In Proceedings of ACM Symposium on Virtual Reality Software and Technology, pp. 11-20, July 1996.

Real-time Multi-resolution Modeling for Complex Virtual Environments

Veysi İşler Rynson W.H. Lau
Computer Graphics and Media Laboratory
Department of Computing
The Hong Kong Polytechnic University, Hong Kong

Mark Green
Department of Computer Science
University of Alberta
Alberta, Canada

Abstract

The rendering and animation of complex scenes containing many objects represented by large numbers of display primitives requires time proportional to the total number of primitives in the scene. To improve the performance of a given graphics engine, multi-resolution modeling techniques are usually used to reduce the total number of primitives required to be rendered. However, existing techniques are usually slow and some of them may not even preserve the topology of the object geometry faithfully. In this paper, we present an efficient and simple algorithm to generate an object model of the desired resolution given a high resolution model of an object and the distance of the object from the view point. The major advantages of the new method are that it preserves the topology of the given model, it can be used to generate multi-resolution models on the fly and in real-time, and its performance is predictable. The last advantage makes it well fit into the framework that we proposed in our work on real-time rendering in virtual reality[1].

1. Introduction

Real-time generation of realistic looking images is essential for interactive computer graphics applications and real-time computer graphics animation. Virtual reality walkthroughs, flight simulation, and scientific visualization are some example applications that require real-time rendering. Although a graphics workstation may be able to generate images in real-time for some particular scene, as the scene becomes more complex, the performance of the rendering and animation system may degrade drastically and eventually, it will fail to deliver images at a high enough rate. Our current research direction [1] is to develop a rendering technique that can deliver images with the highest possible quality within a given time interval, so that the frame rate can be maintained. To be able to generate images within a given time interval, we need to be able to predict the time it takes to process and to render each of the objects in the scene. To be able to generate an image with the highest possible quality within the given time interval, we may consider rendering the objects in the scene adaptively so that objects with higher visual importance are rendered with more details, while those with lower visual importance are rendered with less details. This paper concentrates on the second issue.

Simply representing all objects in the scene with fewer primitives will of course reduce the time it takes to render the image. However, it will lead to inaccurate object models and may result in producing images with misleading information. A common technique used to render objects adaptively is called *multi-resolution modeling* or *level of detail modeling*. Due to the fact that distant objects (objects which are far away from the view point) will occupy smaller pixel areas than nearby objects, they are considered as visually less important than nearby objects. Since the details of these objects are not visible anyway, distant objects may be simplified without sacrificing the accuracy of the resultant image. This can be achieved by having the number of primitives representing an object inversely proportional to the distance of the object from the viewpoint.

Since most existing methods for generating multi-resolution models are slow [2, 3, 4], these models are usually generated in advance. Each object is represented by a few key models with different resolutions, and the distance of the object from the view point at a particular frame determines which model to use for rendering the frame. One limitation of these methods is that they require extra memory space to store multiple models of each object. If a scene database contains a large number of objects, a lot of memory will be needed to store them. In an interactive computer graphics application such as virtual reality walkthrough, all object models in the vicinity must be kept in main memory. The amount of main memory available therefore limits the complexity of the scene. Another limitation of using a fixed number of pre-generated models to represent an object is that when the object crosses the threshold distance, there is a sudden change of models used to represent the object. This may create an objectionable visual effect. Hence, a technique which can generate multi-resolution models fast enough to be used in real-time applications may relieve the need of pre-generating the models and hence solve the above two problems.

In this paper, we present a novel technique for generating multi-resolution models of objects in a scene on the fly and in real time. The algorithm is simple to implement and requires no complicated data structures. The rest of the paper is organized as follows. Section 2 describes previous work on multi-resolution modeling. Section 3 presents our method in detail. Section 4 shows and discusses the timing results of some sample models. Section 5 discusses how we may modify the technique so that the multi-resolution models can be updated incrementally. Such an incremental method will further improve the performance of our multi-resolution technique. Finally, section 6 presents conclusions of the paper and discusses some possible future work.

2. Previous Work

Previous research related to multi-resolution modeling has been performed by scientists from different disciplines including computational geometry, applied mathematics, computer graphics, military simulation, and geographic information systems (GIS). Most of them try to simplify a given model by preserving the topology of the model geometry. The methods proposed vary according to the types of models they simplify. The types of models include polyhedral models, height fields, and curved surfaces. Flight simulators and GIS accept the height fields, also known as a triangulated irregular network (TIN), as input. A height field is typically a rectangular grid of elevation data which is triangulated for rendering. Polyhedral models consist of a set of polygons describing 3D objects in a scene. Virtual environments and scientific visualization applications use this sort of data as input.

