Visibility-Guided Simplification

Eugene Zhang and Greg Turk

GVU Center and College of Computing, Georgia Institute of Technology

Abstract

For some graphics applications, object interiors and hard-to-see regions contribute little to the final images and need not be processed. In this paper, we define a view-independent visibility measure on mesh surfaces based on the visibility function between the surfaces and a surrounding sphere of cameras. We demonstrate the usefulness of this measure with a visibility-guided simplification algorithm.

Mesh simplification reduces the polygon counts of 3D models and speeds up the rendering process. Many mesh simplification algorithms are based on sequences of edge collapses that minimize geometric and attribute errors. By combining the surface visibility measure with a geometric error measure, we obtain simplified models with improvement proportional to the amount of low visibility regions in the original models.

Keywords: Visualization, Visibility, Mesh Simplification, Rendering

1 Introduction

Visibility is important and has been well-studied in computer graphics. In general, visibility refers to determining which surfaces are unoccluded from certain camera positions in an environment.

In this paper, we are primarily interested in describing how some surface points are difficult to see due to object self-occlusions. For instance, the interiors of an object are invisible from any outside viewpoint. Some exterior regions are more difficult to see than others. To describe this view-independent property, we define a surface visibility measure which depends on the visibility function between the surface and a surrounding sphere of cameras (*camera space*). To calculate the surface-camera visibility function, we render the object from a dense set of camera poses in the camera space. For a point on the surface, the visibility measure is the percentage of the camera space from which this point is visible, and the camera space is weighted by the dot product between the point's surface normal and the viewing directions. We use this measure to help mesh simplification.

Mesh simplification algorithms reduce the polygon count of a model while maintain its overall shape and appearance. This is important for reducing the model storage cost and subsequent processing time. Many mesh simplification algorithms are based on a sequence of edge collapses. At each step, one edge is collapsed into a vertex, reducing the polygon count by two. The sequence of the edge collapse operations is designed to minimize geometric and appearance errors. In our study, we observe that many CAD models and medical imaging data sets contain large interiors and concavities, which contribute little to the final images from any outside viewpoint when being rendered as opaque objects. In these regions, our visibility-guided algorithm allows greater geometric and attributes errors.

e-mail: {zhange,turk}@cc.gatech.edu

The remainder of the paper is organized as follows. In Section 2 we review existing methods for visibility calculation and mesh simplification algorithms. We present the definition of our surface visibility measure in Section 3 and then describe how we calculate this measure in Section 4. In Section 5 we present our visibility-guided simplification algorithm, which combines the surface visibility measure with a well-know geometric measure, the quadric measure. Section 6 provides a summary and discuss some future work.

2 Previous Work

In this section, we review previous work in visibility and mesh simplification.

2.1 Visibility

Visibility issues appear in many aspects of graphics. Here, we review some areas that are related to our work.

Visible Surface Determination Problem: The *Visible Surface Determination Problem* (VSD), also called the *Hidden Surface Removal Problem*, is the task of deciding which parts of the opaque objects in a scene are visible from a given viewpoint. In their 1974 survey [22], Sutherland et al classify existing VSD algorithms into those that perform calculations in *object-space*, those that perform calculations in *image-space*, and those that work partly in both, *list-priority*. They further point out these algorithms differ in how they perform sorting and what local coherence information is used to reduce the recalculation cost. The local coherence information used may include: face coherence [20, 24], scan line coherence and edge coherence [2, 25], depth coherence [24], etc. Catmull develops the depth-buffer or z-buffer image-precision algorithm which uses depth coherence [3]. Myers later incorporates the depth-buffer algorithm with the scan-line algorithm [16]. Fuchs et al use BSP tree to establish scene visibility [7]. Appel [1], Weiler and Atherton [27], and Whitted [28] develop ray tracing algorithm which transforms the VSD into ray-object intersection tests.

Aspect Graph: The visibility of a static scene often remains constant if viewpoints are restricted to be inside a limited region. This has led Koenderink and Van Doorn to propose the *aspect graph* to record where visibility changes occur [13]. In this graph, each node represents a general view as seen from a region of viewpoint space. Two neighboring nodes are linked by an edge to represent a *visual event* (visibility change) when the viewpoint moves from one region to another. Algorithms have been developed for computing the aspect graph for 3D convex objects using orthographic views [18] and perspective views [21, 26]. Gigus et al propose algorithms for computing the aspect graph for 3D polyhedra under orthographic views [9]. Unfortunately, computing aspect graphes is expensive. For general polyhedra with n supporting planes, the complexity of computing the aspect graph using orthographic views is $O(n^6)$. One often uses sampling techniques to generate approximations [6, 10]. However, the sampling rate is difficult to set.

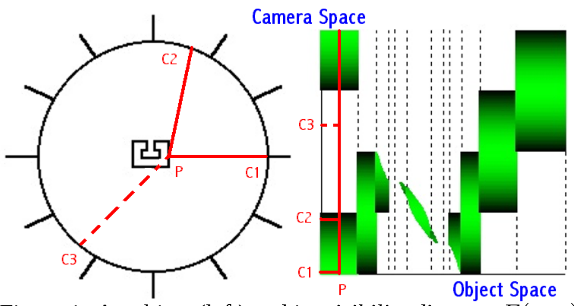

Figure 1: An object (left) and its *visibility diagram* $F(\mathbf{p}, \mathbf{c})$ (right). In the *visibility diagram*, white indicates surface point $\mathbf{p}$ and camera pose $\mathbf{c}$ are mutually occluded. Green indicates they are mutually visible. The intensity of greenness is related to the dot product between the surface normal at $\mathbf{p}$ and the viewing direction of $\mathbf{c}$. Head-on views are in lighter greens and side views are in darker greens. For instance, point $\mathbf{p}$ is visible from both $\mathbf{c1}$ and $\mathbf{c2}$, but occluded from $\mathbf{c3}$. Furthermore, $\mathbf{c1}$ has a better viewing angle for the surface near $\mathbf{p}$ than that $\mathbf{c2}$ does.

