Due to the sheer number of distinct representations this requires, each representation cannot simply created and stored independently. The common information among these representations is used to create a single representation for each simplified object. From this unified representation, a geometric representation that is tuned to the current viewing parameters is extracted. The coherence of the viewing parameters enables incremental modifications to the geometry rendered in the previous frame; this makes the extraction process feasible at interactive frame rates.

[Hoppe 1996] presents a representation called the *progressive mesh*. This representation is simply the original object plus an ordered list of the simplification operations performed on the object. It is generally more convenient to reverse the order of this intuitive representation, representing the simplest *base mesh* plus the inverse of each of the simplification operations. Applying all of these inverse operations to the base mesh will result in the original object representation. A particular level of detail of this progressive mesh is generated by performing some number of these operations.

In [Hoppe 1997], the progressive mesh is reorganized into a vertex hierarchy. This hierarchy is a tree that captures the dependency of each simplification operation on certain previous operations. Similar representations include the *merge tree* of [Xia et al. 1997], the *multiresolution model* of [Klein and Krämer 1997], the *vertex tree* of [Luebke and Erikson 1997], and the *multi-triangulation* of [DeFloriani et al. 1997]. Such hierarchies allow selective refinement of the geometry based on various metrics for screen-space deviation, normal deviation, color deviation, and other important features such as silhouettes and specular highlights. A particular level of detail may be expressed as a *cut* through these graphs, or a *front* of vertex nodes. Each frame, the nodes on the current front are examined, and may cause the graph to be refined at some of these nodes.

[DeFloriani et al. 1997] discuss the properties of such hierarchies in terms of graph characteristics. Examples of these properties include compression ratio, linear growth, logarithmic height, and bounded width. They discuss several different methods of constructing such hierarchies and test these methods on several benchmarks. For example, one common heuristic for building these hierarchies is to choose simplification operations in a greedy fashion according to an error metric. Another method is to choose a set of operations with disjoint areas of influence on the surface and apply this entire set before choosing the next set. The former method does not guarantee logarithmic height, whereas the latter does. Such height guarantees can have practical implications in terms of the length of the chain of dependent operations that must be performed in order to achieve some particular desired refinement.

[DeRose et al. 1993] present a wavelet-based representation for surfaces constructed with subdivision connectivity. [Eck et al. 1995] make this formulation applicable to arbitrary triangular meshes by providing a remeshing algorithm to approximate an arbitrary mesh by one with the necessary subdivision connectivity. Both the remeshing and the filtering/reconstruction of the wavelet representation provide bounded error on the surfaces generated. [Lee et al. 1998] provide an alternate remeshing algorithm based on a smooth, global parameterization of the input mesh. Their approach also allows the user to constrain the parameterization at vertices or along edges of the original mesh to better preserve important features of the input.

5.3 Comparison

Static levels of detail allow us to perform simplification entirely as a pre-process. The real-time visualization system performs only minimal work to select which level of detail to render at any given time. Because the geometry does not change, it may be rendered in retained mode (i.e. from cached, optimized *display lists*). Retained-mode rendering should always be at least as fast as immediate mode rendering, and is much faster on most current high-end hardware. Perhaps the biggest shortcoming of using static levels of detail is that they require that we partition the model into independent "objects" for the purpose of simplification. If an object is large with respect to the user or the environment, especially if the viewpoint is often contained inside the object, little or no simplification may be possible.

This may require that such objects be subdivided into smaller objects, but switching the levels of detail of these objects independently causes visible cracks, which are non-trivial to deal with.

Dynamic levels of detail perform some of simplification as a pre-process, but defer some of the work to be computed by the real-time visualization system at run time. This allows us to provide more fine-tuning of the exact tessellation to be used, and allows us to incorporate more view-dependent criteria into the determination of this tessellation. The shortcoming of such dynamic representations is that they require more computation in the visualization system as well as the use of immediate mode rendering. Also, the memory requirements for such representations are often somewhat larger than for the static levels of detail.