Existing methods to generate multi-resolution models can be classified into *refinement methods* and *decimation methods* [5]. Refinement methods improve the accuracy of a minimal approximation of a model which is initially built [6, 7, 8, 9]. They are also called global reconstruction heuristics. Turk [8] handles the problem as a mesh optimization. He first distributes a number of vertices over the object surface based on the 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 topology for surfaces with high curvature. Hoppe et al. [9] optimizes 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 topology 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 find less accurate, but simpler version of the initial model [10, 11]. These methods are also called local simplification heuristics since they modify the model locally. Schroeder et al. [10] try to delete edges or vertices from almost coplanar adjacent faces. The resultant holes are filled up 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 multi-resolution models. In addition, since the method does not consider feature edges, the multi-resolution models generated may not preserve the original topology of the object geometry. Rossignac et al. [11] cluster the vertices which are close to each other to generate a new vertex for representing them. The polygonal object is then updated accordingly. Although this method is even faster than the previous method as it does not require the triangulation process, it again does not preserve the original topology of the object geometry.

Despite the fact that the two classes of methods can generate multi-resolution models of a given object, there are still some serious problems in this area. Firstly, these methods, especially the refinement methods, are very time consuming and very difficult to implement. Secondly, some of them do not even preserve the original topology of the object geometry. Thirdly, due to the high computational costs of these methods, a fixed number of key models representing an object at different distances are usually generated in advance. As mentioned in the previous section, this may cause objectionable visual effects due to the sudden change of levels of detail when an object moves across the threshold distance. Turk [8] proposed to have a transition period during which a smooth interpolation between the two successive levels of detail is performed to produce models of intermediate resolutions, this method further increases the computational time during the transition period because of the need to process the two levels of detail at the same time. Thus, there is a need to develop more efficient multi-resolution techniques which also preserve the original topology of the object geometry.

3. The Proposed Algorithm

Our goal in this research is to generate multi-resolution models on the fly and in real-time while preserving the topology of the object model as much as possible. The proposed algorithm falls into the class of decimation methods. In contrast to the other decimation algorithms, ours removes triangles instead of vertices from the given model. The removal of a triangle is carried out by merging two or three vertices of the triangle. Merging any two vertices means collapsing an edge of the triangle and the triangle will become a single line. Merging all three vertices means collapsing the triangle itself and the triangle will become a single point. The reason for choosing triangles to remove is to avoid triangulation of the hole resulting from the removal of a vertex. Such triangulation process is non-trivial and time consuming due to the many checks required.

In our method, we assume that all objects in the scene are composed of triangles which are commonly used to represent 3D objects. However, curved surfaces can also be handled by having a pre-process to convert them into triangle meshes. We sort all the triangles of a model in advance according to their visual importance. The visual importance of a triangle is in turn defined by the visual importance of its three vertices. During the rendering process, we remove a suitable number of triangles which have the least visual importance from the model before scan-converting it. However, when there are a large number of objects in the scene, it may be too costly to remove a potentially large number of redundant triangles from some of the models to create lower resolution models because removal of each triangle takes time. Here, we suggest a hybrid option. For each complex object in the scene, two or three key simplified models of the object are generated in advance as in the other non-real-time methods. When generating an image, the key model with the closest, but higher number of triangles than is needed will be chosen. We then remove triangles from the selected model until the total number of triangles in the model is just enough to represent the object at its current distance. Although this hybrid option increases the memory usage of our method, it can further improve the performance of our method by reducing the number of triangles needed to be removed in run time.

In order to define the importance of a triangle, we first classify vertices into three types namely *flat vertices*, *edge vertices* and *feature vertices*. A flat vertex lies on a relatively smooth surface and can be removed when needed. An edge vertex defines a feature edge of an object and can be removed only if some conditions are satisfied. A feature vertex defines the global shape of an object and therefore should be the last to be removed. The visual importance of a triangle is determined according to the classification of its three vertices. Basically, a triangle has a low visual importance if it is not important in defining the topology of the object geometry and hence its removal does not cause a high error between the simplified model and the original model. Since the analytic computation of the error caused by the removal of a triangle is quite expensive and is required for all possible cases, we try to approximate the error by employing some heuristics as described in this section. The algorithm provides two major operations for the removal of a triangle: *edge collapse* and *triangle collapse*. In edge collapse, two vertices of a chosen triangle are merged into one causing the removal of one or two triangles. Edge collapse enables the algorithm to preserve the features of the model as shown in Figure 1. Merging of all three vertices of a triangle results in triangle collapse which may remove from two to four triangles. Triangle collapse, on the other hand, is carried out on the triangles which have similar orientation with their neighbors as shown in Figure 2.

