Multiresolution Modeling: Survey & Future Opportunities

Michael Garland

Computer Science Department, Carnegie Mellon University
Pittsburgh, PA 15213, USA
garland@cs.cmu.edu

Abstract

For twenty years, it has been clear that many datasets are excessively complex for applications such as real-time display, and that techniques for controlling the level of detail of models are crucial. More recently, there has been considerable interest in techniques for the automatic simplification of highly detailed polygonal models into faithful approximations using fewer polygons. Several effective techniques for the automatic simplification of polygonal models have been developed in recent years. This report begins with a survey of the most notable available algorithms. Iterative edge contraction algorithms are of particular interest because they induce a certain hierarchical structure on the surface. An overview of this hierarchical structure is presented,including a formulation relating it to minimum spanning tree construction algorithms. Finally, we will consider the most significant directions in which existing simplification methods can be improved, and a summary of other potential applications for the hierarchies resulting from simplification.

1. Introduction

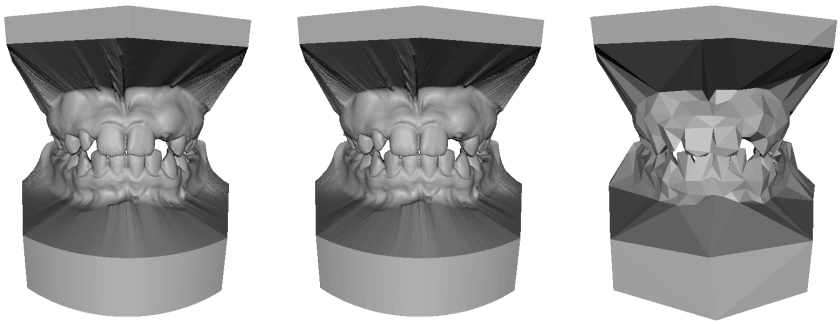
Advances in technology have provided vast databases of polygonal surface models, but these models are often very complex. Surfaces containing millions of polygons are not uncommon. Laser range scanners, computer vision systems, and medical imaging devices can produce models of intricate physical objects. Many companies now design products using computer-aided design (CAD) systems, resulting in very complex, highly detailed surfaces. Models produced by surface reconstruction and isosurface extraction methods can often be very densely sampled meshes with a uniform distribution of points on the surface. Applications in areas ranging from distributed virtual environments to finite element methods to movie special effects rely on polygonal surface models generated by these kinds of systems.

In all these applications, a tradeoff exists between the accuracy with which a surface is modeled and the amount of time required to process it. To achieve acceptable running times, we must often substitute simpler approximations of the original model. A model which captures very fine surface detail may in fact be desirable when creating archival datasets; it helps ensure that applications which later process the model have sufficient and accurate data. However, many applications will require far less detail than is present in the full dataset.

Consequently, there has been considerable interest in techniques for the automatic simplification of highly detailed polygonal models into faithful approximations using fewer polygons. Several effective techniques have been developed in recent years, and they provide valuable tools for tailoring large datasets to the needs of individual applications and for producing more economical surface models. Consider the model shown in Figure 1 at three different levels of detail. The original surface (a), containing nearly a half million triangular faces, is very densely over-sampled. By comparison, approximation (b) contains 86% fewer triangles, but its appearance is virtually identical to that of the original. For many applications, including interactive rendering, this approximation would be a suitable replacement. Approximation (c) contains a mere 1000 faces. While most of the fine detail of the surface is gone, the overall structure remains. An application trying to measure some gross property of the surface, say volume, could arrive at a reasonable initial estimate from this very simple model.

Techniques for controlling the run-time level of surface detail are also very important in real-time rendering systems. For any given system, available hardware capacity — such as frame buffer fill rates, transformation and lighting throughput, and network bandwidth — is essentially fixed. But the complexity of the scene to render may vary considerably. In order to maintain a constant frame rate, of say 30 Hz, we need to keep the level of detail in the scene from exceeding the available hardware capacity. This need arises at the

Published by the Eurographics Association ISSN 1017-4656.



(a) 424,376 faces (b) 60,000 faces (c) 1000 faces

Figure 1: *A scanned dental model with two approximations.*

low end, where computer games and distributed virtual environments must often operate on systems where available resources are highly constrained. The same need is present at the high end as well, where realistic simulation and scientific visualization systems typically have object databases that far exceed the capacity of even the most powerful graphics workstations.

In order to manage the level of detail of an object, we need to represent it as a *multiresolution model* — a surface representation which supports the reconstruction of various approximations which can accommodate a wide range of viewing contexts. As an example, consider a surface model such as the one shown in Figure 2b, containing about 100,000 triangular faces. Suppose the viewer is closely examining the surface as in Figure 2a; the screen is filled by a small portion of the total surface. Under these conditions, the area being examined may well have too few triangles while the rest of the model, which falls beyond the field of view, can be ignored. Now consider a view like that in Figure 2c; the model appears as a few small dots. In this case, the model has far too many polygons for the number of pixels being rendered. Not only must a multiresolution model allow us to extract approximations suitable for these three diverse circumstances, but it must also allow us to change the level of detail without excessive overhead. If the time necessary to switch to and render a lower level of detail exceeds the time necessary to simply render a higher level, we would gain no advantage from the multiresolution model.

This report begins with an overview of the problem of surface simplification. Following this is a survey of the algorithms which have been developed. Simplification algorithms based on iterative contraction are of particular interest because they have been used to construct multiresolution surface representations, and we will consider these described in the subsequent sections. Finally, the most important directions in which existing techniques can be extended and improved are explored.

2. Polygonal Surface Simplification

A polygonal model M is composed of a fixed set of vertices $V = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_r)$ and a fixed set of faces $F = (f_1, f_2, \ldots, f_n)$. It provides a single fixed resolution representation of an object. Without loss of generality, we can assume that the model consists entirely of triangular faces, since any non-triangular polygons may be triangulated in a pre-processing phase. To streamline the discussion, I will assume that models do not contain isolated vertices and edges which are not part of any triangle. For best results in practice, they should be maintained during simplification and rendered at run time[66, 72, 79, 83]. For most algorithms, the only effect of isolated vertices and edges is to complicate the implementation; the underlying algorithms remain the same.

Suppose we have a polygonal model M and we would like an approximation M'. While this approximation will have fewer polygons than the original, it should also be as similar as possible to M. The goal of polygonal *surface simplification* is to automatically produce such approximations. User supervision is generally not feasible. Simplification is naturally targeted towards large and complex datasets which would be very cumbersome to manipulate manually.

A common application of simplification is reducing the complexity of very densely over-sampled models. Models generated by scanning devices and isosurfaces extracted by algorithms such as marching cubes[64] often benefit from simplification. Such models are often uniformly tessellated — an artifact of the nature of most reconstruction algorithms. Triangle density is the same in both flat and highly curved regions. It is usually preferable to be more economical with triangle coverage; local triangle density should adapt to local curvature. The number of triangles can often be reduced by 50 percent or more, and the result will be nearly identical to the original.

More generally, we may want to produce an approximation which is tailored for a specific use. For instance, we might want to produce an approximation of the dragon model in Figure 2 suitable for viewing conditions such as

(a) Close inspection (b) Normal viewing (c) Far in distance

Figure 2: *The same model used in widely differing contexts.*

depicted in Figure 2c. Given the output resolution at which it is displayed, we could probably achieve identical results with fewer than 100 triangles.

In other cases, we would like something more flexible than a single fixed approximation. Suppose that, during an interactive session, a user was viewing the model shown in Figure 2 in the diverse contexts shown. There is no single set of polygons which is appropriate for all of these different viewing conditions. Instead, we would like to have multiple different approximations available, selecting the best one for the current viewing conditions. Rather than a fixed resolution model, we would like a multiresolution model.

A *multiresolution model* is a model representation which captures a wide range of approximations of an object and which can be used to reconstruct any one of them on demand. The cost of reconstructing approximations should be low because we will often need to use many different approximations at run time. It is also important that a multiresolution representation have roughly the same size as the most detailed approximation alone, although a small constant factor increase in size is acceptable. Since rendering systems are of primary concern here, the appropriate surface approximation for a particular model will depend upon current viewing conditions (e.g., distance to the viewer). The appropriate level of detail may also vary considerably over the surface. As we will see, surface simplification algorithms can be used to construct multiresolution representations from the initial surface geometry.

3. Evaluating Surface Approximations

The primary aim of simplification is to produce a surface approximation which is as similar as possible to the original. In order to assess the quality of an approximation, we need some means of quantifying the notion of similarity. Given a polygonal model M and an approximation M', we would like an error metric for which the value $E(M,M')$ measures the *approximation error* of M'.

In general, the preferred similarity criteria will be application-dependent. Rendering systems are one of the primary application areas of interest in the simplification literature, and similarity of appearance is the natural choice for rendering applications[43]. However, in almost all cases, researchers in the field of simplification have chosen to use similarity of shape as the primary criterion for evaluating approximation quality. Not only do shape-based metrics appear to be more computationally convenient, but they are also more appropriate in non-rendering applications such as finite element analysis. Nevertheless, since similarity of appearance is often what we would like to achieve, it is important to consider how we might define it.

3.1. Similarity of Appearance

For a given view, the appearance of a model is determined by the corresponding raster image which a renderer would produce. If I_1 and I_2 are $m \times m$ RGB raster images of models M_1 and M_2, we can define the difference between them as the average sum of squared differences between all corresponding pixels

$$\|I_1 - I_2\|_{img} = \frac{1}{m^2} \sum_u \sum_v \|I_1(u,v) - I_2(u,v)\|^2 \qquad (1)$$

where $\|I_1(u,v) - I_2(u,v)\|$ is the Euclidean length of the difference of the two RGB vectors $I_1(u,v)$ and $I_2(u,v)$. While there are many more elaborate metrics for comparing images[80], this very simple definition appears suitable for the simplification domain. If M_2 is a good approximation of M_1 for the given view, then $\|I_1 - I_2\|_{img}$ should be small. Given this image metric, we can characterize the total difference in appearance between two models by integrating these differences over all possible views. Naturally, we would expect in practice to merely sample these per-image differences over some finite set of viewpoints.

A simplification algorithm guided by an appearance-based metric of this type has several interesting characteristics. Its primary advantage is that it directly measures similarity of appearance, which is precisely what we are interested in preserving in rendering systems. It also allows us to discard occluded details. Suppose that we have some probability distribution on the possible viewpoints that will occur at run time. Any features which are occluded in all possible views can be immediately removed. For example, if we have a complex model of a submarine and we know that the viewpoint will always be outside the hull, we can remove all

polygons on the interior without introducing any error into the approximation.

While appearance-based metrics have some appealing benefits, they also raise some difficult issues. In particular, the foremost problem is the need to adequately sample the possible viewpoints. If we neglect some important part of the viewpoint space, we may very well remove perceptually significant features. And since each sample may involve an expensive rendering step, we cannot make many samples. Indeed, rendering the models for comparison is likely to be quite expensive; simplification is generally performed on models which are prohibitively expensive to render in the first place.

3.2. Geometric Approximation Error

While similarity of appearance is the foremost goal for approximations used in rendering systems, it is generally easier to consider geometric measures of error instead. We can use geometric similarity as a proxy for visual similarity. By striving to produce geometrically faithful results, we can also produce approximations that will be useful in application domains other than rendering.

3.2.1. Function Approximation

Before considering the full problem of measuring approximation error for polygonal models, let us examine a much simpler case: function approximation. This area of study has a long history in the mathematics literature, and it will provide us with some intuition which will carry over into the polygonal domain.