2.2 Mesh Simplification

Mesh simplification, too, is a well-studied problem in computer graphics. Since the literature in this area is extensive, we review only a few of the most relevant methods. Hoppe proposes the framework of the progressive mesh [11] representation to address the issues of progressive transmission, selective refinement and geomorphing. Under this scheme, the polygon count is reduced by a sequence of edge collapses. All edges are put on a priority queue, which is sorted by some error measure. At each step, the edge with the least error is collapsed into a single vertex, therefore removing two polygons. The location of the new vertex and the choice of the error measure are the keys to determining the quality of the simplified models.

Ronfard and Rossignac [19] measure the geometric errors by using the maximum distance of the new vertex location to the supporting planes of the original edge's 1-neighborhood. Garland and Heckbert use similar geometry information, namely, the quadric measure [8] as their error measure. In this measure, determining the location of the new vertex is internally linked to the error measure, defined as the squared sum of the distances of the new vertex location to the supporting planes that contain at least one triangle incident to the edge. The quadric measure is a geometry-based error. Hoppe later extends this to handle attributes such as colors and texture coordinates [12]. The original quadric measure does not use visibility information.

Lindstrom and Turk define a different type of error measure, namely, the image-driven measure [15]. Instead of measuring the geometric deviation caused by the edge collapse operations, they measure the image deviation, that is, the visual differences between the model before and after a certain edge collapse. By creating images of both the original and partially simplified models from a number of different camera poses (such as the center of the faces of an icosahedron) the method determines the order of the edges based on the visual difference that these edges contribute. This measure indirectly takes into account which portions of an object are visible, and it greatly reduces the number of polygons used to represent interior details. However, the processing time required for calculating the image deviation is substantially more than that for the geometric deviation.

3 Visibility Measure Definition

Due to scene occlusions, a point $\mathbf{p}$ is not always visible from a camera pose $\mathbf{c}$. Figure 1 illustrates this object-camera visibility function. In the left image, an object M consisting of line segments is observed from inward-looking orthographic cameras on a surrounding circle S with infinite radius. The center of S coincides with the center of the bounding box of M. Note, to draw both M and S in the same image, their relative size are distorted. The cameras are drawn as small line segments pointing toward the center of the circle. $\mathbf{p}$ is a point on M. $\mathbf{c1}$, $\mathbf{c2}$ and $\mathbf{c3}$ are camera poses on the circle. $\mathbf{p}$ is visible from both $\mathbf{c1}$ and $\mathbf{c2}$, and invisible from $\mathbf{c3}$ due to self-occlusion. $\mathbf{c1}$ has a head-on view of the region near $\mathbf{p}$ while $\mathbf{c2}$ views the same region at a poor angle.

The right image in Figure 1 is a visualization of $F(\mathbf{p}, \mathbf{c})$, which we call the *visibility diagram* of M. The *x*-axis represents points on the perimeter of the shape, as traversed counter-clockwise. The *y*-axis represents camera poses on the surrounding circle, also traversed counter-clockwise. In the *visibility diagram*, the color at point $(\mathbf{p}, \mathbf{c})$ encodes the visibility between point $\mathbf{p}$ on the object M and camera pose $\mathbf{c}$. Green means they are mutually visible, and white means they are mutually occluded. The intensity of greenness is proportional to the dot product between $N(\mathbf{p})$ and $R(\mathbf{c})$, the surface normal at $\mathbf{p}$ and the viewing direction of $\mathbf{c}$, respectively. Lighter green indicates better views. The overall visibility of p from outside views is defined as:

$$V(\mathbf{p}) = \frac{\int_S F(\mathbf{p}, \mathbf{c}) \left(R(\mathbf{c}) \cdot N(\mathbf{p})\right) d\mathbf{c}}{\int_S \left(R(\mathbf{c}) \cdot N(\mathbf{p})\right) d\mathbf{c}} \qquad (3.1)$$

$V(\mathbf{p})$ measures the percentage of camera space that can "see" $\mathbf{p}$, giving more weight to views at better angles. The portion of S over which we integrate is actually a half-sphere, based on the surface normal at $\mathbf{p}$. $V(\mathbf{p})$ is between 0 and 1. For example, any point on a convex object achieves the maximum value. Using the terms from radiosity [5] and under the assumption that there is no scattering or energy loss during light transport, $F(\mathbf{p}, \mathbf{c}) \left(R(\mathbf{c}) \cdot N(\mathbf{p})\right)$ is the form factor between an infinitesimal surface around $\mathbf{p}$ and an infinitesimal surface around $\mathbf{c}$, i.e., the fraction of light which leaves $\mathbf{c}$ that reaches $\mathbf{p}$. Therefore, $V(\mathbf{p})$ measures the fraction of light which leave a sphere infinitely away from the object that can directly reach $\mathbf{p}$. Furthermore, $V(\mathbf{p})$ is related to the measure used by Nooruddin and Turk[17] for surface interior/exterior classification and visualization. For their applications, their measure is a binary measure and all camera views are weighted equally.

Figure 2 shows the measure $V(\mathbf{p})$ for some of our test models. The color coding is as follows: 0-1/3 (interpolating between white and red), 1/3-2/3 (interpolating between red and yellow), 2/3-1 (interpolating between yellow and green). The overall visibility of mesh M is defined as:

$$V(M) = \frac{\int_M V(\mathbf{p})\, d\mathbf{p}}{\int_M d\mathbf{p}} \qquad (3.2)$$

This measure is 1 for convex objects. Table 1 shows the overall surface visibility of some test models. The Stanford Bunny model has a large convex body with the ears and other parts that are attached. This model has a high overall visibility. The Motor and Blade models contain large numbers of interior polygons, resulting in a low overall visibility.

