In Proceedings of IEEE Virtual Reality Annual International Symposium, pp. 20-27, Mar. 1997.

An Adaptive Multi-Resolution Modeling Technique Based on Viewing and Animation Parameters

Rynson W.H. Lau Danny S.P. To
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

Because most existing multi-resolution methods are slow, a common approach is to pre-generate a few key models of the object at different resolutions. During run-time, the object's distance from the viewer determines which model to use for rendering. Although this approach is simple, it suffers from the sudden change in resolution as the object moves across the threshold distance. In addition, the model used to represent an object at a particular frame is not optimized for the given dynamic viewing and animation parameters. The quadtree type of methods for arranging the surface model may allow adaptive multi-resolution modeling in a simple way and it reduces the sudden change of resolution from the object level to the node level. However, the square shape of the node together with the four-time increment in size for representing surfaces limits the types of surfaces that it can handle without creating excessive nodes. In this paper, we present a real-time adaptive multi-resolution method for models of arbitrary topology.

1. Introduction

Although the performance of graphics accelerators has improved tremendously, the demand for even higher performance to handle complex environments is increasing too. To overcome this demand for rendering performance, multi-resolution methods are usually used to reduce the rendering time by reducing the number of triangles (or polygons) needed to be processed. This is by considering the fact that distant objects occupy smaller screen areas than nearby objects, and hence, most of the details in these objects are not visible to the viewer. Multi-resolution methods optimize the rendering performance by representing a nearby object with a more detailed model, but a distant object with a simpler one.

Before we continue, let us introduce two terms here: *the static visual importance (SVI)* and *the dynamic visual importance (DVI)*. SVI refers to the visual importance of a point on the object calculated according to its geometric importance. For example, a point at a corner of an object may have a higher SVI value than a point on a flat surface. The DVI is the visual importance of a point on the object calculated according to some dynamic viewing and animation parameters. For example, a region of an object will have a higher DVI value if it intersects with the viewer's line of sight. A multi-resolution method that takes DVI into consideration is referred to as *adaptive multi-resolution method*.

Although there are many methods developed for generating multi-resolution models [1, 7, 16, 17], most of them focus on the accuracy of the simplification, and hence, are slow. Rossignac et al. in [15] proposes a very efficient multi-resolution method. However, this method does not preserve the topology of the object model. To generate accurate models without sacrificing the performance, the discrete multi-resolution method may be used. This method pre-generates a few key models of the object at different resolutions. During run-time, the object's distance from the viewer determines which model to use for rendering. Although this method is fast and simple, it has two major limitations. Firstly, when the object crosses the threshold distance, there is a sudden change in model resolution and an objectionable visual discontinuity effect can be observed. In [17], Turk proposes 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 computation cost during the transition period because of the need to process two models at the same time. Secondly, because the multi-resolution models are pre-generated, the object's DVI is not con-

sidered. The result of this is a uniform increase or decrease in model resolution, and hence the generated models are not optimized for the given viewing and animation parameters of individual frames. Such optimization is important when an object covers a large depth range, for example, a piece of landscape or a large building. Even though there may only be a small region of the object lying close to the viewer or inside the viewer's line of sight, we still need to use the high resolution model for rendering so that the details in the closest part of the object will not be lost. Sometimes, a small object may also have similar problem if, for example, it is close to the viewer.

To overcome the second limitation, related methods developed for managing large terrain models can be used [2, 5, 12]. These methods basically divide a large terrain surface into square blocks. All the polygons inside a block are arranged in a quadtree structure with the root node representing the whole block and the leaf nodes representing individual polygons. Each successive lower level of the tree represents a four-time increase in resolution. With such data structure, it is possible to calculate the DVI values of individual high level nodes during run-time. A node with high DVI value can be rendered using its lower level subnodes, i.e., with more details, while a node with low DVI value can be rendered using its higher level nodes. However, the major limitation of these methods is that the object has to be divided into regular square tiles, While this may be fine for representing smooth landscape, it may not be the best way to represent objects of arbitrary shapes because those objects may contain many sharp edges, thus, causing a lot of nodes to be generated around them.