The two most commonly used error metrics are the L_∞ and L_2 norms[73]. Suppose a real-valued function $f(t)$, an approximation $g(t)$, and an interval of interest $[a,b]$ are given. The L_∞ norm, which measures the maximum deviation between the original and the approximation is defined by

$$\|f-g\|_\infty = \max_{a\le t\le b} |f(t)-g(t)| \tag{2}$$

The L_2 norm defined by

$$\|f-g\|_2 = \sqrt{\int_a^b (f(t)-g(t))^2\,dt} \tag{3}$$

provides a measure of the average deviation between the two functions. A piecewise-linear approximation $g(t)$ composed of n segments is called *optimal* if there is no other n-segment approximation having a smaller error. The L_∞ norm is generally regarded as a stronger measure of error in the function approximation literature[73]. Because it provides a global absolute bound on the distance between the original and the approximation, it is often easier to prove quality guarantees. However, the L_2 norm is somewhat more general. Certain functions, such as $f(t)=t^{-1/3}$ on the interval $[0,1]$, have a well-defined L_2 norm but no L_∞ norm.

The L_∞ norm is most useful because it provides absolute distance bounds which are a useful error guarantee. However, it can be overly sensitive to any noise that might be present in the original model. In contrast, the L_2 norm better reflects overall fit, but may discount large, but highly localized, deviations. For example, consider the curves shown in

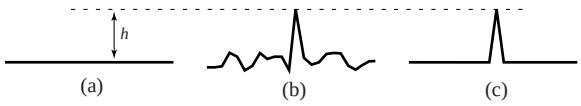


Figure 3: *Two approximations to the same base curve.*

Figure 3. The two approximations (b) and (c) have the same L_∞ error, namely the distance h. However, curve (c) certainly seems to be a better overall approximation. The L_2 norm would assign a higher error to curve (b) than curve (c). Now consider the curves shown in Figure 4. The base

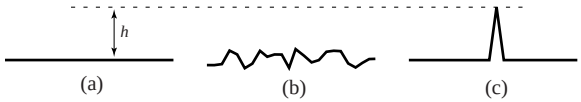


Figure 4: *Alternative approximations to the same base curve.*

curve (a) and the approximation (c) are the same as before. We can choose the size of the tent in approximation (c) such that both (b) and (c) have the same L_2 error. However, there are certainly cases in which (b) is a preferable approximation given that (c) deviates significantly further from the base curve. Also, suppose that we allow the width ε of the tent in (c) to approach 0. The L_2 error of (c) will also approach 0, while its L_∞ error will remain h.

3.2.2. Surface Approximation

We can formulate surface-based analogs of both the L_2 and L_∞ function approximation norms. First, we need to generalize the notion of deviation between the original and the approximation. In the functional case outlined in the previous section, we measured deviation as the vertical distance $|f(x)-g(x)|$. When comparing general surfaces, there is no single distinguished direction along which to measure distances. Instead, we will measure distances between closest pairs of points. The distance from a point $\mathbf{v}$ to the model M is defined to be the distance to the closest point $\mathbf{w}$ on the model:

$$d_{\mathbf{v}}(M) = \min_{\mathbf{w}\in M} \|\mathbf{v}-\mathbf{w}\| \tag{4}$$

where $\|\cdot\|$ is the usual Euclidean vector length operator.

One commonly used geometric error measure is the Hausdorff distance[74], an analog the L_∞ metric, which can be defined as

$$E_{max}(M_1,M_2) = \max\left(\max_{\mathbf{v}\in M_1} d_{\mathbf{v}}(M_2), \max_{\mathbf{v}\in M_2} d_{\mathbf{v}}(M_1)\right) \tag{5}$$

The Hausdorff error measures the maximum deviation between the two models. If $E_{max}(M,M') < \varepsilon$, then we know that every point of the approximation is within ε of the original surface and that every point of the original is within

ε of the approximation. Along similar lines, we can define E_{avg}, an analog of the L_2 metric, which measures the average squared distance between the two models as

$$E_{avg}(M_1,M_2) = \frac{1}{w_1}\int_{\mathbf{v}\in M_1} d^2_{\mathbf{v}}(M_2) + \frac{1}{w_2}\int_{\mathbf{v}\in M_2} d^2_{\mathbf{v}}(M_1) \tag{6}$$

where w_1, w_2 are the surface areas of M_1, M_2. Note the symmetric construction of both E_{max} and E_{avg}. It is not sufficient to simply consider every point on M_1 and find the closest corresponding point on M_2. We must also do the same for every point on M_2.

In practice, these error metrics can be prohibitively expensive to compute exactly. It is common to formulate approximations of these ideal metrics based on sampling the distance $d_{\mathbf{v}}$ at a discrete set of points X_1, X_2 on the surfaces of M_1, M_2, respectively. These sets should, at a minimum, contain all the vertices of their respective models.

Some simplification algorithms have used these metrics to directly guide the construction of approximations. For instance, the E_{dist} energy term used by Hoppe *et al.* [51, 47] is very similar to E_{avg}. It differs only in omitting the averaging terms and by measuring the asymmetric distance from M_1 to M_2. And most published results which attempt to assess the objective quality of approximations[9, 63, 31] use one or both of these metrics, which can be conveniently calculated using the Metro[10] model comparison tool.

To further reduce the cost of evaluating these error metrics, others[88, 57, 7, 58] define *localized* versions of the underlying distance function $d_{\mathbf{v}}$. As defined above, $d_{\mathbf{v}}(M)$ finds the distance of $\mathbf{v}$ to the closest point on M. However, we can restrict our search to a small region R of M and evaluate the localized distance $d_{\mathbf{v}}(R)$. Many surface simplification algorithms produce a correspondence between vertex neighborhoods on the approximation and regions on the original surface. Thus, we can quite naturally define a localized error metric based on measuring distances to these corresponding regions.

4. Survey of Polygonal Simplification Methods

The problems of surface simplification and multiresolution modeling have received increasing attention in recent years. The underlying concept of multiresolution surface models is not particularly new; Clark[11] discussed the general idea twenty-five years ago. However, with the exception of work done on simpler objects such as curves and height fields, most of the results in the field are fairly recent.

This section surveys some of the notable simplification algorithms. It is by no means exhaustive; rather, I have tried to select representative algorithms. The survey by Heckbert and Garland[44] provides more complete coverage of several algorithms, particularly for height field simplification. Data on the relative performance of various simplification algorithms can be found in the survey of Cignoni *et al.* [9] and elsewhere[63, 7, 31].

The two most common methodologies in surface simplification are *refinement* and *decimation*. A refinement algorithm is an iterative algorithm which begins with an initial coarse approximation and adds elements at each step. Essentially the opposite of refinement, a decimation algorithm begins with the original surface and iteratively removes elements at each step. Both refinement and decimation share a very important characteristic: they seek to derive an approximation through a *transformation* of some initial surface.

An important distinction between algorithms is whether they perform topological simplification on the surface. Most methods fall into one of three categories. Some specifically *prohibit* any topological alteration[13]. The majority of algorithms simplify the topology *implicitly*. In other words, they make choices based on geometric criteria, but they may simplify the topology as a side-effect. Finally, some algorithms *explicitly* consider the simplification of surface topology[42, 23] along with geometric simplification.

Clearly, there are applications in which topological simplification must be prevented. For example, in some medical imaging applications, preserving a hole in the heart wall may be much more important than preserving the exact shape of the surface. However, in most rendering applications, not only can we safely simplify the topology of models, it is often desirable to do so. Consider a model of a sponge. When examined closely, the intricate structure of holes in the sponge is a visually important feature. However, when viewed from a distance, these holes are imperceptible. The entire sponge can be adequately approximated by a simple block, particularly if we can apply an appropriate texture image to the block that simulates the texture of the original.

Surface models are often assumed to be *manifolds*[45] — surfaces for which all points have neighborhoods topologically equivalent to a disk. Many surfaces encountered in practice tend to be manifolds, and many surface-based algorithms require manifold input. It is possible to apply such algorithms to non-manifold surfaces by cutting the surface into manifold components and subsequently stitching them back together[40]. However, it can be advantageous for simplification algorithms to explicitly allow non-manifold surfaces, particularly when topological simplification is allowed. Imagine a model of a metal plate with many small holes drilled in it. The common contraction-based approach for removing a hole from this model would begin by collapsing one end of the hole into a single point, resulting in a non-manifold vertex neighborhood. While it is possible to explicitly cut and re-stitch the surface during simplification[85], this can add substantial complexity to the algorithm.

Before considering general surface simplification, let us briefly examine two lower-dimensional problem domains — the simplification of curves and height fields.

4.1. Curves and Functions

Not surprisingly, the simplification of functions and curves has the longest history. Within this area, the work on the simplification of piecewise-linear curves is most closely related

to the problem of simplifying polygonal surfaces. It has developed in, among other fields, cartography (under the name "generalization"), computer vision, and computer graphics.

Suppose that we have a piecewise-linear curve with n vertices, and we would like an approximation with $m < n$ vertices. For these simple geometric objects, we can actually construct optimal approximations — those which use the minimum number of vertices necessary to achieve a given error tolerance. Algorithms have been developed for constructing L_∞–optimal approximations of functions[53], plane curves[53], and 3-D space curves[52]. However, finding these optimal solutions quickly becomes expensive. While the algorithm for finding optimal approximations of functions has a time complexity of $O(n)$, the algorithms for plane curves and space curves have much higher complexities of $O(n^2 \log n)$ and $O(n^3 \log m)$, respectively. This makes them rather impractical for very large datasets.

Perhaps the most widely used algorithm for curve simplification is a simple refinement algorithm, commonly referred to as the Douglas–Peucker[20] algorithm. This algorithm begins with with some minimal approximation, normally a single line segment from the first to last vertex. This segment is split at the point on the original curve which is furthest from the approximation. Each of the two new subsegments can be recursively split until the approximation meets some termination criteria. This is evidently a rather natural algorithm for curve approximation, since it was independently invented by a number of people[44].

Decimation algorithms, which in essence are the Douglas–Peucker refinement algorithm in reverse, have also been developed[4, 61]. While the quality of their results is at least as good, they tend to be less efficient. Broadly speaking, the time and memory requirements of these iterative algorithms depend on the size of the current approximations being tracked through successive iterations. The refinement approach begins with a minimal approximation and gradually refines it, rather than starting with the full model and gradually simplifying it. Therefore, the intermediate approximations which it constructs tend to be fairly small, particularly if the target approximation is only say 10% or less the size of the original.

4.2. Height Fields

Height fields are among the simplest types of surface. They can be defined as the set of points satisfying an equation of the form $z = f(x,y)$ where x and y range over a subset of the Cartesian plane.

In contrast to curve simplification, it is not feasible to construct optimal approximations of height fields. Agarwal and Suri[2] have shown that computing an L_∞–optimal approximation of a height field is NP-Hard[14]. In other words, an optimal approximation cannot be computed in less than exponential time. Polynomial time approximation algorithms have been developed[2, 1] which can generate approximations with some L_∞ error ε using $O(k \log k)$ triangles, where there are k triangles in the optimal approximation. However, their running time is at best $O(n^2)$ for a height field with n input points — too high for practical use on large datasets.

Refinement is the most popular approach for terrain approximation, as it was in the case of curves. One particularly common algorithm begins with a minimal approximation and iteratively inserts the point where the approximation and the original are farthest apart. This *greedy insertion* technique has received significant attention and has been independently rediscovered repeatedly[44]. Incremental Delaunay triangulation[41] is often used to triangulate the selected vertices, but other data-dependent triangulations can produce approximations with lower error[75, 32].

