Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments

Thomas A. Funkhouser and Carlo H. Séquin
University of California at Berkeley[‡]

Abstract

We describe an adaptive display algorithm for interactive frame rates during visualization of very complex virtual environments. The algorithm relies upon a hierarchical model representation in which objects are described at multiple levels of detail and can be drawn with various rendering algorithms. The idea behind the algorithm is to adjust image quality adaptively to maintain a uniform, user-specified target frame rate. We perform a constrained optimization to choose a level of detail and rendering algorithm for each potentially visible object in order to generate the "best" image possible within the target frame time. Tests show that the algorithm generates more uniform frame rates than other previously described detail elision algorithms with little noticeable difference in image quality during visualization of complex models.

CR Categories and Subject Descriptors:
[Computer Graphics]: I.3.3 Picture/Image Generation – *viewing algorithms*; I.3.5 Computational Geometry and Object Modeling – *geometric algorithms, object hierarchies*; I.3.7 Three-Dimensional Graphics and Realism – *virtual reality*.

1 Introduction

Interactive computer graphics systems for visualization of realistic-looking, three-dimensional models are useful for evaluation, design and training in virtual environments, such as those found in architectural and mechanical CAD, flight simulation, and virtual reality. These visualization systems display images of a three-dimensional model on the screen of a computer workstation as seen from a simulated observer's viewpoint under interactive control by a user. If images are rendered smoothly and quickly enough, an illusion of real-time exploration of a virtual environment can be achieved as the simulated observer moves through the model.

[‡]Computer Science Division, Berkeley, CA 94720

It is important for a visualization system to maintain an interactive frame rate (e.g., a constant ten frames per second). If frame rates are too slow, or too jerky, the interactive feel of the system is greatly diminished [3]. However, realistic-looking models may contain millions of polygons – far more than currently available workstations can render at interactive frame rates. Furthermore, the complexity of the portion of the model visible to the observer can be highly variable. Tens of thousands of polygons might be simultaneously visible from some observer viewpoints, whereas just a few can be seen from others. Programs that simply render all potentially visible polygons with some predetermined quality may generate frames at highly variable rates, with no guaranteed upper bound on any single frame time.

Using the UC Berkeley Building Walkthrough System [5] and a model of Soda Hall, the future Computer Science Building at UC Berkeley, as a test case, we have developed an adaptive algorithm for interactive visualization that guarantees a user-specified target frame rate. The idea behind the algorithm is to trade image quality for interactivity in situations where the environment is too complex to be rendered in full detail at the target frame rate. We perform a constrained optimization that selects a level of detail and a rendering algorithm with which to render each potentially visible object to produce the "best" image possible within a user-specified target frame time. In contrast to previous culling techniques, this algorithm guarantees a uniform, bounded frame rate, even during visualization of very large, complex models.

2 Previous Work

2.1 Visibility Determination

In previous work, visibility algorithms have been described that compute the portion of a model potentially visible from a given observer viewpoint [1, 11]. These algorithms cull away large portions of a model that are occluded from the observer's viewpoint, and thereby improve frame rates significantly. However, in very detailed models, often more polygons are visible from certain observer viewpoints than can be rendered in an interactive frame time. Certainly, there is no upper bound on the complexity of the scene visible from an observer's viewpoint. For instance, consider walking through a very detailed model of a fully stocked department store, or viewing an assembly of a complete airplane engine. In our model of Soda Hall, there are some viewpoints from which an observer can see more than eighty thousand polygons. Clearly, visibility processing alone is not sufficient to guarantee an interactive frame rate.

2.2 Detail Elision

To reduce the number of polygons rendered in each frame, an interactive visualization system can use *detail elision*. If a model can be described by a hierarchical structure of *objects*, each of which is represented at multiple *levels of detail* (LODs), as shown in Figure 1, simpler representations of an object can be used to improve frame rates and memory utilization during interactive visualization. This technique was first described by Clark [4], and has been used by numerous commercial visualization systems [9]. If different representations for the same object have similar appearances and are blended smoothly, using transparency blending or three-dimensional interpolation, transitions between levels of detail are barely noticeable during visualization.

241 Polygons 44 Polygons

Figure 1: Two levels of detail for a chair.

Previously described techniques for choosing a level of detail at which to render each visible object use static heuristics, most often based on a threshold regarding the size or distance of an object to the observer [2, 8, 9, 13], or the number of pixels covered by an average polygon [5]. These simple heuristics can be very effective at improving frame rates in cases where most visible objects are far away from the observer and map to very few pixels on the workstation screen. In these cases, simpler representations of some objects can be displayed, reducing the number of polygons rendered without noticeably reducing image quality.

Although static heuristics for visibility determination and LOD selection improve frame rates in many cases, they do not generally produce a *uniform* frame rate. Since LODs are computed independently for each object, the number of polygons rendered during each frame time depends on the size and complexity of the objects visible to the observer. The frame rate may vary dramatically from frame to frame as many complex objects become visible or invisible, and larger or smaller.