In [6], Hoppe proposes 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 an efficient way. In addition, if some ancestor information of each edge collapse is stored, limited selective refinement is possible to increase the resolution of certain region of the model.

In this paper, we describe an adaptive multi-resolution method, which calculates the DVI values of different regions of an object based on some run-time viewing and animation parameters. Hence, the resolution of the output model may be non-uniform; regions of the object with lower DVI values contain less triangles and those with higher DVI values contain more triangles. The new method is based on the real-time multi-resolution method that we have developed [8, 11]. The rest of this paper is organized as follows. Section 2 briefly describe our real-time multi-resolution method. Section 3 presents the adaptive multi-resolution method in detail. Section 4 discusses the management of the adaptive models. Section 5 shows some example outputs of the new method and compares the new method with other similar methods. Finally, section 6 presents conclusions of the paper and discusses some possible future work.

2. Real-Time Multi-Resolution Method

In our earlier paper [8], we presented a real-time multi-resolution method which simplifies a given triangle model with two efficient operators called *edge collapse* and *triangle collapse*. The edge collapse operator collapses a triangle edge into a point each time and removes two triangles. The triangle collapse operator collapses a triangle into a point and removes four triangles. The whole simplification process is divided into two stages: the *pre-processing stage* and the *run-time stage*. In the pre-processing stage, we calculate the visual importance of each triangle and sort the triangles according to their importance values. During the run-time stage, we simplify the model by removing triangles from it according to the sorted triangle list.

In our recent paper [11], we presented a refined real-time multi-resolution method. The new method improves significantly on the performance of the simplification process. The major changes are that we only provide the edge collapse operator to further simplify the algorithm, and instead of sorting all the triangles according to their importance values, we group the triangle edges into a table according to the edge importance values. During the pre-processing stage, triangle edges are assigned importance values as follows:

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

where $V_{1,imp}$ and $V_{2,imp}$ are the importance values of the two vertices V_1 and V_2 respectively of an edge E. $|E|$ is the length of E and L_{ref} is a reference length. Hence, a short edge will have a lower edge importance, i.e., higher priority for deletion. To calculate the vertex importance, we first determine 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 can be approximated as follows:

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

If V_{imp} is smaller than a given threshold, 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 assumed to be on a feature edge and is called an *edge vertex*. Any other vertices are considered as important in defining the model topology and referred to as *corner vertices*.

We divide the maximum range of edge importance values into a fixed number of groups. Each group is a linked list of triangle edges whose importance values fall inside the range covered by the group. Each list is not sorted and is maintained in a first in first out manner as in Figure 1. During run-time, we simplify the model by collapsing triangle edges one at a time taken from the list of the lowest visual importance group.

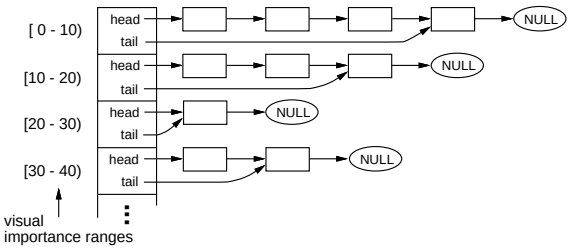

Figure 1. The visual importance table.