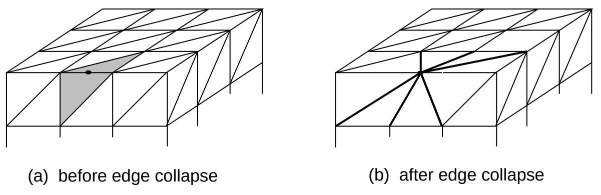


Figure 1: Collapsing of an edge into a point.

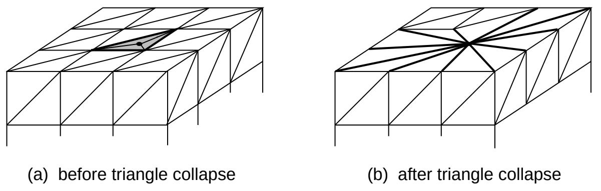


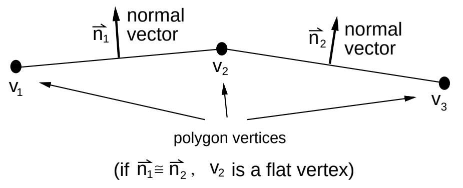
Figure 2: Collapsing of a triangle into a point.

At this point, it may be useful to compare our method with other decimation methods. The method proposed by Schroeder et al. [10] is based on vertex removal. The hole resulted from the removal of a vertex is patched using a rather complex triangulation method that must create non-intersecting and non-degenerate triangles. In some cases, the triangulation may fail when the chosen vertex and surrounding triangles are not removed from the triangle mesh. Certain special cases that occur during the triangulation process are treated by incorporating many checks in the algorithm. The simplification algorithms based on vertex removal must include such a triangulation process, which is quite expensive and hard to implement. It also requires the use of complex data structures. An interesting property of our method is that after the collapse, all the neighboring triangles that are connected to the original two (edge collapse) or three (triangle collapse) vertices remain triangular. Hence there is no need for a triangulation process. Both Hoppe et al. [9] and Scarlatos and Pavlidis [12] have also utilized the collapse of an edge and the shifting of vertices to simplify a given triangle mesh. Our approach is different from theirs in the following respects. Hoppe et al. consider the problem as an optimization problem which is solved accurately but slowly. The maximum dihedral angle of the edges at the destination vertex after the edge collapse is checked to decide whether the collapse should be rejected or not. The conditions and the evaluation of the edge collapse in our algorithm are based on the vertex types and the error measures that we define later in this paper. Scarlatos and Pavlidis, on the other hand, assume that the triangulation approximates a surface represented by regular height field. Their proposed algorithm does not apply to general triangle meshes. The error measure used is different from and more expensive than ours. Furthermore, their simplification method is based on the curvature equalization where curvature measure includes the mean curvature and the Gaussian curvature [12], which can be expensive to compute. In the following subsections, we presents a more detail description of our method.

3.1. Visual Importance of a Vertex

Similar to a triangle, a vertex has a low visual importance if it is not important in defining the topology of the object geometry and therefore, its removal does not cause a high error between the simplified model and the original model. As in other simplification algorithms, differences between normal vectors of the triangles around a vertex give the major clue about this. If the normal vectors are all similar, that vertex is not important in defining the topology of the model, and thus, may be removed without giving rise to a high error. Such a vertex is called a *flat vertex*, where all incident triangles have similar orientation. This is visualized in 2D in Figure 3. Since at most two line segments can be incident in a vertex in 2D space, a flat vertex will have two line segments with similar slope. When the slopes are significantly different, the vertex may no longer be a flat vertex. In 3D space, the importance of a vertex is computed as the largest angle between any two normals of the triangles incident in that vertex. Since this computation will be done for all vertices in advance and for vertices which are modified during the simplification process, it should be efficient to compute. This largest angle can be computed by comparing all of the normals with each other, which is an expensive operation. Comparing any two 3D vectors requires three floating point multiplications and two floating point additions. For the sake of efficiency, we have approximated this by finding the minimum and maximum values $(x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})$ of all triangle normals around a vertex. The absolute values of the differences between maximum and minimum values are used in determining the importance of the vertex:

$$V_{imp} = \text{Max}\begin{pmatrix} |x_{max} - x_{min}|, \\ |y_{max} - y_{min}|, \\ |z_{max} - z_{min}| \end{pmatrix} \quad (1)$$



$\vec{n}_1$ normal vector
$\vec{n}_2$ normal vector
v_1
v_2
v_3
polygon vertices
(if $\vec{n}_1 \cong \vec{n}_2$, v_2 is a flat vertex)

Figure 3: Flat vertex in 2D space.