6. SURFACE DEVIATION ERROR BOUNDS

Measuring the deviation of a polygonal surface as a result of simplification is an important component of the simplification process. This surface deviation error gives us an idea of the quality of a particular simplification. It helps guide the simplification process to produce levels of detail with low error, determine when it is appropriate to show a particular level of detail of a given surface, and optimize the levels of detail for an entire scene to achieve a high overall image quality for the complexity of the models actually rendered.

6.1 Distance Metrics

Before discussing the precise metrics and methods used by several researchers for measuring surface deviation, we consider two formulations of the distance between two surfaces. These are the Hausdorff distance and the mapping distance. The Hausdorff distance is a well-known concept from topology, used in image processing as well as surface modeling, and the mapping distance is a commonly used metric for parametric surfaces.

6.1.1 Hausdorff Distance

The Hausdorff distance is a distance metric between point sets. Given two sets of points, A and B, the Hausdorff distance is defined as

$$H(A,B) = \max(h(A,B), h(B,A)) , \tag{3}$$

where

$$h(A,B) = \max_{a \in A} \min_{b \in B} \left\| a - b \right\|. \tag{4}$$

Thus the Hausdorff distance measures the farthest distance from a point in one point set to its closest point in the other point set (notice that h(A,B) ≠ h(B,A)). Because a surface is a particular type of continuous point set, the Hausdorff distance provides a useful measure of the distance between two surfaces.

6.1.2 Mapping Distance

The biggest shortcoming of the Hausdorff distance metric for measuring the distance between surfaces is that it makes no use of the point neighborhood information inherent in the surfaces. The function h(A,B) implicitly assigns to each point of surface A the closest point of surface B. However, this mapping may have discontinuities. If points i and j are "neighboring" points on surface A (i.e. there is a path on the surface of length no greater than ε that connects them), their corresponding points, i' and j', on surface B may not be neighboring points. In addition, the mapping implied by h(A,B) is not identical to the mapping implied by h(B,A).

For the purpose of simplification, we would like to establish a continuous mapping between the surface's levels of detail. Ideally, the correspondences described by this mapping should coincide with a viewer's perception of which points are "the same" on the surfaces. Given such a continuous mapping

$$F: A \rightarrow B$$

the mapping distance is defined as

$$D(F) = \max_{a \in A} \left\| a - F(a) \right\|. \tag{5}$$

Because there are many such mappings, there are many possible mapping distances. The minimum mapping distance is simply

$$D_{min} = \min_{F \in M} D(F), \tag{6}$$

where *M* is the set of all such continuous mapping functions. Note that although D_{min} and its associated mapping function may be difficult to compute, all continuous mapping functions provide an upper bound on D_{min}.

6.2 Surface Deviation Algorithms

We now classify several simplification algorithms according to how they measure the surface deviation error of their levels of detail.

6.2.1 Mesh Optimization

[Hoppe et al. 1993] pose the simplification problem in terms of optimizing an energy function. This function has terms corresponding to number of triangles, surface deviation error, and a heuristic spring energy. To quantify surface deviation error, they maintain a set of point samples from the original surface and their closest distance to the simplified surface. The sum of squares of these distances is used as the surface deviation component of the energy function. The spring energy term is required because the surface deviation error is only measured in one direction: it approximates the closest distance from the original surface to the simplified surface, but not vice versa. Without this term, small portions of the simplified surface can deviate quite far from the original surface, as long as all the point samples are near to some portion of the simplified surface.

6.2.2 Vertex Clustering

[Rossignac and Borrel 1993] present a simple and general algorithm for simplification using vertex clustering. The vertices of each object are clustered using several different sizes of uniform grid. The surface deviation in this case is a Hausdorff distance and must be less than or equal to the size of grid cell used in determining the vertex clusters. This is a very conservative bound, however. A slightly less conservative bound is the maximum distance from a vertex in the original cluster to the single representative vertex after the cluster is collapsed. Even this bound is quite conservative in many cases; the actual maximum deviation from the original surface to the simplified surface may be considerably smaller than the distance the original vertices travel during the cluster operation.

