Visibility Culling Using Hierarchical Occlusion Maps *

Hansong Zhang Dinesh Manocha Tom Hudson Kenny Hoff
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175
{zhangh,manocha,hudson,hoff}@cs.unc.edu

Abstract: We present hierarchical occlusion maps (HOM) for visibility culling on complex models with high depth complexity. The culling algorithm uses an object space bounding volume hierarchy and a hierarchy of image space occlusion maps. Occlusion maps represent the aggregate of projections of the occluders onto the image plane. For each frame, the algorithm selects a small set of objects from the model as occluders and renders them to form an initial occlusion map, from which a hierarchy of occlusion maps is built. The occlusion maps are used to conservatively cull away a portion of the model not visible from the current viewpoint. The algorithm is applicable to all models and makes no assumptions about the size, shape, or type of occluders. It has been implemented on current graphics systems and has been applied to large models composed of hundreds of thousands of polygons. In practice, it achieves significant speedup in interactive walkthroughs of models with high depth complexity.
Key Words and Phrases: visibility culling, interactive display, image pyramid, occlusion culling, hierarchical data structures

1 Introduction

Interactive display and walkthrough of large geometric models currently pushes the limits of graphics technology. Environments composed of millions of primitives (e.g. polygons) are not uncommon in applications such as simulation-based design of large mechanical systems, architectural visualization, or walkthrough of outdoor scenes. Although throughput of graphics systems has increased considerably over the years, the size and complexity of these environments has been growing even faster. In order to display such models at interactive rates, the rendering algorithms need to use techniques based on visibility culling, levels-of-detail, texturing, etc. to limit the number of primitives rendered in each frame. In this paper, we focus on conservative visibility culling algorithms.

*Supported in part by a Sloan fellowship, ARO Contract P-34982-MA, NSF grant CCR-9319957, NSF grant CCR-9625217, ONR contract N00014-94-1-0738, ARPA contract DABT63-93-C-0048 and NSF/ARPA Science and Technology Center for Computer Graphics & Scientific Visualization NSF Prime contract No. 8920219.

Their goal is to cull away large portions of the environment not visible from the current viewpoint.

Our criteria for an effective visibility culling algorithm are *generality*, *interactive performance*, and *significant culling*. Additionally, in order for it to be *practical*, it should be implementable on current graphics systems and work well on large real-world models. To the best of our knowledge, no earlier algorithm can achieve all these goals simultaneously.

Main Contribution: In this paper, we present a new algorithm for visibility culling in complex environments with high depth complexity. At each frame, the algorithm carefully selects a small subset of the model as occluders and renders them to build *hierarchical occlusion maps* (HOM). The hierarchy is an image pyramid and each map in the hierarchy is composed of pixels corresponding to rectangular blocks in the screen space. The pixel value records the *opacity* of the block. The algorithm decomposes the visibility test for an object into a conservative two-dimensional overlap test, performed against the occlusion map hierarchy, and a conservative Z test to compare the depth. The overall approach combines an *object space* bounding volume hierarchy (also useful for view frustum culling) with the *image space* occlusion map hierarchy to cull away a portion of the model not visible from the current viewpoint. Some of the main features of the algorithm are:

1. **Generality:** The algorithm requires no special structures in the model and places *no restriction* on the types of occluders. The occluders may be polygonal objects, curved surfaces, or even not be geometrically defined (e.g. a billboard).

2. **Occluder Fusion:** A key characteristic of the algorithm is the ability to *combine* a "forest" of small or disjoint occluders, rather than using only large occluders. In most cases, the union of a set of occluders can occlude much more than what each of them can occlude taken separately. This is very useful for large mechanical CAD and outdoor models.

3. **Significant Culling:** On high depth complexity models, the algorithm is able to cull away a significant fraction (up to 95%) of the model from most viewpoints.

4. **Portability:** The algorithm can be implemented on most current graphics systems. Its main requirement is the ability to read back the frame-buffer. The construction of hierarchical occlusion maps can be accelerated by texture mapping hardware. It is not susceptible to degeneracies in the input and can be parallelized on multiprocessors.

5. **Efficiency:** The construction of occlusion maps takes a few milliseconds per frame on medium- to high-end graphics systems. The culling algorithm achieves significant speedup in interactive walkthroughs of models with high depth complexity. The algorithm involves no significant preprocessing and is applicable to dynamic environments.