We implemented and tested this method on an SGI workstation with a 195MHz R10000 CPU. The method can delete 8,700 triangles per second. At first glance, this number may appear to be small. However, it represents an incremental change in the number of triangles per second. The method also represents a three-time performance improvement over the original method when tested under the same machine. Although this method is fast, it suffers from one major limitation. As an object moves away from the viewer, we may incrementally remove triangles from the model by collapsing edges. However, if the object moves toward the viewer, we need to insert edges (or triangles) into the model. Unfortunately, there is no information available as to where the triangles should be best inserted. Hence, although the difference between two consecutive frames may be only a few triangles, we need to simplify from the high resolution model until the intended resolution in every frame. If a lot of objects are moving toward the viewer simultaneously, the cost of simplification will become too high to be practical. Hence, in the paper, we also suggested to keep a data structure called the *simplification list*. The simplification list basically caches the most recent edge collapse sequence. We may perform uncollapses simply by following the reverse order of the sequence. To take this idea to the extreme, we may pre-generate a complete simplification list of the model. This idea of pre-computing the sequence of edge collapses is similar to the progressive meshes proposed by Hoppe [6], though we developed our ideas independently [8]. There are also differences in our ideas, but they are not the main concerns here. We implemented and tested the real-time multi-resolution method with the simplification list on the same workstation. The system can collapse 133,333 triangles and uncollapse 160,000 triangles per second.

3. Adaptive Multi-Resolution Method

The new method is based on our real-time multi-resolution method discussed in section 2.

Whenever the viewing and animation parameters change, the DVI of each region of the model changes too. In the extreme situation, we may re-calculate the visual importance of each triangle according to both the SVI value and the run-time DVI value of the triangle. The visual importance table is then updated to represent the current importance priorities of the edges. These operations, however, are too expensive to perform in real-time. Instead, we group triangles into patches. The SVI values of the triangles within a patch are calculated as in the original method and a simplification list is created for each patch. However, a DVI value is calculated for each patch during run-time according to the viewing and animation parameters. Given a number representing the maximum number of triangles that we want the model to have in a particular frame, this number is distributed among all the patches according to their DVI values. The number of triangles to be removed from or inserted to a patch is then determined. Starting from the patch with the largest number of triangles to be removed, the simplification list is traversed to update the resolution of the patch. If a boundary edge is to be collapsed and the same edge in the adjacent patch is not collapsed, we switch to simplify the adjacent patch until it is also collapsed.

To group the triangles in a model into patches, we have developed a simple method similar to the superface method [10]. We randomly pick a triangle that has all three vertices being flat. From there, we group neighboring triangles of similar orientations to it to form a patch. The grouping process stops in a particular direction when it reaches a feature edge. This is to maximize the flatness, and hence, the degree of simplification, of each patch. In order to limit the size of a patch, we also bound a patch with a bounding sphere. We insert the triangle into the current patch only if the distances of all three vertices of the triangle from the center of the patch are smaller than the radius of the sphere. To maintain the size of a patch to roughly the

same throughout the simplification, all patch boundaries are considered as feature edges. Hence, a boundary vertex never collapses to a non-boundary vertex, although a non-boundary vertex may be collapsed to a boundary vertex. When a patch is formed, a patch normal vector is calculated by averaging the normal vectors of all triangles within it. After dividing the model into patches, we merge those having too few triangles to nearby patches with similar patch normal vectors.

During run-time, an importance value, $P_{i,imp}$, and an importance ratio, $P_{i,irat}$, are calculated for each patch i in the model according to the number of triangles, T_i, and the DVI value, $P_{i,DVI}$, of the patch at a particular frame:

$$\begin{aligned} P_{i,imp} &= P_{i,DVI} * P_{i,SVI} \\ P_{i,irat} &= \frac{1}{T_i} * P_{i,imp} \end{aligned}$$

where $P_{i,SVI}$ represents the roughness of the patch and is calculated only once in the pre-processing stage by averaging all the vertex importance values within the patch. This value is not changed even though the resolution of a patch may change during run-time. In the second equation, if the number of triangles in the patch is high, more triangles may be deleted from it and hence, a lower importance ratio is assigned to it. $P_{i,DVI}$ may be calculated as follows:

$$P_{i,DVI} = I_{dist} * I_{sight} * I_{move} * I_{obliq} * I_{dof}$$

where I_{dist}, I_{sight}, I_{move}, I_{obliq} and I_{dof} are all between zero and one, and described in the next section.