[Luebke and Erikson 1997] take a similar approach, but their system uses an octree instead of a single-resolution uniform grid. This allows them to take a more dynamic approach, folding and unfolding octree cells at run-time and freely merging nearby objects. The measure of surface deviation remains the same, but they allow a more flexible choice of error tolerances in their run-time system. In particular, they use different tolerances for silhouette and non-silhouette clusters.

6.2.3 Superfaces

[Kalvin and Taylor 1996] present an efficient simplification algorithm based on merging adjacent triangles to form polygonal patches, simplifying the boundaries of these patches, and finally retriangulating the patches themselves. This algorithm guarantees a maximum deviation from vertices of the original surface to the simplified surface and from vertices of the simplified surface to the original surface. Unfortunately, even this bidirectional bound does not guarantee a maximum deviation between points on the simplified surface and points on the original surface. For instance, suppose we have two adjacent triangles that share an edge, forming a non-planar quadrilateral. If we retriangulate this quadrilateral by performing an edge swap operation, the maximum deviation between these two surfaces is non-zero, even though their four vertices are unchanged (thus the distance measured from vertex to surface is zero).

6.2.4 Error Tolerance Volumes

[Guéziec 1995] presents a simplification system that measures surface deviation using error volumes built around the simplified surface. These volumes are defined by spheres, specified by their radii, centered at each of the simplified surface's vertices. We can associate with any point in a triangle a sphere whose radius is a weighted average of the spheres of the triangle's vertices. The error volume of an entire triangle is the union of the spheres of all the points on the triangle, and the error volume of a simplified surface is the union of the error volumes of its triangles. As edge collapses are performed, not only are the coordinates of the new vertex computed, but new sphere radii are computed such that the new error volume contains the previous error volume. The maximum sphere radius is a bound on the Hausdorff

distance of the simplified surface from the original, and thus provides a bound for surface deviation in both 3D and 2D (after perspective projection).

6.2.5 Simplification Envelopes

The simplification envelopes technique of [Cohen and Varshney et al. 1996] bounds the Hausdorff distance between the original and simplified surfaces without actually making measurements during the simplification process. For a particular simplification, the input surface is surrounded by two envelope surfaces, which are constructed to deviate by no more than a specified tolerance, ε, from the input surface. As the simplification progresses, the modified triangles are tested for intersection with these envelopes. If no intersections occur, the simplified surface is within distance ε from the input surface. Similar constructions are built to constrain error around the borders of bordered surfaces. By including extensive self-intersection testing as well, the algorithm provides complete global topology preservation. This algorithm does an excellent job at generating small-triangle-count surface approxima-tions for a given error bound. The biggest limitations are the up-front processing costs required for envelope construction (for each level of detail to be generated) and the conserva-tive nature of the envelopes themselves, which do not expand beyond the point of self-intersection.

6.2.6 Error Quadrics

[Ronfard and Rossignac 1996] describe a fast method for approximating surface devia-tion. They represent surface deviation error for each vertex as a sum of squared distances to a set of planes. The initial set of planes for each vertex are the planes of its adjacent faces. As vertices are merged, the sets of planes are unioned. This metric provides a useful and efficient heuristic for choosing an ordering of edge collapse operations, but it does not provide any guarantees about the maximum or average deviation of the simplified surface from the original.

[Garland and Heckbert 1997] present some improvements over [Ronfard and Rossignac 1996]. The error metric is essentially the same, but they show how to approximate a vertex's set of planes by a quadric form (represented by a single 4x4 matrix). These matrices are simply added to propagate the error as vertices are merged. Using this metric, it is possible to

choose an optimal vertex placement that minimizes the error. In addition, they allow the merging of vertices that are not joined by an edge, allowing increased topological modification. [Erikson and Manocha 1998] further improve this technique by automating the process of choosing which non-edge vertices to collapse and by encouraging such merging to preserve the local surface area.