Furthermore, static heuristics for visibility determination and LOD selection do not even guarantee a *bounded* frame rate. The frame rate can become arbitrarily slow, as the scene visible to the observer can be arbitrarily complex. In many cases, the frame rate may become so slow that the system is no longer interactive. Instead, a LOD selection algorithm should adapt to overall scene complexity in order to produce uniform, bounded frame rates.

2.3 Adaptive Detail Elision

In an effort to maintain a specified *target frame rate*, some commercial flight simulators use an adaptive algorithm that adjusts the size threshold for LOD selection based on feedback regarding the time required to render previous frames [9]. If the previous frame took longer than the target frame time, the size threshold for LOD selection is increased so that future frames can be rendered more quickly.

This adaptive technique works reasonably well for flight simulators, in which there is a large amount of coherence in scene complexity from frame to frame. However, during visualization of more discontinuous virtual environments, scene complexity can vary radically between successive frames. For instance, in a building walkthrough, the observer may turn around a corner into a large atrium, or step from an open corridor into a small, enclosed office. In these situations, the number and complexity of the objects visible to the observer changes suddenly. Thus the size threshold chosen based on the time required to render previous frames is inappropriate, and can result in very poor performance until the system reacts. Overshoot and oscillation can occur as the feedback control system attempts to adjust the size threshold more quickly to achieve the target frame rate.

In order to *guarantee* a bounded frame rate during visualization of discontinuous virtual environments, an adaptive algorithm for LOD selection should be *predictive*, based on the complexity of the scene to be rendered in the current frame, rather than *reactive*, based only on the time required to render previous frames. A predictive algorithm might estimate the time required to render every object at every level of detail, and then compute the largest size threshold that allows the current frame to be rendered within the target frame time. Unfortunately, implementing a predictive algorithm is nontrivial, since no closed-form solution exists for the appropriate size threshold.

3 Overview of Approach

Our approach is a generalization of the predictive approach. Conceptually, every potentially visible object can be rendered at any level of detail, and with any rendering algorithm (e.g., flat-shaded, Gouraud-shaded, texture mapped, etc.). Every combination of objects rendered with certain levels of detail and rendering algorithms takes a certain amount of time, and produces a certain image. We aim to find the combination of levels of detail and rendering algorithms for all potentially visible objects that produces the "best" image possible within the target frame time.

More formally, we define an *object tuple*, (O, L, R), to be an instance of object O, rendered at level of detail L, with rendering algorithm R. We define two heuristics for object tuples: $Cost(O, L, R)$ and $Benefit(O, L, R)$. The *Cost* heuristic estimates the time required to render an object tuple; and the *Benefit* heuristic estimates the "contribution to model perception" of a rendered object tuple. We define S to be the set of object tuples rendered in each frame. Using these formalisms, our approach for choosing a level of detail and rendering algorithm for each potentially visible object can be stated:

$$\begin{aligned} &\text{Maximize :} && \textstyle\sum_S Benefit(O, L, R) \\ &\text{Subject to :} && \textstyle\sum_S Cost(O, L, R) \leq TargetFrameTime \end{aligned} \quad (1)$$

This formulation captures the essence of image generation with real-time constraints: "do as well as possible in a given amount of time." As such, it can be applied to a wide variety of problems that require images to be displayed in a fixed amount of time, including adaptive ray tracing (i.e., given a fixed number of rays, cast those that contribute most to the image), and adaptive radiosity (i.e., given

a fixed number of form-factor computations, compute those that contribute most to the solution). If levels of detail representing "no polygons at all" are allowed, this approach handles cases where the target frame time is not long enough to render all potentially visible objects even at the lowest level of detail. In such cases, only the most "valuable" objects are rendered so that the frame time constraint is not violated. Using this approach, it is possible to generate images in a short, fixed amount of time, rather than waiting much longer for images of the highest quality attainable.

For this approach to be successful, we need to find *Cost* and *Benefit* heuristics that can be computed quickly and accurately. Unfortunately, *Cost* and *Benefit* heuristics for a specific object tuple cannot be predicted with perfect accuracy, and may depend on other object tuples rendered in the same image. A perfect *Cost* heuristic may depend on the model and features of the graphics workstation, the state of the graphics system, the state of the operating system, and the state of other programs running on the machine. A perfect *Benefit* heuristic would consider occlusion and color of other object tuples, human perception, and human understanding. We cannot hope to quantify all of these complex factors in heuristics that can be computed efficiently. However, using several simplifying assumptions, we have developed approximate *Cost* and *Benefit* heuristics that are both efficient to compute and accurate enough to be useful.

4 Cost Heuristic

The $Cost(O, L, R)$ heuristic is an estimate of the time required to render object O with level of detail L and rendering algorithm R. Of course, the actual rendering time for a set of polygons depends on a number of complex factors, including the type and features of the graphics workstation. However, using a model of a generalized rendering system and several simplifying assumptions, it is possible to develop an efficient, approximate *Cost* heuristic that can be applied to a wide variety of workstations. Our model, which is derived from the *Graphics Library Programming Tools and Techniques* document from Silicon Graphics, Inc. [10], represents the rendering system as a pipeline with the two functional stages shown in Figure 2:

- *Per Primitive:* coordinate transformations, lighting calculations, clipping, etc.
- *Per Pixel:* rasterization, z-buffering, alpha blending, texture mapping, etc.