Decimation algorithms for simplifying height fields have also been proposed[60, 82]. However, as was the case with curves, they do not seem to be as widely used as refinement methods. Depending on the exact algorithms chosen, decimation may produce higher quality results than refinement. But the greater speed and smaller memory requirements of refinement seem to have made it the more common choice.

4.3. Surfaces

Successful algorithms for simplifying curves and height fields were developed twenty years ago[20, 27], but the work on more general surface simplification is much more recent. Note that, since height fields are a special case of general surfaces, optimally approximating a surface is NP-Hard.

4.3.1. Manual Preparation

The traditional approach to multiresolution surface models has been manual preparation. A human designer must construct various levels of detail by hand. Manual techniques have been in use in the flight simulator field for decades[15], and similar techniques are in use today by game developers[90]. While this process may be aided by a specially designed surface editor[28], it can still be a time-consuming and difficult task. The general goal of the work done on surface simplification has been to automate this task.

4.3.2. Polyhedral Refinement

Only a small number of algorithms for progressively refining polygonal surfaces have been proposed[25, 18, 19]. While refinement has traditionally been the method of choice for approximating curves and height fields, decimation has been much more widely used for simplifying more general surfaces. Perhaps the primary difficulty with refinement in this case involves actually constructing the base approximation. If we limit ourselves to refining via simple subdivision rules, then the initial approximation must necessarily have the same topology as the original model. However, not only does this prevent us from simplifying the topology, but it is not always easy to discover the topology of the input surface.

4.3.3. Vertex Clustering

Vertex clustering methods[79, 66, 83] spatially partition the vertex set into a set of clusters and unify all vertices within the same cluster. They are generally very fast and work on arbitrary collections of triangles. Unfortunately, they can often produce relatively poor quality approximations.

The simplest clustering method is the uniform vertex clustering algorithm described by Rossignac and Borrel[79]. A simple example of uniform clustering is shown in Figure 5. The vertex set is partitioned by subdividing a bounding

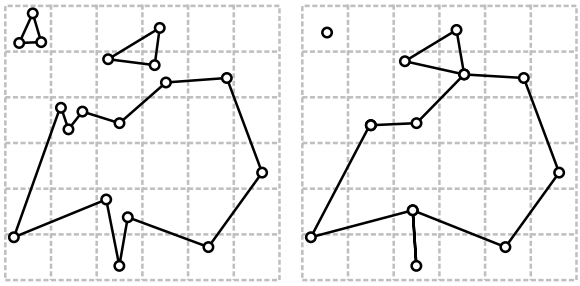


Figure 5: *Uniform clustering in two dimensions.*

box on a regular grid, and the new representative vertex for each cell is computed using cheap heuristics based on criteria such as edge length. This process can be implemented quite efficiently. The algorithm also tends to make substantial alterations to the topology of the original model. Looking at Figure 5, we can see that the triangle in the upper left corner is reduced to a point, and two separate components along the top are joined together. Note that, much like uniform subsampling of images, the results of this algorithm can be quite sensitive to the actual placement of the grid cells. It is also incapable of simplifying features larger than the cell size. A planar rectangle consisting of many triangles all larger than the cell size will not be simplified at all, even though it can be approximated using two triangles without error.

The most natural way to extend uniform clustering is to use an adaptive partitioning scheme such as octrees[67]. Centering cells around important vertices, rather than merely partitioning space, can also lead to improved approximations[66].

Clustering methods tend to work well if the original model is highly over-sampled and the required degree of simplification is not too great. They also tend to perform better when the surface triangles are smaller than the cell size. Since no vertex moves further than the diameter of its cell, clustering algorithms provide guaranteed bounds on the Hausdorff approximation error sampled at the vertices of M and M'. However, to achieve substantial simplification, the required cell size increases quite rapidly, making the error bound rather weak. In particular, at more aggressive simplification levels, the quality of the resulting approximations can quickly degrade.

4.3.4. Region Merging

A handful of simplification algorithms[54, 46, 37] operate by merging surface regions together. For example, the "superfaces" algorithm of Kalvin and Taylor[54] partitions the surface into disjoint connected regions based on a planarity criterion. Each region is replaced by a polygonal patch whose boundary is simplified, and the resulting region is retriangulated. These algorithms are generally restricted to manifold surfaces, and do not alter the topology of the model. The algorithms of Hinker & Hanson[46] and Gourdon[37] appear to be best suited for smooth surfaces that are not highly curved. However, Kalvin and Taylor's algorithm seems to produce good quality results, and it provides bounds on the approximation error.

Region merging techniques do not seem to have become widespread. This may well be because they are somewhat more complicated to implement in comparison to other algorithms without offering superior approximations. And in contrast to iterative edge contraction, they do not produce a natural multiresolution representation.

4.3.5. Wavelet Decomposition

Wavelet methods[91] provide a fairly clean mathematical framework for the decomposition of a surface into a base shape plus a sequence of successively finer surface details. Approximations can be generated by discarding the least significant details. They have been used quite successfully for producing multiresolution representations of signals and images[68, 91].

Lounsbery *et al.* [65] developed a method for generating a wavelet decomposition of surfaces with subdivision connectivity. Consequently, the resulting approximations may be relatively far from optimal because they may use a large number of triangles simply to preserve subdivision connectivity. Wavelet decompositions are also generally unable to resolve creases on the surface unless they fall along edges in the base mesh; Hoppe[47] provides a good illustration of this effect. Eck *et al.* [22] developed a procedure for producing a subdivision mesh from a surface with arbitrary connectivity. However, this pre-process introduces some level of error into the base shape, although this error can be limited by a specified tolerance value. Like other subdivision-based schemes, wavelet methods cannot easily construct approximations with a topology different from the original surface.

4.3.6. Vertex Decimation

One of the more widely used algorithms is vertex decimation, an iterative simplification algorithm originally proposed by Schroeder *et al.* [86]. In each step of the decimation process, a vertex is selected for removal, all the faces adjacent to that vertex are removed from the model, and the resulting hole is retriangulated (see Figure 6). Since this retriangulation requires a projection of the local surface onto a plane, these algorithms are generally limited to manifold surfaces. The fundamental operation of vertex deletion is also incapable of simplifying the topology of the model.

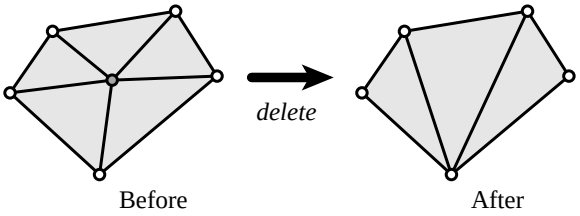


Figure 6: *A vertex is removed and the resulting hole is retriangulated.*

Schroeder[85] was able to lift these restrictions by incorporating cutting and stitching operations into the simplification process. The original vertex decimation algorithm[86] used a fairly conservative estimate of approximation error. More recent methods[88, 57, 7, 58] use more accurate error metrics, like the localized Hausdorff error. They maintain links between points on the original surface and the corresponding neighborhood on the approximation, and the distances between these points and the associated faces define the approximation error.

The algorithm of Schroeder *et al.* is reasonably efficient, both in time and space, but it seems to have some difficulty preserving smooth surfaces (c.f. Schroeder[85] Fig. 9). The body of the turbine blade is initially smooth, but becomes quite rough during simplification. While the other vertex decimation algorithms produce higher quality results, they are substantially slower and consume more space.

This methodology of vertex decimation is in fact closely related to iterative contraction (discussed in the next section). In particular, note that the vertex removal pictured in Figure 6 can just as easily be accomplished by contracting the bottom edge. Removing a vertex by edge contraction[85] is generally more robust than projecting the neighborhood onto a plane and retriangulating[86].

4.3.7. Iterative Contraction

The final major class of algorithms is based on the iterative contraction of vertex pairs[3, 33, 34, 39, 63, 59, 47, 72, 77]. Some algorithms have been formulated using face contraction[35], but since a face can be contracted by contracting two of its edges, the distinction is minor. Contraction algorithms have become increasingly popular in recent years, and I will focus on them in greater detail.

A vertex pair contraction, denoted $(\mathbf{v}_i, \mathbf{v}_j) \rightarrow \bar{\mathbf{v}}$, modifies the surface in three steps:

1. Move the vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ to the position $\bar{\mathbf{v}}$;
2. Replace all occurrences of $\mathbf{v}_j$ with $\mathbf{v}_i$;
3. Remove $\mathbf{v}_j$ and all faces which become degenerate — that no longer have three distinct vertices.

The first step modifies the geometry of the surface, and the second step modifies the connectivity of its mesh. Unless the topology is explicitly preserved, this may also implicitly alter the topology of the surface (e.g., by closing holes). The final step simply removes elements of the surface which are no longer needed.

When an edge is contracted, its end points are replaced by a single point and triangles which degenerate to edges are removed (see Figure 7). Also note that the fundamental operation of contraction does not require the immediate neighborhood to be manifold. In fact, contraction can be applied to any simplicial complex. Thus contraction-based algorithms can more conveniently deal with non-manifold surfaces than vertex decimation algorithms.

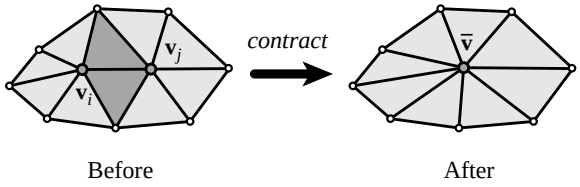


Figure 7: *Edge $(\mathbf{v}_i, \mathbf{v}_j)$ is contracted. The darker triangles become degenerate and are removed.*

General pair contractions, where the vertices $\mathbf{v}_i, \mathbf{v}_j$ need not be connected by an edge, have been proposed to provide a means of merging separate topological components during simplification[33, 72]. The effect of an edge contraction such as the one pictured in Figure 7 is to remove one vertex and one or more faces from the model. In contrast, contracting a non-edge pair will remove one vertex and join previously unconnected regions of the surface (see Figure 8). In general, performing topological simplification via pair contractions requires the algorithm to support non-manifold surfaces. At the instant when two separate components are joined together, a non-manifold region will almost certainly be created. For instance, the resulting neighborhood pictured in Figure 8 is non-manifold because the faces surrounding $\bar{\mathbf{v}}$ form two separate fans.

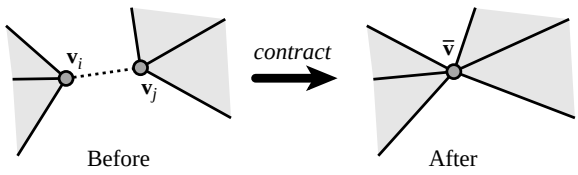


Figure 8: *Non-edge pair $(\mathbf{v}_i, \mathbf{v}_j)$ is contracted, joining previously unconnected areas. No triangles are removed.*

To perform the contraction $(\mathbf{v}_i, \mathbf{v}_j) \rightarrow \bar{\mathbf{v}}$, we must choose a target position $\bar{\mathbf{v}}$. *Subset placement*, where we select one of the endpoints as the target position, is the simplest strategy that we can adopt. We can often produce better approximations using *optimal placement* where $\bar{\mathbf{v}}$ is allowed to float freely in space in order to minimize some error metric. This will generally result in higher quality approximations, but as we will see, the storage requirements for multiresolution representations will be higher.