6.2.7 Mapping Error

[Bajaj and Schikore 1996] perform simplification using the vertex remove operation, and measure surface deviation using local, bijective (one-to-one and onto) mappings in the plane between points on the surface just before and just after the simplification operation. This approach provides a fairly tight bound on the maximum deviation over all points on the surface, not just the vertices (as does [Guéziec 1995]) and provides pointwise mappings between the original and simplified surfaces.

A similar technique is employed by [Cohen et al. 1997], who perform mappings in the plane for the edge collapse operation. They present rigorous and efficient techniques for finding a plane in which to perform the mapping, as well as applying the mapping and propagating error from operation to operation. The computed mappings are used not only to guide the simplification process in its choice of operations, but also to assign texture coordinates to the post-collapse vertices and to control the switching of levels of detail in interactive graphics applications.

6.2.8 Hausdorff Error

[Klein et al. 1996] measure a one-sided Hausdorff distance (with appropriate locality restrictions) between the original surface and the simplified surface. By definition, this approach produces the smallest possible bound on maximum one-sided surface deviation, but the one-sided formulation does not guarantee a true bound on overall maximum deviation. At each step of the simplification process, the Hausdorff distance must be measured for each of the original triangles mapping to the modified portion of the surface. The computation time for each simplification operation grows as the simplified triangles cover more and more of the mesh, but of course, there are also fewer and fewer triangles to simplify. [Klein and Krämer 1997] present an efficient implementation of this algorithm.

6.2.9 Memory-efficient Simplification

[Lindstrom and Turk 1998] demonstrate the surprising result that good simplifications are possible without measuring anything with respect to the original model. All errors in this method are measured purely as incremental changes in the local surface. The error metric used preserves the total volume while minimizing volume changes of each triangle. Another interesting aspect of this work is that they perform after-the-fact measurements to compare the "actual" mean and maximum simplification errors of several algorithm implementations. These measurement use the *Metro* geometric comparison tool [Cignoni et al. 1996], which uniformly samples the simplified surface, computes correspondences with the original surface, and measures the error of the samples.

7. APPEARANCE ATTRIBUTE PRESERVATION

We now classify several algorithms according to how they preserve the appearance attributes of their input models.

7.1 Scalar Field Deviation

The mapping algorithm presented in [Bajaj and Schikore 1996] allows the preservation of arbitrary scalar fields across a surface. Such scalar fields are specified at the mesh vertices and linearly interpolated across the triangles. Their approach computes a bound on the maximum deviation of the scalar field values between corresponding points on the original surface and the simplified surface.

7.2 Color Preservation

[Hughes et al. 1996] describes a technique for simplifying colored meshes resulting from global illumination algorithms. They use a logarithmic function to transform the vertex colors into a more perceptually linear space before applying simplification. They also experiment with producing mesh elements that are quadratically- or cubically-shaded in addition to the usual linearly-shaded elements.

[Hoppe 1996] extends the error metric of [Hoppe et al. 1993] to include error terms for scalar attributes and discontinuities as well as surface deviation. Like the surface deviation,

the scalar attribute deviation is measured as a sum of squared Euclidean distances in the attribute space (e.g. the RGB color cube). The distances are again measured between sampled points on the original surface and their closest points on the simplified surface. This metric is useful for prioritizing simplification operations in order of increasing error. However, it does not provide much information about the true impact of attribute error on the final appearance of the simplified object on the screen. A better metric should incorporate some degree of area weighting to indicate how the overall illuminance of the final pixels may be affected.

[Erikson and Manocha 1998] present a method for measuring the maximum attribute deviation in Euclidean attribute spaces. Associated with each vertex is an attribute volume for each attribute being measured. The volume is a disc of the appropriate dimension (i.e. an interval in 1D, a circle in 2D, a sphere in 3D, etc.). Each attribute volumes is initially a point in the attribute space (an *n*-disk with radius zero). As vertex pairs are merged, the volumes grow to contain the volumes of both vertices.