If V_{imp} is almost zero, or does not exceed a given threshold value, the vertex is considered to be a flat vertex. Otherwise, the vertex may be classified as either an edge vertex or a feature vertex. An edge vertex appears along a feature edge of an object while a feature vertex appears at the intersection point of multiple feature edges as shown in Figure 4. Because a feature vertex is located at such important geometric position, it is considered important in defining the topology of the model geometry. Hence, it should not be removed unless there are no other vertices to choose from. Although an edge vertex helps to define the topology of a feature edge, it is considered as less important than the feature vertex. This vertex can be removed if either the removal of it does not affect the topology of the object or there are no other flat vertices to choose from.

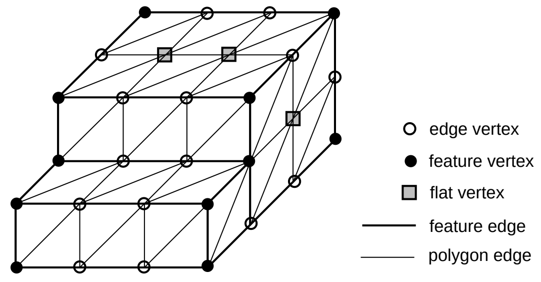


Figure 4: Vertex Types: Flat, Edge and Feature.

A vertex is an edge vertex if the normal vectors of the triangles connected to it can be classified into exactly two groups with respect to their directions. The orientations of the normal vectors within each group should not differ by more than a given threshold value. The normal vectors can be compared with each other using the absolute differences between the maximum and minimum values of the coordinates as above. Hence, the importance of an edge vertex is given as:

$$V_{imp_1} = \text{Max}\begin{pmatrix} |x1_{max} - x1_{min}|, \\ |y1_{max} - y1_{min}|, \\ |z1_{max} - z1_{min}| \end{pmatrix} \quad (2)$$

$$V_{imp_2} = \text{Max}\begin{pmatrix} |x2_{max} - x2_{min}|, \\ |y2_{max} - y2_{min}|, \\ |z2_{max} - z2_{min}| \end{pmatrix} \quad (3)$$

$$V_{imp} = \text{Max}(V_{imp_1}, V_{imp_2}) \quad (4)$$

Where the values $x1_{min}$, $x1_{max}$, $y1_{min}$, $y1_{max}$, $z1_{min}$ and $z1_{max}$ are the minimum and maximum values of the normal vector coordinates in the first group of triangles. Similarly, the other values are for the second group of triangles. The values V_{imp_1} and V_{imp_2} are importance measures, similar to those computed for flat vertices, but for the two groups of triangles connected to the edge vertex. These two measures tell us how close the triangles in each group are to each other. Their maximum is chosen as the importance of the edge vertex. All three types of vertices are shown on a sample model in Figure 4.

3.2. Triangle Collapse

In most existing decimation methods, a vertex which is not considered important, such as a flat vertex, is removed and the resultant hole is triangulated using a costly algorithm. In addition, they only distinguish flat and non-flat vertices. Hence an edge vertex defined above cannot be removed even if it is not important in describing the topology. In our method, a flat vertex which is not important is removed by merging it with another vertex (or vertices). An edge vertex is similarly removed by merging it with another vertex along the feature edge of the model, as long as it satisfies one of two conditions mentioned in the last subsection. Depending on the vertex type and the move, some of the triangles in the model will be deleted and some of them will be modified. There are two major types of vertex removal which result in triangle removal: triangle collapse and edge collapse. The following cases show how we collapse a triangle according to the types of its three vertices.

1. When all three vertices of a triangle are of type flat, the triangle will have edge and vertex neighboring triangles all with similar orientations. Therefore, we can collapse this triangle by merging all three vertices at a point over the triangle without any hesitation. The question is how we determine the location of the new point to minimize the error. To simplify the computation, in the current implementation, we use one of the three vertices as the new point. Collapsing a triangle will result in the removal of that triangle and all of the edge neighboring triangles. Hence if the triangle has three edge neighbors, the number of triangles in the model will decrease by four, while the number of vertices will decrease by two. Figure 5 shows an example of a triangle collapse.

