Rynson W.H. Lau
Mark Green*
Danny To
and
Janis Wong
Computer Graphics and Media Laboratory
Department of Computing
The Hong Kong Polytechnic University
Hong Kong
cswhlau@comp.polyu.edu.hk

Real-Time Continuous Multiresolution Method for Models of Arbitrary Topology

Abstract

Many multiresolution methods have been proposed. Most of them emphasize accuracy and hence are slow. Some methods may be fast, but they may not preserve the geometry of the model. Although there are a few real-time multiresolution methods available, they are developed mainly for handling large terrain models. In this paper, we present a very efficient multiresolution method for continuously reducing the resolution of a triangle model by incrementally removing triangles from it. The algorithm is simple to implement, requires no complicated data structures, and has a linear triangle deletion rate. We also present a method for caching the most recent sequence of triangle removal operations into a list, called the *simplification list,* so that it is possible to continuously increase the resolution of the model by inserting triangles in the reverse order of the sequence. We will compare our method with Hoppe's progressive meshes. Towards the end of the paper, we discuss the performance and memory usage of our method.

1 Introduction

Because visual feedback plays a very important role in human communication, most existing VR applications require the use of expensive graphics hardware to provide high rendering throughput. To improve the overall performance of a given graphics system, our current research is directed toward developing a framework for time-critical rendering in VR (Green, 1996). That is, we are looking at different aspects of the image-rendering process, trying to maintain a constant frame rate by trading off the least important factors. For example, when there are too many objects to be processed and rendered within the given frame time, we may consider using flat shading instead of Gouraud shading or using low-resolution object models when rendering objects of lower visual importance. To achieve this objective, we have proposed a model for predicting the time needed to render the objects. However, unlike other predictive models such as (Funkhouser and Séquin, 1993), ours also addresses the issues of how various aspects of the hardware affect rendering time. This paper considers only one aspect of the total problem, the dynamic construction of multiresolution models of 3D objects. The number of triangles used to represent an object is one of the major factors affecting the object's display rate. The method presented here allows us to dynamically adjust the number of triangles

Presence, *Vol. 7, No. 1, February 1998,* 22–35

* Department of Computer Science, University of Alberta, Alberta, Canada

a model, in real time, and thus control the time required to display the model. The other aspects of our approach to time critical rendering are presented in Green (1996).

Although the performance of graphics accelerators is increasing rapidly, the demand for even higher performance to handle more complex environments is increasing too. To overcome this demand for improved rendering performance, a technique called multiresolution modeling is attracting a lot of attention. The technique is based on the fact that distant objects occupy smaller screen areas after projection than nearby objects do. As such, details of distant objects are less visible to the viewer than those of the nearby objects. To take advantage of this situation, multiresolution modeling optimizes the rendering performance by representing nearby objects with more detailed models, containing more triangles, and distant objects with simplier models, containing fewer triangles.

In this paper, we present a very efficient multiresolution method for continuously reducing the resolution of a given triangle model by incrementally removing triangles from the model. The algorithm is simple to implement, requires no complicated data structures, and has a linear triangle deletion rate resulting in a predictable performance. We also present a method for caching the most recent sequence of triangle removal operations into a list, called the *simplification list*, so that it is possible to continuously increase the resolution of the model by inserting triangles in the reverse order of the sequence. The rest of the paper is organized as follows. Section 2 describes previous work on multiresolution modeling. Section 3 gives an overview of the idea presented in this paper, and Section 4 presents our method in detail. Section 5 presents the simplification list algorithm. Section 6 shows some experimental results and discusses the advantages and limitations of the method. Finally, Section 7 presents conclusions and discusses some possible future work.

2 Related Work

Existing methods for generating multiresolution models can be classified into *refinement methods* and *decimation methods* (Heckbert and Garland, 1995). Refinement methods improve the accuracy of a minimal approximation of a model that has been initially built (DeHaemer and Zyda, 1991; Hoppe et al., 1993; Turk, 1992). They are also called global reconstruction heuristics. DeHaemer and Zyda (1991) use an adaptive subdivision method to recursively subdivide each quadrilateral polygon into four until the error is within a given tolerance. The location for the subdivision is selected in such a way that the resulting error after the subdivision is minimum. Turk (1992) handles the problem as mesh optimization. He first distributes a number of vertices over the object surface based on its curvature and then triangulates the vertices to obtain a higher-resolution model with the specified number of vertices. Although the algorithm works well for curved surfaces, it is slow and may fail to preserve the geometry of surfaces with high curvature. Hoppe et al. (1993) optimize an energy function over a surface iteratively to minimize the distance of the approximating surface from the original, as well as the number of vertices in the resultant approximation. This may preserve the geometry of the model better, but it is too slow for on-the-fly real-time computations.

Decimation methods, on the other hand, remove vertices from an exact representation of a model to create less accurate, but simpler versions of the initial model (Cohen, et al., 1996; Eck et al., 1995; Lounsbery, DeRose, & Warren, 1993; Rossignac and Borrel, 1992; Schroeder, Zarge, and Lorensen, 1992). These methods are also called local simplification heuristics since they modify the model locally. Schroeder et al. (1992) try to delete edges or vertices from almost coplanar faces. The resultant holes are filled by a local triangulation process. Although their method is generally faster than the refinement methods described above, the need to process and triangulate the models in multiple passes can be computationally expensive and hence difficult to use in real-time generation of multiresolution models. In addition, since the method does not consider feature edges, the generated models may not preserve the geometry of the model. Lounsbery et al. (1993) proposed the multiresolution analysis on arbitrary triangle models with subdivision connectivity, i.e., models obtained from a simple base model by recursively splitting each triangle in the model into four. The method is based on repeatedly decomposing the surface function into a lower-resolution part and a set of wavelet coefficients representing the surface details. Eck et al. (1995) later proposed a method to overcome the subdivision connectivity limitation by approximating an input arbitrary triangle model with one having subdivision connectivity. From the performance figures given at the end of this paper, this method can simplify a few hundred triangles per second. This is much faster than most existing multiresolution methods, but it may still be slow for real-time multiresolution modeling. Cohen et al. (1996) proposed the idea of simplification envelopes for model simplification. Two envelopes, the inner envelope and the outer envelope, are created to constraint the simplification of a model to within a given error tolerance. Rossignac and Borrel (1992) propose a very efficient multiresolution method. The object model is uniformly subdivided into 3D cells. The resolution of the model may be reduced by clustering all vertices within a cell to form a single new vertex. This method can greatly simplify the simplification process and is therefore very fast. However, because the geometry of the model is not considered when subdividing it, it is possible that important vertices (and the associated triangles) that define the geometry of the model are clustered first before the less important ones. Hence this method fails to preserve the geometry of the model.