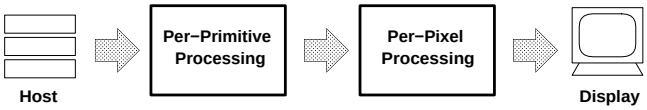


Figure 2: Two-stage model of the rendering pipeline.

Since separate stages of the pipeline run in parallel, and must wait only if a subsequent stage is "backed up," the throughput of the pipeline is determined by the speed of the slowest stage – i.e., the bottleneck. If we assume that the host is able to send primitives to the graphics subsystem faster than they can be rendered, and no other operations are executing that affect the speed of any stage of the graphics subsystem, we can model the time required to render an object tuple as the maximum of the times taken by any of the stages.

We model the time taken by the *Per Primitive* stage as a linear combination of the number of polygons and vertices in an object tuple, with coefficients that depend on the rendering algorithm and machine used. Likewise, we assume that the time taken by the *Per Pixel* stage is proportional to the number of pixels an object covers. Our model for the time required to render an object tuple is:

$$Cost(O, L, R) = max \left\{ \begin{array}{c} C_1 Poly(O, L) + C_2 Vert(O, L) \\ C_3 Pix(O) \end{array} \right\}$$

where O is the object, L is the level of detail, R is the rendering algorithm, and C_1, C_2 and C_3 are constant coefficients specific to a rendering algorithm and machine.

For a particular rendering algorithm and machine, useful values for these coefficients can be determined experimentally by rendering sample objects with a wide variety of sizes and LODs, and graphing measured rendering times versus the number of polygons, vertices and pixels drawn. Figure 3a shows measured times for rendering four different LODs of the chair shown in Figure 1 rendered with flat-shading. The slope of the best fitting line through the data points represents the time required *per polygon* during this test. Using this technique, we have derived cost model coefficients for our Silicon Graphics VGX 320 that are accurate within 10% at the 95% confidence level. A comparison of actual and predicted rendering times for a sample set of frames during an interactive building walkthrough is shown in Figure 3b.

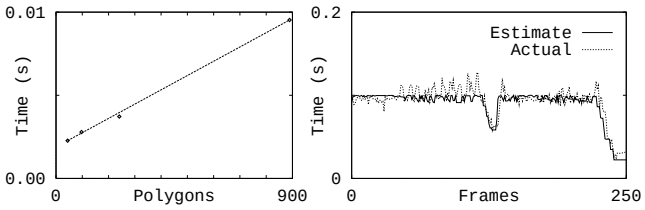


Figure 3: Cost model coefficients can be determined empirically. The plot in (a) shows actual flat-shaded rendering times for four LODs of a chair, and (b) shows a comparison of actual and estimated rendering times of frames during an interactive building walkthrough.

5 Benefit Heuristic

The $Benefit(O, L, R)$ heuristic is an estimate of the "contribution to model perception" of rendering object O with level of detail L and rendering algorithm R. Ideally, it predicts the amount and accuracy of information conveyed to a user due to rendering an object tuple. Of course, it is extremely difficult to accurately model human perception and understanding, so we have developed a simple, easy-to-compute heuristic based on intuitive principles.

Our *Benefit* heuristic depends primarily on the size of an object tuple in the final image. Intuitively, objects that appear larger to the observer "contribute" more to the image (see Figure 4). Therefore, the base value for our *Benefit* heuristic is simply an estimate of the number of pixels covered by the object.

Our *Benefit* heuristic also depends on the "accuracy" of an object tuple rendering. Intuitively, using a more detailed representation or a more realistic rendering algorithm for an object generates a higher quality image, and therefore conveys more accurate information to the user. Conceptually, we evaluate the "accuracy" of an object tuple rendering by comparison to an *ideal image* generated with an

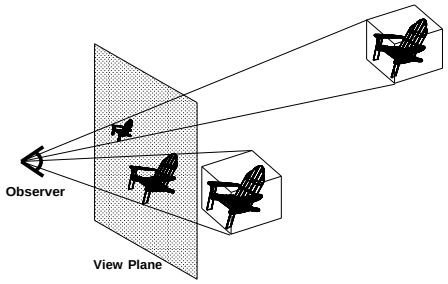
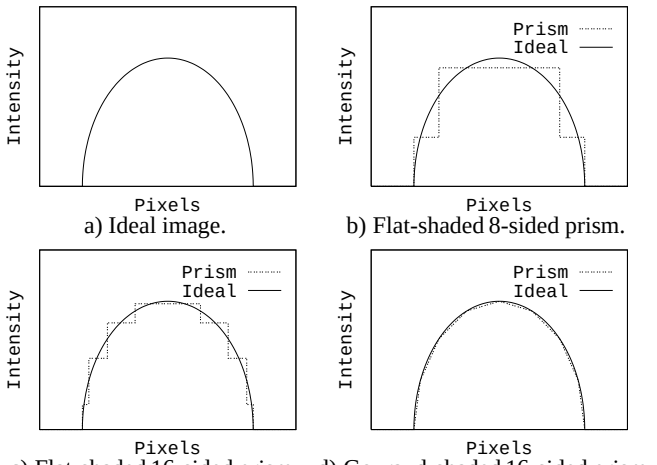