4 Visibility Measure Calculation

Calculating the exact mesh visibility function for large models is computationally prohibitive. Nooruddin and Turk [17]

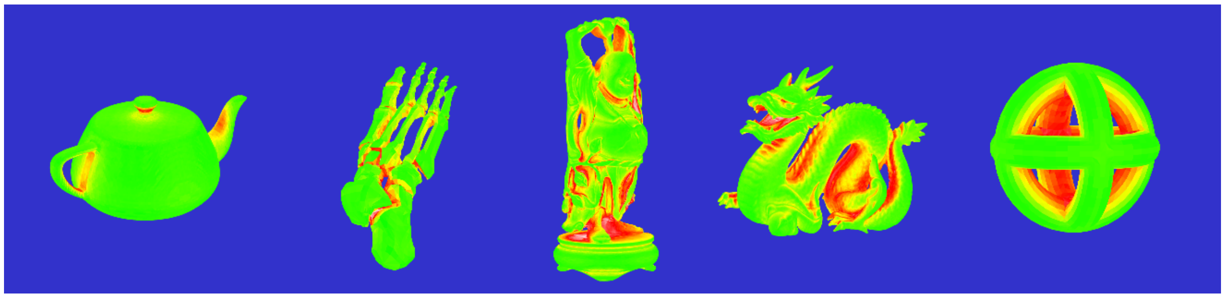

Figure 2: This image illustrates the visibility measures for some test models: the Utah Teapot, a foot bone model, Happy Buddha, Dragon, and a CAD model of three interlocking tori.

have used a sampling approach in both the object space M and the camera space S for interior/exterior classification. Here, we use a similar approach. First, we subdivide the mesh surface until all triangles are small. Next, we choose a finite number of camera positions that are evenly distributed in the camera space. Finally, we render M from each of these camera positions with the help of graphics hardware to quickly compute a table of visibility between the camera positions and the surface triangles. This table is a discrete version of the *visibility diagram* (Figure 1).

To obtain uniformly spaced camera poses, we construct a tessellation of the camera space S by subdividing the faces of an octahedron three times and placing sample cameras at every vertex of the resulting mesh. We assume a camera pose $\mathbf{v}$ sees a triangle $\mathbf{t}$ if and only if at least part of $\mathbf{t}$ is visible from $\mathbf{v}$. We now adapt all our definitions in Section 3 as follows. $F(\mathbf{t},\mathbf{v})$ is defined as 0 if $\mathbf{t}$ is entirely invisible from $\mathbf{v}$, and 1 otherwise. $N(\mathbf{t})$ is the normal of $\mathbf{t}$, and $R(\mathbf{v})$ is the viewing direction of $\mathbf{v}$. We assume the tessellation of the camera space is even. Thus, $area(\mathbf{v})$ is the same for all $\mathbf{v}$.

$$V(\mathbf{t}) = \frac{\sum_{\mathbf{v}\in S} F(\mathbf{t},\mathbf{v}) * (R(\mathbf{v}) \cdot N(\mathbf{t})) * area(\mathbf{v})}{\sum_{\mathbf{v}\in S} (R(\mathbf{v}) \cdot N(\mathbf{t})) * area(\mathbf{v})} \tag{4.3}$$

Here, we make the assumption that the visibility between a triangle $\mathbf{t}$ and a view triangle $\mathbf{v}$ is constant across both $\mathbf{t}$ and $\mathbf{v}$. In general this is not true. However, when triangles in both M and S are small enough, the error introduced in the above formula becomes negligible. From each viewpoint $\mathbf{v}$, we render the mesh object with a color-encoding of the polygon ID using graphics hardware. Then we record $F(\mathbf{t},\mathbf{v}) = 1$ if at least one pixel has color $\mathbf{t}$ in the color buffer.

This approach has two problems that need attention: triangles that are too large, and triangles that are too small or sliver-shaped. Large triangles increase the error since $F(\mathbf{t},\mathbf{v})$ is far away from being constant. Small triangles result in aliasing, or so-called popping effect. When being viewed from rather poor angles, depending on how the scan-conversion is done in the rendering system, the presence of pixels for a particular triangle in the image is not consistent.

To handle triangles that are too large, we subdivide them such that the length of the longest edge of each triangle after subdivision is limited by a threshold l. l is calculated based on the aspect ratio and worst viewing angle (we use 75 degrees, where the dot product between the surface normal and light rays is 0.25). To perform the subdivision, we add vertices to the middle of any edge that is longer than l. For each triangle, based on the number of the new vertices added to its edges, we divide it into sub-triangles. This process is repeated until all mesh edges are shorter than l.

While mesh subdivision removes large triangles, it maintains or even creates small and sliver triangles, which are subject to sampling problems. This affects the accuracy of $F(\mathbf{t},\mathbf{v})$ more for the side views than then the head-on views. Since $V(\mathbf{t})$ is defined to favor the head-on views, it is less sensitive to the sampling problems. Nonetheless, we alleviate t he situation by storing a depth buffer along with the color buffer for each camera pose. To determine $F(\mathbf{t},\mathbf{v})$ for a small triangle $\mathbf{t}$, we compare the depths of its vertices to

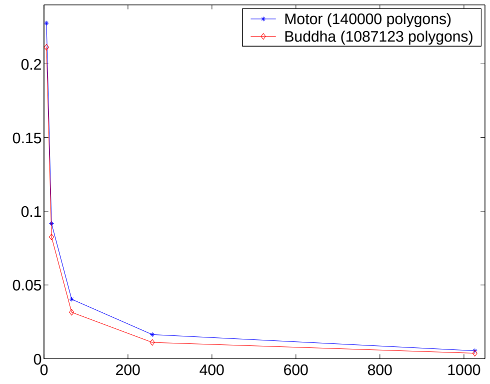

Figure 3: This diagram shows the tradeoff between the visibility errors and the number of cameras (calculation time) for the Motor and the Buddha models. The visibility is calculated with 6, 18, 66, 258, 1026 cameras, and is compared to the visibility calculated with 4096 cameras. The *X*-axis represents the number of cameras used for visibility calculation, and the *Y*-axis represents the visibility error.

Surface Visibility Measure			
	Model Size (# polygons)	Visibility	Processing Time (mm:ss)
Bunny	69,451	0.958	6:07
Teapot	28,744	0.890	4:58
Dragon	871,414	0.804	24:38
Buddha	1,087,416	0.764	30:14
Locking Tori	9,708	0.728	4:35
Foot Bone	8,790	0.724	4:33
Blade	1,688,933	0.466	58:19
Skull	1,169,608	0.444	37:44
Motor	140,113	0.264	10:56

Table 1: This table shows the surface visibility measure for several models along with the processing time. The timing measurements were taken on a SGI Octane of 195 MHz CPU.

the depths of their respective neighbor pixels. Even without pixels in the color buffer indicating $\mathbf{t}$ is visible, our algorithm considers it visible if the depth at any of its vertices is within a tolerance to the depth of a neighbor pixel. With this method, we are able to use a relatively low resolution (480×480) during the rendering process.