Despite the fact that the two classes of methods can generate multiresolution models, there are still some limitations in this area. These methods, especially the refinement methods, have high computational cost, and some of them do not even preserve the geometry of the model. To overcome the performance limitation, a common method is to generate a few key models of the object at different resolutions in advance. During run-time, the object distance from the viewer determines which of these pre-generated models to use for rendering a particular frame. This method is referred to as *discrete multiresolution method.* Although this method is fast and simple, it has a major limitation. Since there is usually a considerable visual difference between two successive models, when the object crosses the threshold distance, the sudden change in resolution may create an objectionable visual effect. Turk (1992) proposed to have a transition period during which a smooth interpolation between the two successive models is performed to produce models of intermediate resolutions. This method, however, further increases the computational cost during the transition period because of the need to process two models at the same time.

There are a few real-time multiresolution methods. They are used primarily for handling large terrain models (Falby et al., 1993; Herzen & Barr, 1987; Lindstrom et al., 1996). Falby et al. (1993) divide a large terrain surface into 50×50 square blocks. All the polygons inside a block are arranged in a quadtree structure such that the root node of the tree represents the whole block and the leaf nodes represent individual polygons. Each successive lower level of the tree represents a four-time increase in resolution. This quadtree structure helps in culling polygons that are outside the field of view much more quickly and provides a multiresolution representation for rendering. The major limitation of using a quadtree to represent a surface is that the surface has to be divided into regular square tiles. Although this may be fine for representing smooth landscapes, it may not be the best way to represent objects or surfaces of arbitrary topology because those objects may contain a lot of sharp edges (or feature edges), which will cause a lot of nodes to be generated around them.

Hoppe (1996) proposed an idea called progressive meshes. A progressive mesh stores a pre-computed list of edge collapses (or edge splits). By following the list in the forward or backward direction, the resolution of the model can be modified in a very efficient way. This idea is similar to the simplification list we propose here although we developed our idea independently (İşler, Lau, and Green, 1996). The major advantages of Hoppe's method as compared to the quadtree type of real-time multiresolution methods are that it works on surfaces of arbitrary topology, and the models do not have to be regularly subdivided.

3 Overview of Our Method

One may argue that it is not important to preserve the geometry of an object model since the objects will be too far away for the changes to be visible. However, this argument is not valid for two reasons. First, in continuous multiresolution modeling, triangles (or vertices) are being removed from the model in a continuous manner as the object moves away from the viewer. If the geometry of the model is not considered, it is possible that triangles that form important features of the model are removed in an earlier stage while the object is still near the viewer. This may result in giving the incorrect perception to the viewer that the object is deforming as it moves. Second, in some VR applications, there may not be any distant objects. However, if the graphics accelerator is overloaded, in order to maintain a constant frame rate, we may need to simplify some objects that are considered as visually less important, even though these objects may not be too far away from the viewer. For example, in an architectural walkthough, where there are typically no distant objects, a sofa in an office may be considered as visually less important and, hence, can be simplified. However, as we still want it to look like a sofa, we must preserve its geometry while removing the details.

Most existing multiresolution methods either spend a lot of computational time trying to preserve the geometry of the original model in order to produce the best possible multiresolution model, or concentrate on the performance of the method and not on preserving the geometry of the original model. The method that we propose here takes a somewhat middle course. Although it is important to preserve the geometry of the model, we argue that it is not necessary to spend a large amount of computational time so as to find the best triangles to remove in order to minimize some error function when reducing the resolution of the model. The result of this approach is a continuous multiresolution method that is fast enough to simplify object models on-the-fly in real-time; at the same time it continuously updates a data structure representing the geometry of the model, to make sure that the geometry is preserved as much as possible.

There are two major contributions of our present work. The first one is the introduction of the continuous multiresolution method. The method is based on edge collapse and can be applied to triangle models of arbitrary topology. The second contribution is a simple enhancement method to the first, which makes real-time and on-the-fly multiresolution modeling possible even with a less powerful computer. The cost is only a small increase in memory. This increase is achieved by storing the sequence of edge collapses in a simplification list. During run time, we may decrease or increase the resolution of the model by following the list in the forward or backward direction. Although our idea of the simplification list is similar to the progressive meshes proposed by Hoppe, it differs from his in the following respects:

1. We believe that in a virtual world we may have deformable objects as well as rigid objects. Our method is based on a fast multiresolution method, which also has a low pre-processing time. As such, it can handle objects with changing geometry. On the other hand, Hoppe's method is based on a slow but accurate multiresolution method. The progressive meshes must be constructed offline, and hence cannot be used to handle objects with changing geometry. (As an example, Hoppe [1996] indicates that a model containing 13,546 triangles took 23 minutes to simplify on a 150 MHz SGI Indigo^2 workstation.) In this sense, our method is a more general approach to multiresolution modeling.
2. One of the advantages of our algorithm is that a complete simplification list for the object need not be constructed. The programmer can state the size of the simplification list associated with each object, and only the most recent sequence of edge collapses will be stored. The programmer has more control over memory resources and over the tradeoff between memory and processor time than with Hoppe's algorithm, where a complete reconstruction of the object is stored in the progressive mesh.
3. The objective of our research is to develop a real-time continuous multiresolution method. Although the method that we have developed (to be discussed in Section 4) can reduce the resolution of an object model efficiently, it is unable to increase the resolution of the model because there is no information on where triangles can be inserted into the model. Hence, as the object moves away from

the viewer, we may incrementally remove triangles from the model during each frame time. As the object moves towards the viewer, however, we need to simplify the model starting from the highest resolution until the intended resolution is reached during every frame time. Our idea of caching, in a simplification list, the most recent sequence of edge collapses was developed to solve this limitation. This list contains a minimal amount of information and is not meant to replace the data structure of the object model. By following the reverse order of the list, we may perform edge uncollapses to increase the resolution of the model. This idea was first mentioned in İşler, Lau, & Green (1996) as possible future work. On the other hand, it appears to us that Hoppe developed his idea oriented towards progressive transmission of object models as he puts all the information needed to recreate the object model, such as the vertex coordinates, into the progressive mesh data structure. In this way, transmission of the model can be achieved by transmitting the corresponding progressive mesh alone, starting with the low resolution end. At the receiving side, the model may be reconstructed to whatever resolution desired, depending on the time available for receiving the data.