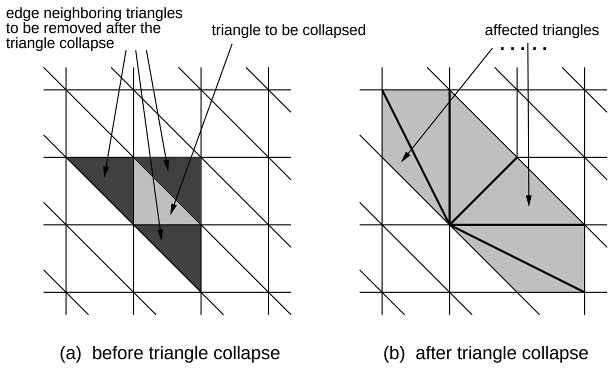

edge neighboring triangles to be removed after the triangle collapse
triangle to be collapsed
affected triangles
(a) before triangle collapse
(b) after triangle collapse

Figure 5: An example of triangle collapse.

2. When two vertices of a triangle are of type flat, the third vertex should be kept unaltered while the two flat vertices may be moved onto this vertex. This is a triangle collapse where two flat vertices are merged at the third non-flat vertex. In this case, again two vertices and up to four triangles are removed from the model definition.

3. When only one vertex of a triangle is of type flat, we can still remove some triangles without changing the topology of the model. In this case, the only flat vertex of the triangle is moved onto either one of the two non-flat vertices, reducing the number of vertices and triangles in the model by one and two, respectively. This move of a single flat vertex can also be considered as edge collapse which will be described later in the next subsection.

3.3. Edge Collapse

In addition to flat vertices, edge vertices may also be moved and merged with other vertices either with limited or no effect on the topology of the model geometry. Merging an edge vertex with a nearby vertex gives rise to edge collapse where one vertex and two triangles are removed from the model definition. To simplify the computation, in our current implementation, we simply move one of the two vertices onto the other vertex instead of creating a new one somewhere between the original two vertices. However, not every edge vertex of a triangle can be moved while preserving the topology. Let us consider the triangle T with vertices V_1, V_2 and V_3 in Figure 6. Vertex V_1 is a feature vertex, whereas vertices V_2 and V_3 are both edge vertices. As can be seen from the diagram, performing a triangle collapse of T will certainly destroy the topology, while collapsing the edge V_1–V_3 by moving vertex V_3 onto vertex V_1 will not affect too much the topology. On the other hand, it is not possible to move either V_1 or V_2 onto another vertex without seriously affecting the topology.

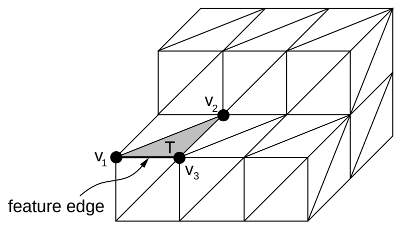


Figure 6: Edge collapse on a feature edge.

As defined earlier, an edge vertex is associated with two normal vectors representing two groups of triangles incident at that vertex. Whether an edge vertex can be moved or not is determined by its visual importance calculated in Equation (4), which basically measures the similar in orientations between the triangles within each group. If they are similar in orientation, which means that the edge vertex has a low visual importance, the vertex can be moved onto the vertex at the other end of the feature edge.

3.4. Detecting the Crossover of Edges

Normally, when we collapse a triangle or an edge, the topology of the geometry remains essentially the same because the orientations of the neighboring triangles do not expect to change too much after the collapse. However, there is a situation when this is not true. When collapsing a triangle or an edge, some neighboring edges may crossover the others and result in a dramatic change in orientations for some triangles. We must be able to detect this situation and prevent such a collapse. Figure 7 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 7(b). This will cause triangle S_9 to have a dramatic change in orientation and result in the change of local topology as shown in the 3D view of Figure 7(b).

To prevent the occurrence of this situation, we check the visual importance of the survived vertex (V_1 in Figure 7) if the collapse were performed. If there will be a dramatic change in visual importance, we do not perform the collapse. Since this crossover problem does not occur very often and we need to calculate the visual importance of the survived vertex anyway, the cost of this method is only an extra check.