There are two situations that we want to update the resolution of the model. In a time critical rendering environment [3, 4], the rendering manager may specify the maximum number of triangles, T_{max}, that the model may have in a particular frame in order that the rendering hardware may render the object within the allowable time. If the model has N patches, the number of triangles, $\overline{P_i}$, that patch i will have can be calculated using linear distribution:

$$\overline{T_i} = \frac{P_{i,irat}}{\sum_{k=1}^{N} P_{k,irat}} * T_{max}$$

In the second situation, we may just want to generate a model of optimized resolution given a set of viewing and animation parameters. Let T_{min} and T_{max} be the numbers of triangles of patch i when $P_{i,imp}$ is equal to zero and one respectively. We may simply use the patch importance value to determine the number of triangles that patch i should have:

$$\overline{T_i} = P_{i,imp} * (T_{max} - T_{min}) + T_{min}$$

Once we have determined the number of triangles patch i should have in the next frame, the incremental difference in the number of triangles of patch i is then:

$$T_{i,del} = T_i - \overline{T_i}$$

A positive $T_{i,del}$ indicates the number of triangles to be removed from the model and a negative $T_{i,del}$ indicates the number of triangles to be inserted into the model.

4. Management of Models

The new adaptive multi-resolution method modifies the resolution of a model according to some viewing and animation parameters. These parameters can be considered as independent of each other and which one(s) to use depends on the application. In this section, we look at some of these parameters and discuss how they can be integrated into our method.

4.1. Distance from the View Point

In our original method [8, 11], the whole object is considered to have a single depth value when calculating the multi-resolution model. With the new method, the distance of each patch from the viewer can be measured individually. Hence, even though an object may cover a large depth range such as a terrain model, our method can adaptively reduce the model resolution according to the individual patch distances.

In 2D, the projected length of a line varies in proportion to the distance of the line from the viewer. In 3D, the projected area of an object varies in quadratic to the distance of the object from the viewer. To apply this in our method, the DVI of a patch due to its distance from the viewer is:

$$I_{dist} \propto (\frac{D_{max} - D_x}{D_{max}})^2$$

where D_{max} is the distance of the patch from the viewer when the model is at its minimum resolution and D_x is the current distance of the patch from the viewer. Figure 2(a) shows the relationship of I_{dist} and D_x.

4.2. Line of Sight

Studies have shown that when an object is located outside the line of sight, the viewer is unable to perceive much detail from the object [13, 18]. Degradation of peripheral visual detail can improve rendering performance and reduce perceptual impact.

Many eye tracking systems are already available for detecting line of sight [9]. They include using skin electrodes for detecting the potential difference generated

from eye movement, illuminating the eye with infrared light and then detecting the eye position with an infrared camera, and wearing a magnetic coil in the form of a contact lens. Because most of these systems are still too expensive for the general public, some applications simply assume that the viewer's line of sight is always at the center of the screen.

As the angle between the line of sight and the line joining an object to the viewer increases, the DVI of the object decreases exponentially. To apply this in our method, the DVI of a patch due to the angle, θ_g, measured between the line of sight and the line joining the viewer and the patch is:

$$I_{sight} \propto e^{-K_s\theta_s}$$

where K_s is a constant for adjusting the decrement rate. Figure 2(b) shows the relationship of I_{sight} and θ_s.

4.3. Object Movement

Another issue is the moving speed of an object. We can easily see the details on a static object, but not a moving object. The faster the object moves, the less detail we can observe from it. Hence, we may use a simpler model for rendering if an object moves.

When an object is close to the viewer, a small movement can seriously reduce the viewer's visual perception. However, as the object gets further away from the viewer, the same movement will gradually have less effect on the viewer's visual perception. To apply this in our method, the DVI of a patch due to its movement is related to its angular velocity, $\delta\theta$, [13] as follows:

$$I_{move} \propto \text{Max}(1 - K_m|\delta\theta|,\ 0)$$

where K_m is a constant for adjusting the decrement. Figure 2(c) shows the relationship of I_{move} and $\delta\theta$.