Figure 4: Objects that appear larger "contribute" more to the image.

ideal camera. For instance, consider generating a gray-level image of a scene containing only a cylinder with a diffusely reflecting Lambert surface illuminated by a single directional light source in orthonormal projection. Figure 5a shows an intensity plot of a sample scan-line of an ideal image generated for the cylinder.

First, consider approximating this ideal image with an image generated using a flat-shaded, polygonal representation for the cylinder. Since a single color is assigned to all pixels covered by the same polygon, a plot of pixel intensities across a scan-line of such an image is a stair-function. If an 8-sided prism is used to represent the cylinder, at most 4 distinct colors can appear in the image (one for each front-facing polygon), so the resulting image does not approximate the ideal image very well at all, as shown in Figure 5b. By comparison, if a 16-sided prism is used to represent the cylinder, as many as 8 distinct colors can appear in the image, generating a closer approximation to the ideal image, as shown in Figure 5c.



a) Ideal image. b) Flat-shaded 8-sided prism.

c) Flat-shaded 16-sided prism. d) Gouraud-shaded 16-sided prism.

Figure 5: Plots of pixel intensity across a sample scan-line of images generated using different representations and rendering algorithms for a simple cylinder.

Next, consider using Gouraud shading for a polygonal representation. In Gouraud shading, intensities are interpolated between vertices of polygons, so a plot of pixel intensities is a continuous, piecewise-linear function. Figure 5d shows a plot of pixel intensities across a scan line for a Gouraud shaded 16-sided prism. Compared to the plot for the flat-shaded image (Figure 5b), the Gouraud shaded image approximates the ideal image much more closely.

More complex representations (e.g., parametric or implicit surfaces) and rendering techniques (e.g., Phong shading, antialiasing or ray tracing) could be used to approximate the ideal image even more closely. Based on this intuition, we assume that the "error," i.e., the difference from the ideal image, decreases with the number of samples (e.g., rays/vertices/polygons) used to render an object tuple, and is dependent on the type of interpolation method used (e.g., Gouraud/flat). We capture these effects in the *Benefit* heuristic by multiplying by an "accuracy" factor:

$$Accuracy(O, L, R) = 1 - Error = 1 - \frac{BaseError}{Samples(L, R)^m}$$

where Samples(L, R) is #pixels for ray tracing, or #vertices for Gouraud shading, or #polygons for flat-shading (but never more than #pixels); and m is an exponent dependent on the interpolation method used (flat = 1, Gouraud = 2). The *BaseError* is arbitrarily set to 0.5 to give a strong error for a curved surface represented by a single flat polygon, but still account for a significantly higher benefit than not rendering the surface at all.

In addition to the size and accuracy of an object tuple rendering, our *Benefit* heuristic depends on on several other, more qualitative, factors, some of which apply to a static image, while others apply to sequences of images:

- **Semantics:** Some types of object may have inherent "importance." For instance, walls might be more important than pencils to the user of a building walkthrough; and enemy robots might be most important to the user of a video game. We adjust the *Benefit* of each object tuple by an amount proportional to the inherent importance of its object type.

- **Focus:** Objects that appear in the portion of the screen at which the user is looking might contribute more to the image than ones in the periphery of the user's view. Since we currently do not track the user's eye position, we simply assume that objects appearing near the middle of the screen are more important than ones near the side. We reduce the *Benefit* of each object tuple by an amount proportional to its distance from the middle of the screen.

- **Motion Blur:** Since objects that are moving quickly across the screen appear blurred or can be seen for only a short amount of time, the user may not be able to see them clearly. So we reduce the *Benefit* of each object tuple by an amount proportional to the ratio of the object's apparent speed to the size of an average polygon.

- **Hysteresis:** Rendering an object with different levels of detail in successive frames may be bothersome to the user and may reduce the quality of an image sequence. Therefore, we reduce the *Benefit* of each object tuple by an amount proportional to the difference in level of detail or rendering algorithm from the ones used for the same object in the previous frame.