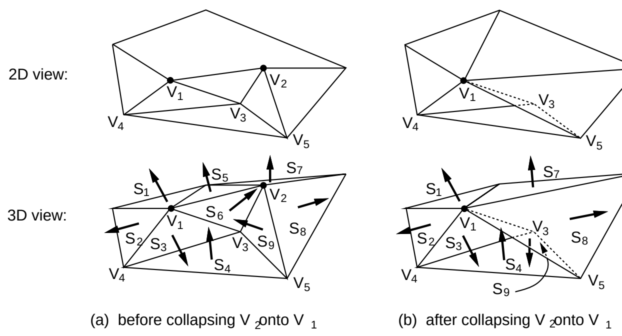

2D view:
3D view:
V1
V2
V3
V4
V5
S1
S2
S3
S4
S5
S6
S7
S8
S9
(a) before collapsing V2 onto V1
(b) after collapsing V2 onto V1

Figure 7: Crossover of Triangle Edges.

3.5. Visual Importance of a Triangle

To select which triangle to be removed, we need to determine the visual importance of each triangle; that is how much it will affect the resulting topology of the model if it is removed. Currently, we consider two factors, the area of the triangle and the visual importance of its three vertices. For example, the importance of triangle T in Figure 6 is calculated as follows:

$$T_{imp} \;=\; T_{area} \times \mathrm{Min}(V_{v1_{imp}}, V_{v2_{imp}}, V_{v3_{imp}}) \qquad (5)$$

where $V_{v1_{imp}}$, $V_{v2_{imp}}$ and $V_{v3_{imp}}$ are the visual importance of the three vertices v_1, v_2 and v_3, respectively, of triangle T and T_{area} is the area of T. A small triangle covers relatively smaller area of the object and causes less error if deleted than a larger one and hence it is considered to have a lower visual importance. This is due to the fact that small details will not be visible from a far distance.

This importance calculation is computed for each triangle in the model at the pre-processing stage and the triangles are then sorted with respect to their computed importance values. When reducing the resolution of a model, unimportant triangles are deleted before the triangles with higher importance values until the specified number of triangles are removed from the model. While triangles are deleted from the model, the importance of some triangles may change as a result of triangle collapse or edge collapse. This requires resorting of the affected triangles into the triangle list.

4. Results

We have tried our method on three simple models and one complex model. The results are shown in Figures 9, 10, 11 and 12 at the end of the paper. The first simple model, as shown in Figure 9, is an exact cube with 24 triangles where each face of the cube is represented by 4 triangles. Obviously, this cube can be represented by 12 triangles which is the minimum number of triangles for a cube. The algorithm classifies the 8 corner vertices as feature vertices which should not be changed. The rest of vertices, totally 6, one on each face are classified as flat vertices, which can be removed when needed.

The second model, shown in Figure 10, is a deformed cube defined by 36 triangles in total. In this sample model, the edge vertices play an important role in the simplification process. There is an edge vertex on each edge of the cube. The edge vertices are moved onto the corner vertices of the cube one by one according to the amount of deviation of the edge vertices from the edge of an exact cube. The resultant simplified model is approximated to an exact cube formed with corner vertices classified as feature vertices.

The third example, as shown in Figure 11, is a deformed stair like model with 60 triangles. This model contains all three types of vertices, namely flat, edge and feature. This example includes both edge collapse and triangle collapse. The resultant approximation easily preserves each stair step which are the major features of the model. The optimal approximated representation of this model contains 20 triangles.

The last sample model, as shown in Figure 12, is a human face model with 4356 triangles. We use this model to illustrate how the algorithm preserves the topology of the geometry on a real world object. The major features such as eyes, nose and lips are not changed after simplification.

We implemented the algorithm in C++ and ran it on an SGI Indigo^2 Workstation with Extreme Graphics Accelerator. Although the program is not optimized and the workstation was in multi-user mode during our experiments, the results are in fact very encouraging. Figure 8 shows the performance of removing triangles from the face model shown in Figure 12. We can see that the resultant curve is very close to linear. (Note that these figures do not include the preprocessing stage where normals, visual importance and neighbors of triangles are found and the vertices of the model are classified.)