6. **Approximate Visibility Culling:** Our approach can also use the hierarchy of maps to perform *approximate* culling. By varying an *opacity threshold* parameter

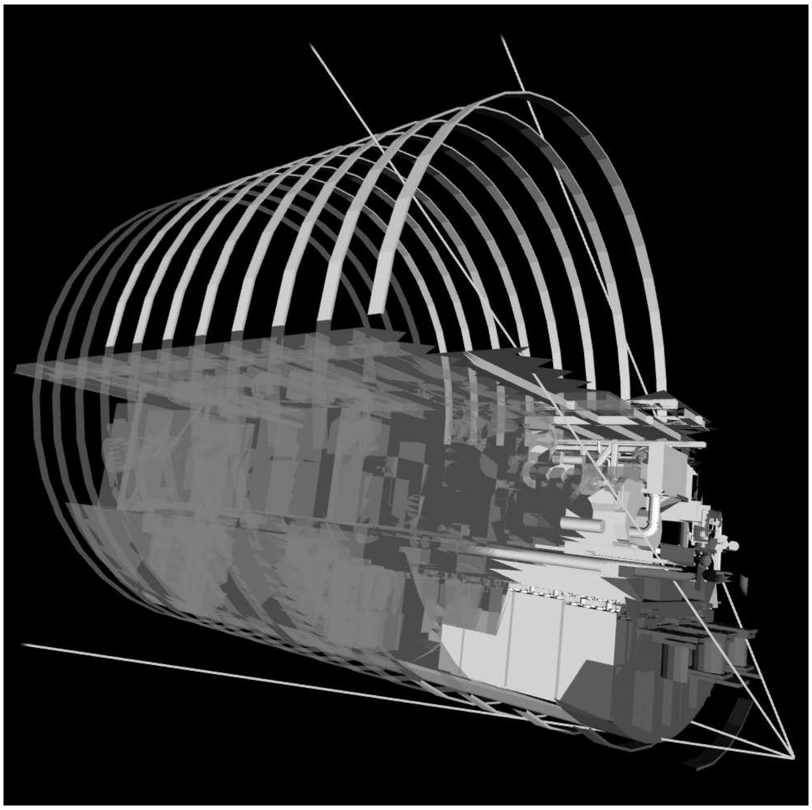

Figure 1: Demonstration of our algorithm on the CAD model of a submarine's auxiliary machine room. The model has 632,252 polygons. The green lines outline the viewing frustum. The blue color indicates objects selected as occluders, gray the objects not culled by our algorithm and transparent red the objects culled away. For this particular view, 82.7% of the model is culled.

the algorithm is able to fill small transparent holes in the occlusion maps and to cull away portions of the model which are visible through small gaps in the occluders.

The resulting algorithm has been implemented on different platforms (SGI Max Impact and Infinite Reality) and applied to city models, CAD models, and dynamic environments. It obtains considerable speedup in overall frame rate. In Figure 1 we demonstrate its performance on a submarine's Auxiliary Machine Room.

Organization: The rest of the paper is organized in the following manner. We briefly survey related work in Section 2 and give an overview of our approach in Section 3. Section 4 describes occlusion maps and techniques for fast implementation on current graphics systems. In Section 5 we describe the entire culling algorithm. We describe its implementation and performance in Section 6. Section 7 analyses our algorithm and compares it with other approaches. Finally, in Section 8 we briefly describe some future directions.

2 Related Work

Visibility computation and hidden surface removal are classic problems in computer graphics [FDHF90]. Some of the commonly used visibility algorithms are based on Z-buffer [Cat74] and view-frustum culling [Cla76, GBW90]. Others include Painter's Algorithm [FDHF90] and area-subdivision algorithms [War69, FDHF90].

There is significant literature on visible surface computation in computational geometry. Many asymptotically efficient algorithms have been proposed for hidden surface removal [Mul89, McK87]. See [Dor94] for a recent survey. However, the practical utility of these algorithms is unclear at the moment.

Efficient algorithms for calculating the visibility relationship among a static group of 3D polygons from arbitrary viewpoints have been proposed based on the binary space-partitioning (BSP) tree [FKN80]. The tree construction may involve considerable pre-processing in terms of time and space requirements for large models. In [Nay92], Naylor has given an output-sensitive visibility algorithm using BSPs. It uses a 2D BSP tree to represent images and presents an algorithm to project a 3D BSP tree, representing the model in object space, into a 2D BSP tree representing its image.

Many algorithms structure the model database into *cells* or regions, and use a combination of off-line and on-line algorithms for cell-to-cell visibility and the conservative computation of the potentially visible set (PVS) of primitives [ARB90, TS91, LG95]. In particular, Teller et al. [TS91, Tel92, TH93] have presented analytic algorithms for cell-to-cell visibility and also applied them to efficient calculation of form factors for radiosity. Such approaches have been successfully used to visualize architectural models, where the division of a building into discrete rooms lends itself to a natural division of the database into cells. It is not apparent that cell-based approaches can be generalized to an arbitrary model.

Other algorithms for densely-occluded but somewhat less-structured models have been proposed by Yagel and Ray [YR96]. They used regular spatial subdivision to partition the model into cells and describe a 2D implementation. However, the resulting algorithm is very memory-intensive and does not scale well to large models.