4.4. Obliquity to the View Point

When a surface is oblique to the line of projection, the details are less visible to the viewer. This is due to the reduction in screen area covered by the surface. To apply this in our method, the obliquity of a patch can be used to determine its visual importance. If the normal vector of a patch is parallel to the line of projection, it has the maximum projected area. However, if it is perpendicular to the line of projection, it has the minimum projected area and most of the details on the patch will not be seen by the viewer. Thus, the DVI of a patch due to its obliquity is:

$$I_{obliq} \propto \text{Max}(cos(\theta_o),\ {P_{SVI}}^{K_o})$$

where θ_o is the angle between the line of projection and the normal vector of the patch. P_{SVI} represents the roughness of the patch described in Section 3. It determines the minimum value of I_{obliq}. This term is needed because the silhouette of an object is important to human visual system and hence, if a patch is rough, we should not reduce the resolution of the patch too much even if θ_o is close to 90°. K_o is a scaling factor of P_{SVI} usually between zero and one; a small K_o gives P_{SVI} a larger influence on the minimum I_{obliq} value. Figure 2(d) shows the relationship of I_{obliq} and θ_o.

4.5. Depth of Field

The human eye has a limited depth of field [14] and the absence of the depth of field effect causes the surreal appearance of the generated images. If the depth of field effect is to be simulated in the output image, an object which is outside the depth of field region can be rendered with a lower resolution model. However, it is common to have objects that lie both inside and outside the depth of field region. With our method, the depth of field effect can be taken into consideration when calculating the DVI of a patch. A patch inside the depth of field region should have a higher visual importance value than those outside.

The diameter of the circle of confusion, C_{diam}, provides an indication of the amount of blurring of a patch at distance D_x from the viewer:

$$C_{diam} \approx \frac{F^2}{n\ D_x}\left|1 - \frac{D_x}{D_f}\right| \qquad \text{for } D_f \gg F$$

where F is the focal length of the lens and n is the aperture number. D_f is the distance of an object point at which the eye focuses. To apply this in our method, the DVI of a patch due to the depth of field effect is:

$$I_{dof} \propto 1 - \left\{\frac{F^2}{n(D_x + K_d)}\left|1 - \frac{D_x}{D_f}\right|\right\}$$

where K_d is added to bound the output value when D_x approaches to zero. In our experiments, we set it to $2F^2/n$ so that the output value will be 0.5 instead of zero when D_x approaches to zero. This is because when a patch is near to the viewer, some of the details although may be blurred can still be visible in the final image and hence, we do not want to use the minimum resolution of the patch for rendering. Figure 2(e) shows the relationship of I_{dof} and D_x.

5. Results and Discussions

Figure 4 shows some output models of the original real-time multi-resolution method and the adaptive multi-resolution method. Figure 4(a) shows the

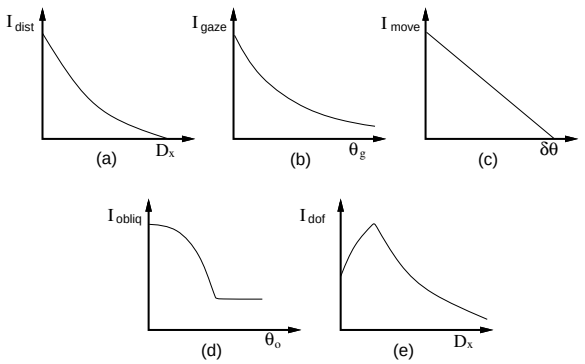

Figure 2. Parameters affecting the visual importance: (a) patch distance, (b) line of sight, (c) angular velocity of patch, (d) angle of obliquity, and (e) depth of field.