Each of these qualitative factors is represented by a multiplier between 0.0 and 1.0 reflecting a possible reduction in object tuple benefit. The overall *Benefit* heuristic is a product of all the aforementioned factors:

$$Benefit(O, L, R) = Size(O) * Accuracy(O, L, R) * Importance(O) * Focus(O) * Motion(O) * Hysteresis(O, L, R)$$

This *Benefit* heuristic is a simple experimental estimate of an object tuple's "contribution to model perception." Greater *Benefit* is assigned to object tuples that are larger (i.e., cover more pixels in the image), more realistic-looking (i.e., rendered with higher levels of detail, or better rendering algorithms), more important (i.e.,

semantically, or closer to the middle of the screen), and more apt to blend with other images in a sequence (i.e., hysteresis). In our implementation, the user can manipulate the relative weighting of these factors interactively using sliders on a control panel, and observe their effects in a real-time walkthrough. Therefore, although our current *Benefit* heuristic is rather ad hoc, it is useful for experimentation until we are able to encode more accurate models for human visual perception and understanding.

6 Optimization Algorithm

We use the *Cost* and *Benefit* heuristics described in the previous sections to choose a set of object tuples to render each frame by solving equation 1 in Section 3.

Unfortunately, this constrained optimization problem is NP-complete. It is the Continuous Multiple Choice Knapsack Problem [6, 7], a version of the well-known Knapsack Problem in which elements are partitioned into candidate sets, and at most one element from each candidate set may be placed in the knapsack at once. In this case, the set *S* of object tuples rendered is the knapsack, the object tuples are the elements to be placed into the knapsack, the target frame time is the size of the knapsack, the sets of object tuples representing the same object are the candidate sets, and the *Cost* and *Benefit* functions specify the "size" and "profit" of each element, respectively. The problem is to select the object tuples that have maximum cumulative benefit, but whose cumulative cost fits in the target frame time, subject to the constraint that only one object tuple representing each object may be selected.

We have implemented a simple, greedy approximation algorithm for this problem that selects object tuples with the highest *Value* ($Benefit(O,L,R)/Cost(O,L,R)$). Logically, we add object tuples to *S* in descending order of *Value* until the maximum cost is competely claimed. However, if an object tuple is added to *S* which represents the same object as another object tuple already in *S*, only the object tuple with the maximum benefit of the two is retained. The merit of this approach can be explained intuitively by noting that each subsequent portion of the frame time is used to render the object tuple with the best available "bang for the buck." It is easy to show that a simple implementation of this greedy approach runs in $O(n \log n)$ time for n potentially visible objects, and produces a solution that is at least half as good as the optimal solution [6].

Rather than computing and sorting the *Benefit*, *Cost*, and *Value* for all possible object tuples during every frame, as would be required by a naive implementation, we have implemented an incremental optimization algorithm that takes advantage of the fact that there is typically a large amount of coherence between successive frames. The algorithm works as follows: At the start of the algorithm, an object tuple is added to *S* for each potentially visible object. Initially, each object is assigned the LOD and rendering algorithm chosen in the previous frame, or the lowest LOD and rendering algorithm if the object is newly visible. In each iteration of the optimization, the algorithm first increments the accuracy attribute (LOD or rendering algorithm) of the object that has the highest subsequent *Value*. It then decrements the accuracy attributes of the object tuples with the lowest current *Value* until the cumulative cost of all object tuples in *S* is less than the target frame time. The algorithm terminates when the same accuracy attribute of the same object tuple is both incremented and decremented in the same iteration.

This incremental implementation finds an approximate solution that is the same as found by the naive implementation if *Values* of object tuples decrease monotonically as tuples are rendered with greater accuracy (i.e., there are diminishing returns with more complex renderings). In any case, the worst-case running time for the algorithm is $O(n \log n)$. However, since the initial guess for the LOD and rendering algorithm for each object is generated from the previous frame, and there is often a large amount of coherence from frame to frame, the algorithm completes in just a few iterations on average. Moreover, computations are done in parallel with the display of the previous frame on a separate processor in a pipelined architecture; they do not increase the effective frame rate as long as the time required for computation is not greater than the time required for display.

7 Test Methods

To test whether this new cost/benefit optimization algorithm produces more uniform frame rates than previous LOD selection algorithms, we ran a set of tests with our building walkthrough application using four different LOD selection algorithms:

a) **No Detail Elision:** Each object is rendered at the highest LOD.

b) **Static:** Each object is rendered at the highest LOD for which an average polygon covers at least 1024 pixels on the screen.

c) **Feedback:** Similar to *Static* test, except the size threshold for LOD selection is updated in each frame by a feedback loop, based on the difference between the time required to render the previous frame and the target frame time of one-tenth of a second.

d) **Optimization:** Each object is rendered at the LOD chosen by the cost/benefit optimization algorithm described in Sections 3 and 6 in order to meet the target frame time of one-tenth of a second. For comparison sake, the *Benefit* heuristic is limited to consideration of *object size* in this test, i.e., all other *Benefit* factors are set to 1.0.

All tests were performed on a Silicon Graphics VGX 320 workstation with two 33MHz MIPS R3000 processors and 64MB of memory. We used an *eye-to-object* visibility algorithm described in [12] to determine a set of potentially visible objects to be rendered in each frame. The application was configured as a two-stage pipeline with one processor for visibility and LOD selection computations and another separate processor for rendering. Timing statistics were gathered using a $16\mu s$ timer.