Object space algorithms for occlusion culling in general polygonal models have been presented by Coorg and Teller [CT96a, CT96b] and Hudson et al. [Hu96]. These algorithms dynamically compute a subset of the objects as occluders and use them to cull away portions of the model. In particular, [CT96a, CT96b] compute an arrangement corresponding to a linearized portion of an aspect graph and track the viewpoint within it to check for occlusion. [Hu96] use shadow frusta and fast interference tests for occlusion culling. All of them are object-space algorithms and the choice of occluder is restricted to convex objects or simple combination of convex objects (e.g. two convex polytope sharing an edge). These algorithms are unable to combine a "forest" of small non-convex or disjoint occluders to cull away large portions of the model.

A hierarchical Z-buffer algorithm combining spatial and temporal coherence has been presented in [GKM93, GK94, Gre95]. It uses two hierarchical data structures: an octree and a Z-pyramid. The algorithm exploits coherence by performing visibility queries on the Z-pyramid and is very effective in culling large portions of high-depth complexity models. However, most current graphics systems do not support the Z-pyramid capability in hardware, and simulating it in software can be relatively expensive. In [GK94], Greene and Kass used a quadtree data structure to test visibility throughout image-space regions for anti-aliased rendering.

More recently, Greene [Gre96] has presented a hierarchical tiling algorithm using coverage masks. It uses an image hierarchy named a "coverage pyramid" for visibility culling. Traversing polygons from front to back, it can process densely occluded scenes efficiently and is well suited to anti-aliasing by oversampling and filtering.

For dynamic environments, Sudarsky and Gotsman [SG96] have presented an output-sensitive algorithm which minimizes the time required to update the hierarchical data structure for a dynamic object and minimize the number of dynamic objects for which the structure has to be updated.

A number of techniques for interactive walkthrough of large geometric databases have been proposed. Refer to [Br96] for a recent survey. A number of commercial systems like *Performer* [RH94], used for high performance graphics, and *Brush* [SBM+94], used for visualizing architectural and CAD models, are available. They use techniques based on view-frustum culling, levels-of-detail, etc., but have little support for occlusion culling on arbitrary models.

The structure of hierarchical occlusion maps is similar to some of the hierarchies that have been proposed for images, such as image pyramids [TP75], MIP maps [Wil83], Z-pyramids [GKM93], coverage pyramids [Gre96], and two-dimensional wavelet transforms like the non-standard decomposition [GBR91, SDS96].

3 Overview

In this paper we present a *conservative* solution to the visibility problem. Given a scene database and a viewpoint, our algorithm culls a subset of the objects not visible. The heart of the algorithm is a hierarchy of occlusion maps, which records the aggregate projection of occluders onto the image plane at different resolutions. We use occlusion maps because they can be built quickly and have several unique properties (described later in the paper). The use of occlusion maps reflects a decomposition of the visibility problem into two sub-problems: a two-dimensional overlap test and a depth test. The former decides whether the screen space projection of the potential occludee lies completely within the screen space projection of the union of all occluders. The latter determines whether or not the potential occludee is behind the occluders. We use occlusion maps for the overlap tests, and a *depth estimation buffer* for the conservative depth test. In the conventional Z-buffer algorithm (as well as in the hierarchical Z-buffer algorithm), the overlap test is implicitly performed as a side effect of the depth comparison by initializing the Z-buffer with large numbers.

The algorithm renders the occluders at each frame and builds a hierarchy (pyramid) of occlusion maps. In addition to the model database, the algorithm maintains a separate *occluder database*, which is derived from the model database as a preprocessing step. Both databases are represented as bounding volume hierarchies. The rendering pipeline with our algorithm incorporated is illustrated in Figure 1. The shaded blocks indicate new stages introduced due to our algorithm. For each frame, the pipeline executes in

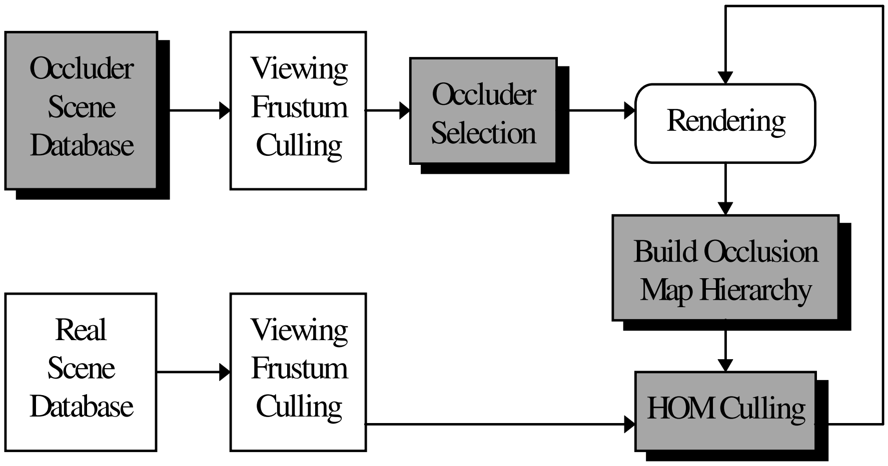

Figure 2: Modified graphics pipeline showing our algorithm. The shaded blocks indicate components unique to culling with hierarchical occlusion map.