4. In order to reduce the bandwidth in transmitting the progressive meshes, Hoppe suggests encoding the vertex coordinates stored in the meshes. This approach will certainly decrease the overall performance of the method when used in real-time multiresolution modeling. On the other hand, our simplification list is not developed for model transmission. Hence, no compression is applied to reduce the size of the list. The list stores only enough information for multiresolution modeling. As will be shown later, the simplification list method is very efficient and requires a small amount of additional memory.

4 Real-Time Multiresolution Method

In our method, we assume that the object model is composed of triangles. However, curved surfaces can be handled by first converting them into triangle meshes. The process of generating multiresolution models is divided into two stages, *the pre-processing stage* and *the run-time stage.* During the pre-processing stage, we calculate the visual importance value (or simply *the importance value*) of each vertex in the model according to the local geometry. Based on these vertex importance values, we also calculate the importance value of each triangle edge in the model. All edges are then divided into groups according to their importance values. During the run-time stage, we remove a specified number of triangles from the model. This number is determined by the time available to render the object (Green, 1996) and the distance of the object from the viewer at a particular frame. To remove triangles without re-triangulation, we introduce an operator called *edge collapse.* Edge collapse works by merging one of the two vertices of an edge onto the other one; the former vertex is referred to as the *collapsed vertex* and the latter one as the *surviving vertex.* The removal of triangles is achieved by collapsing edges starting with those in the lowest importance group. The method is summarized in the following table, and we discuss the various operations in detail in the following subsections.

Pre-processing stage
- Calculate the visual importance of all vertices.
- Calculate the visual importance of all edges.
- Create the visual importance table by dividing all edges into groups.

Run-time stage
- Select an edge to collapse from the visual importance table.
- Check for edge crossover.
- Collapse the edge, and update neighboring geometric information.
- Recalculate the visual importance of all affected vertices and edges.

4.1 Visual Importance of a Triangle Vertex

In order to define the importance of a vertex, we classify vertices into three types, namely *flat vertices, edge vertices,* and *corner vertices.* A flat vertex is located on a

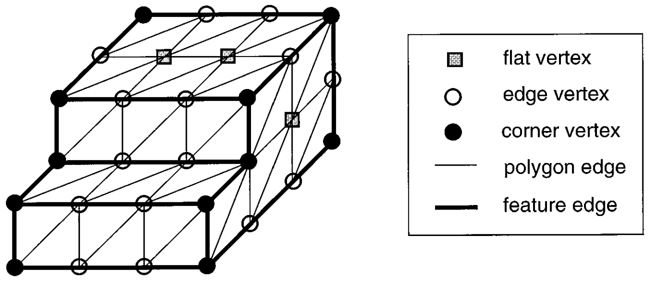


Figure 1. *Three types of vertices: flat, edge, and corner.*

locally flat surface and, therefore, all triangles connected to it have similar orientations. A flat vertex can be removed if needed. By contrast, an edge vertex is located along a feature edge of an object and hence all the triangles connected to it can roughly be divided into two groups according to their orientations. An edge vertex can be removed if the collapse is along the feature edge. Finally, a corner vertex is located at a sharp corner of the surface, which is usually the intersection point of multiple feature edges. The removal of a corner vertex will certainly affect the geometry of the object model and hence it should be the last to be removed. Figure 1 shows examples of the three types of vertices.

To determine the visual importance of a vertex, we compute the largest angle between any two normal vectors of the triangles connected to the vertex. Although this largest angle can be computed by comparing all the normal vectors with each other, this comparison is expensive to do at run-time when we need to recalculate the visual importance of all affected vertices and edges after an edge-collapse operation. Instead, we approximate the angle by finding the minimum and maximum values $(x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})$ of all triangle normal vectors around the vertex. The vertex importance is then calculated as follows:

$$V_{imp} = (x_{max} - x_{min}) + (y_{max} - y_{min}) + (z_{max} - z_{min}) \quad (1)$$

If V_{imp} is smaller than a given threshold, V_{Thres}, the vertex is considered as a flat vertex; otherwise, if the normal vectors of the triangles connected to the vertex can be classified into exactly two groups with respect to their orientations, the vertex is considered as an edge vertex. The importance values V_{imp_1} and V_{imp_2} of the two groups are calculated independently with Equation 1. The importance value of the edge vertex is then calculated as:

$$V_{imp} = \mathrm{Max}(V_{imp_1}, V_{imp_2}) \quad (2)$$

If a vertex is neither a flat vertex nor an edge vertex, it is considered as a corner vertex.

4.2 Visual Importance of a Triangle Edge

Once we have calculated the importance of all vertices, we proceed to calculate the importance value of each triangle edge. This importance value indicates how much the geometry of the model will be affected if the edge is collapsed. Currently, we consider two factors, the length of the edge and the importance values of its two vertices as follows:

$$E_{imp} = \frac{|E|}{L_{ref}} * \mathrm{Min}(V_{1imp}, V_{2imp}) \quad (3)$$

where V_{1imp} and V_{2imp} are the importance values of the two vertices V_1 and V_2, respectively, of edge E. $|E|$ is the length of E. L_{ref} is a reference length for bounding $|E|$ so that the edge importance values can be quantized into groups. In our implementation, L_{ref} is proportional to the maximum extent of the object in the x, y, or z direction. Both V_{1imp} and V_{2imp} are already bounded by the threshold value, V_{Thres}. We consider $|E|$ in calculating the edge importance because it indicates how important the two adjacent triangles are. A short edge indicates that the adjacent triangles are either small or degenerate. A small triangle covers relatively small area and causes less error than a larger one if removed. A degenerate triangle reduces the performance of the graphics hardware and is more prone to rendering and numerical errors. In either case, the triangle should have a higher priority to be removed, and hence the edge should be assigned a lower visual importance value. For the second term of the equation, the reason for choosing the minimum of the two vertex importance values is that we may perform an edge collapse when one of the two vertices has a low visual importance value. The details of this will be discussed in Section 4.5.