Most of the iterative contraction algorithms which have been developed follow a simple greedy procedure to select a sequence of edge contractions. Each pair being considered for contraction is assigned a "cost". The way in which this cost is determined is the primary differentiating factor between algorithms of this type. Generally, this cost of contraction is meant to reflect the amount of error introduced into the approximation by the contraction of the pair in question. At each iteration, the lowest cost pair is contracted. Once a contraction is added to the sequence, it is never reconsidered.

Figure 9 illustrates the simplification of a small model by a series of edge contractions. The original model M^0 is a square with a hole cut through it; the total model consists of eight triangular faces. By contracting the highlighted pair of vertices $(\mathbf{v}_5, \mathbf{v}_8)$, we produce the approximation M^1. Now, by contracting the highlighted pair $(\mathbf{v}_6, \mathbf{v}_7)$, we arrive at the approximation M^2; notice that the hole is now closed. Assuming that contractions were selected based on purely geometric criteria, this example illustrates how the topology of the model may be implicitly simplified by iterative contraction.

Hoppe's algorithm[47] for constructing progressive meshes is based on minimization of an energy function. One of its primary components is a geometric error term very much like E_{avg}. The algorithm maintains a set of sample points on both the original surface, and the distances between these points and the corresponding closest points on the approximation determine the geometric error. This algorithm produces some of the highest quality results among currently available methods. However, the price of this precision is a very long running time. Hoppe reports[48] running times of around an hour for a model of about 70,000 faces. The original algorithm could simplify topology by closing holes in the surface, and the extension by Popović and Hoppe[72] can join unconnected regions. The mesh optimization algorithm of Hoppe *et al.* [51] is an earlier form of the progressive mesh construction algorithm[47]. It performs explicit search rather than simple greedy contraction. Consequently, it exhibits even longer running times, but may produce the highest quality results.

The algorithm developed by Guéziec[38, 39] maintains a tolerance volume around the approximation such that the original surface is guaranteed to lie within that volume. The volume itself is defined by spheres located at each vertex of the approximation. The convex combination of these spheres over the faces of the model creates so called "fat triangles" which comprise the tolerance volume. Vertices of the approximation are positioned to preserve the volume of the object. While this algorithm appears somewhat slow, it is faster than Hoppe's algorithm, and it appears to generate good quality results.

In the algorithm of Ronfard and Rossignac[77], each vertex in the approximation has an associated set of planes, and the error at that vertex is defined by the maximum of squared distances to the planes in this set. These sets are merged when vertices are contracted together. While it is necessarily less precise, Ronfard and Rossignac[78] show that localized Hausdorff error bounds can be derived from their metric. The resulting approximations appear to have generally good quality, and the algorithm is fairly efficient compared to the more exact algorithms.

The quadric error metric developed by Garland and Heckbert[33, 31] also defines error in terms of distances to sets of planes. However, it uses a much more efficient implicit representation of these sets. Each vertex is assigned a single symmetric 4×4 matrix which can measure the sum of squared distances of a point to all the planes in the set. Under suitable conditions, the eigenvectors and eigenvalues of a quadric accumulated over a smooth surface region are determined by the principal directions and principal curvatures of the surface[31]. While the quadric metric sacrifices some precision in assessing the approximation error, the resulting algorithm can produce quality approximations very rapidly. For example, only 7 seconds are require to simplify a model containing 70,000 triangles.

The "memoryless" algorithm recently developed by Lindstrom and Turk[63] is interesting in that, unlike most algorithms, it makes decisions based purely on the current approximation alone. No information about the original shape is retained. They use linear constraints, based primarily on conservation of volume, in order to select an edge for contraction and the position at which the remaining vertex will be located. In fact, the "volume optimization" component of this metric is identical to a variation on the quadric error metric[31]. The reported results suggest that it can generate good quality results, and that it is fairly efficient, particularly in memory consumption. By way of comparison, simplifying a 70,000 triangle model requires about 2.5 minutes.

One of the major benefits of iterative contraction is the hierarchical structure that it induces on the surface. This quite naturally leads to a useful multiresolution surface representation which we will explore in subsequent sections.

4.4. Material Properties

Much of the work done on simplifying surfaces has focused exclusively on the geometry of the surface. But in practice, models may often have various material properties. For example, models intended for use in rendering systems might often have color and texture attached to the surface.

Certain *et al.* [5] outlined a technique for adding surface color to a wavelet surface decomposition. Hoppe[47] explicitly included attributes in his error metric which supports both per-vertex (or scalar) attributes and per-face (discrete) attributes. As was the case with geometric fidelity, this algorithm seems to produce high quality results at the cost of rather high running times. The quadric error metric can also be generalized[34] to consider material properties.

An alternative approach is to treat attributes as maps on the surface. In this case, we would focus on reparameterizing the maps rather than preserving actual attribute values.

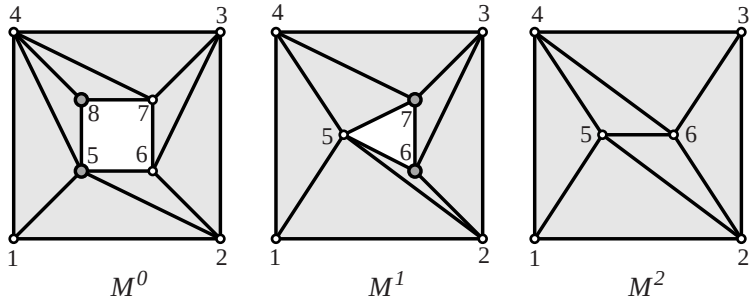


Figure 9: *Simplification of a simple planar object.*

Cohen *et al.* [12] developed an algorithm capable of reparameterizing both texture and normal maps as a surface is simplified. Others — including Maruya[69], Soucy *et al.* [87], and Cignoni *et al.* [8] — decouple attributes and geometry even further. They first compute a simplified surface, without regard for the surface attributes. Given the final approximation, they resample the attributes of the original into a texture map on the approximation.

The primary attraction of this approach is that texturing is widely supported in rendering systems, and it decouples the resolution of the geometry (vertex count) from the resolution of the attributes (texture dimensions). If the target rendering system supports bump mapping or normal mapping, this can be an effective technique for retaining high levels of visual fidelity[8, 12]. However, current approaches for generating this mapping into texture space produce highly fragmented textures — neighboring triangles on the surface will not occupy neighboring regions in the texture. The unfortunate consequence is that we cannot construct image pyramids for the resulting textures because neighboring texels may be mapped onto entirely separate parts of the model. It also makes this approach ill-suited for multiresolution modeling applications, since each level of detail would require its own individual texture map.

5. Discrete Multiresolution Models

The simplest method for creating multiresolution surface models is to generate a set of increasingly simpler approximations. For any given frame, a renderer could select which model to use and render that model in the current frame[29]. In this case, we would be using a series of *discrete levels of detail;* our multiresolution model would consist of the set of levels — such as in Figure 10 — and the threshold parameters to control the switching between them. The simplicity of the discrete multiresolution approach is its primary attraction. If we can produce good surface approximations, we can produce discrete multiresolution models.

5.1. Level of Detail Blending

Simply switching levels of detail between frames by substituting one whole discrete model will often incur negligible overhead at display time. Many systems are designed to transmit all the geometry of the world to the graphics subsystem at each frame. Thus, ignoring external factors such as paging the relevant geometry into main memory, switching levels of detail simply involves transmitting different geometry for the current frame. If the graphics subsystem supports caching several levels of detail in pre-compiled display lists, we might not even have to transmit any new geometry at all. However, it can potentially cause significant visual artifacts. In most cases, the number of polygons in the two models will differ significantly, and this may cause their appearances in the output image to be significantly different as well. Making such a substantial change in appearance between two consecutive frames can lead to "popping" artifacts. This effect can be mitigated by extending the level of detail transition over several frames and using alpha blending to perform a smooth cross-dissolve between the images of the two models[30]. Visual artifacts are still evident, but are much less objectionable. Unfortunately, this technique causes the overall rendering cost to increase during transitions since the system must render two levels of the model at the same time.

5.2. Geomorphing

Another alternative is to smoothly interpolate between the geometries of two consecutive levels over several frames. This *geomorphing* technique has been used in line-based[70] and terrain-specific[15, 26] systems for some time. Provided that we have a correspondence between the vertices of successive levels of detail, we can also apply geomorphs to general polygonal surfaces[47]. Suppose that we are transitioning between a model M and a simpler model M' and that every vertex $\mathbf{v} \in V$ corresponds to a vertex $\phi(\mathbf{v}) \in V'$. Iterative contraction algorithms generate exactly this sort of correspondence. During the transition, the model will have the same mesh connectivity as the more complex model M, but its geometry will vary continuously between that of M and M'. For each vertex $\mathbf{v}$ in M, we substitute an interpolated position $t\mathbf{v} + (1-t)\phi(\mathbf{v})$. At $t = 0$, the model will have exactly the same shape as M, and at $t = 1$, the model will have the shape of M'. By moving t between 0 and 1 over several successive frames, we can smoothly transition between the two

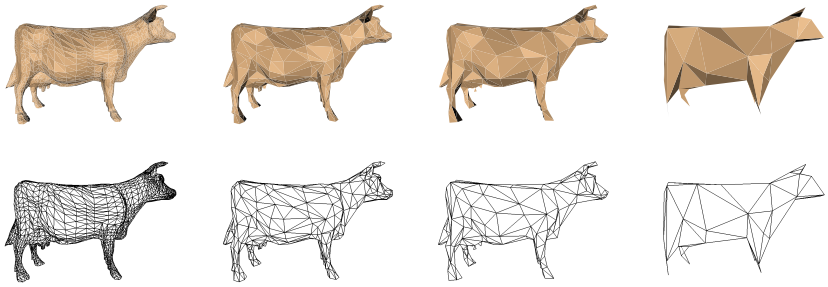


Figure 10: *Fixed set of levels of detail for a cow model.*

models. Unlike the alpha blending approach, the geometric complexity of the object being rendered is the same as M. While this is less than ideal, because we have determined that the required level of complexity is only that of M', it is certainly lower than the combined size of both levels. However, there is the additional overhead of interpolating the vertex positions for each frame. Whether this is less expensive than blending the images of M and M' may depend on the hardware architecture.

The principal drawback of discrete multiresolution models is that the levels of detail available at run time are rather limited. A renderer would be forced to pick one of our pregenerated models, even if it needed an intermediate level. Thus, the renderer would either have to pick a model without sufficient detail (and sacrifice image quality) or choose a model with excess detail (and waste time). Unless the model is divided into interchangeable blocks, the renderer would also be unable to vary the level of detail over different parts of the model. Suppose, for example, that we are standing near the corner of a building looking down one side. At the corner nearest the viewpoint, the renderer needs a high level of detail to maintain image quality. However, as the walls recede into the distance, the renderer could potentially use less and less detail. If the renderer is forced to use the same level of detail over the whole model, it must again choose to use an insufficient level and sacrifice quality or use an excessive level and waste time.

Despite this limitation, discrete multiresolution models can be quite useful in certain situations. If an object is displayed such that the entire surface is at roughly the same scale, then discrete multiresolution models are an effective means of controlling level of detail. For instance, the discrete method seems to have been effective in the walkthrough system described by Funkhouser and Séquin[29]. Support for discrete levels of detail has also been included in a number of commercial rendering systems, including RenderMan[94], Open Inventor[95], and IRIS Performer[76]. The RenderMan interface provides for "mixing" successive levels of detail together, but leaves the exact mechanism undefined. Performer provides explicit support for both alpha blending and geomorphing. Discrete levels of detail have also been used for accelerating the computation of radiosity solutions[81].