two major phases:

1. Construction of the Occlusion Map Hierarchy: The occluders are selected from the occluder database and rendered to build the occlusion map hierarchy. This involves:

- **View-frustum culling:** The algorithm traverses the bounding volume hierarchy of the occluder database to find occluders lying in the viewing frustum.
- **Occluder selection:** The algorithm selects a subset of the occluders lying in the viewing frustum. It utilizes temporal coherence between successive frames.
- **Occluder rendering and depth estimation:** The selected occluders are rendered with special shading parameters to form an image in the framebuffer which is the highest resolution occlusion map. A depth estimation buffer is built to record the depth of the occluders.
- **Building the Hierarchical Occlusion Maps:** After occluders are rendered, the algorithm recursively filters the rendered image down by averaging blocks of pixels. This process can be accelerated by texture mapping hardware on many current graphics systems.

2. Visibility Culling with Hierarchical Occlusion Maps: Given an occlusion map hierarchy, the algorithm traverses the bounding volume hierarchy of the model database to perform visibility culling. The main components of this stage are:

- **View-frustum Culling:** The algorithm applies standard view-frustum culling to the model database.

- **Depth Comparison:** For each potential occludee, the algorithm conservatively checks whether it is behind the occluders.
- **Overlap test with Occlusion Maps:** The algorithm traverses the occlusion map hierarchy to conservatively decide if each potential occludee's screen space projection falls completely within the opaque areas of the maps.

Only objects that fail one of the latter two tests (depth or overlap) are rendered.

Our approach shares a number of characteristics with earlier work. This includes dynamically selecting the occluders, as in [CT96b, CT96a] and [Hu96], and using a combination of object space and image space hierarchy, as in [GKM93] and [Gre96]. Differences between their approaches and our approach are highlighted in Section 7.

4 Occlusion Maps

In this section, we present occlusion maps, algorithms using texture mapping hardware for fast construction of the hierarchy of occlusion maps, and state a number of properties of occlusion maps which are used by the visibility culling algorithm.

When an opaque object is projected to the screen, the area of its projection is made opaque. The *opacity* of a block on the screen is defined as the ratio of the sum of the opaque areas in the block to the total area of the block. An *occlusion* map is a two-dimensional array in which each pixel records the opacity of a rectangular block of screen space. Any rendered image can have an accompanying occlusion map which has the same resolution and stores the opacity for each pixel. In such a case, the occlusion map is essentially the α channel [FDHF90] of the rendered image (assuming α values for objects are set properly during rendering), though generally speaking a pixel in the occlusion map can correspond to a block of pixels in the screen space. Our hierarchical occlusion maps have a number of similarities to and differences from coverage pyramids [Gre96], as discussed in Section 7.3.

4.1 Image Pyramid

Given the lowest level occlusion map, the algorithm constructs from it a hierarchy of occlusion maps (HOM) by recursively applying the average operator to rectangular blocks of pixels. This operation forms an *image pyramid* as shown in Figure 3. The resulting hierarchy represents the occlusion map at multiple resolutions. It greatly accelerates the overlap test and is used for approximate culling. In the rest of the paper, we follow the convention that the *highest* resolution occlusion map of a hierarchy is at *level* 0.

The algorithm first renders the occluders into an image, which forms the lowest-level and highest resolution occlusion map. This image represents an *image-space fusion* of all occluders in the object space. The occlusion map hierarchy is built by recursively filtering from the highest-resolution map down to some minimal resolution (e.g. 4×4). The highest resolution need not match that of the image of the model database. Using a lower image resolution for rendering occluders may lead to inaccuracy for occlusion culling near the edges of the objects, but it speeds up the time for constructing the