The accuracy of our algorithm depends on the sampling pattern in the camera space. In general, more cameras means more accurate results. On the other hand, more cameras means longer calculation time. Figure 3 shows the relation between the visibility errors with respect to the number of cameras used for the Motor and Happy Buddha models. Here, we subdivide an octahedron up to 5 times to generate 6 camera sampling patterns, namely, 6, 18, 66, 258, 1026 and 4098 cameras evenly spaced on a sphere. Assuming the visibility is accurate using 4098 cameras, we obtain the visibility errors for the other sampling patterns by calculating the area-weighted average of the visibility difference. As one can see, the visibility errors quickly converge, and we find that 258 cameras seem to be a good comprise between time and accuracy for all test models.

5 Visibility-Guided Simplification

5.1 Algorithm

We observe that many applications do not require processing invisible and low visibility concavity regions. We can be less concerned with the geometry errors at those parts of the surface. To put this into practice, we combine our surface visibility measure with a well-known geometric error measure called the quadric measure [8], which is defined for each edge in the mesh object. Let $\mathbf{e}$ be the next edge to collapse into a point $\mathbf{v}$, represented in homogeneous coordinates as $(x_0, y_0, z_0, 1)^T$. Let T be all the triangles in M that are adjacent to at least one vertex of $\mathbf{e}$, i.e., T is the union of the 1-ring neighborhoods of both vertices of edge $\mathbf{e}$, allowing the triangles in both neighborhoods to be counted twice. Each triangle $\mathbf{t}$ has a plane equation

$$A_t x + B_t y + C_t z + D_t = 0 \tag{5.4}$$

The quadric measure is then defined as

$$E_{\mathbf{q}}(\mathbf{e}) = \sum_{\mathbf{t} \in T} (distance(\mathbf{v}, \mathbf{t}))^2 \tag{5.5}$$

i.e.,

$$E_{\mathbf{q}}(\mathbf{e}) = \sum_{\mathbf{t} \in T} (A_t x_0 + B_t y_0 + C_t z_0 + D_t)^2 = \mathbf{v}^T \mathbf{Q} \mathbf{v} \tag{5.6}$$

where

$$\mathbf{Q} = \sum_{\mathbf{t} \in T} \begin{pmatrix} A_t^2 & A_t B_t & A_t C_t & A_t D_t \\ A_t B_t & B_t^2 & B_t C_t & B_t D_t \\ A_t C_t & B_t C_t & C_t^2 & C_t D_t \\ A_t D_t & B_t D_t & C_t D_t & D_t^2 \end{pmatrix} \tag{5.7}$$

To combine our visibility measure with the quadric measure we note that the quadric measure is the sum of the squared distance from a point to many planes. If edge $\mathbf{e}$ is adjacent to some triangles with low visibility, then the distance from $\mathbf{v}$ to this plane makes less visual impact than the distances from $\mathbf{v}$ to high visibility triangles if the geometric errors are the same. Our visibility-guided error measure is defined as

$$E_{\mathbf{v}}(\mathbf{e}) = \sum_{\mathbf{t} \in T} (distance(\mathbf{v}, \mathbf{t})\, V(\mathbf{t}))^2 \tag{5.8}$$

$E_{\mathbf{v}}(\mathbf{e})$ guides which edges are collapsed, that is, this measure is used to order the priority queue.

Recall the meaning of $V(\mathbf{t})$ as the weighted sum of dot products between a triangle's normal with incoming ray directions, our visibility-guided error measure for one triangle is the weighted average projected distance from all viewing directions. This means edges with higher geometric errors can be chosen for removal if they situate in extremely low visibility regions, such as interiors and creases. We use the original quadric matrix to select the best new vertex location for the collapsed edge as described in [8]. For computational purpose, our measure is written as

$$E_{\mathbf{v}}(\mathbf{e}) = \mathbf{v}^T \mathbf{Q}_{\mathbf{v}} \mathbf{v} \tag{5.9}$$

where

$$\mathbf{Q}_{\mathbf{v}} = \sum_{\mathbf{t} \in T} \left[\begin{pmatrix} A_t^2 & A_t B_t & A_t C_t & A_t D_t \\ A_t B_t & B_t^2 & B_t C_t & B_t D_t \\ A_t C_t & B_t C_t & C_t^2 & C_t D_t \\ A_t D_t & B_t D_t & C_t D_t & D_t^2 \end{pmatrix} V^2(\mathbf{t}) \right] \tag{5.10}$$

5.2 Results

To compare the quality of the two simplification methods, we select the following image-based root-mean-squared (RMS) error, based on the method of Lindstrom and Turk [15]. For the original model M_0 and the simplified model M_i, we render both models from the twenty vertices of a surrounding dodecahedron using flat shading. The RMS "image" error between the images is calculated as:

$$RMS(M_i, M_0) = \sqrt{\sum_{n=1}^{20} D_i^n} \tag{5.11}$$

Here, D_i^n is the squared sum of pixel-wise intensity difference between the n-th image of M_i and M_0. Essentially we are evaluating how similar the original and simplified models appear when rendered.

For each of the six test model, we select seven target polygon counts, and apply both the quadric-based (QB) method [8] and our visibility-guided (VG) method. Figure 4 shows the comparisons between the image errors and the geometric errors obtained using the Metro program [4] for the simplified models of the same polygon counts. Results obtained for the QB method are painted using blue lines, and those for the VG method are painted using red lines. The image errors are painted using regular lines with diamonds, and the geometric errors are painted using wider lines. This figure shows that the VG method in general generates smaller image errors but incurs greater geometric errors than the QB method. The greater geometric errors occur in low visibility regions.