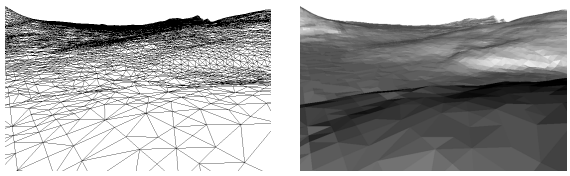


Figure 11: *Large terrain viewed from near the surface.*

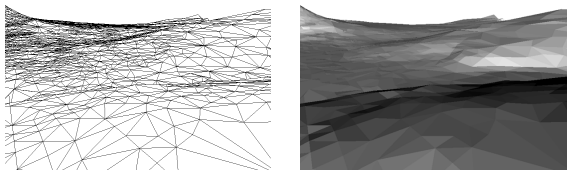


Figure 12: *Identical view with adaptive tessellation.*

6. Continuous Multiresolution Models

As we have just seen, discrete multiresolution models are sufficient in some circumstances, but there are other cases in which they are inadequate. A large surface, such as a terrain, being viewed at close range from an oblique angle is particularly problematic. Consider the example shown in Figure 11. The viewpoint is positioned just above the surface, looking out towards the horizon. Notice how the screen-space density of the triangulation increases as the surface recedes into the distance. An approximation with a constant level of detail would either be too dense in the distance (as in Figure 11) or too sparse near the viewpoint. We would prefer an approximation where the level of detail is allowed to vary *continuously* over the surface. In particular, we would like the level of detail of a particular neighborhood to be *view dependent*. Figure 12 demonstrates the results. While the approximation shown in Figure 11 contains many distant triangles whose projected screen size is minute, the approximation shown in Figure 12 uses a much lower level of detail for distant surface regions. The result is an approximation which is specifically tailored to the current viewpoint. Thus, we are looking for a multiresolution representation that con-

tinuously adapts the surface at run time based on viewing conditions. Run-time adaptation can be combined with the blending and geomorphing techniques described earlier to produce smooth transitions.

The need for adaptive level of detail control is particularly pronounced in the case of terrains, and continuous multiresolution models have been in use by flight simulator systems for twenty years[15]. Several effective adaptive terrain techniques are available[21, 62]. Many are based on a regular subdivision (e.g., quadtrees) of the terrain surface. Using regular subdivisions helps to minimize the run-time overhead incurred by maintaining an adaptive level of detail.

There has been comparatively less work on continuous multiresolution representations for general triangulated surfaces. Multi-triangulations[17, 16] provide a fairly general framework that can describe most commonly used multiresolution representations. Vertex hierarchies[48, 50, 67, 97] are a particular multiresolution representation that have received considerable attention. An important feature of vertex hierarchies is that they can be constructed as a by-product of contraction-based simplification algorithms. While the associated overhead is often acceptable, it is certainly higher than that of discrete multiresolution models. For instance, Hoppe[48] reports that model adaptation consumed 14% of the frame time for his implementation.

Depending on the application at hand, discrete or continuous multiresolution models may be more appropriate. Discrete methods are simpler and require less overhead. Continuous methods are more flexible but have higher overhead. This flexibility is important for models such as terrains, but may not be necessary for objects such as chairs in a room. Indeed, the best solution is to support different multiresolution representations which are tailored to different classes of model.

7. Incremental Representations

Starting with the original model M^0, iterative simplification algorithms generate a sequence of approximations

$$M^0 \rightarrow M^1 \rightarrow M^2 \rightarrow \cdots \rightarrow M^k$$

arriving at the final approximation M^k. Note that this differs from the progressive mesh notation used by Hoppe[47] where M^0 is the base mesh which, by a sequence of refinements, is transformed into the original mesh M^k. An *incremental representation* is one which encodes the original model M^0, the final model M^k, and all the intermediate approximations $M^1, \ldots, M^{k-1}$. This is a multiresolution representation because it allows us to extract a fairly wide range of levels of detail. However, the available approximations are restricted to exactly those which were generated during simplification.

7.1. Simplification Streams

The direct by-product of iterative contraction is an incremental representation which I will term a *simplification stream*. During the process of simplification, we generate the sequence of models

$$M^0 \xrightarrow{\phi^1} M^1 \xrightarrow{\phi^2} M^2 \xrightarrow{\phi^3} \cdots \xrightarrow{\phi^k} M^k$$

where each step $M^{i-1} \rightarrow M^i$ corresponds to the application of a single contraction ϕ^i. Thus each intermediate approximation M^i can be expressed as the result of applying some prefix of the total sequence of contractions

$$M^i = [\phi^i](M^0) = (\phi^i \circ \cdots \circ \phi^1)(M^0) \tag{7}$$

Suppose that along with the original mesh M^0 we store a representation of each contraction ϕ^i. By simply applying (7), we can reconstruct any intermediate model M^i in addition to the original and final models. In essence, by storing a record of the simplification process, we can re-apply the same contraction sequence but choose an earlier stopping point.

At a minimum, the pieces of information that must be recorded for a contraction ϕ^i are

1. identifiers for the vertices $(\mathbf{v}_j, \mathbf{v}_k)$ being contracted, and
2. the final position $\overline{\mathbf{v}}$.

Note that the actual size of this data may vary. If we are using an optimal placement strategy, $\overline{\mathbf{v}}$ might require as much as 3 floating point numbers to determine the new vertex position. And if the vertices have associated material attributes, further data will be required for each attribute. On the other hand, with a fixed placement strategy, $\overline{\mathbf{v}}$ can be encoded implicitly by the order of the vertices $(\mathbf{v}_j, \mathbf{v}_k)$.

Simplification streams do provide an incremental representation of the surface. By applying some prefix of the contraction sequence, we can reconstruct any of the intermediate approximations generated during simplification. However, their practical utility is limited. Lau *et al.* [59] used simplification streams of bounded size as a cache of recently generated approximations for run-time simplification. Recording the last k contractions allows their system to refine the current approximation by at most k steps. If further refinement is required, simplification begins again from the original model. For certain limited applications, this might yield acceptable results, but simplification streams are unsuitable for representing a wide range of approximations. Since we store the entire original model M^0 plus the contraction sequence, the resulting representation is necessarily larger than the original model. If we assume that our original model is very large and our desired approximation is quite small, certainly a common case, we are faced with a more significant problem. In order to reconstruct a small approximation, we must apply a large number of contractions to a large model. Thus, the smaller the approximation the greater the time required to extract it. We would clearly prefer reconstruction cost to be proportional to the desired approximation size. Fortunately, a closely related representation can solve both of these problems.

7.2. Progressive Meshes

The progressive mesh (PM) structure, originally introduced by Hoppe[47, 49], provides the same functionality as a simplification stream. However, it has two important advantages. First, the resulting representation can actually be smaller than the original model. Second, reconstruction time is proportional to the desired approximation size.

A progressive mesh is, in essence, a reversed simplification stream. It exploits the fact that the contraction operator is invertible. For each contraction ϕ^i we can define a corresponding inverse ψ^i. This operation, called a *vertex split*, is an inverse of ϕ^i such that $\psi^i(M^i) = M^{i-1}$. Thus, we begin with the final approximation M^k and produce a sequence of models

$$M^k \xrightarrow{\psi^k} \cdots \xrightarrow{\psi^3} M^2 \xrightarrow{\psi^2} M^1 \xrightarrow{\psi^1} M^0$$

terminating at the original model. The model M^k is called the *base mesh*. Along with this base mesh, we can store the vertex split sequence $\psi^k, \psi^{k-1}, \ldots, \psi^1$. Each item in the sequence must encode the vertex being split, positions for the two resulting vertices, and which triangles to introduce into the mesh[47]. We can reconstruct some intermediate approximation M^i by applying a prefix of the vertex split sequence:

$$M^i = [\psi^{i+1}](M^k) = (\psi^{i+1} \circ \cdots \circ \psi^k)(M^k) \qquad (8)$$

Not only does this encode many different levels of detail of the original model, but Hoppe[47] demonstrated that progressive meshes are also an effective technique for compressing the input geometry.

Progressive meshes are generally developed as the result of iterative edge contraction. Naturally, they can also be used with alternative contraction primitives. For instance, a progressive mesh based on face contraction has exactly the same structure as one based on edge contraction. However, because edge contraction is the finest-grained contraction primitive, it generates progressive meshes with the largest number of intermediate models. Simplification by pair contraction, and hence progressive meshes, can also be generalized to operate on arbitrary simplicial complexes[72].

8. Simplification and Spanning Trees

As outlined in the previous section, the sequence of contractions computed during simplification can be used to construct an incremental multiresolution representation of the model. But the process of iterative contraction actually induces further structure on the surface. In this section, we shall see that iterative edge contraction is a close analog of a minimum spanning tree algorithm.

Let us consider a model M as a graph. A *spanning tree* is an acyclic subgraph of M which connects all the vertices of M. A *spanning forest* is a collection of disjoint trees which collectively cover all the vertices of M. Suppose that every edge i has an associated weight, or cost, w_i. The weight of a graph is the sum of the weights of its edges, and a *minimum spanning tree* is a spanning tree whose total edge cost is minimal.

Figure 14 provides a simple illustration of the connection between iterative contraction and spanning trees. In the right-hand column is a sequence of approximations $M^2, \ldots, M^7$. In this example, we are using a fixed placement strategy; in other words, each contraction is of the form $(\mathbf{v}_i, \mathbf{v}_j) \to \mathbf{v}_i$. I will indicate this by writing the contraction in the more compact form $\mathbf{v}_j \to \mathbf{v}_i$. For each model, the edge being contracted to produce the next model is drawn as an arrow indicating the contraction in the same manner. Note that I have labeled the initial model M^2 because it corresponds to the final model shown in Figure 9.

Each vertex in a given approximation corresponds to some set of vertices in the original model. In particular, it corresponds to itself plus all the vertices which have been contracted into it. These sets are disjoint and they completely partition the original set of vertices. In the original model M^2, each vertex $\mathbf{v}_i$ corresponds to the singleton set $\{\mathbf{v}_i\}$. After the contraction $\mathbf{v}_5 \to \mathbf{v}_1$, the vertex $\mathbf{v}_1$ corresponds to the set $\{\mathbf{v}_1, \mathbf{v}_5\}$ in the original model. In the left-hand column of Figure 14, I have indicated the structure of these sets by enclosing them in shaded regions.

It is easy to show inductively that the construction of these sets is equivalent to the construction of a spanning forest. In the initial model, each set contains a single vertex. The set of all vertices is clearly a spanning forest. In general, a single edge contraction joins exactly two sets together. Assuming that each set is already a spanning tree, connecting these trees by a single edge (the edge being contracted) results in a larger spanning tree. Notice, however, that there is not one unique spanning tree corresponding to the simplification process. When two regions are merged, any duplicate edges connecting them to the same region are removed. Thus, an edge connecting two vertices in the approximation corresponds to multiple edges in the original graph. For example, when producing the tree for M^6 in Figure 14, there are three different edges which connect the corresponding parts of the spanning forest.

This view of iterative contraction is very similar to the minimum spanning tree algorithm of Cheriton and Tarjan[6, 92]. The primary difference is that in their algorithm each edge is assigned a constant weight, but in the simplification algorithm the weights assigned to the edges change over time. For example, the quadric error metric[33] assigns the weight of an edge as a function of the quadrics associated with its endpoints.