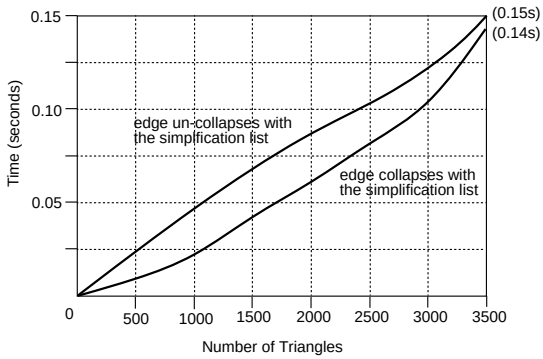

Figure 3. Performance of the new method in simplifying and refining the resolution of the face model.

original face model with surface rendering. The model contains a total of 4,356 triangles. Figure 4(b) shows the face model after patch growing with wireframe rendering. We can see that most of the patches are similar in size. Figures 4(c) and 4(d) show the simplification of the face model using the original multi-resolution method with wireframe and surface rendering respectively. Both of them have 2,000 triangles. Figures 4(e) and 4(f) are generated by applying the adaptive multi-resolution method on the model in Figure 4(c). The distance from the viewer and the viewer's line of sight are considered. The viewer is located at the left side of the diagram and the line of sight is assumed to be looking towards the model's right face. Both diagrams have 1,600 triangles. By looking at Figure 4(d) and Figure 4(f), it is difficult to tell the difference, although the latter represents a 20% reduction in the number of triangles. Figure 4(g) is similar to Figure 4(e) but has 3,580 triangles. Figure 4(h) is generated from Figure 4(g) by considering also the surface obliquity to the viewer. The model has 3,250 triangles. This represents a 10% reduction in the number of triangles. Figure 4(i) shows the surface rendering of the model in Figure 4(h). There is no appearance difference in the object profile between this and Figure 4(a). Figure 4(j) is the output of a terrain model after applying the adaptive multi-resolution method. The distance from the viewer, the viewer's line of sight and the surface obliquity to the viewer are all considered. The viewer's line of sight is assumed to be looking at top right hand corner of the diagram. Figure 3 shows the performance of the new method in simplifying and refining the resolution of the face model. It is roughly ten-time slower than our original multi-resolution method with the simplification list.

The new method has many advantages over other similar methods. When compared to the discrete multi-resolution method, our method optimizes the performance of the graphics hardware by adaptively changing the resolution of different region of the model. In addition, our method gradually removes triangles from (or inserts triangles to) the model allowing a graceful degradation (or refinement) of the model resolution. In the discrete multi-resolution method, the model selected for rendering at a particular frame must be high enough so that the details in the high DVI patches will not be lost. Even though most part of the object is visually less important, it is rendered with the same high resolution model. In addition, this method also suffers from the sudden change in resolution as the object moves across the threshold distance. When compared to the quadtree type of methods, our method works on models of arbitrary topology. The quadtree type of methods divides a surface into square regions and hence, they do not adapt to the surface topology as well without creating excessive nodes. Some of these methods even create cracks between adjacent patches. Hoppe's method [6], developed in concurrent with our work, allows models of arbitrary topology to be refined adaptively. However, we believe that his method for selective refinement does not preserve the topology of the model. This is because when refining the resolution of the model not according to the sequence specified in the progressive mesh, the triangular structure will be changed. This problem is accumulative; as we continue refining and simplifying the resolution of the model, the model will eventually become unrecognizable. In addi-

tion, the program must be able to handle the change in triangular structure of the model properly; otherwise, holes may appear in the models. To overcome these two problems, we need to refine (or to simplify) the model from the lowest (or highest) resolution every time there is a change in the model resolution. This reduces the overall performance of the method. The method that we propose here solves the problems and allows the resolution of the model to be increased and decreased continuously.

Although, our method has a number of advantages, it is not as fast as those methods using the quadtree structure. In those methods, the multi-resolution models are created in advance in the form of a quadtree. The operation involved during run-time is only to determine what nodes to use for rendering. In our method, triangles are removed on the fly. Even though the operation involved is reduced to minimal, it is still more expensive than those methods.