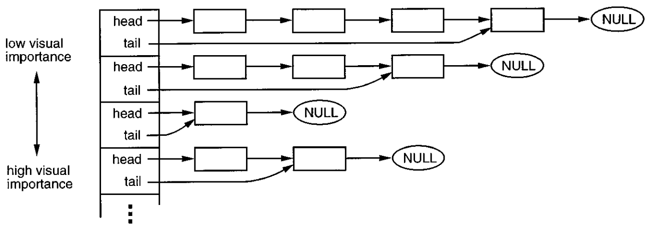


Figure 2. *The visual importance table.*

4.3 The Visual Importance Table

In our earlier method (İşler, Lau, & Green, 1996), we perform a full sorting on all triangles according to their importance values. Triangles are removed starting with the one having the lowest value. During run time, this sorting process is also needed to reorder the priorities of some triangles whose importance values are changed by the latest edge collapse operation. The complexity of the sorting process is $O(nT)$, where n is the number of edges to be removed, and T is the number of triangles in the model. To eliminate this sorting process, we adopt a quantization method here. We divide the maximum range of edge importance values into a fixed number of groups. In our current implementation, we use 100 to 1,000 groups depending on the size of the model. Each group is basically a linked list of triangle edges whose importance values fall inside the range covered by the group. The list is not sorted and is maintained in a first-in first-out manner. When an edge is inserted in a list, it is added to the end of the list. When an edge is to be removed from a list, it is taken from the beginning of the list. Figure 2 shows an example of the visual importance table.

We set up the visual importance table in the pre-processing stage. During the run-time stage, we simplify the object model by collapsing triangle edges one at a time taken from the edge list of the lowest visual importance group. Normally, two triangles will be removed as a result of an edge collapse, except at the boundary of a surface where only one triangle will be removed. After an edge collapse, all the edges that were originally connected to the collapsed vertex are now connected to the surviving vertex. We need to re-compute their importance values. If the new importance value of an affected edge is outside the importance range of its current group, it is moved to the end of the edge list in the appropriate group.

The idea of quantizing the edge importance values is to trade off accuracy for efficiency. This quantization method is much more efficient than the original sorting method. However, it causes errors, as all edges within a group are considered as equally important. A way to reduce this quantization error is by increasing the number of groups in the table, but this will increase the memory cost too.

4.4 Edge Collapse

In our earlier work (İşler, Lau, & Green, 1996), we proposed two operators, *edge collapse* and *triangle collapse,* for simplifying a triangle mesh, with triangle collapse being the preferred operator because it can remove four triangles each time instead of two, as with the edge collapse operator. In this work, we suggest the use of the edge collapse operator alone for simplifying object models for the following reasons:

- The triangle-collapse operation can be separated into two edge-collapse operations.
- The use of the edge collapse operator alone further simplifies the multiresolution method because only one operator is used.
- There is only one type of data structure stored in the simplification list instead of two.

An edge collapse merges two vertices of an edge into a single point. Ideally, the final point should be somewhere along the edge to be collapsed so as to minimize the error of the approximation. However, we choose to have the final point as one of the two original vertices because it is fast and does not require the creation of new vertices. Figure 3 shows an example of the edge collapse operation.

4.5 Edge Collapse Criteria

In order to minimize the change in the geometry of the model, edges are collapsed according to their importance values. An edge has a low importance value if it

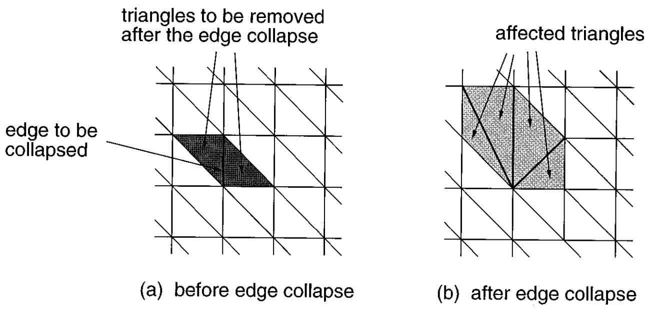


Figure 3. *Collapsing an edge into a point.*

is not important in defining the geometry of the model, and hence its removal does not cause a high error between the simplified and the original models.

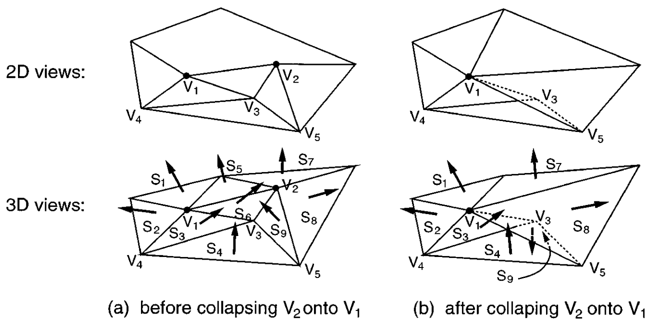
Since each edge has two vertices and each vertex may be classified as flat, edge, or corner, there are six possible edge types. If we denote a flat vertex as *F*, an edge vertex as *E*, and a corner vertex as *C*, the six edge types are *F–F*, *F–E*, *E–E*, *C–F*, *C–E*, and *C–C*. If both vertices are of type flat, we can always perform an edge collapse by merging the vertex with lower visual importance to the other one. If only one of the two vertices is of type flat, we can still perform an edge collapse by merging the flat vertex to the other vertex. If both of them are of type corner, we should not perform a collapse unless all the remaining vertices in the model are of type corner too. An edge collapse in this case will certainly affect the geometry of the model. However, at this time, the object may already be too far away for the change to be visible. For the remaining types, *E–E* and *C–E*, we may perform an edge collapse only if the two vertices share a common feature edge. The following table shows a summary of the six cases.

4.6 Detecting Crossover Edges and Collinear Edges

Normally, when we collapse an edge, the geometry of the model remains essentially the same since the orientations of the neighboring triangles do not change significantly after the collapse. However, there is an exceptional situation. When we collapse an edge, some neighboring edges may cross over the others, resulting in a dramatic change in orientations for some triangles.