5. Discussions

In the previous sections, we have presented the algorithm which generates multi-resolution object models with predictable performance. Although the algorithm is efficient, removing a large number of triangles may use up a lot of the available rendering time. To reduce the number of triangles needed to be removed at any one time, we have suggested a hybrid option which is to create two or three key simplified models of the objects in advance and the key model with the closest but higher number of triangles than is needed will be chosen for

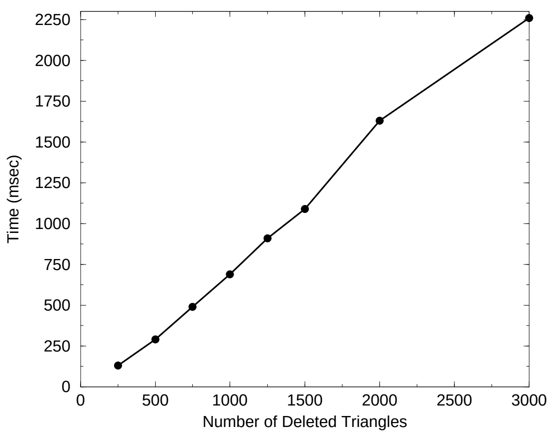


Figure 8: Timing for simplifying the human face model.

rendering. This option can greatly reduce the number of triangles needed to be deleted.

To further improve the performance of generating multi-resolution models, we are currently working on an interesting enhancement to our method. It is the incremental update of the multi-resolution models. In an interactive walkthrough, for examples, most objects either are stationary or move by a small amount from one frame to the next. To save time deleting triangles from the key models to generate the new models, we may delete triangles from the most recently generated models to create the new models. To implement this, we need to keep information on the list of triangles being removed from the key models. Each model is associated with one triangle removal list. When an object moves further away from the view point, triangles are removed from the multi-resolution model generated in the previous frame and added to the triangle removal list. When the object moves closer to the view point, the last inserted triangles are removed from the triangle removal list and added to the multi-resolution model.

The major idea of the incremental method is to spread the cost of producing a new model over multiple frames. Hence, in each update, there is only a small number of triangles to be removed from a given model; therefore less time is required, leaving more time for the scan-conversion and shading processes. The tradeoff of this is the extra memory needed to store the lists. However, the performance gained from this method may out-weight this limitation.

6. Conclusion

Real-time generation of multi-resolution models with predictable performance is essential in virtual reality applications. We have presented such an algorithm for simplifying a given triangular object model to generate multi-resolution models in real-time. The new algorithm removes triangles from a given object model with two operations, namely edge collapse and triangle collapse, while preserving its topology as much as possible in an efficient way. This is achieved by determining the visual importance of each triangle and ranking them according to the importance values during the pre-processing phase. In addition, one other advantage of the algorithm is that the storage requirement is minimal since no additional data structure is needed and only a couple of key models of each object are stored. The amount of storage required also affects the speed of the algorithm when the model contains large numbers of triangles or the scene contains too many models.

Our future work includes utilization of this algorithm in a virtual environment by incorporating techniques for the management of the generated multi-resolution models. In this respect, the method we use to calculate the visual importance of a triangle becomes very important. We are currently working on a more general model of it. Furthermore, we also intend to incorporate the method we describe here into the MR Toolkit [13]. Another future research direction is to improve the algorithm to minimize the error for curved surfaces. Although this algorithm can simplify curved surfaces with minimal error between the resultant and original models, this error may be further reduced if we calculate new vertices on the triangles and on the edges.

7. Acknowledgement

This work is currently supported by the University Central Research Grant, #351/084.

References

[1] M. Green. A framework for real-time rendering in virtual reality. *Also appear in this proceedings.*

[2] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[3] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 247–254, August 1993.

[4] P. S. Heckbert and M. Garland. Multiresolution modeling for fast rendering. In *Graphics Interface '94*, pages 43–50. Canadian Inf. Soc., May 1994.

[5] P. S. Heckbert and M. Garland. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, Carnegie Mellon University, School of Computer Science, August 1995.

[6] F. J. Schmitt, B. A. Barsky, and W. Du. An adaptive subdivision method for surface-fitting from sampled data. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 179–188, July 1986.

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

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

[9] P. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 19–26, August 1993.

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

[11] J. R. Rossignac and P. Borrel. Multi-resolution 3d approximations for rendering complex scenes. Technical Report RC 17697 (#77951), IBM Research Division, T. J. Watson Research Center, 1992.

[12] L. L Scarlatos and T. Pavlidis. Optimizing triangulations by curvature equalization. In *Visualization'92*, pages 333–339. IEEE Computer Society Press, 1992.

[13] C. Shaw, M. Green, J. Liang, and Y. Sun. Decoupled simulation in virtual reality with the mr toolkit. *ACM Transactions on Information Systems*, 11(3):287–317, 1993.