We can construct a directed graph by creating an edge $\mathbf{v}_j \to \mathbf{v}_i$ whenever we perform the corresponding contraction. Figure 13 illustrates the resulting graphs for models M^5 and M^7. By starting at a node and following the arcs, we arrive at the currently active vertex which has accumulated the node at which we began. This structure is a common representation for disjoint sets[14, 92]; it is often referred to as a disjoint-set (or union–find) forest. In fact, these structures are exactly those used by Cheriton and Tarjan[6] to track

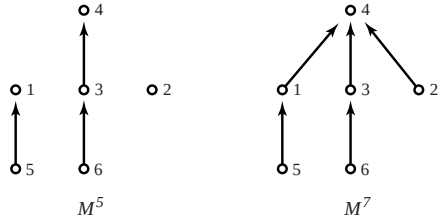


Figure 13: *Disjoint-set graph for models shown in Fig. 14.*

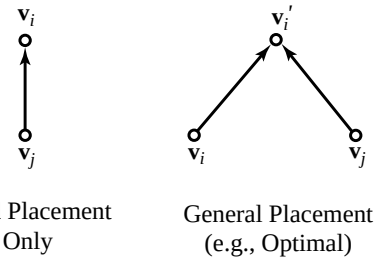


Figure 15: *Graph structure of a single contraction.*

disjoint sets of vertices in their minimum spanning tree algorithm. When using these graphs to simply track disjoint sets, it is common practice to apply a path compression heuristic — whenever we traverse a path, we update all the nodes encountered to point directly to the root of their tree. Consequently, all nodes will ultimately point directly to the root. Path compression leads to more efficient set membership queries; however, the additional structure of the uncompressed graph has some very useful multiresolution applications.

Edge contraction has also been used explicitly for constructing minimum spanning trees by Karger *et al.* [55]. This technique is similar to the algorithm of Xia and Varshney[97], which simplifies the surface by iteratively contracting maximal independent sets of edges.

9. Vertex Hierarchies

Progressive meshes provide a useful multiresolution structure, but they are somewhat restrictive. They only allow us to reconstruct models which were generated during the original simplification process. This is because we always perform contractions in the order in which they were discovered, but this total ordering of contractions is not necessary. By using a less restrictive partial ordering, we can achieve a much more flexible continuous multiresolution representation which will allow us to generate novel approximations that were never constructed during simplification.

Let us return to the simplification sequence pictured in Figure 14. Consider the contraction $\mathbf{v}_5 \rightarrow \mathbf{v}_1$ (which produces M^3) and the contraction $\mathbf{v}_6 \rightarrow \mathbf{v}_3$ (which produces M^4). With a progressive mesh, we would always perform these two contractions in this order, because that is the order in which they were initially performed during simplification. However, by inspection, we can see that they are independent. We can just as easily perform them in the opposite order, and the resulting meshes would be just as valid as the ones shown in the figure.

These two contractions are interchangeable because the sets of vertices involved in the contraction are disjoint. We can see this fact reflected in the structure of the graphs shown in Figure 13. There is no path which includes both contractions $\mathbf{v}_5 \rightarrow \mathbf{v}_1$ and $\mathbf{v}_6 \rightarrow \mathbf{v}_3$. If we interpret these disjoint-set forests as dependency graphs, we can determine whether any given pair of contractions can be performed independently or not.

The construction pictured in Figure 13 is only applicable when we are using a fixed placement scheme. In general, we would like to perform contractions $(\mathbf{v}_i, \mathbf{v}_j) \rightarrow \mathbf{v}_i'$ where $\mathbf{v}_i' \neq \mathbf{v}_i$. In order to accommodate such general placement, we can treat $\mathbf{v}_i'$ as a new vertex instance and link both $\mathbf{v}_i$ and $\mathbf{v}_j$ to it. This structure is illustrated in Figure 15.

The result of applying this rule over the entire simplification process will be a binary forest. Assuming that we simplify a model completely to a single vertex, we will have a binary tree. The resulting graph is a *vertex hierarchy*; Figure 16 illustrates the hierarchy resulting from the simplification in Figures 9 and 14. This vertex hierarchy structure was developed independently by several authors[48, 67, 97].

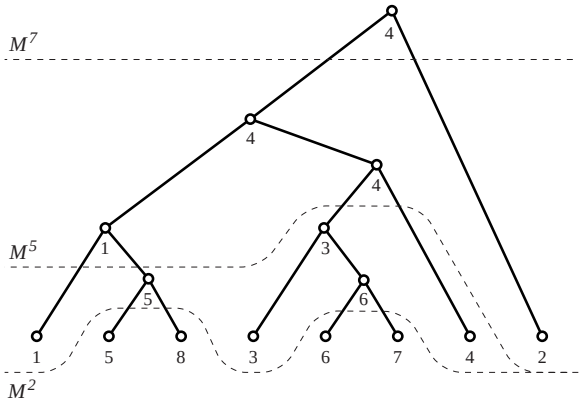


Figure 16: *The vertex hierarchy resulting from the simplification process shown in Figures 9 and 14.*

In a vertex hierarchy, cuts through the tree correspond to allowable approximations. Figure 16 shows the three different cuts corresponding to models M^2, M^5, and M^7. Any given cut divides the tree into some number of components. The component containing the root remains a tree, essentially a pruned version of the original. I will call the leaves of this pruned tree *active vertices*. Each active vertex is the root of some subtree in the hierarchy. For every active vertex, we can perform all the contractions described by that subtree, and we can perform contractions in separate subtrees in any order. The result is a valid approximation of the

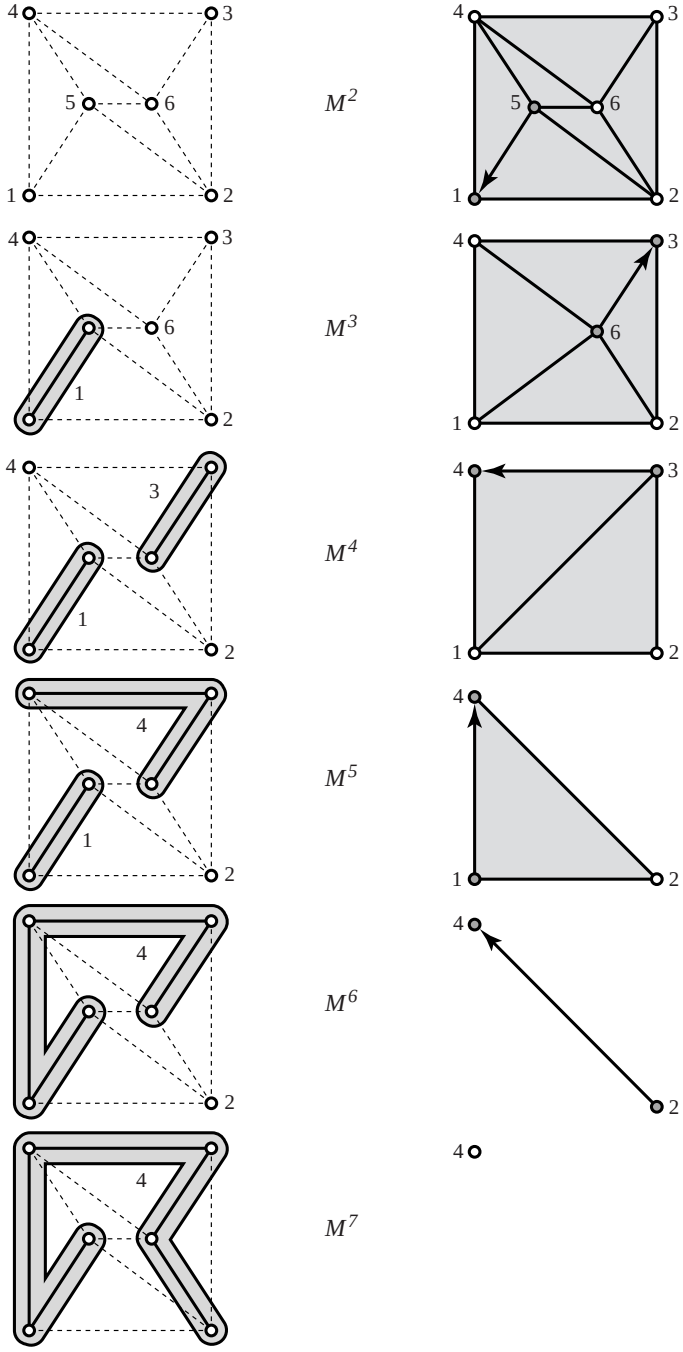


Figure 14: *Simplification process and corresponding spanning tree.*

original model. However, it is not guaranteed to be free of artifacts such as mesh fold-over. To prevent such degeneracies, we must either perform run-time consistency checks or encode further dependencies in the hierarchy itself.

9.1. Adaptive Refinement

The primary application for which vertex hierarchies have been used is view-dependent refinement of models for real-time rendering[48, 50, 67, 97]. Suppose that we have a large surface, such as a terrain, that we would like to render using a continuously varying level of detail. Furthermore, let us assume that there is generally a high degree of frame-to-frame coherence in the viewpoint, such as we would expect in a flight simulator. Now, imagine that our model is represented using a vertex hierarchy, and that we have selected a cut through the tree. Any change in this cut, either up or down the tree, will result in a new, closely related approximation. By maintaining an active cut through the tree, and incrementally adjusting it for each frame, we can adapt the current model to the new viewing conditions.

Vertex hierarchies can also accommodate further synthetic refinement of the model past the original level of detail. Any mesh refinement which can be implemented by edge splitting can be added into the hierarchy. An edge split,

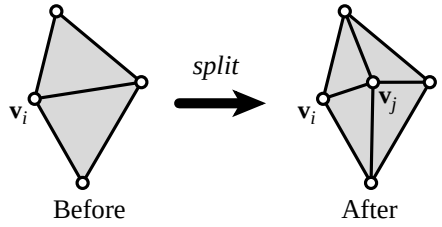


Figure 17: *Refining a mesh with a single edge split.*

such as the one shown in Figure 17, can be directly encoded as a vertex split operation on the vertex $\mathbf{v}_i$. The leaves of the vertex hierarchy represent the original vertices, but we can add temporary levels below the leaves corresponding to further refinement of the initial mesh. Consider the example of modeling a large terrain surface. By extracting some information from the input data, we might synthesize additional detail, consistent with the form of the surface, using fractal-based edge subdivision.

Naturally, vertex hierarchies are not required to be binary trees. If we allow generalized contraction operations, each node may have an arbitrary number of children[67]. A very deep hierarchy has the disadvantage that many adaptation steps are required to smoothly transition from the bottom of the hierarchy to the top, or vice versa. In some cases, it may be desirable to reduce this cost by compressing entire subtrees so that all the leaves point directly to the root of the subtree and all internal nodes are eliminated. An extreme example of this strategy would be to pick a small number of cuts through the tree, and remove all internal nodes separating them so that each level was linked directly to the next. This sort of *stratified* vertex hierarchy corresponds directly to the discrete multiresolution models discussed earlier.

10. Future Directions

Recent research in the field of surface simplification has produced several effective techniques for constructing approximations and multiresolution representations. Several commercial packages have begun to include simplification facilities. The spectrum of algorithms now available offers several possible tradeoffs between efficiency and quality. At one end are very high quality, very slow algorithms such as mesh optimization[51]. At the other are the very fast, but low quality vertex clustering algorithms[79]. Between these extremes are a number of algorithms, such as the quadric error metric[33] or vertex decimation[86], which provide various compromises between speed and approximation quality. However, there are a number of areas in which current simplification methods could be improved. The following avenues appear particularly important or promising.