TABLE I

Cases	Edge types	Collapse directions	Conditions
1	*F–F*	↔	Collapse to the vertex with lower visual importance.
2	*F–E*	→	Always collapse to the edge vertex.
3	*E–E*	↔	Collapse only when they share a common feature edge.
4	*C–F*	←	Always collapse to the corner vertex.
5	*C–E*	←	Collapse only when they share a common feature edge, and always collapse to the corner vertex.
6	*C–C*	x	No collapse.



2D views:
3D views:
V1
V2
V3
V4
V5
S1
S2
S3
S4
S5
S6
S7
S8
S9
(a) before collapsing V2 onto V1
(b) after collaping V2 onto V1

Figure 4. *Crossover of triangle edges.*

We need to detect this situation and prevent such a collapse. Figure 4 shows an example of this situation. Although both V_1 and V_2 may be considered as flat vertices, as we collapse V_2 onto V_1, edge V_2–V_5 will become V_1–V_5 and cross over edge V_3–V_4 as shown in the 2D view of Figure 4(b). This will cause triangle S_9 to have a dramatic change in orientation, resulting in a visual discontinuity as shown in the 3D view of Figure 4(b).

To prevent this result, we compare the normal vectors of each affected triangle before and after the collapse. If the change in orientation of a triangle is found to be too high, the collapse is not performed. Since this crossover problem does not occur very often and we need to update the normal vector of the triangle anyway if the col-

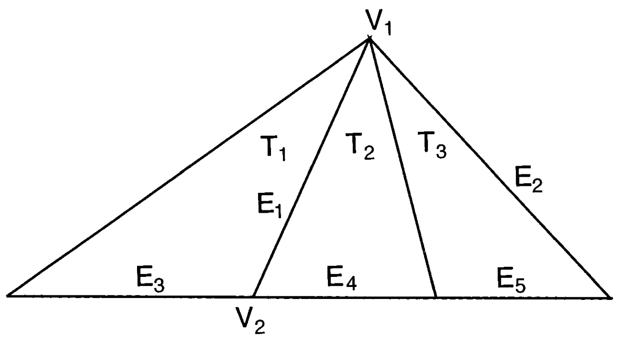


Figure 5. *Edge collapse problem with collinear edges.*

lapse is performed, the cost of this method is only an extra comparison.

In addition, it is sometimes possible that collapsing an edge will remove three or more triangles. This happens when some edges are collinear. For example, in Figure 5, E_3, E_4, and E_5 are collinear edges. If we collapse V_1 onto V_2, all three triangles will be removed. This situation can also be detected with the above method. If E_1 in Figure 5 is collapsed first, triangle T_3 will become a line and its normal vector after the collapse will be zero. To prevent creating a new type of data structure to handle this situation, we do not allow E_1 to collapse. Instead, we may consider collapsing E_5 first, for example, before we consider collapsing E_1.

5 The Simplification List

In the previous section, we presented a method for generating real-time multiresolution models. Although the method is very efficient in removing triangles as the object moves further away from the viewer, it is not efficient if the object moves towards the viewer. Consider the situation when an object is moving away from the viewer. Between two consecutive frames the object moves a little farther away from the viewer and hence an additional few triangles are removed from the object model. In this situation, the method works fine even in environments containing a large number of objects. However, if the object moves towards the viewer, between two consecutive frames the object moves a little bit closer to the viewer and hence an additional few triangles need to be added to the model to provide more detail. Unfortunately, we do not know where the triangles should be inserted. Hence, although the difference may be only a few triangles, we need to simplify from the high-resolution model to the intended resolution during every frame. If a lot of objects are moving simultaneously towards the viewer, the cost of simplification will become too high to be practical.

We have developed a simple method to overcome this problem: cache the most recent sequence of edge collapses in a last-in-first-out circular buffer, called the simplification list. A record is stored into the buffer whenever an edge is removed from the model. When the buffer is full, the oldest records will be replaced by the most recent records. If the object moves farther away from the viewer, edges are removed from the models and records are inserted into the buffer. If the object moves towards the viewer, we perform an ''un-collapse'' operation by reconstructing the model from the information stored in the buffer. The most recently inserted records provide information on which edges (or triangles) are to be reconstructed. With this improvement, we need to simplify from the initial high-resolution model only when the object moves a large distance toward the viewer, which results in exhausting the buffer. (Here, a low watermark may actually be used to trigger the start of the simplification process before the buffer is completely exhausted.) This method relieves the need to simplify from the high-resolution model to the intended resolution once per frame, to once every several frames as the object moves towards the viewer. The actual number of frames depends on the size of the buffer. This improvement basically trades off memory for efficiency. The data structure for storing the information about an edge collapse is as follows:

```
typedef struct {
 short colvertex; /* colapsed vertex of the edge */
 short survertex; /* surviving vertex of the edge */
 short remedge[2]; /* two removed edges */
 short suredge [2]; /* two surviving edges */
} EdgeRecord;

typedef EdgeRecord simplificationlist[MaxEdges];
```

The information needed in order to execute an edge un-collapse operation is stored in an EdgeRecord. Here

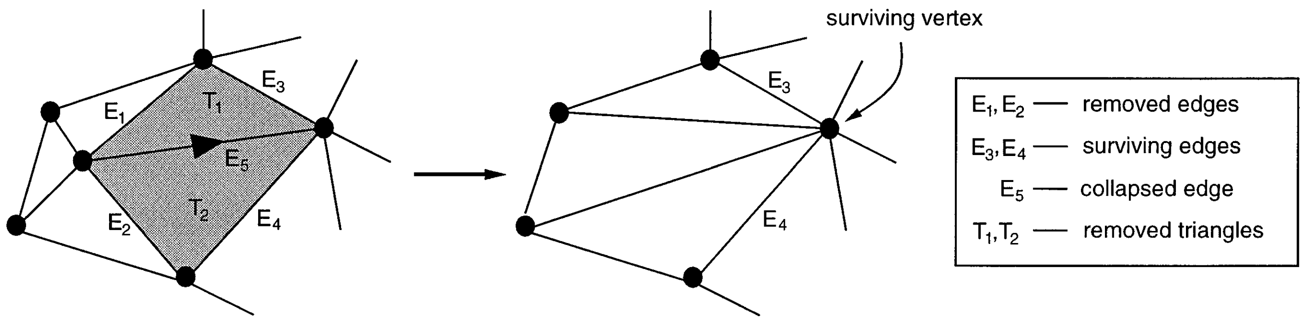


Figure 6. *Triangle edges affected by an edge collapse.*

colvertex and survertex store the IDs for the collapsed vertex and the surviving vertex, respectively, of an edge collapse. The two locations in remedge store the IDs of the two removed edges due to the removal of the two adjacent triangles, and those in suredge store the IDs of the two surviving edges of the two triangles as shown in Figure 6.