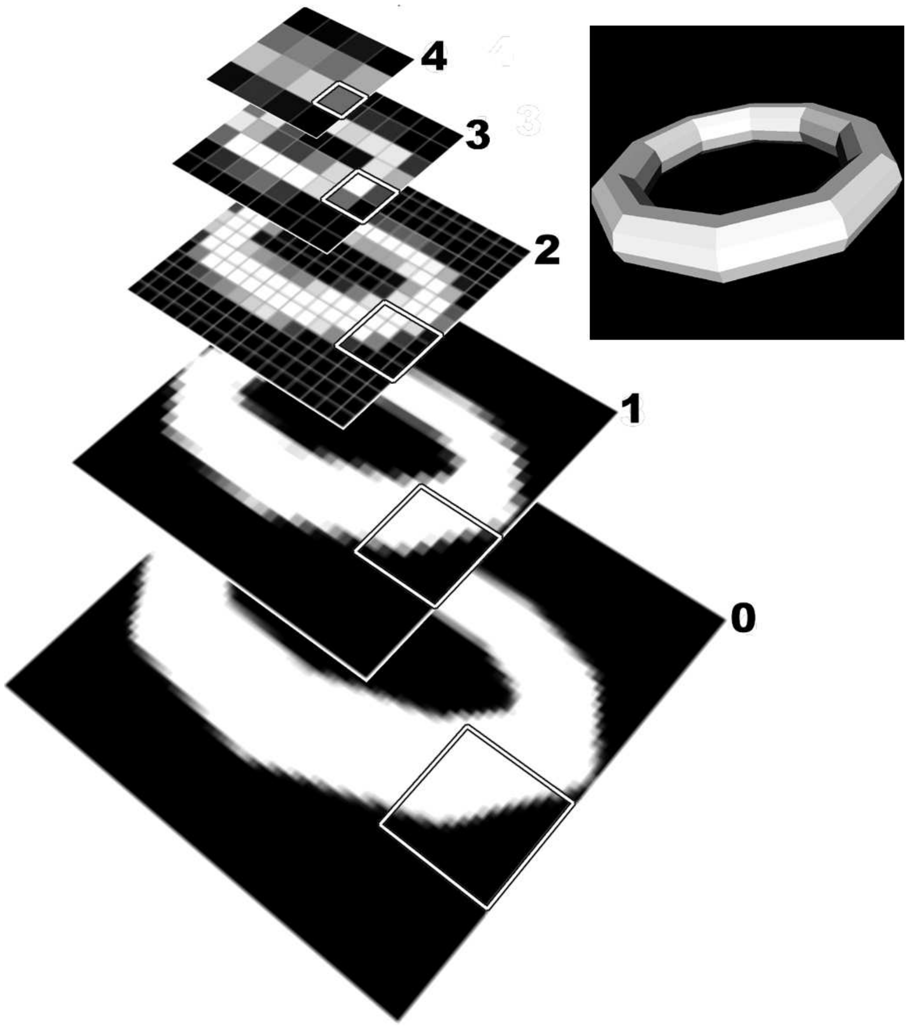

Figure 3: The hierarchy of occlusion maps. This particular hierarchy is created by recursively averaging over 2 blocks of pixels. The outlined square marks the correspondence of one top-level pixel to pixels in the other levels. The image also shows the rendering of the torus to which the hierarchy corresponds.

4.2 Fast Construction of the Hierarchy

When filtering is performed on 2×2 blocks of pixels, hierarchy construction can be accelerated by graphics hardware that supports bilinear interpolation of texture maps. The averaging operator for 2×2 blocks is actually a special case of *bilinear interpolation*. More precisely, the bilinear interpolation of four scalars or vectors $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ is:

$$(1-\alpha)(1-\beta)\mathbf{v}_0 + \alpha(1-\beta)\mathbf{v}_1 + \alpha\beta\mathbf{v}_2 + (1-\alpha)\beta\mathbf{v}_3,$$

where $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$ are the weights. In our case, we use $\alpha = \beta = 0.5$ and this formula produces the average of the four values. Thus, by carefully setting the texture coordinates, we can filter an $2N \times 2N$ occlusion map to $N \times N$ by drawing a two dimensional rectangle of size $N \times N$, texturing it with the $2N \times 2N$ occlusion map, and reading back the rendered image as the $N \times N$ occlusion map. Figure 4 illustrates this process.

The graphics hardware typically needs some setup time for the required operations. When the size of the map to be filtered is relatively small, setup time may dominate the computation. In such cases, the use of texture mapping hardware may slow down the computation of occlusion maps rather than accelerate it and hierarchy building is faster on the host CPU. The break-even point between hardware and software hierarchy construction varies with different graphics systems.

[BM96] have presented a technique for generating mipmaps by using a hardware accumulation buffer. We did not use this method because the accumulation buffer is less commonly supported in current graphics systems than texture mapping.

4.3 Properties of Occlusion Maps

The hierarchical occlusion maps for an occluder set have several desirable properties for accelerating visibility culling. The visibility culling algorithm presented in Section 5 utilizes these properties.

1. Occluder Fusion: Occlusion maps represent the fusion of small and possibly disjoint occluders. No assumptions are made on the shape, size, or geometry of the occluders. Any object that is renderable can serve as an occluder.

2. Hierarchical Overlap Test: The hierarchy allows us to perform a fast overlap test in screen space for visibility culling. This test is described in more detail in Section 5.1.