In each test, we used the sample observer path shown in Figure 6 through a model of an auditorium on the third floor of Soda Hall. The model was chosen because it is complex enough to differentiate the characteristics of various LOD selection algorithms (87,565 polygons), yet small enough to reside entirely in main memory so as to eliminate the effects of memory management in our tests. The test path was chosen because it represents typical behavior of real users of a building walkthrough system, and highlights the differences between various LOD selection algorithms. For instance, at the observer viewpoint marked 'A', many complex objects are simultaneously visible, some of which are close and appear large to the observer; at the viewpoint marked 'B', there are very few objects visible to the observer, most of which appear small; and at the viewpoint marked 'C', numerous complex objects become visible suddenly as the observer spins around quickly. We refer to these marked observer viewpoints in the analysis, as they are the viewpoints at which the differences between the various LOD selection algorithms are most pronounced.

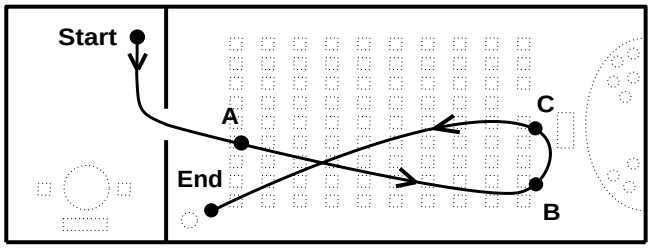


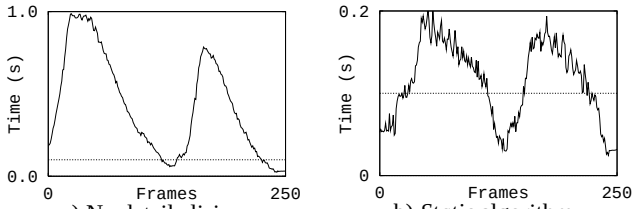
Figure 6: Test observer path through a model of an auditorium.

LOD Selection Algorithm	Compute Time		Frame Time		
	Mean	Max	Mean	Max	StdDev
None	0.00	0.00	0.43	0.99	0.305
Static	0.00	0.01	0.11	0.20	0.048
Feedback	0.00	0.01	0.10	0.16	0.026
Optimization	0.01	0.03	0.10	0.13	0.008

Table 1: Cumulative statistics for test observer path (in seconds).

8 Results and Discussion

Figure 7 shows plots of the frame time (seconds per frame) for each observer viewpoint along the test path for the four LOD selection algorithms tested. Table 1 shows cumulative compute time (i.e., time required for execution of the LOD selection algorithm) and frame time statistics for all observer viewpoints along the test path.



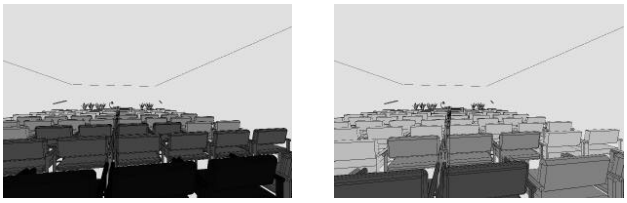
a) No detail elision. b) *Static* algorithm.

c) *Feedback* algorithm. d) *Optimization* algorithm.

Figure 7: Plots of frame time for every observer viewpoint along test observer path using a) no detail elision, b) static algorithm, c) feedback algorithm, and d) optimization algorithm. Note: the "Frame Time" axis in plot (a) is five-times larger than the others.

If no detail elision is used, and all potentially visible objects are rendered at the highest LOD, the time required for each frame is generally long and non-uniform, since it depends directly on the number and complexity of the objects visible to the observer (see Figure 7a). In our test model, far too many polygons are visible from most observer viewpoints to generate frames at interactive rates without detail elision. For instance, at the observer viewpoint marked 'A' in Figure 6, 72K polygons are simultaneously visible, and the frame time is 0.98 seconds. Overall, the mean frame time for all observer viewpoints on the test path is 0.43 seconds per frame.

If the *Static* LOD selection algorithm is used, objects whose average polygon is smaller than a size threshold fixed at 1024 pixels per polygon are rendered with lower LODs. Even though the frame rate is much faster than without detail elision, there is still a large amount of variability in the frame time, since it depends on the size and complexity of the objects visible from the observer's viewpoint (see Figure 7b). For instance, at the observer viewpoint marked 'A', the frame time is quite long (0.19 seconds) because many visible objects are complex and appear large to the observer. A high LOD is chosen for each of these objects independently, resulting in a long overall frame time. This result can seen clearly in Figure 8a which depicts the LOD selected for each object in the frame for observer viewpoint 'A' – higher LODs are represented by darker shades of gray. On the other hand, the frame time is very short in the frame at the observer viewpoint marked 'B' (0.03 seconds). Since all visible objects appear relatively small to the observer, they are rendered at a lower LOD even though more detail could have been rendered within the target frame time. In general, it is impossible to choose a single size threshold for LOD selection that generates uniform frame times for all observer viewpoints.