Figure 5 shows the visual comparisons for the Motor model, a car engine model with 140,113 polygons (middle). This model has a large number of invisible polygons with high curvatures occluded by its main exterior feature, a box. The exterior also contains several mechanical parts. For the simplified models (QB on the left, and VG on the right) with the same polygon count of 15,000, the VG method has 51.77% less image error as the QB method. In fact, the image error for the VG method of polygon count 12,000 is about equal to that for the QB method of 27,000 polygons (Figure 4). This is not surprising since the quadric errors for the original Motor model's interior edges are higher than that of most exterior features. When the regions with low

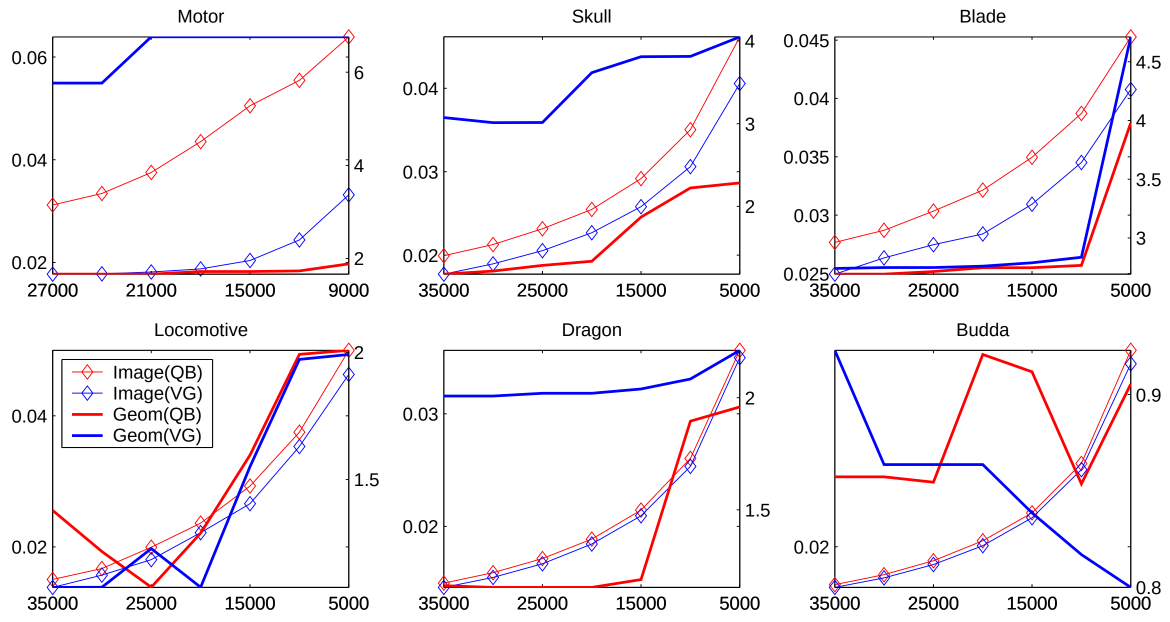

Figure 4: Image error (diamond) and geometric error (wider line) comparisons between the quadric simplification method (blue) and our visibility-guided method (red) at 7 levels in the progressive meshes. The *X*-axis represents the numbers of polygons for each level. The left *Y*-axis represents represents the image error and the right *Y*-axis represents the geometric error. Motor, Skull, Blade and Locomotive models show significant improvement due to the large percentage of interiors. Buddha and Dragon models show small but noticeable improvement due to the large percentage of low visibility regions.

Simplification comparisons (running time in minutes:seconds)					
	Model Size (# polygons)	Visibility	Time Calculating Visibility	Time Simplification (quadric and visibility)	Less Image Error Incurred with VG than with QB under Flat Shading
Motor	140,113	0.264	10:56	0:24	51.77%
Skull	1,169,608	0.444	37:44	4:18	11.51%
Blade	1,688,933	0.446	58:19	6:36	10.22%
Locomotive	183,450	0.538	11:14	0:32	7.41%
Dragon	871,414	0.804	24:38	2:31	2.44%
Buddha	1,087,416	0.764	30:14	3:06	2.12%

Table 2: This table shows the comparison between six test models, with their polygon counts, visibility measure, image error, and the calculation time the visibility measure as well as processing time for simplification. The timing measurements were taken on a SGI Octane with a 195 MHz CPU.

quadric errors have been simplified, the QB method starts simplifying the exterior features. The VG method simplifies the interior regions despite their relatively high quadric errors.

Also shown in Figure 6 is the Blade model created from CT data of an actual turbine blade (middle: original model with 1,688,933 polygons, left: QB method, and right: VG method, both with 15,000 polygons). It also contains a large number of interior polygons. Again, the VG method performs better than the QB method. Note the difference at the letters on the base (bottom row) and the features along both sides of the blade (both rows).

Medical imaging data sets often contain large interiors and concavities. The Skull model, created from 124 CT scan slices, is shown in Figure 7 (middle: the original model with 1,169,608 polygons, left: QB method, and right: VG method, both with 10,000 polygons). To remove the contour lines that are artifacts of the reconstruction algorithm, we performed Taubin's smoothing method [23] before simplification. This model does not have many invisible polygons, but it has a large number of polygons with low visibility. The VG method maintains better triangulations than the QB method around the regions of the teeth and their roots, as well as the forehead.

To understand the range of applicability of our method, we tested our method against models that has a relatively small amount of low visibility regions. The Buddha model and the Dragon model (not shown), created using range scan and surface reconstruction, belong to this category. As shown in Figure 4, the visibility-guided method consistently perform better although the improvement is less than that of other models. Figure 8 shows the visual comparisons for the Buddha model (bottom middle: original model with 1,087,416 polygons, bottom left: QB method, and bottom right: VG method, both with 20,000 polygons). The main difference is mainly around the face. Note the features in this regions are better maintained using our VG method than using the QB method.

From the analysis above, it appears that the amount of improvement is related to the overall visibility measure of the surface, see Table 2. The last column lists the average of the percentage differences in the image errors incurred using

VG method than using the QB method for the seven levels. The table also lists the processing time for each test model, including the time to calculate the visibility measure, and the time to perform visibility-guided simplification. Note that the time required for the QB method and the VG method differ very little ($< 1\%$). Therefore, if a model's visibility measure has been pre-computed, the visibility-guided simplification does not require more time than that is needed by other edge-collapse mesh simplification algorithms.

6 Conclusion and Future Work

In this paper, we defined a view-independent visibility measure to classify mesh surface regions based on how easy they are to see from the outside. This visibility measure depends on the visibility function between the mesh surface and a surrounding sphere of cameras. We combined our visibility measure with the quadric measure to perform mesh simplification. The visibility-guided method produces improvements (measured according to image differences) that are related to the amount of low-visibility regions in the mesh.