3. High-level Opacity Estimation: The opacity values in a low-resolution occlusion map can give an estimate of the opacity values in higher-resolution maps. For instance, if a pixel in a higher level map has a very low intensity value, it implies that almost all of its descendant pixels have low opacities, i.e. there is a low possibility of occlusion. This is due to the fact that occlusion maps are based on the average operator rather than the minimum or maximum operators. This property allows for a conservative early termination of the overlap test.

4. Approximate Visibility Culling: The hierarchy provides a natural method for approximate occlusion culling. It may be used to cull away portions of the model visible only through small gaps in or among occluders. A high opacity value of a pixel in a low resolution map implies that most of its descendant pixels are opaque. The algorithm uses the *opacity threshold* parameter to control the degree of approximation. More details are given in Section 5.4.

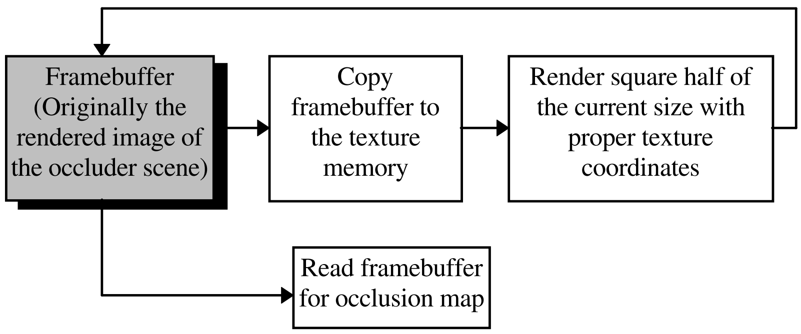

Figure 4: Use of texture-mapping hardware to build occlusion maps

5 Visibility Culling with Hierarchical Occlusion Maps

An overview of the visibility culling algorithm has been presented in Section 3. In this section, we present detailed algorithms for overlap tests with occlusion maps, depth comparison, and approximate culling.

5.1 Overlap Test with Occlusion Maps

The two-dimensional overlap test of a potential occludee against the union of occluders is performed by checking the opacity of the pixels it covers in the occlusion maps. An exact overlap test would require a scan-conversion of the potential occludee to find out which pixels it touches, which is relatively expensive to do in the software. Rather, we present a simple, efficient, and *conservative* solution for the overlap test.

For each object in the viewing frustum, the algorithm conservatively approximates its projection with a screen-space bounding rectangle of its bounding box. This rectangle covers a superset of the pixels covered by the actual object. The extremal values of the bounding rectangle are computed by projecting the corners of the bounding box. The main advantage of using the bounding rectangle is the reduced cost of finding the pixels covered by a rectangle compared to scan-converting general polygons.

The algorithm uses the occlusion map hierarchy to accelerate the overlap test. It begins the test at the level of the hierarchy where the size of a pixel in the occlusion map is approximately the same size as the bounding rectangle. The algorithm examines each pixel in this map that overlaps the bounding rectangle. If any of the overlapping pixels is not completely opaque [1], the algorithm recursively descends from that pixel to the next level of the hierarchy and checks all of its sub-pixels that are covered by the bounding rectangle. If all the pixels checked are completely opaque, the algorithm concludes that the occludee's projection is completely inside that of the occluders. If not, the algorithm conservatively concludes that the occludee may not be completely obscured by the occluders, and it is rendered.

[1] By definition, a pixel is completely opaque if its value is above or equal to the *opacity threshold*, which is defined in Section 5.4.

The algorithm supports *early termination* in overlap tests. If the opacity of a pixel in a low-resolution map is too small, there is small probability that we can find high opacity values even if we descend into the sub-pixels. So the overlap test stops and concludes that the object is not occluded. The *transparency thresholds* are used to define lower-bounds on opacity below which traversal of the hierarchy is terminated.

5.2 Depth Comparison

Occlusion maps do not contain depth information. They provide a necessary condition for occlusion in terms of overlap tests in the image plane, but do not detect whether an object is in front of or behind the occluders. The algorithm manages the depth information separately to complete the visibility test. In this section, we propose two algorithms for depth comparison.

5.2.1 Single Z plane

One of the simplest ways to manage the depth is to use a single Z plane. It is based on defining a plane parallel to and beyond the near plane . This plane separates the occluders from the potential occludees so that any object lying beyond the plane is farther away than any occluder. As a result, an object which is contained within the projection of the occluders and lies beyond the parallel plane is completely occluded. This is an extremely simple and conservative method which gives a rather coarse bound for the depth values of all occluders.