6. Conclusions

In this paper, we have presented an adaptive multi-resolution method, which generates multi-resolution models according to the run-time viewing and animation parameters. We have also discussed some issues in managing the adaptive multi-resolution models generated with the new method. Finally, we show some results from the new method and compare the new method with other similar methods.

When compared to our original real-time multi-resolution method, the new method is much slower, We have found that most of the computation time is spent on switching between patches when refining or simplifying the resolution of the model. We are currently looking at this to increase the overall performance of the new method.

Acknowledgements

We would like to thank the reviewers for their useful comments. We would also like to acknowledge that this work is currently supported by the University Central Research Grant, #351/084.

References

[1] M. DeHaemer and M. Zyda. Simplification of Objects Rendered by Polygonal Approximations. *Computers & Graphics*, **15**(2):175–184, 1991.

[2] J. Falby, M. Zyda, D. Pratt, and R. Mackey. NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation. *Computers & Graphics*, **17**(1):65–69, 1993.

[3] T. Funkhouser and C. Séquin. Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments. In *ACM SIGGRAPH'93*, volume **27**, pages 247–254, August 1993.

[4] M. Green. A Framework for Real-Time Rendering in Virtual Reality. In *ACM Symposium on Virtual Reality Software and Technology*, pages 3–10, July 1996.

[5] B. Herzen and A. Barr. Accurate Triangulations of Deformed, Intersecting Surfaces. In *ACM SIGGRAPH'87*, volume **21**, pages 103–110, July 1987.

[6] H. Hoppe. Progressive Meshes. In *ACM SIGGRAPH'96*, pages 99–108, August 1996.

[7] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh Optimization. In *ACM SIGGRAPH'93*, volume **27**, pages 19–26, August 1993.

[8] V. İşler, R. Lau, and M. Green. Real-Time Multi-Resolution Modeling for Complex Virtual Environments. In *ACM Symposium on Virtual Reality Software and Technology*, pages 11–20, July 1996.

[9] R. Jacob. Chapter 7: Eye tracking in advanced interface design. In *Virtual Environments and Advanced Interface Design, W. Barfield and T. Furness (Eds.)*, pages 258–288. Oxford University Press, 1995.

[10] A. Kalvin and R. Taylor. Superfaces: Polygonal Mesh Simplification with Bounded Error. *IEEE Computer Graphics and Applications*, **16**(3):64–77, May 1996.

[11] R. Lau, M. Green, D. To, and J. Wong. Real-Time Continuous Multi-Resolution Method for Models of Arbitrary Topology. *Submitted for publication*, 1996.

[12] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-Time, Continuous Level of Detail Rendering of Height Fields. In *ACM SIGGRAPH'96*, pages 109–118, August 1996.

[13] T. Ohshima, H. Yamamoto, and H. Tamura. Gaze-Directed Adaptive Rendering for Interacting with Virtual Space. In *IEEE Virtual Reality Annual International Symposium*, pages 103–110, July 1996.

[14] P. Rokita. Generating Depth-of-Field Effects in Virtual Reality Applications. *IEEE Computer Graphics and Applications*, **16**(2):18–21, March 1996.

[15] J. 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.

[16] W. Schroeder, J. Zarge, and W. Lorensen. Decimation of Triangle Meshes. In *ACM SIGGRAPH'92*, volume **26**, pages 65–70, July 1992.

[17] G. Turk. Re-tiling Polygonal Surfaces. In *ACM SIGGRAPH'92*, volume **26**, pages 55–64, July 1992.

[18] B. Watson, N. Walker, and L. Hodges. Effectiveness of Spatial Level of Detail Degradation in the Periphery of Head-Mounted Displays. In *ACM CHI'96)*, pages 227–228, April 1996.