To take this idea to the extreme, we may pre-generate a complete simplification list of the model. We simplify an object model from the highest resolution to the lowest one and save all the edge collapse records into a *full simplification list* in advance. During run time, we use the full simplification list to reconstruct the model of a particular resolution, whether the object moves closer to or farther away from the viewer. The major advantage of creating a full simplification list is speed. We no longer need to determine at run time which edges to collapse. The sequence of edge collapses are all determined in advance. In the multiresolution method discussed in the previous section, in order to achieve real-time performance, we simplify a lot of tests when selecting edges to collapse. For example, we divide all edges into groups according to their visual importance to save sorting time. If the simplification process is done in advance, we may perform more checks to produce more accurate multiresolution models. This approach of generating more accurate models is taken by Hoppe (1996).

6 Results and Discussions

We implemented our method in C++ and OpenInventor. Figure 7 shows some output images of a face model with the multiresolution method. Images in column (i) show the model in full resolution (4,356 triangles). Those in columns (ii), (iii), and (iv) have 3,000, 2,000, and 860 triangles, respectively. On the other hand, the images in row (a) show the model at different resolutions and placed at appropriate distances from the viewer. The images in row (b) show the close-up views of the model at different resolutions. The images in row (c) show the triangle mesh of the model at different resolutions. Figure 8 shows the output images of a landscape at different resolutions (triangles: 9,620 → 5,690 → 1,680).

We tested the program on an SGI Indigo^2 workstation with a 195MHz R10000 CPU and Solid Impact graphics accelerator. Figure 9 shows the performance of our multiresolution method, i.e. the run-time performance of the edge collapse operator without the simplification list, in simplifying the face model shown in Figure 7. We can see that the number of triangles removed is close to linear in the time taken to remove them. We have determined that the method can eliminate about 8,700 triangles per second. At first glance, this number may appear to be small. However, this number represents an incremental change in the number of triangles per second. On the other hand, we would expect to be able to achieve a two to three times performance improvement if we could implement the algorithm in a low-level language. We have compared the new method with our earlier method (İşler, Lau, & Green, 1996) under the same system configuration. The new method is over three times faster than the old one. We have also measured the pre-processing time of the new method on the face model as shown in Table 2.

From the figures, we can see that the pre-processing time of the multiresolution method is short, and the