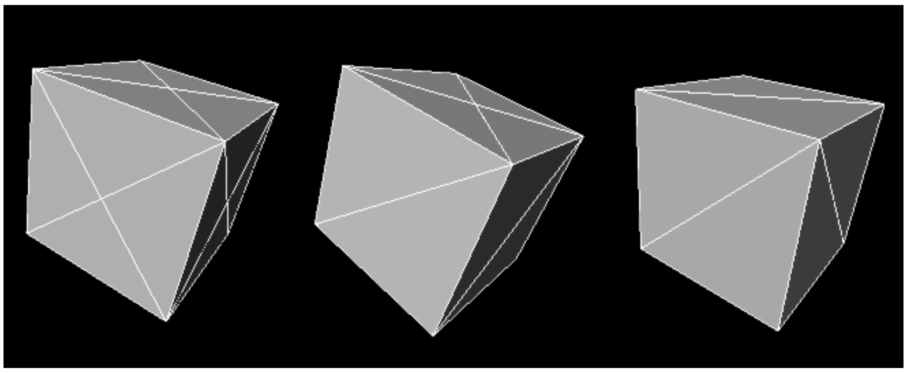


Figure 9: Simplification of a cube (Triangles: $24 \rightarrow 18 \rightarrow 12$).

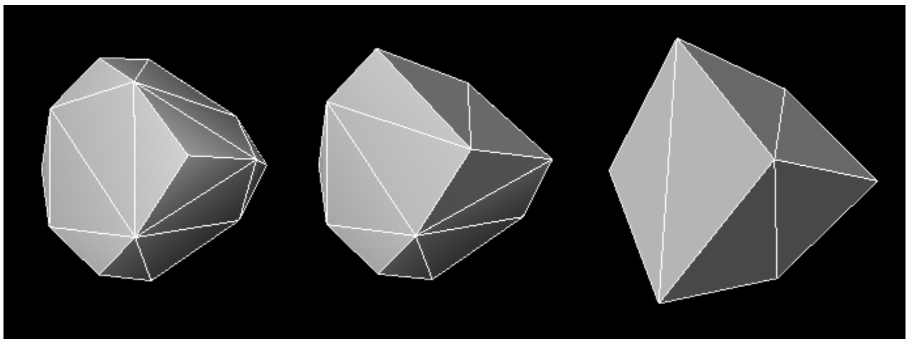


Figure 10: Simplification of a deformed cube (Triangles: $36 \rightarrow 18 \rightarrow 12$).

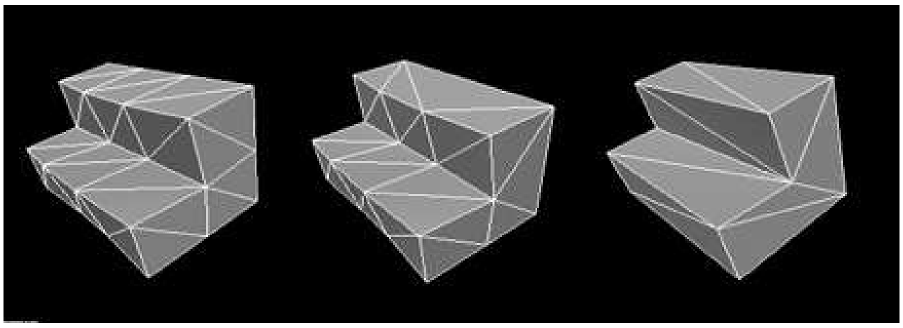


Figure 11: Simplification of a stair (Triangles: $60 \rightarrow 40 \rightarrow 20$).

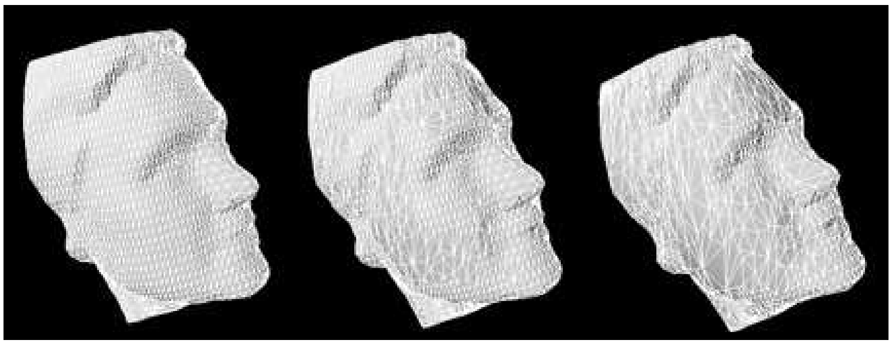


Figure 12: Simplification of a human face (Triangles: $4356 \rightarrow 3356 \rightarrow 2356$).