[Garland and Heckbert 1998] extend the algorithm of [Garland and Heckbert 1997] to consider color and texture coordinate error as well as geometry. The error quadrics are lifted to higher dimensions to accommodate the combined attribute spaces (e.g. 3 dimensions for RGB color and 2 dimensions for texture coordinates). The associated form matrices grow quadratically with the dimension, but standard hardware-accelerated rendering models typically require a dimension of 9 or less. The error is thus measured and optimized for all attributes simultaneously. The method makes the simplifying assumption that the errors in all these attribute values may be measured as in a Euclidean space.

[Certain et al. 1996] present a method for preserving vertex colors in conjunction with the wavelet representation for subdivision surfaces [DeRose et al. 1993]. The geometry and color information are stored as two separate lists of wavelet coefficients. Coefficients may be added or deleted from either of these lists to adjust the complexity of the surface and its geometric and color errors. They also use the surface parameterization induced by the subdivision to store colors in texture maps to render as textured triangles for machines that support texture mapping in hardware.

[Bastos et al. 1997] use texture maps with bicubic filtering to render the complex solutions to radiosity illumination computations. The radiosity computation often dramatically increases the number of polygons in the input mesh in order to create enough vertices to store the resulting colors. Storing the colors instead in texture maps removes unnecessary geometry, reducing storing requirements and rasterization overhead.

The appearance-preserving simplification technique of [Cohen et al. 1998] is in some sense a generalization of this "radiosity as textures" work. Colors are stored as texture maps before the simplification is applied. Mappings are computed as in [Cohen et al. 1997], but this time in the 2D texture domain, effectively measuring the 3D displacements of a texture map as a surface is simplified. Whereas [Bastos et al. 1997] reduces geometry complexity to that of the pre-radiositized mesh, [Cohen et al. 1998] simplify complex geometry much farther, quantifying the distortions caused by the simplification of non-planar, textured surfaces. [Cignoni et al. 98] describe a method for compactly storing attribute values into map structures that are customized to a particular simplified mesh.

7.3 Normal Vector Preservation

[Xia et al. 1997] associate a cone of normal vectors with each vertex during their simplification preprocess. These cones initially have an angle of zero, and grow to contain the cones of the two vertices merged in an edge collapse. Their run-time, dynamic simplification scheme uses this range of normals and the light direction to compute a range of reflectance vectors. When this range includes the viewing direction, the mesh is refined, adapting the simplification to the specular highlights. The results of this approach are visually quite compelling, though they do not allow increased simplification of the highlight area as it gets smaller on the screen (i.e. as the object gets farther from the viewpoint).

[Klein 1998] maintains similar information about the cone of normal deviation associated with each vertex. The refinement criterion takes into account the spread of reflected normals (i.e. the specular exponent, or shininess) in addition to the reflectance vectors themselves. Also, refinement is performed in the neighborhood of silhouettes with respect to the light sources as well as specular highlights. Again, this normal deviation metric does not allow

increased simplification in the neighborhood of the highlights and light silhouettes as the object gets smaller on the screen.

[Cohen et al. 1998] apply their appearance-preserving technique to normals as well as colors by storing normal vectors in normal maps. Figure 11 shows a view of a complex "armadillo" model. Applying the appearance-preserving algorithm to this model generates the simplified versions of Figure 12 and Figure 13, in which it is nearly impossible to distinguish the simplifications from the original. Compared this to the bunnies in Figure 3 and Figure 4. Although the positions of the surfaces are preserved quite well, as evidenced by the similarity of the silhouettes of the bunnies, the shading makes it quite easy to tell which bunnies have been simplified and which have not (i.e. the appearance has not been totally preserved).

The appearance-preserving approach to normal preservation has the advantage that the normal values need not be considered in the simplification process – only texture distortion error constrains the simplification process. In fact, the error in the resulting images can be characterized entirely by the number of pixels of deviation of the textured surface on the screen. The major disadvantage to this approach is that it assumes a per-pixel lighting model is applied to shade the normal-mapped triangles. Per-pixel lighting is still too computation-ally expensive for most graphics hardware, though support for such lighting is making its way into standard graphics APIs such as OpenGL.