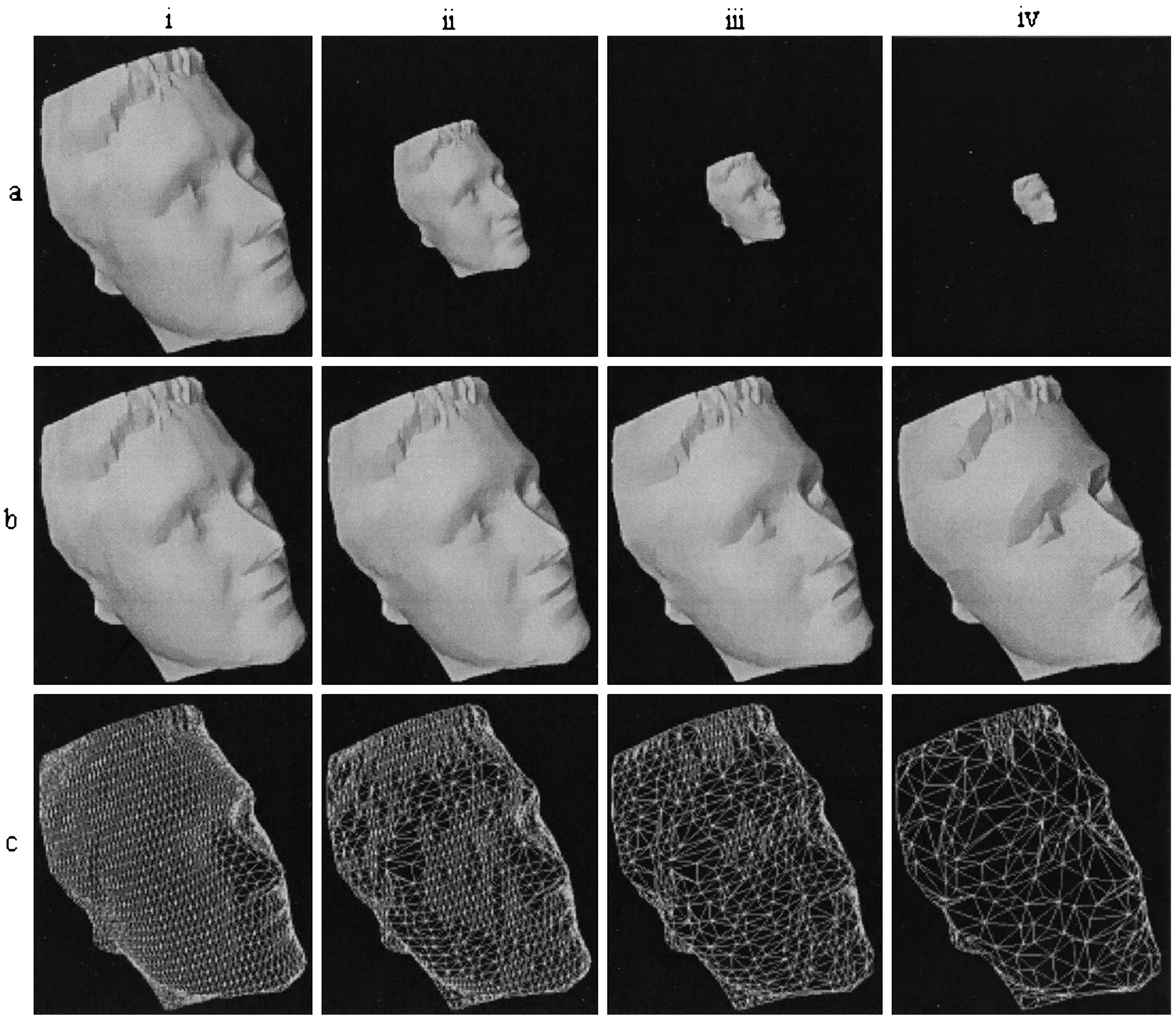


Figure 7. *A face model generated at different resolutions.*

creation of the full simplification list is also very efficient. (Note that the 0.47 s already includes the time needed to collapse all the edges in the model using the multiresolution method.) The computational complexities of both the pre-processing algorithm and the run-time algorithm are proportional to the number of edges to collapse. Since, in most situations, an edge collapse causes two triangles to be removed, the computational complexity of the method is $O(n)$, where n is the number of triangles in the model.

Figure 10 shows the performance of edge collapses and edge uncollapses when the simplification list is used. We have determined that the system can collapse approximately 133,333 triangles per second and can uncollapse approximately 160,000 triangles per second.

Figure 11 shows the frame rates of rendering the face model shown in Figure 7 as it moves from the closest point (point 0 in the diagram) to the point farthest away (point 80 in the diagram). When the model is at the closest point, it covers roughly one-third of the screen and when it is at the farthest point, it covers less than 50 pixels. When there is no simplification to the model (i.e.,

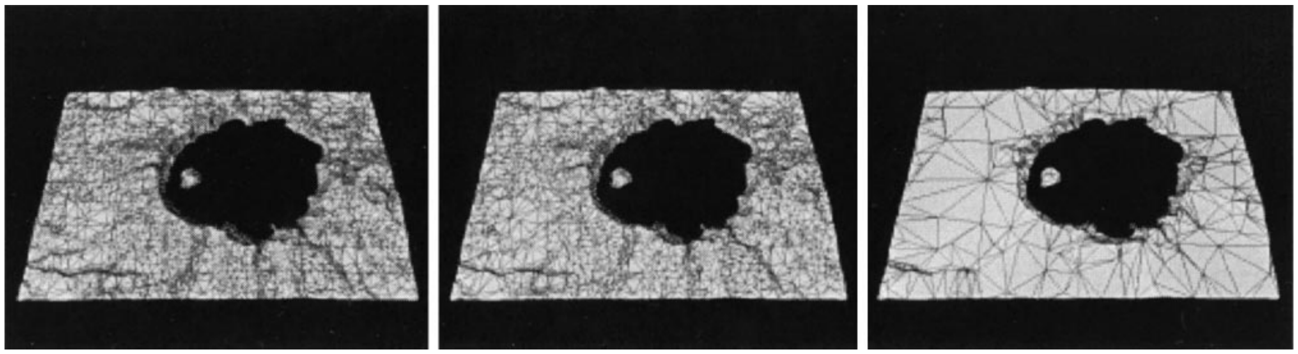


Figure 8. *A landscape model generated at different resolutions.*

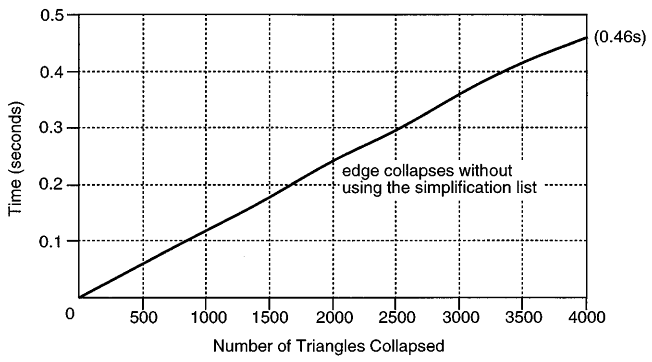


Figure 9. *Performance of the multiresolution method.*

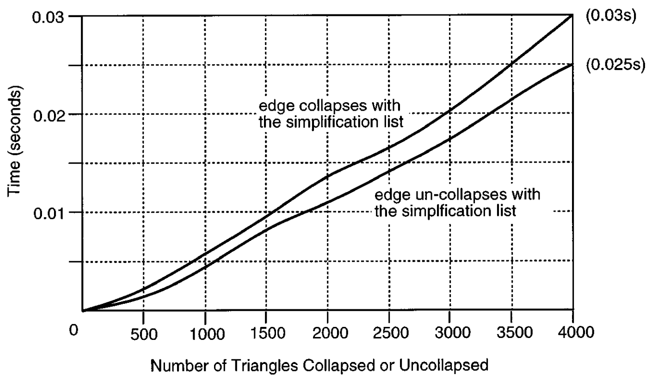


Figure 10. *Performance of edge collapses and uncollapses with the simplification list.*

TABLE 2

New method (on face model)	Time
Calculation of vertex and edge importances	0.03 s
Creating the visual importance table	0.00 s
Creating the full simplification list	0.47 s

when the model contains 4,356 triangles) the rendering speed is always 36 frames per second. Therefore, the frame rate does not depend on the distance or the projected size of the model. However, when simplification is used, the frame rate increases as we remove more and more triangles from the model. The numbers shown at the top of the diagram indicate the total numbers of triangles removed from the model at the corresponding distances from the viewer. The time needed for the simplification process is considerably smaller and depends on the moving speed of the model. For example, if we maintain a frame rate of 20 in a particular virtual reality application and the model takes 20 seconds to move from the closest point to the farthest point, the total number of frames for the duration will be 400. This means that we only need to remove 10 triangles from the model during each frame time, and the time spent on removing these triangles will only be 75 μs with the simplification list.