As a possible next step, we would like to find algorithms to calculate the visibility function more accurately. One possibility is to allow the visibility function to have values between 0 and 1 as a probability for views at poor angles or insufficient resolutions. Also, we would like to perform out-of-core visibility calculations for large models such as those obtained through the digital Michelangelo project [14].

We are also exploring other applications for our visibility measure, including shape matching and texture mapping.

The visibility function and the visibility measure describe the self-occlusion properties of mesh objects. Therefore, it is possible that the distribution of the visibility measure can be used in shape matching and feature recognition.

In texture mapping, the surface of an object is often divided into patches. Every patch is independently unfolded onto a plane before all the patches are packed into the texture map. Since mesh interiors do not contribute the final images for opaque objects, we do not need to assign space for them. Also, regions with low visibility measure will be viewed from poor angles, allowing us to be less concerned about their stretch during the texture unfolding process.

7 Acknowledgement

We would like to thank the following people and groups for the 3D models they provided: Will Schroeder, Ken Martin, Bill Lorensen, Bruce Teeter, Terry Yoo, Mark Levoy and the Stanford Graphics Group. Also, we would like to thank Michael Garland for the QSlim code. Finally, we thank the anonymous reviewers for their excellent suggestions.

This work is funded by NSF grant ACI 0083836.

References

[1] Appel, A., Some Techniques for Shading Machine Renderings of Solids , *SJCC*, 1968, pp. 37-45.

[2] Bouknight, W.J., A procedure for Generation of Three-Dimensional Half-toned Computer Graphics Representations, *Comm, ACM 13, 9*, September 1969.

[3] Catmull, E., A Subdivision Algorithm for Computer Display of Curved Surfaces, *Ph.D Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah*, Salt Lake City, UT, December 1974.

[4] Cignoni, P., Rocchini, C., and Scopigno, R., Metro: measuring error on simplified surfaces, *Computer Graphics Forum*, vol. 17, no. 2, pp. 167-174, June 1998.

[5] Cohen, M.F., and Greenberg, D.P., The Hemi-Cube: A Radiosity Solution for Complex Environments, *Computer Graphics Proceedings*, vol. 19, no. 3, pp. 31-40, 1985.

[6] Fekete, G., and Davis, L.S., Property spheres: a new representation for 3-D object recognition, *Proceedings of the Workshop on Computer Vision: Representation and Control* , Annapolis, MD, April 30-May 2, 1984, pp. 192-201.

[7] Fuchs, H., Kedem, Z.M., and Naylor, B.F., On Visible Surface Generation by A Priori Tree Structures, *SIGGRAPH 80*, 1980, pp. 124-133.

[8] Garland, M. and Heckbert, P.S., Surface Simplification using Quadric Error Metrics, *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 97), pp. 209-216.

[9] Gigus, Z., and Malik, J., Computing the aspect graph for the line drawings of polyhedral objects, *IEEE Trans. on Pat. Matching & Mach. Intelligence*, February 1990, 12(2).

[10] Hebert, M., and Kanade, T., The 3-D profile method for object recognition, *Proceedings, CVPR '85 (IEEE Computer Society Conference on Computer Vision and Pattern Recognition)*, San Francisco, CA, June 1985, pp. 458-463.

[11] Hoppe, H., Progressive Meshes, *Computer Graphics*, (SIGGRAPH 1996 Proceedings), pages 99-108.

[12] Hoppe, H., New quadric metric for simplifying meshes with appearance attributes, *IEEE Visualization 1999*, October 1999, pp. 59-66.

[13] Koenderink, J.J., and vanDoorn, A.J., The singularities of the visual mapping, *BioCyber*, 1976, 24(1), pp. 51-59.

[14] Levoy, M., Pulli, K., Curless, B., Ruskinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J., Shade, J., and Fulk, D., The Digital Michelangelo Project: 3D Scanning of Large Statues , *SIGGRAPH Proceedings 2000*, 2000, pp. 131-144.

[15] Lindstrom, P. and Turk, G., Image-Driven Simplification, *ACM Transactions on Graphics*, 19(3), July 2000, pp. 204-241.

[16] Myers, A.J., An Efficient Visible Surface Program, *Report to National Science Foundation, Computer Graphics Research Group, Ohio State University, Columbus, OH*, July 1975.

[17] Nooruddin, F.S, and Turk, G., Interior/Exterior Classification of Polygonal Models, *Visualization 2000 Conference*, Salt Lake City, Utah, Oct. 8-13, 2000, pp. 415-422.

[18] Plantinga, W.H., and Dyer, C.R., An algorithm for constructing the aspect graph, *27th Annual Symposium on Foundations of Computer Science*, Los Angeles, Ca., USA, October 1986, pp. 123-131.

[19] Ronfard, R., and Rossignac, J., Full-range approximations of triangulated polyhedra, *Proceedings of Eurographics96, Computer Graphics Forum*, pp. C-67, Vol. 15, No. 3, August 1996.

[20] Schumacker, R.A., Brand, B., Gilliland, M., and Sharp, W., Study for Applying Computer Generated Images to Visual Simulation, *AFHRL-TR-69-14, US Air Force Human Resources Laboratory*, September 1969.

[21] Stewman, J., and Bowyer, K., Direct construction of the perspective projection aspect graph of convex polyhedra, *Computer Vision, Graphics and Image Processing*, July 1990, 51(1), pp. 20-37.

[22] Sutherland, I.E., Sproull, R.F. and Schumacker, R.A., A characterization of ten hidden-surface algorithms, *ACM Computing Survey*, 6(1), March 1974, pp. 1-55.

[23] Taubin, G., A signal Processing Approach to Fail Surface Design, *Computer Graphics Proceedings*, (SIGGRAPH 1995 Proceedings) August 1995, pp. 351-358.

[24] Warnock, J.E., A Hidden-Surface Algorithm for Computer-Generated Halftone Pictures, *Computer Science Department, University of Utah, TR 4-15*, June 1969.

[25] Watkins, G.S., A Real-Time Visible Surface Algorithm, *Computer Science Department, University of Utah, UTECH-CSC-70-101*, June 1970.

[26] Watts, N.A., Calculating the principle views of polyhedron, *Ninth International Conference on Pattern Recognition*, Rome, Italy, November 1988, pp. 316-322.