Figure 11: "Armadillo" model: 249,924 triangles

249,924 triangles   7,809 triangles

Figure 12: Medium-sized "armadillos"

249,924 triangles   975 triangles

Figure 13: Small-sized "armadillos"

8. CONCLUSIONS

As is the case for many classes of geometric algorithms, there does not seem to be any single best simplification algorithm or scheme. An appropriate scheme depends not only on the characteristics of the input models, but also the final application to which the multi-resolution output will be applied.

For poorly-behaved input data (mostly non-manifold or triangle soups), the vertex clustering algorithms [Rossignac and Borrel 1992], [Luebke and Erikson 1997] should yield the fastest and most painless success. For cleaner input data, one of the many methods which respect topology will likely produce more appealing results.

When even pre-computation time is of the essence, a fast algorithm such as [Garland and Heckbert 1997] may be appropriate, while applications required better-controlled visual fidelity should invest some extra pre-computation time in an algorithm such as [Cohen et al. 1998], [Guéziec 1995], or [Hoppe 1996], to achieve guaranteed or at least higher quality.

For applications and machines with extra processing power to spare, dynamic level of detail techniques such as [Hoppe 1997] and [Luebke and Erikson 1997] can provide smooth level-of-detail transitions with minimal triangle counts. However, for applications requiring maximal triangle throughput (including display lists) or need to actually employ their CPU(s) for application-related processing, static levels of detail (possibly with geomorphs between levels of detail) are often preferable (they also add less complexity to application code).

The construction and use of levels of detail have become essential tools for accelerating the rendering process. The field has now reached a level of maturity at which there is a rich "bag of tricks" from which to choose when considering the use of levels of detail for a particular application. Making sense of the available techniques as well as when and how well they work is perhaps the next step towards answering the question, "What is a good simplification?", both statically, and over the course of an interactive application.

9. ACKNOWLEDGMENTS

10. REFERENCES

Agarwal, Pankaj K. and Subhash Suri. Surface Approximation and Geometric Partitions. *Proceedings of 5th ACM-SIAM Symposium on Discrete Algorithms.* 1994. pp. 24-33.

Bajaj, Chandrajit and Daniel Schikore. Error-bounded Reduction of Triangle Meshes with Multivariate Data. *SPIE.* vol. 2656. 1996. pp. 34-45.

Barequet, Gill and Subodh Kumar. Repairing CAD Models. *Proceedings of IEEE Visualization '97.* October 19-24. pp. 363-370, 561.

Bastos, Rui, Mike Goslin, and Hansong Zhang. Efficient Rendering of Radiosity using Texture and Bicubic Interpolation. *Proceedings of 1997 ACM Symposium on Interactive 3D Graphics.*

Brönnimann, H. and Michael T. Goodrich. Almost Optimal Set Covers in Finite VC-Dimension. *Proceedings of 10th Annual ACM Symposium on Computational Geometry.* 1994. pp. 293-302.

Certain, Andrew, Jovan Popovic, Tony DeRose, Tom Duchamp, David Salesin, and Werner Stuetzle. Interactive Multiresolution Surface Viewing. *Proceedings of SIGGRAPH 96.* pp. 91-98.

Cignoni, Paolo, Claudio Montani, Claudio Rocchini, and Roberto Scopigno. A General Method for Recovering Attribute Values on Simplified Meshes. *Proceedings of IEEE Visualization '98.* pp. 59-66, 518.

Cignoni, Paolo, Claudio Rocchini, and Roberto Scopigno. Metro: Measuring Error on Simplified Surfaces. Technical Report B4-01-01-96, Instituto I. E. I.- C.N.R., Pisa, Italy, January 1996.