10.1. Broader Applicability

Most recent work has focused on the simplification of triangulated surface models. However, there are other model classes where simplification techniques would also be quite valuable. Like scanned surface models, tetrahedral volume models are often very complex and can benefit from simplification[93, 89]. For the same reasons we might like to reduce the number of triangles in a polygonal surface model, we might like to reduce the number of patches in a piecewise-polynomial model[84, 36].

All current simplification methods assume that the surface being simplified is rigid. This covers a large class of models used in practice, including those composed of many individual moving, yet rigid, parts such as an engine model. However, there are many other applications where surfaces are changing over time. For example, many animation systems represent characters as surfaces attached to articulated skeletons. As the skeletal joints bend, the surface is deformed. Current simplification methods must be extended to handle this more generalized class of models.

The hierarchical structures resulting from simplification (e.g., vertex hierarchies) have been used primarily for view-dependent refinement. However, they have many potentially important applications. There are a large class of problems for which the structure itself, which we can think of as a hierarchy of regions on the surface, is useful. Rather than extracting a single approximation from the hierarchy, we can perform computations on the hierarchy. For example, these hierarchical regions can be used to construct a hierarchy of bounding volumes for applications such as collision detection or ray tracing. Instead of partitioning space around the model, these bounding volumes are intrinsically linked to the surface. Many kinds of simulation problems rely on hierarchies of successively larger surface regions. Radiosity methods, for instance, can be significantly accelerated by employing simplification techniques[96].

10.2. More Effective Simplification

There are several areas in which current simplification methods could be made more effective. One obvious goal is to devise an algorithm which can produce approximations (for general surfaces) which are provably close to optimal. An algorithm which could preserve higher level surface characteristics, such as symmetry, would also be quite useful. Current simplification methods all seem to perform poorly at extremely low levels of simplification, of say less than 50 triangles. This may very well be a weakness of their shared approach: almost all algorithms derive approximations by repeated transformation (e.g., edge contraction) of the original. At these very low levels, a human could presumably do a substantially better job. This suggests that semi-automatic tools which could incorporate some human interaction in the final stages of simplification could be quite useful.

It would also be desirable for simplification systems to incorporate better control over topology simplification. Most methods that allow topological simplification perform it only implicitly. It would seem that explicit topological simplification might require a more volumetric approach to surface models, and early methods[23, 42] have moved in this direction. This may also lead to more effective aggregation of separate components. The method of aggregation via pair contraction[33, 72, 24] has not been entirely successful. For instance, it would appear to be overly dependent on the placement of the vertices on the original surface[31].

New methods for measuring approximation error are also needed. For rendering applications, similarity of appearance is the ultimate goal. It would be helpful to have an appearance-based metric for reliably comparing the visual similarity of two models. Even if we are primarily concerned with preserving the shape of an object, the error metrics E_{max} and E_{avg} have some drawbacks. They do not directly address the measurement of attribute error. We can treat surface properties (e.g., color) as points in a Euclidean space and thus incorporate attributes into these metrics. But this is not a strictly accurate way of assessing attribute error; color space is not Euclidean, for example. It is also unclear whether metrics like E_{max} and E_{avg} adequately reflect the similarity of an approximation whose topology has been simplified.

10.3. Decoupling Analysis and Synthesis

Another promising avenue for improving the quality of automatically generated approximations may well be to decouple the analysis and synthesis phases of the simplification process. As an example, consider the quadric error metric algorithm[33]. A particular vertex begins with a quadric constructed from its immediate neighborhood. As other vertices are repeatedly contracted into this vertex, it accumulates a quadric that represents ever larger regions of the surface. In some sense, this is a shape analysis process. The algorithm is constructing information about ever larger regions of the surface. However, the current algorithm actually performs a contraction on the mesh immediately after accumulating the quadrics for the endpoints — it immediately synthesizes a new approximation.

Now imagine that we were to decouple these processes into two separate phases. The first phase would be an analysis phase, gaining information about the structure of the surface at ever coarser levels of detail. This might involve accumulating quadrics over progressively larger regions, constructing face cluster hierarchies, or some completely different technique. After this phase is completed, we could begin simplifying the surface. The current algorithm, when considering the contraction of an edge, can only consider the shape of the immediate neighborhood of this edge, represented by the quadrics of the endpoints. It suffers from a certain shortsightedness because it can only assess the local and immediate effect of a contraction. However, if an earlier analysis phase had already been performed, it could consider the effect of a contraction at several levels of detail, from the immediately local to the more global.

10.4. Non-Greedy Frameworks

Many simplification methods are based on the same framework: greedy application of simplification operators. In the case of iterative contraction, this naturally produces a sequence of edge contractions which lead to multiresolution representations such as progressive meshes and vertex hierarchies. However, greedy decimation can limit the quality of the final result. Since it only iteratively picks what appears to be the best local operation to perform, a bad decision at some point can lead to results that are far from optimal.

Alternative frameworks are possible. For instance, mesh optimization [51] uses a simulated annealing-like process. Other search techniques, more general than greedy decimation, could also be applied. However, this type of search algorithm generally does not produce a single sequence of contractions that transform the original object into the final result. Consequently, they cannot be used to construct progressive meshes and related multiresolution representations.

Another interesting possibility is an algorithm based on graph partitioning, which has been used in several areas, including sparse matrix computations[71], often in conjunction with contraction-based coarsening algorithms[56]. Suppose we have an algorithm based on iterative edge contraction. Notice that not only do we know where simplification begins (with the original model), but we also know the final configuration which it will reach. In the end, every connected component will have collapsed to a single vertex. Let us consider a single connected component. The last operation before the final state of a single vertex must have involved the contraction of two vertices together. These two vertices correspond to two disjoint sets of vertices in the original mesh: the set of vertices which were contracted into the remaining vertices. In other words, we can partition the entire mesh into two separate regions, one for each vertex. Now, each one of these vertices was formed by the contraction of two earlier vertices. This leads to a recursive binary partitioning of the mesh. At each phase, all the vertices within a single partition

are collapsed into a single representative vertex. In essence, this computes a sequence of contractions in reverse, from last to first.

11. Acknowledgements

This research was supported in part by the National Science Foundation (grants CCR-9357763, CCR-9619853, & CCR-9505472) and by the Schlumberger Foundation. Thanks to Paul Heckbert, Jarek Rossignac, Martial Hebert, and Andy Witkin for their many helpful comments and suggestions. Thanks as well to Tim Vadnais of Iris Development who provided the dental model.

References

1. Pankaj K. Agarwal and Pavan K. Desikan. An efficient algorithm for terrain simplification. In *Proc. ACM-SIAM Sympos. Discrete Algorithms*, pages 139–147, 1997.
2. Pankaj K. Agarwal and Subhash Suri. Surface approximation and geometric partitions. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 24–33, 1994. (Also available as Duke U. CS tech report, ftp://ftp.cs.duke.edu/dist/techreport/1994/1994-21.ps.Z).
3. María-Elena Algorri and Francis Schmitt. Mesh simplification. *Computer Graphics Forum*, 15(3), August 1996. Proc. Eurographics '96.
4. Laurence Boxer, Chun-Shi Chang, Russ Miller, and Andrew Rau-Chaplin. Polygonal approximation by boundary reduction. *Pattern Recognition Letters*, 14(2):111–119, February 1993.
5. Andrew Certain, Jovan Popović, Tony DeRose, Tom Duchamp, David Salesin, and Werner Stuetzle. Interactive multiresolution surface viewing. In *SIGGRAPH 96 Conference Proceedings*, pages 91–98, August 1996.
6. David Cheriton and Robert E. Tarjan. Finding minimum spanning trees. *SIAM Journal of Computing*, 5(4):724–742, December 1976.
7. A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno. Multiresolution decimation based on global error. *The Visual Computer*, 13(5):228–246, 1997.
8. P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. A general method for preserving attribute values on simplified meshes. In *IEEE Visualization 98 Conference Proceedings*, pages 59–66,518, Oct 1998.
9. P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, 1998.
10. P. Cignoni, C. Rocchini, and R. Scopigno. Metro: measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–74, June 1998. http://vcg.iei.pi.cnr.it/metro.html.
11. James H. Clark. Hierarchical geometric models for visible surface algorithms. *CACM*, 19(10):547–554, October 1976.
12. Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-preserving simplification. In *Proceedings SIGGRAPH 98*, pages 115–122, 1998.
13. Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification envelopes. In *SIGGRAPH '96 Proc.*, pages 119–128, August 1996. http://www.cs.unc.edu/~geom/envelope.html.
14. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.
15. Michael A. Cosman and Robert A. Schumacker. System strategies to optimize CIG image content. In *Proceedings of the Image II Conference*, pages 463–480. Image Society, Tempe, AZ, June 1981.
16. Leila De Floriani, Paola Magillo, and Enrico Puppo. Efficient implementation of multi-triangulations. In *IEEE Visualization 98 Conference Proceedings*, pages 43–50,517, Oct 1998.
17. Leila De Floriani, Enrico Poppo, and Paola Magillo. A formal approach to multiresolution hypersurface modeling. In W. Straßer, R. Klein, and R. Rau, editors, *Geometric Modeling: Theory and Practice*. Springer-Verlag, 1997.
18. Michael DeHaemer, Jr. and Michael J. Zyda. Simplification of objects rendered by polygonal approximations. *Computers and Graphics*, 15(2):175–184, 1991.
19. Hervé Delingette. Simplex meshes: A general representation for 3D shape reconstruction. In *Conf. on Computer Vision and Pattern Recognition (CVPR '94)*, June 1994. http://zenon.inria.fr/epidaure/personnel/delingette/delingette.html.
20. David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, December 1973.
21. Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. ROAMing terrain: Real-time optimally adapting meshes. In *IEEE Visualization 97 Conference Proceedings*, pages 81–88, October 1997. http://www.llnl.gov/graphics/ROAM/.
22. Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH '95 Proc.*, pages 173–182. ACM, August 1995. http://www.cs.washington.edu/research/projects/grail2/www/pub/pub-author.html.
23. Jihad El-Sana and Amitabh Varshney. Controlled simplification of genus for polygonal models. In *IEEE*

Visualization 97 Conference Proceedings, pages 403–410, 1997.

24. Carl Erikson and Dinesh Manocha. GAPS: General and automatic polygonal simplification. In *Proc. Symposium on Interactive 3D Graphics '99*, pages 79–88,225, 1999. http://www.cs.unc.edu/~eriksonc/Research/Paper/.

25. Olivier Faugeras, Martial Hebert, P. Mussi, and Jean-Daniel Boissonnat. Polyhedral approximation of 3-D objects without holes. *Computer Vision, Graphics, and Image Processing*, 25:169–183, 1984.

26. R. L. Ferguson, R. Economy, W. A. Kelley, and P. P. Ramos. Continuous terrain level of detail for visual simulation. In *Proceedings of the 1990 Image V Conference*, pages 145–151. Image Society, Tempe, AZ, June 1990.

27. Robert J. Fowler and James J. Little. Automatic extraction of irregular network digital terrain models. *Computer Graphics (SIGGRAPH '79 Proc.)*, 13(2):199–207, August 1979.

28. Thomas A. Funkhouser. *Database and Display Algorithms for Interactive Visualization of Architectural Models*. PhD thesis, CS Division, UC Berkeley, 1993.

29. Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics (SIGGRAPH '93 Proc.)*, 1993.

30. Thomas A. Funkhouser, Carlo H. Séquin, and Seth J. Teller. Management of large amounts of data in interactive building walkthroughs. In *1992 Symposium on Interactive 3D Graphics*, pages 11–20, 1992. Special issue of Computer Graphics.