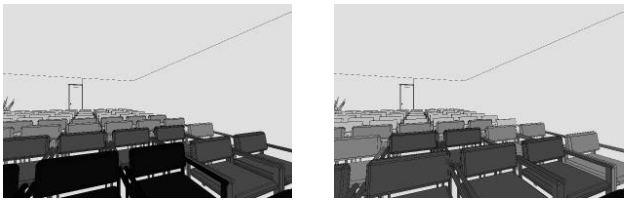


a) *Static* algorithm b) *Optimization* algorithm

Figure 8: Images depicting the LODs selected for each object at the observer viewpoints marked 'A' using the *Static* and *Optimization* algorithms. Darker shades of gray represent higher LODs.

The *Feedback* algorithm adjusts the size threshold for LOD selection adaptively based on the time taken to render previous frames in an effort to maintain a uniform frame rate. This algorithm generates a fairly uniform frame rate in situations of smoothly varying scene complexity, as evidenced by the relatively flat portions of the frame time curve shown in Figure 7c (frames 1–125). However, in situations where the complexity of the scene visible to the observer changes suddenly, peaks and valleys appear in the curve. Sometimes the frame time generated using the *Feedback* algorithm can be even longer than the one generated using the *Static* algorithm, as the *Feedback* algorithm is lured into a inappropriately low size threshold during times of low scene complexity. For instance, just before the viewpoint marked 'C', the observer is looking at a relatively simple scene containing just a few objects on the stage, so frame times are very short, and the size threshold for LOD selection is reduced to zero. However, at the viewpoint marked 'C', many chairs become visible suddenly as the observer spins around quickly. Since the adaptive size threshold is set very low, inappropriately high LODs are chosen for most objects (see Figure 9a), resulting in a frame time of 0.16 seconds. Although the size threshold can often adapt quickly after such discontinuities in scene complexity, some effects related to this feedback control (i.e., oscillation, overshoot, and a few very slow frames) can be quite disturbing to the user.

In contrast, the *Optimization* algorithm predicts the complexity of the model visible from the current observer viewpoint, and chooses an appropriate LOD and rendering algorithm for each object to meet the target frame time. As a result, the frame time generated using the *Optimization* algorithm is much more uniform than using any of



a) *Feedback* algorithm b) *Optimization* algorithm

Figure 9: Images depicting the LODs selected for each object at the observer viewpoints marked 'C' using the *Feedback* and *Optimization* algorithms. Darker shades of gray represent higher LODs.

the other LOD selection algorithms (see Figure 7d). For all observer viewpoints along the test path, the standard deviation in the frame time is 0.008 seconds, less than one third of any of the other three algorithms tested. The longest frame time is 0.13 seconds, and the shortest is 0.075 seconds.

As the *Optimization* algorithm adjusts image quality to maintain a uniform, interactive frame rate, it attempts to render the "best" image possible within the target frame time for each observer viewpoint. As a result, there is usually little noticeable difference between images generated using the *Optimization* algorithm and ones generated with no detail elision at all. A comparison of images for observer viewpoint 'A' generated using a) no detail elision, and b) using the *Optimization* algorithm to meet a target frame time of one tenth of a second are shown in Figure 10. Figure 10a has 72,570 polygons and took 0.98 seconds to render, whereas Figure 10b has 5,300 polygons and took 0.10 seconds. Even though there are less than a tenth as many polygons in Figure 10b, the difference in image quality is barely noticeable. For reference, the LOD chosen for each object in Figure 10b is shown in Figure 8b. Note that reduction in rendering time does not map to a linear reduction in polygon count since polygons representing lower levels of detail tend to be bigger on average.

The *Optimization* algorithm is more general than other detail elision algorithms in that it also adjusts the rendering algorithm (and possibly other attributes in the future) for each object independently. Examine Figure 11, which shows three images of a small library on the sixth floor of Soda Hall containing several textured surfaces. Figure $11a_1$, shows an image generated using no detail elision – it contains 19,821 polygons and took 0.60 seconds to render. Figures $11b_1$ and $11c_1$ show images generated for the same observer viewpoint using the *Optimization* algorithm with target frame times of b) 0.15 seconds (4,217 polygons), and c) 0.10 seconds (1,389 polygons). Although the *Optimization* algorithm uses lower levels of detail for many objects (see Figures $11b_1$ and $11c_1$), and generates images that are quite different than the one generated with no detail elision (see Figures $11b_2$ and $11c_2$), all three images look very similar. Notice the reduced tessellation of chairs further from the observer, and the omission of texture on the bookshelves in Figure $11b_1$. Similarly, notice the flat-shaded chairs, and the omission of books on bookshelves and texture on doors in Figure 11 c_1.

Having experimented with several LOD selection algorithms in an interactive visualization application, we are optimistic that variation in image quality is less disturbing to a user than variation in the frame times, as long as different representations for each object appear similar, and transitions between representations are not very noticeable. Further experimentation is required to determine which types of rendering attributes can be blended smoothly during interactive visualization.