Clarkson, Kenneth L. Algorithms for Polytope Covering and Approximation. *Proceedings of 3rd Workshop on Algorithms and Data Structures.* 1993. pp. 246-252.

Cohen, Jonathan, Dinesh Manocha, and Marc Olano. Simplifying Polygonal Models using Successive Mappings. *Proceedings of IEEE Visualization '97.* pp. 395-402.

Cohen, Jonathan, Marc Olano, and Dinesh Manocha. Appearance-Preserving Simplification. *Proceedings of ACM SIGGRAPH 98.* pp. 115-122.

Cohen, Jonathan, Amitabh Varshney, Dinesh Manocha, Gregory Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification Envelopes. *Proceedings of SIGGRAPH 96.* pp. 119-128.

Das, G. and D. Joseph. The Complexity of Minimum Convex Nested Polyhedra. *Proceedings of 2nd Canadian Conference on Computational Geometry.* 1990. pp. 296-301.

DeFloriani, Leila, Paola Magillo, and Enrico Puppo. Building and Traversing a Surface at Variable Resolution. *Proceedings of IEEE Visualization '97.* pp. 103-110.

DeRose, Tony, Michael Lounsbery, and J. Warren. Multiresolution Analysis for Surfaces of Arbitrary Topology Type. Technical Report TR 93-10-05. Department of Computer Science, University of Washington. 1993.

Dörrie, H. Euler's Problem of Polygon Division. *100 Great Problems of Elementary Mathematics: Their History and Solutions.* Dover, New York.1965. pp. 21-27.

Eck, Matthias, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution Analysis of Arbitrary Meshes. *Proceedings of SIGGRAPH 95.* pp. 173-182.

El-Sana, Jihad and Amitabh Varshney. Controlled Simplification of Genus for Polygonal Models. *Proceedings of IEEE Visualization'97.* pp. 403-410.

Erikson, Carl. Polygonal Simplification: An Overview. Technical Report TR96-016. Department of Computer Science, University of North Carolina at Chapel Hill. 1996.

Erikson, Carl and Dinesh Manocha. Simplification Culling of Static and Dynamic Scene Graphs. Technical Report TR98-009. Department of Computer Science, University of North Carolina at Chapel Hill. 1998.

Garland, Michael and Paul Heckbert. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. *Proceedings of IEEE Visualization '98.* pp. 263-269, 542.

Garland, Michael and Paul Heckbert. Surface Simplification using Quadric Error Bounds. *Proceedings of SIGGRAPH 97.* pp. 209-216.

Gieng, Tran S., Bernd Hamann, Kenneth I. Joy, Gregory L. Schlussmann, and Isaac J. Trotts. Smooth Hierarchical Surface Triangulations. *Proceedings of IEEE Visualization '97.* pp. 379-386.

Guéziec, André. Surface Simplification with Variable Tolerance. *Proceedings of Second Annual International Symposium on Medical Robotics and Computer Assisted Surgery (MRCAS '95).* pp. 132-139.

Hamann, Bernd. A Data Reduction Scheme for Triangulated Surfaces. *Computer Aided Geometric Design.* vol. 11. 1994. pp. 197-214.

He, Taosong, Lichan Hong, Amitabh Varshney, and Sidney Wang. Controlled Topology Simplification. *IEEE Transactions on Visualization and Computer Graphics.* vol. 2(2). 1996. pp. 171-814.

Heckbert, Paul and Michael Garland. Survey of Polygonal Simplification Algorithms. *SIGGRAPH 97 Course Notes.* 1997.

Hoppe, Hugues. Progressive Meshes. *Proceedings of SIGGRAPH 96.* pp. 99-108.

Hoppe, Hugues. View-Dependent Refinement of Progressive Meshes. *Proceedings of SIGGRAPH 97.* pp. 189-198.

Hoppe, Hugues, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh Optimization. *Proceedings of SIGGRAPH 93.* pp. 19-26.