31. Michael Garland. *Quadric-Based Polygonal Surface Simplification*. PhD thesis, Carnegie Mellon University, CS Dept., 1999. Tech. Rept. CMU-CS-99-105. http://www.cs.cmu.edu/~garland/thesis/.

32. Michael Garland and Paul S. Heckbert. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, Comp. Sci. Dept., Carnegie Mellon University, September 1995. http://www.cs.cmu.edu/~garland/scape.

33. Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Proc.*, pages 209–216, August 1997. http://www.cs.cmu.edu/~garland/quadrics/.

34. Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *IEEE Visualization 98 Conference Proceedings*, pages 263–269,542, October 1998. http://www.cs.cmu.edu/~garland/quadrics/.

35. Tran S. Gieng, Bernd Hamann, Kenneth I. Joy, Gregory L. Schussman, and Isaac J. Trotts. Constructing hierarchies for triangle meshes. *IEEE Trans. on Visualization and Computer Graphics*, 4(2):145–161, April–June 1998.

36. M. Gopi and D. Manocha. A unified approach for simplifying polygonal and spline models. In *IEEE Visualization 98 Conference Proceedings*, pages 271–278,543, Oct 1998.

37. Alexis Gourdon. Simplification of irregular surface meshes in 3D medical images. In *Computer Vision, Virtual Reality, and Robotics in Medicine (CVRMed '95)*, pages 413–419, April 1995.

38. André Guéziec. Surface simplification with variable tolerance. In *Second Annual Intl. Symp. on Medical Robotics and Computer Assisted Surgery (MRCAS '95)*, pages 132–139, November 1995.

39. André Guéziec. Surface simplification inside a tolerance volume. Technical report, Yorktown Heights, NY 10598, March 1996. IBM Research Report RC 20440, http://www.watson.ibm.com:8080/search_paper.shtml.

40. André Guéziec, Gabriel Taubin, Francis Lazarus, and William Horn. Converting sets of polygons to manifold surfaces by cutting and stitching. In *IEEE Visualization 98 Conference Proceedings*, pages 383–390,553, Oct 1998.

41. Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):75–123, 1985.

42. T. He, L. Hong, A. Varshney, and S. Wang. Controlled topology simplification. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):171–184, June 1996.

43. Paul S. Heckbert and Michael Garland. Multiresolution modeling for fast rendering. In *Proc. Graphics Interface '94*, pages 43–50, Banff, Canada, May 1994. Canadian Inf. Proc. Soc. http://www.cs.cmu.edu/~ph.

44. Paul S. Heckbert and Michael Garland. Survey of polygonal surface simplification algorithms. In *Multiresolution Surface Modeling Course Notes*. ACM SIGGRAPH, 1997. http://www.cs.cmu.edu/~ph, draft of Carnegie Mellon University tech. report, to appear.

45. Michael Henle. *A Combinatorial Introduction to Topology*. Dover, New York, 1994. Reprint of 1979 edition of W. H. Freeman.

46. Paul Hinker and Charles Hansen. Geometric optimization. In *Proc. Visualization '93*, pages 189–195, San Jose, CA, October 1993. http://www.acl.lanl.gov/Viz/vis93_abstract.html.

47. Hugues Hoppe. Progressive meshes. In *SIGGRAPH '96 Proc.*, pages 99–108, August 1996. http://research.microsoft.com/~hoppe/.

48. Hugues Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH 97 Proc.*, pages 189–198, August 1997. http://research.microsoft.com/~hoppe/.

49. Hugues Hoppe. Efficient implementation of progressive meshes. *Computers & Graphics*, 22(1):27–36, 1998.

50. Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Visualization 98 Conference Proceedings*, pages 35–42,516, Oct 1998.

51. Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *SIGGRAPH '93 Proc.*, pages 19–26, August 1993. http://research.microsoft.com/~hoppe/.

52. Insung Ihm and Bruce Naylor. Piecewise linear approximations of digitized space curves with applications. In N. M. Patrikalakis, editor, *Scientific Visualization of Physical Phenomena*, pages 545–569, Tokyo, 1991. Springer-Verlag.

53. Hiroshi Imai and Masao Iri. Polygonal approximations of a curve – formulations and algorithms. In G. T. Toussaint, editor, *Computational Morphology*, pages 71–86. Elsevier Science, 1988.

54. Alan D. Kalvin and Russell H. Taylor. Superfaces: Polygonal mesh simplification with bounded error. *IEEE Computer Graphics and Appl.*, 16(3), May 1996. http://www.computer.org/pubs/cg&a/articles/g30064.pdf.

55. David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321–328, March 1995.

56. George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

57. Reinhard Klein, Gunther Liebich, and W. Straßer. Mesh reduction with error control. In *Proceedings of Visualization '96*, pages 311–318, October 1996.

58. Leif Kobbelt, Swen Campagna, and Hans Peter Seidel. A general framework for mesh decimation. In *Proc. Graphics Interface '98*, pages 43–50, June 1998.

59. Rynson Lau, Mark Green, Danny To, and Janis Wong. Real-time continuous multi-resolution method for models of arbitrary topology. *Presence: Teleoperators and Virtual Environments*, 7(1):22–35, February 1998. http://www.cs.cityu.edu.hk/~rynson/pub-cgvr.html.

60. Jay Lee. A drop heuristic conversion method for extracting irregular network for digital elevation models. In *GIS/LIS '89 Proc.*, volume 1, pages 30–39. American Congress on Surveying and Mapping, November 1989.

61. J.-G. Leu and L. Chen. Polygonal approximation of 2-D shapes through boundary merging. *Pattern Recognition Letters*, 7(4):231–238, April 1988.

62. Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory A. Turner. Real-time, continuous level of detail rendering of height fields. In *SIGGRAPH '96*, pages 109–118, August 1996.

63. Peter Lindstrom and Greg Turk. Fast and memory efficient polygonal simplification. In *IEEE Visualization 98 Conference Proceedings*, pages 279–286,544, Oct 1998.

64. William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface reconstruction algorithm. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):163–170, July 1987.

65. Michael Lounsbery, Tony D. DeRose, and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Trans. on Graphics*, 16(1):34–73, 1997. http://www.cs.washington.edu/research/graphics/pub/.

66. Kok-Lim Low and Tiow-Seng Tan. Model simplification using vertex-clustering. In *1997 Symposium on Interactive 3D Graphics*. ACM SIGGRAPH, 1997. http://www.iscs.nus.sg/~tants/.

67. David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH 97 Proc.*, pages 199–208, August 1997.

68. Stephane G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11(7):674–693, July 1989.

69. Makoto Maruya. Generating a texture map from object-surface texture data. *Computer Graphics Forum*, 14(3):397–405,506–507, 1995. Proc. Eurographics '95.

70. Nelson L. Max. Computer graphics distortion for IMAX and OMNIMAX projection. In *Nicograph '83 Proceedings*, pages 137–159, December 1983.

71. Gary L. Miller, Shang-Hau Teng, and Steven A. Vavasis. Automatic mesh partitioning. In A. George, J. R. Gilbert, and J. W.-H. Liu, editors, *Sparse Matrix Computations: Graph Theory Issues and Algorithms*. Springer-Verlag, 1993.

72. Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In *SIGGRAPH 97 Proc.*, pages 217–224, 1997. http://research.microsoft.com/~hoppe/.

73. P. M. Prenter. *Splines and Variational Methods*. John Wiley & Sons, New York, 1975.

74. Franco P. Preparata and Michael I. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, New York, NY, 1985.

75. Shmuel Rippa. Adaptive approximation by piecewise linear polynomials on triangulations of subsets of scattered data. *SIAM J. Sci. Stat. Comput.*, 13(5):1123–1141, September 1992.

76. John Rohlf and James Helman. IRIS Performer: A high performance multiprocessing toolkit for real-time 3D graphics. In *SIGGRAPH '94 Proc.*, pages 381–394, July 1994. http://www.sgi.com/software/performer/.

77. Rémi Ronfard and Jarek Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum*, 15(3), August 1996. Proc. Eurographics '96.

78. Rémi Ronfard and Jarek Rossignac. Full-range approximation of triangulated polyhedra. Technical Report IBM Research Report RC 20423, IBM T. J. Watson Research, Yorktown Heights, NY, 1996.

79. Jarek Rossignac and Paul Borrel. Multi-resolution 3D approximations for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, 1993.

80. Y. Rubner and C. Tomasi. Texture metrics. In *Proc. IEEE Intl. Conf. on Systems, Man, and Cybernetics*, pages 4601–4607, 1998.

81. Holly E. Rushmeier, Charles Patterson, and Aravindan Veerasamy. Geometric simplification for indirect illumination calculations. In *Proc. Graphics Interface '93*, pages 227–236, Toronto, Ontario, May 1993. Canadian Inf. Proc. Soc. http://www.cc.gatech.edu/gvu/people/Phd/Charles.Patterson/research/gsii/gsii.html.

82. Lori Scarlatos. *Spatial Data Representations for Rapid Visualization and Analysis*. PhD thesis, CS Dept, State U. of New York at Stony Brook, 1993.

83. G. Schaufler and W. Stürzlinger. Generating multiple levels of detail from polygonal geometry models. In M. Göbel, editor, *Virtual Environments '95 (Eurographics Workshop)*, pages 33–41. Springer Verlag, January 1995.

84. Francis J. M. Schmitt, Brian A. Barsky, and Wen-Hui Du. An adaptive subdivision method for surface-fitting from sampled data. *Computer Graphics (SIGGRAPH '86 Proc.)*, 20(4):179–188, August 1986.

85. William J. Schroeder. A topology modifying progressive decimation algorithm. In *IEEE Visualization 97 Conference Proceedings*, pages 205–212,545, 1997.

86. William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proc.)*, 26(2):65–70, July 1992.

87. Marc Soucy, Guy Godin, and Marc Rioux. A texture-mapping approach for the compression of colored 3D triangulations. *The Visual Computer*, 12(10):503–514, 1996.

88. Marc Soucy and Denis Laurendeau. Multiresolution surface modeling based on hierarchical triangulation. *Computer Vision and Image Understanding*, 63(1):1–14, 1996.

89. Oliver G. Staadt and Markus H. Gross. Progressive tetrahedralizations. In *IEEE Visualization 98 Conference Proceedings*, pages 397–402,555, Oct 1998.

90. Paul Steed. The art of low-polygon modeling. *Game Developer*, pages 62–69, June 1998. http://www.gdmag.com/backissue1998.htm#jun98.

91. Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgann Kaufmann, San Francisco, CA, 1996.

92. Robert Endre Tarjan. *Data Structures and Network Algorithms*. Number 44 in Regional Conference Series in Applied Mathematics. SIAM, Philadelphia, 1983.

93. Isaac J. Trotts, Bernd Hamann, Kenneth I. Joy, and David F. Wiley. Simplification of tetrahedral meshes. In *IEEE Visualization 98 Conference Proceedings*, pages 287–295, Oct 1998.

94. Steve Upstill. *The Renderman Companion*. Addison Wesley, Reading, MA, 1990.

95. Josie Wernecke. *The Inventor Mentor: Programming Object-Oriented 3D Graphics With Open Inventor, Release 2*. Addison-Wesley, 1994.

96. Andrew J. Willmott, Paul S. Heckbert, and Michael Garland. Face cluster radiosity. In *Eurographics Workshop on Rendering*, June 1999. http://www.cs.cmu.edu/~ajw/paper/fcr-eg99/.

97. Julie C. Xia and Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. In *Proceedings of Visualization '96*, pages 327–334, October 1996.