[27] Weiler, K., and Atherton, P., Hidden Surface Removal Using Polygon Area Sorting, *SIGGRAPH 77*, pp. 214-222.

[28] Whitted, T., An Improved Illumination Model for Shaded Display, *ACAM, 23(6)*, June 1980, pp. 343-349.

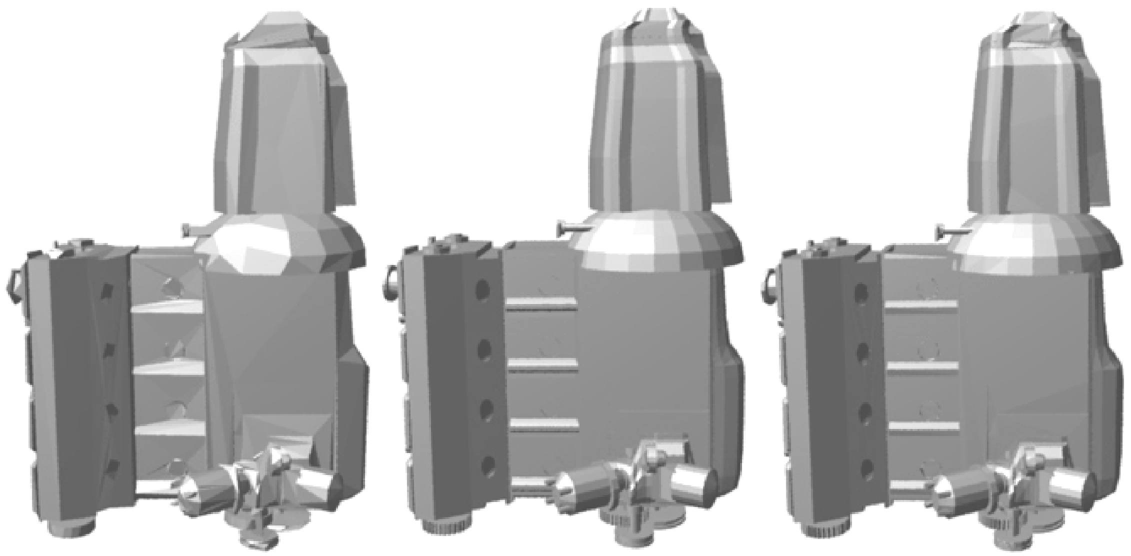

Figure 5: Visual comparisons between the original Motor model (Middle, 140,113 polygons), and the simplified versions using the quadric-only method (Left, 15,000 polygons) and the visibility-guided method (Right, 15,000 polygons). All images are rendered using flat shading. Compare the overall shape of the trunk and mechanical parts.

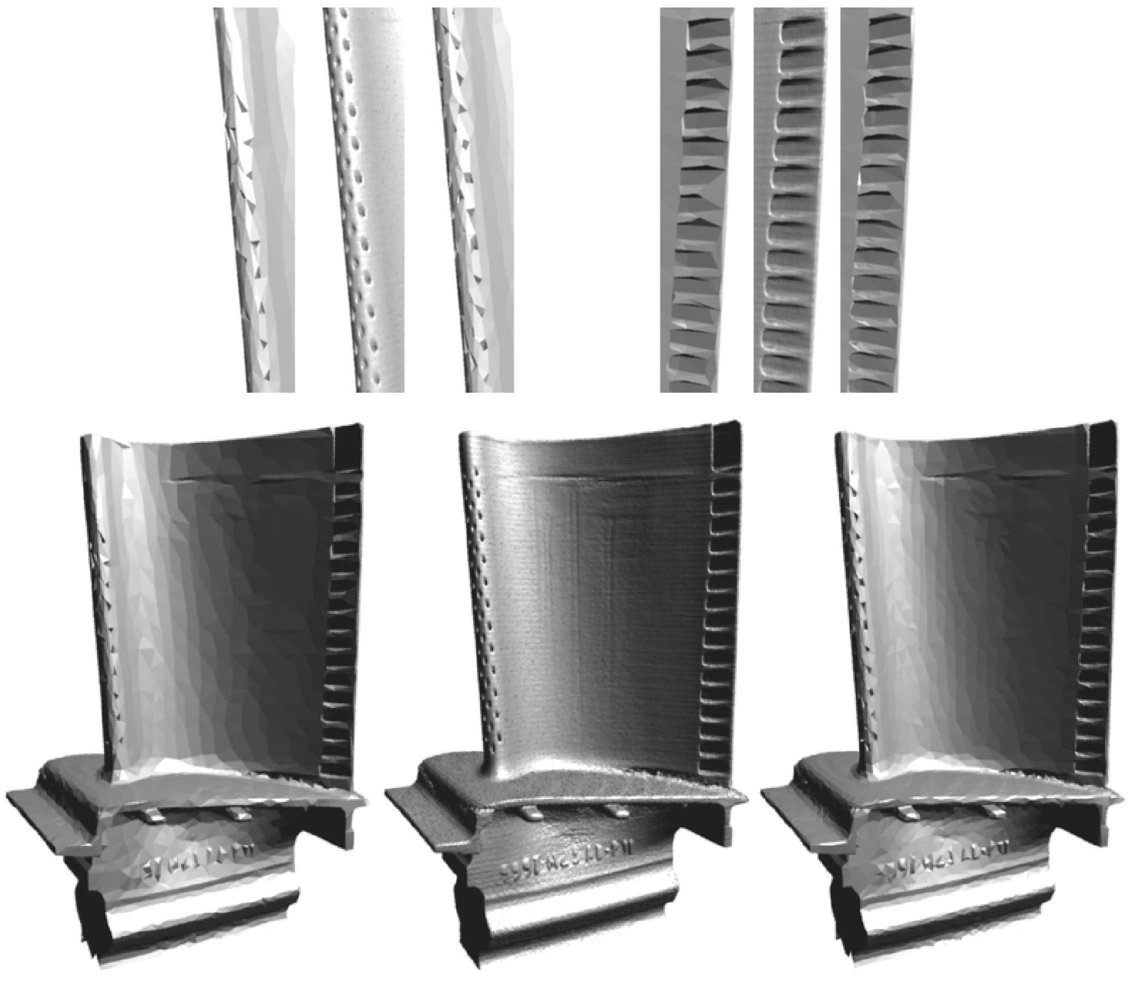

Figure 6: Visual comparisons between the original Blade model (Middle, 1,688,933 polygons), and the simplified versions using the quadric-only method (Left, 15,000 polygons) and the visibility-guided method (Right, 15,000 polygons). All images are rendered using flat shading. Compare features such as the letters on the base (bottom row), the column of rectangular vents along the right edge, and the small holes along the left edge (both rows).