Hughes, Merlin., Anselmo Lastra, and Eddie Saxe. Simplification of Global-Illumination Meshes. *Proceedings of Eurographics '96, Computer Graphics Forum.* pp. 339-345.

Kalvin, Alan D. and Russell H. Taylor. Superfaces: Polygonal Mesh Simplification with Bounded Error. *IEEE Computer Graphics and Applications.* vol. 16(3). 1996. pp. 64-77.

Klein, Reinhard. Multiresolution Representations for Surface Meshes Based on the Vertex Decimation Method. *Computers and Graphics.* vol. 22(1). 1998. pp. 13-26.

Klein, Reinhard and J. Krämer. Multiresolution Representations for Surface Meshes. *Proceedings of Spring Conference on Computer Graphics 1997.* June 5-8. pp. 57-66.

Klein, Reinhard, Gunther Liebich, and Wolfgang Straßer. Mesh Reduction with Error Control. *Proceedings of IEEE Visualization '96.*

Krishnamurthy, Venkat and Marc Levoy. Fitting Smooth Surfaces to Dense Polygon Meshes. *Proceedings of SIGGRAPH 96.* pp. 313-324.

Lindstrom, Peter and Greg Turk. Fast and Memory Efficient Polygonal Simplification. *Proceedings of IEEE Visualization '98.* pp. 279-286, 544.

Luebke, David and Carl Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. *Proceedings of SIGGRAPH 97.* pp. 199-208.

Mitchell, Joseph S. B. and Subhash Suri. Separation and Approximation of Polyhedral Surfaces. *Proceedings of 3rd ACM-SIAM Symposium on Discrete Algorithms.* 1992. pp. 296-306.

Murali, T. M. and Thomas A. Funkhouser. Consistent Solid and Boundary Representations from Arbitrary Polygonal Data. *Proceedings of 1997 Symposium on Interactive 3D Graphics.* April 27-30. pp. 155-162, 196.

O'Rourke, Joseph. *Computational Geometry in C.* Cambridge University Press 1994. 357 pages.

Plouffe, Simon and Neil James Alexander Sloan. The Encyclopedia of Integer Sequences. Academic Press 1995. pp. 587.

Rohlf, John and James Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real--Time 3D Graphics. *Proceedings of SIGGRAPH 94.* July 24-29. pp. 381-395.

Ronfard, Remi and Jarek Rossignac. Full-range Approximation of Triangulated Polyhedra. *Computer Graphics Forum.* vol. 15(3). 1996. pp. 67-76 and 462.

Rossignac, Jarek and Paul Borrel. Multi-Resolution 3D Approximations for Rendering. *Modeling in Computer Graphics.* Springer-Verlag1993. pp. 455-465.

Rossignac, Jarek and Paul Borrel. Multi-Resolution 3D Approximations for Rendering Complex Scenes. Technical Report RC 17687-77951. IBM Research Division, T. J. Watson Research Center. Yorktown Heights, NY 10958. 1992.

Schikore, Daniel and Chandrajit Bajaj. Decimation of 2D Scalar Data with Error Control. Technical Report CSD-TR-95-004. Department of Computer Science, Purdue University. 1995.

Schroeder, William J., Jonathan A. Zarge, and William E. Lorensen. Decimation of Triangle Meshes. *Proceedings of SIGGRAPH 92.* pp. 65-70.

Turk, Greg. Re-tiling Polygonal Surfaces. *Proceedings of SIGGRAPH 92.* pp. 55-64.

van Dam, Andries. PHIGS+ Functional Description, Revision 3.0. *Computer Graphics.* vol. 22(3). 1988. pp. 125-218.

Varshney, Amitabh. Hierarchical Geometric Approximations. Ph.D. Thesis. Department of Computer Science. University of North Carolina at Chapel Hill. 1994.

Xia, Julie C., Jihad El-Sana, and Amitabh Varshney. Adaptive Real-Time Level-of-Detail-Based Rendering for Polygonal Models. *IEEE Transactions on Visualization and Computer Graphics.* vol. 3(2). 1997. pp. 171-183.