9 Conclusion

We have described an adaptive display algorithm for fast, uniform frame rates during interactive visualization of large, complex virtual environments. The algorithm adjusts image quality dynamically in order to maintain a user-specified frame rate, selecting a level of detail and an algorithm with which to render each potentially visible object to produce the "best" image possible within the target frame time.

Our tests show that the *Optimization* algorithm generates more uniform frame rates than other previously described detail elision algorithms with little noticeable difference in image quality during visualization of complex models. Interesting topics for further study include algorithms for automatic generation of multiresolution models, and experiments to develop measures of *image quality* and *image differences*.

10 Acknowledgements

We are grateful to Thurman Brown, Delnaz Khorramabadi, Priscilla Shih and Maryann Simmons for their efforts constructing the building model. Silicon Graphics, Inc. has been very generous, allowing us to use equipment, and donating a VGX 320 workstation to this project as part of a grant from the Microelectronics Innovation and Computer Research Opportunities (MICRO 1991) program of the State of California. We appreciate the assistance of Greg Ward, Sharon Fischler, and Henry Moreton who helped generate the color prints for this paper. Finally, we thank Seth Teller for his spatial subdivisions, visibility algorithms, and other important contributions to this project.

References

[1] Airey, John M., Rohlf, John H., and Brooks, Jr., Frederick P. Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. *ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics* , 24, 2 (1990), 41-50.

[2] Blake, Edwin H. A Metric for Computing Adaptive Detail in Animated Scenes using Object-Oriented Programming. *Eurographics '87*. G. Marechal (Ed.), Elsivier Science Publishers, B.V. (North-Holland), 1987.

[3] Brooks, Jr., Frederick P. Walkthrough - A Dynamic Graphics System for Simulating Virtual Buildings. *Proceedings of the 1986 Workshop on Interactive 3D Graphics* .

[4] Clark, James H. Hierarchical Geometric Models for Visible Surface Algorithms. *Communications of the ACM*, 19, 10 (October 1976), 547-554.

[5] Funkhouser, Thomas A., Séquin, Carlo H., and Teller, Seth J. Management of Large Amounts of Data in Interactive Building Walkthroughs. *ACM SIGGRAPH Special Issue on 1992 Symposium on Interactive 3D Graphics* , 11-20.

[6] Garey, Michael R., and Johnson, David S. Computers and Intractibility: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York, 1979.

[7] Ibaraki, T., Hasegawa, T., Teranaka, K., and Iwase J. The Multiple Choice Knapsack Problem. *J. Oper. Res. Soc. Japan* 21, 1978, 59-94.

[8] Rossignac, Jarek, and Borrel, Paul. Multi-resolution 3D approximations for rendering complex scenes. *IFIP TC 5.WG 5.10 II Conference on Geometric Modeling in Computer Graphics*, Genova, Italy, 1993. Also available as IBM Research Report RC 17697, Yorktown Heights, NY 10598.

[9] Schachter, Bruce J. (Ed.). *Computer Image Generation*. John Wiley and Sons, New York, NY, 1983.

[10] *Graphics Library Programming Tools and Techniques* , Document #007-1489-01, Silicon Graphics, Inc., 1992.

[11] Teller, Seth J., and Séquin, Carlo H. Visibility Preprocessing for Interactive Walkthroughs. Proceedings of SIGGRAPH '91. In *Computer Graphics* 25, 4 (August 1991), 61-69.

[12] Teller, Seth J. *Visibility Computations in Densely Occluded Polyhedral Environments*. Ph.D. thesis, Computer Science Division (EECS), University of California, Berkeley, 1992. Also available as UC Berkeley technical report UCB/CSD-92-708.

[13] Zyda, Michael J. Course Notes, Book Number 10, Graphics Video Laboratory, Department of Computer Science, Naval Postgraduate School, Monterey, California, November 1991.

a) No detail elision

b) *Optimization* algorithm (0.10 seconds)

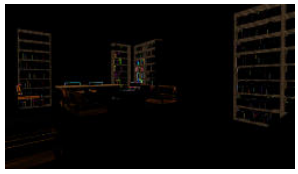
Figure 10: Images for observer viewpoint 'A' generated using a) no detail elision (72,570 polygons), and b) the *Optimization* algorithm with a 0.10 second target frame time (5,300 polygons).

a_1

a) No detail elision

b_1

b_2



b_3

b) *Optimization* algorithm (0.15 seconds)

c_1

c_2

c_3

c) *Optimization* algorithm (0.10 seconds)

Figure 11: Images of library generated using a) no detail elision (19,821 polygons), and the *Optimization* detail elision algorithm with target frame times of b) 0.15 seconds (4,217 polygons), and c) 0.10 seconds (1,389 polygons). LODs chosen for objects in b_1 and c_1 are shown in b_2 and c_c – darker shades of gray represent higher LODs. Pixel-by-pixel differences $abs(a_1 - b_1)$ and $abs(a_1 - c_1)$ are shown in b_3 and c_3 – brighter colors represent greater difference.