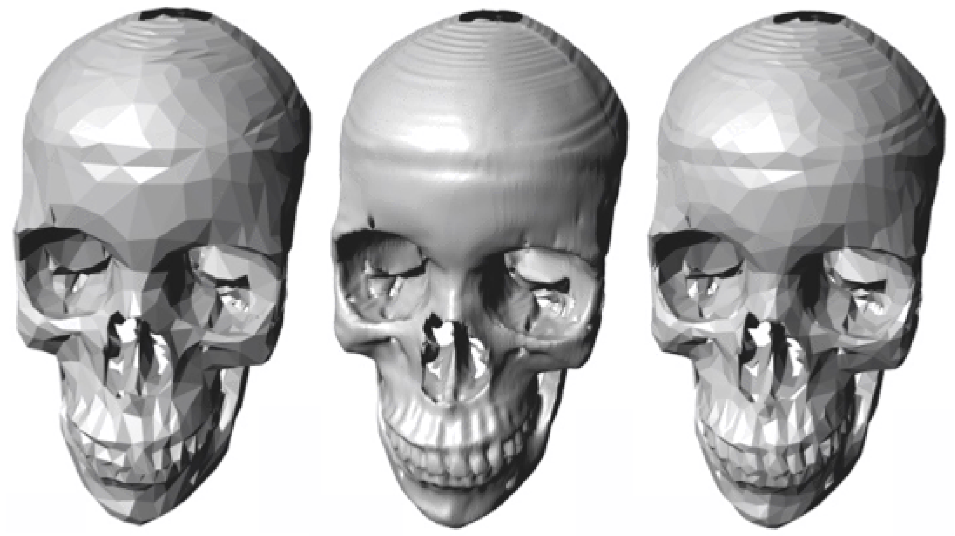

Figure 7: Visual comparisons between the original Skull model (Middle, 1,169,608 polygons), and the simplified versions using both the quadric-only method (Left, 10,000 polygons) and the visibility-guided method (Right, 10,000 polygons). All images are rendered using flat shading. Compare features such as the teeth and forehead.

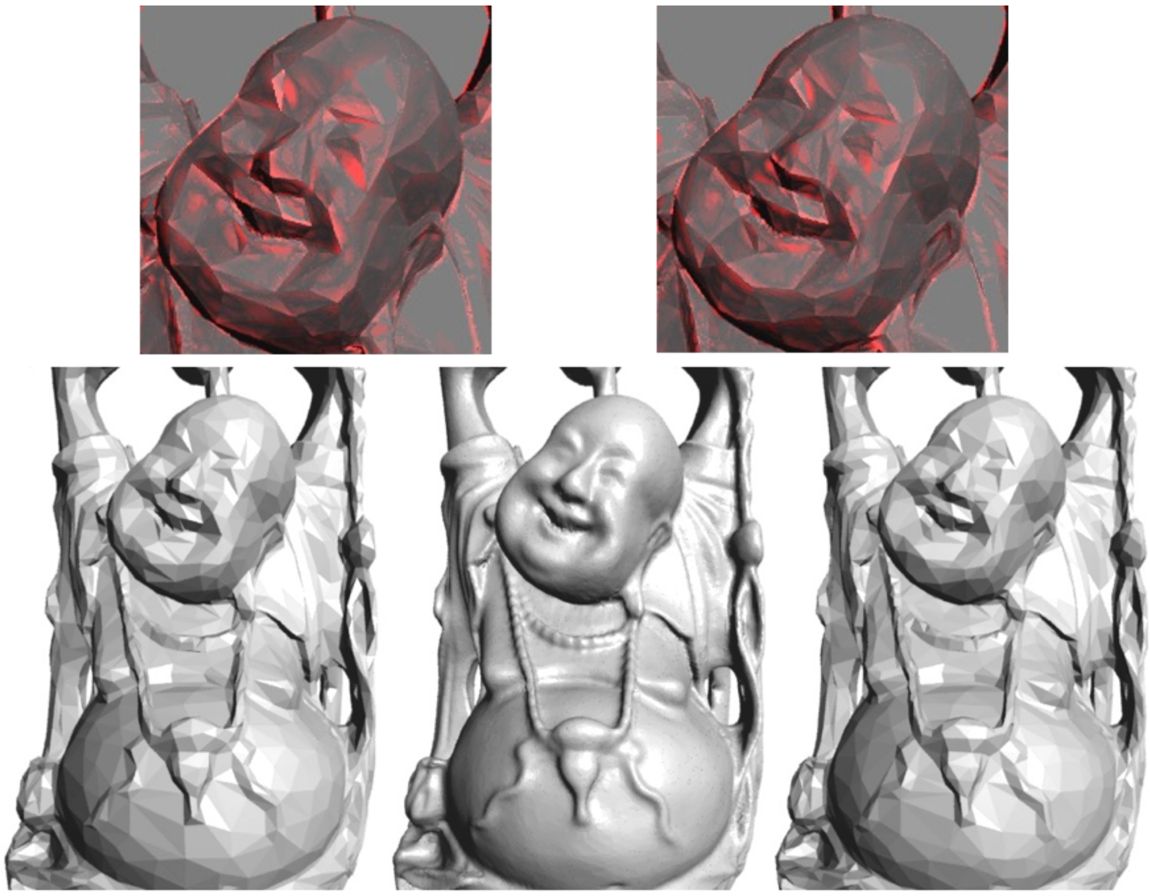

Figure 8: Visual comparisons between the original Buddha model (Bottom Middle, 1,087,416 polygons), and the simplified versions using the quadric-only method (Bottom Left, 20,000 polygons) and the visibility-guided method (Bottom Right, 20,000 polygons). All images are rendered using flat shading. The top row images use the red channel to encode image differences between the bottom row images. Note that the top left image (difference between the bottom center image and the bottom left image) has more areas of red than the top right image (difference between the bottom center image and the bottom right image).