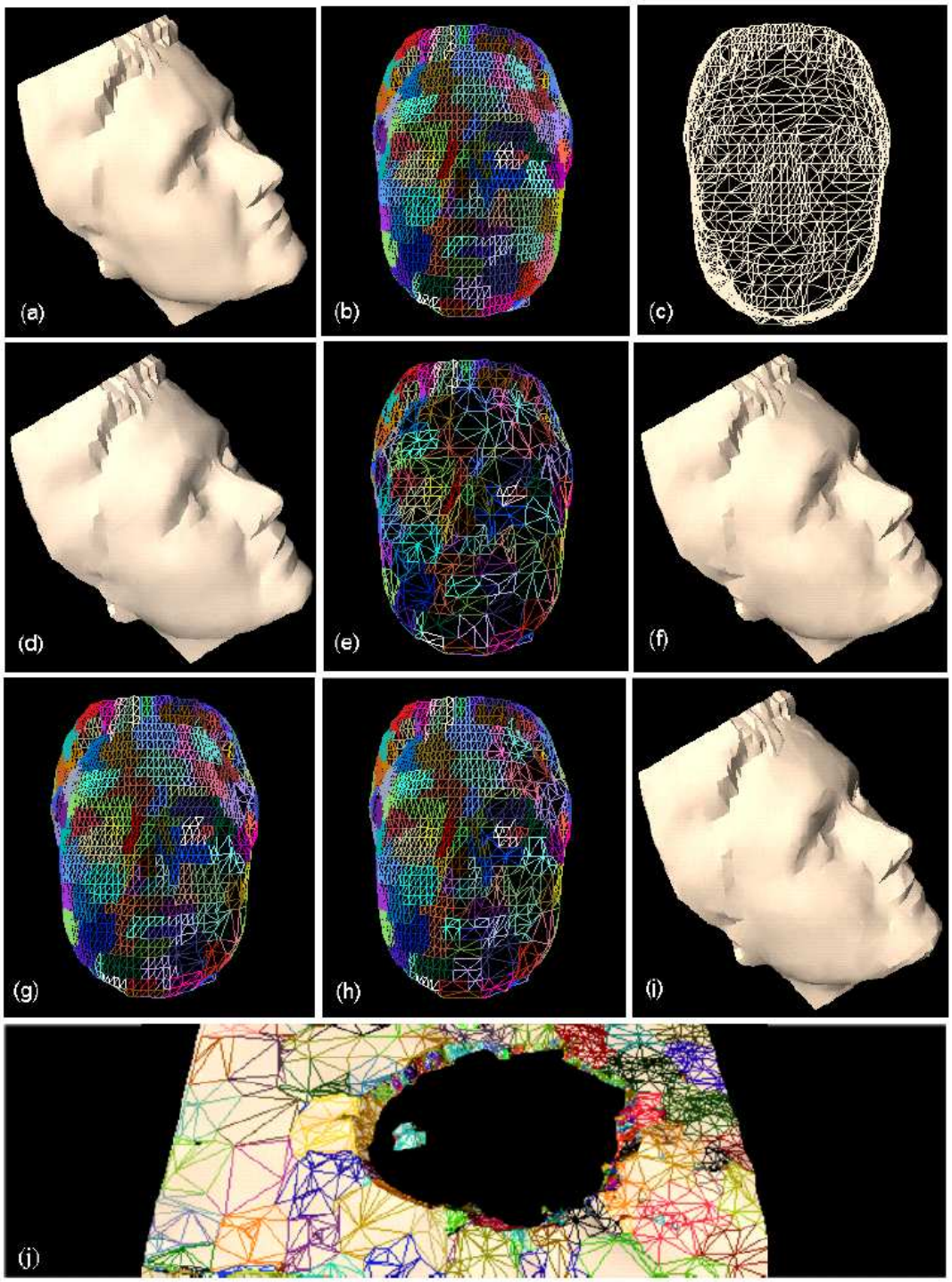

Figure 4. Simplification examples.