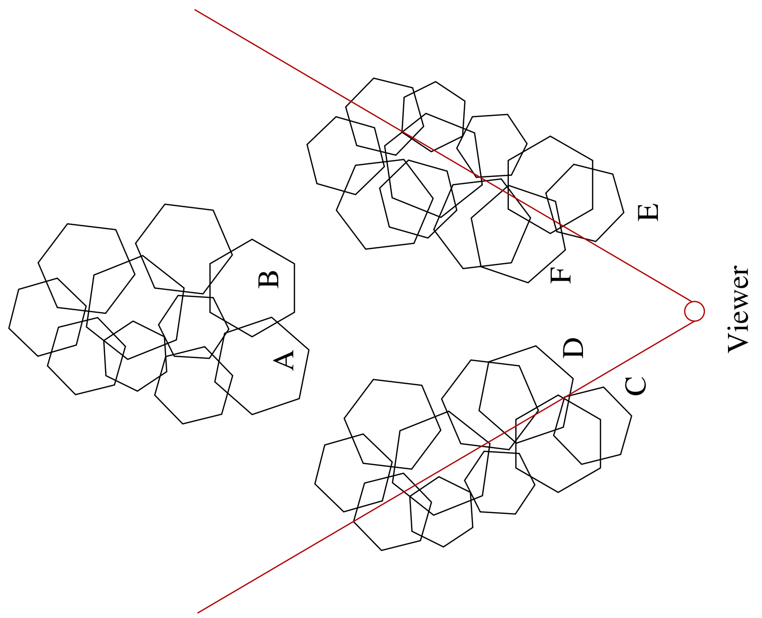

Figure 5: Distance criterion for dynamic selection

5.2.2 Depth Estimation Buffer

The depth estimation buffer is a software buffer that provides a more general solution for conservatively estimating the depth of occluders. Rather than using a single plane to capture the depth of the entire set of occluders, the algorithm partitions the screen-space and uses a separate plane for each region of the partition. By using a separate depth for each region of the partition, the algorithm obtains a finer measure of the distances to the occluders. The depth estimation buffer essentially is general-purpose software Z buffer that records the farthest distances instead of the nearest.

An alternative to using the depth estimation buffer might be to read the accurate depth values back from a hardware Z buffer after rendering the occluders. This approach was not taken mainly because it involves further assumptions of hardware features (i.e. there is a hardware Z-buffer, and we are able read Z-values reasonably fast in a easily-usable format).

Construction of the depth estimation buffer: The depth estimation buffer is built at every frame, which requires determining the pixels to which the occluders project on the image plane. Scan-converting the occluders to do this would be unacceptably expensive. As we did in constructing occlusion maps, we conservatively estimate the projection and depth of an occluder by its screen-space bounding rectangle and the Z value of its bounding volume's farthest vertex. The algorithm checks each buffer entry covered by the rectangle for possible updates. If the rectangle's Z value is greater than the old entry, the entry is updated. This process is repeated for all occluders.

Conservative Depth Test: To perform the conservative depth test on a potential occludee, it is approximated by the screen space bounding rectangle of its bounding box (in the same manner as in overlap tests), which is assigned a depth value the same as that of the nearest vertex on the bounding box. Each entry of the depth estimation buffer covered by the rectangle is checked to see if any entry is greater than the rectangle's Z value. If this is the case then the object is conservatively regarded as being partly in front of the union of all occluders and thus must be rendered.

The cost of the conservative Z-buffer test and update, though far cheaper than accurate operations, can still be expensive as the resolution of the depth estimation buffer increases. Furthermore, since we are performing a conservative estimation of the objects' scree space extents, there is a point where increasing resolution of the depth estimation buffer does not help increase the accuracy of depth information. Normally the algorithm uses only coarse-grain resolution (e.g. 64×64).

5.3 Occluder Selection

At each frame, the algorithm selects an occluder set. The *optimal* set of occluders is exactly the visible portion of the model. Finding this optimal set is the visible surface computation problem itself. Another possibility is to pre-compute global visibility information for computing the useful occluders at every viewpoint. The fastest known algorithm for computing the effects on global visibility due to a single polyhedron with m vertices can take $O(m^6 \log m)$ time in the worst case [GCS91].

We present algorithms to estimate a set of occluders that are used to cull a significant