We also tested the method on a PC with a Pentium 166 MHz CPU; the system can both collapse and uncollapse 80,000 triangles per second. In terms of memory usage, each edge collapse stores two vertex IDs and four edge IDs. If each ID requires two bytes to store, each edge collapse will require 12 bytes. The face model shown in Figure 7 has a total of 6,633 edges. The total amount of memory needed for the full simplification list is approximately 78 kilobytes. Of course, if memory space is tight, we may only keep a small section of the

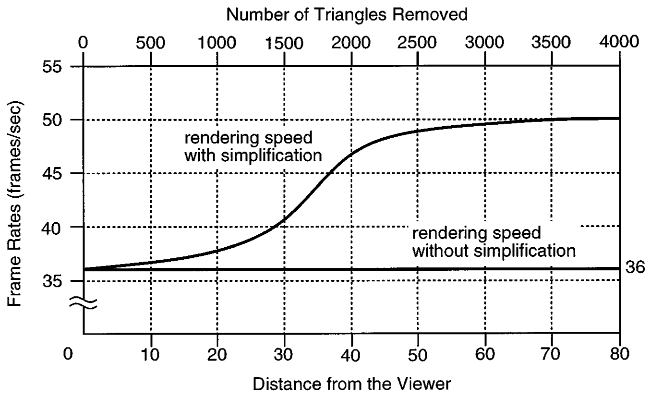


Figure 11. *Frame-rate improvement with model simplification.*

simplification list in main memory and use a prediction algorithm to pre-fetch other sections on demand.

It would be interesting here to consider the performance of the simplification list method if the face model deforms. When the surface deforms, we need to recalculate the importance values of all vertices and edges, recreate the visual importance table, and recreate the full simplification list. If the surface deforms by changing the 3D positions of the vertices in the original high-resolution face model, the data structures of the model do not need to change. We may gradually increase the resolution of the model based on the order of the simplification list as the surface is deforming and the vertex positions are being updated. As soon as the deformation stops, the simplification list is recomputed by a separate processor. The current processor continues to use the high-resolution models for rendering until the list is recomputed. The amount of time needed to recompute the full simplification list is 0.5 s (0.03 s + 0.00 s + 0.47 s). Of course, if the deformation is small, we may only need to update the vertex positions in the model without recomputing the simplification list.

7 Conclusions

In this paper, we have presented an efficient multiresolution method. Not only is this method fast, but it also preserves the geometry of the model as much as possible. We have also presented an improved algorithm to the multiresolution method based on the simplification list. Similar to Hoppe's method, this enhancement stores the pre-computed sequence of edge collapses. With the simplification list, it is possible to decrease or increase the resolution of the model simply by following the list in the forward or backward direction. Unlike Hoppe's method, our method is based on a fast multiresolution method and hence can be applied to deformable objects. Towards the end of the paper, we have discussed the performance and memory usage of the method. Since the new method has linear triangle insertion and deletion rates, the time needed to increase or decrease the resolution of a model by a certain amount can easily be predicted. Hence, the new method fits well into our framework for time-critical rendering in VR (Green, 1996).

Currently, we are studying the possibility of parallelizing the multiresolution method. There are two approaches. The first approach is to distribute the task of creating the full simplification list to different processors in the machine to improve the performance of creating the list if the object deforms. The second approach is to cluster objects of different behaviors into groups and distribute these object clusters to other processors when the current processor is not fast enough to update the resolutions of all objects. We are also working on an adaptive approach to the multiresolution method described here, so that different parts of a large model may have different amount of simplification depending on some run-time factors such as the viewer's line of sight. Such a method will allow local deformations to be handled more efficiently. (Lau, To, and Green, 1997) described some initial results of this idea.

Acknowledgments

We would like to thank the anonymous reviewers for their useful comments and suggestions. This work is supported in part by the University Central Research Grant number 351/084.

References

Cohen, J., Varshney, A., Manocha, D., Turk, G., Weber, H., Agarwal, P., Brooks, F., & Wright, W. (1996). Simplification envelopes. *ACM Computer Graphics (SIGGRAPH'96)*, 119–128.

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

Eck, M., DeRose, T., Hoppe, H., Lounsbery, M., & W. Stuetzle. (1995). Multiresolution analysis of arbitrary meshes. *ACM Computer Graphics (SIGGRAPH'95)*, *29*, 173–182.

Falby, J., Zyda, M., Pratt, D., & Mackey, R. (1993). NPSNET: Hierarchical data structures for real-time three-dimensional visual simulation. *Computers and Graphics*, *17*(1), 65–69.

Funkhouser, T., & Séquin, C. (1993). Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *ACM Computer Graphics (SIGGRAPH'93)*, *27*, 247–254.

Green, M. (1996). A framework for real-time rendering in virtual reality. *ACM Symposium on Virtual Reality Software and Technology*, 3–10.

Heckbert, P., & Garland, M. (1995). Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, School of Computer Science, Carnegie Mellon University.

Herzen, B., & Barr, A. (1987). Accurate triangulations of deformed, intersecting surfaces. *ACM Computer Graphics (SIGGRAPH'87)*, *21*, 103–110.

Hoppe, H. (1996). Progressive meshes. *ACM Computer Graphics (SIGGRAPH'96)*, 99–108.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., & Stuetzle, W. (1993). Mesh optimization. *ACM Computer Graphics (SIGGRAPH'93)*, *27*, 19–26.

İşler, V., Lau, R. W. H., & Green, M. (1996). Real-time multiresolution modeling for complex virtual environments. *ACM Symposium on Virtual Reality Software and Technology*, 11–20.

Lau, R. W. H., To, D., & Green, M. (1997). An adaptive multi-resolution modeling technique based on viewing and animation parameters. *IEEE Virtual Reality Annual International Symposium*, 20–27.

Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L., Faust, N., & Turner, G. (1996). Real-time, continuous level of detail rendering of height fields. *ACM Computer Graphics (SIGGRAPH'96)*, 109–118.

Lounsbery, M., DeRose, T., & Warren, J. (1993). Multiresolution analysis for surfaces of arbitrary topological type. Technical Report TR931005b, Department of Computer Science and Engineering, University of Washington.

Rossignac, J., & Borrel, P. (1992). Multi-resolution 3D approximations for rendering complex scenes. Technical Report RC 17697 (#77951), IBM Research Division, T.J. Watson Research Center.

Schroeder, W., Zarge, J., & Lorensen, W. (1992). Decimation of triangle meshes. *ACM Computer Graphics (SIGGRAPH'92)*, *26*, 65–70.

Turk, G. (1992). Re-tiling polygonal surfaces. *ACM Computer Graphics (SIGGRAPH'92)*, *26*, 55–64.