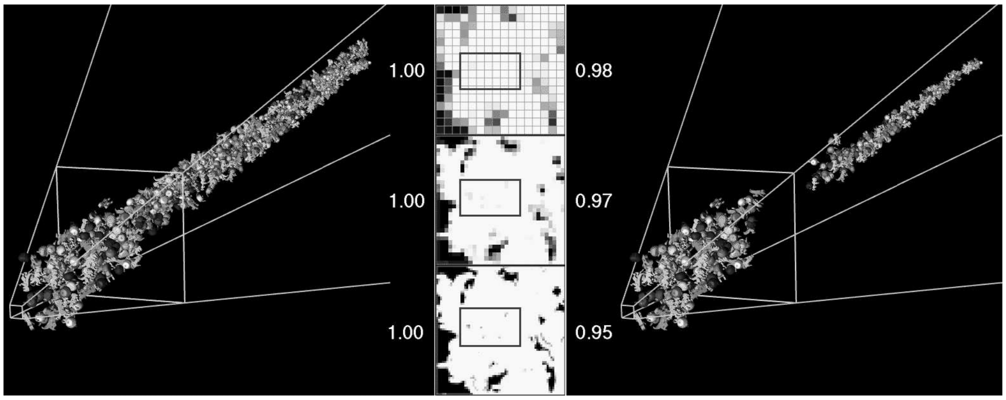

Figure 6: Demonstration of approximate culling. The blue rectangles highlight the holes in the occlusion maps. The holes dissolve as we go up in the hierarchy. The numbers on the left and right hand sides of the occlusion maps are the opacity thresholds used by the occlusion culling algorithm to generate the left and right images, respectively. In the right image, the algorithm culls away 50% more of the model.

fraction of the model. We perform preprocessing to derive an occluder database from the model. At runtime the algorithm dynamically selects a set of occluders from that database.

5.3.1 Building the Occluder Database

The goal of the pre-processing step is to discard objects which do not serve as good occluders from most viewpoints. We use the following criteria to select good occluders from the model database:

- **Size:** Small objects will not serve as good occluders unless the viewer is very close to them.
- **Redundancy:** Some objects, e.g. a clock on the wall, provide redundant occlusion and are removed from the database.
- **Rendering Complexity:** Objects with a high polygon count or rendering complexity are not preferred, as scan-converting them may take considerable time and it affects the overall frame rate.

5.3.2 Dynamic Selection

At runtime, the algorithm selects a set of objects from the occluder database. The algorithm uses a distance criterion, size, and temporal coherence to select occluders.

The single Z-plane method for depth comparison, presented in Section 5.2.1, is also an occluder selection method. All objects within the Z-plane are occluders.

When the algorithm uses the depth estimation buffer, it dynamically selects occluders based on a distance criterion and a limit ($\mathcal{L}$) on the number of occluder polygons. These two variables may vary between frames as a function of the overall frame rate and percentage of model culled. Given $\mathcal{L}$, the algorithm tries to find a set of good occluders whose total polygon count is less than $\mathcal{L}$.

The algorithm considers each object in the occluder database lying in the viewing frustum. The distance between the viewer and the center of an object's bounding volume is used as an estimate of the distance from the viewer to the object. The algorithm sorts these distances, and selects the nearest objects as occluder until their combined polygon count exceeds $\mathcal{L}$. This works well for most situations, except when a good occluder is relatively far away. One such situation has been shown in Figure 5. The distance criterion will select C, D, E, F, etc. as occluders, but $\mathcal{L}$ will probably be exceed *before* A and B are selected. Thus, we lose significant occlusion that would have been contributed by A and B. In other words, there is a hole in the occlusion map which decreases the culling rate.

We use a feedback mechanism to handle this problem, making use of temporal coherence. If the culling rate and frame rate from the previous frame are too low, we examine pixels in the lowest-resolution occlusion map (normally 4×4) for existence of major holes, i.e. pixel values that are lower than some threshold. Simple heuristics are used to combine such pixels into as few rectangular regions as possible. We next find occluders that project to these regions. A sub-frustum lying within the view frustum is formed by the region with the lowest average opacity and the viewing point. Occluders which intersect this frustum and have *not* been selected in the last frame are sorted by distance and the nearest ones are selected until a polygon count (some fraction of $\mathcal{L}$) is exceeded. After selecting these "hole-filling" occluders, the algorithm chooses the rest of the occluders by the standard occluder selection criteria. At the end of the frame, if we observed significant increase in culling rate over the last frame, we append the "hole-filling" occluders into a queue. Occluders in the queue are always selected. The queue is considered full if the total polygon count of the occluders in the queue exceeds some fixed limit. Attempts to add an object into a full queue causes the object at the head of the queue to be removed.

5.4 Approximate Visibility Culling

An additional feature of our algorithm is to perform approximate visibility culling, which ignores objects only visible through small holes in or among the occluders. This ability is based on an inherent property of HOM that it naturally represents the combined occluder projections at different levels of details.

In the process of filtering maps to build the hierarchy, a pixel in a low resolution map can obtain high opacity value even if a small number of its descendant pixels have low opacity. Intuitively, a small group of low-opacity pixels (a "hole") in a high-resolution map can *dissolve* as the average operation (which involves high opacity values from neighboring pixels) is recursively applied to build lower-resolution maps.

The opacity value above which the pixel is considered completely opaque is called the *opacity threshold*, which is by default 1.0. The visibility culling algorithm varies the degree of approximation by changing the opacity thresholds. As the threshold is lowered, the algorithm becomes more approximate. This effect of the opacity threshold is based on the fact that if a pixel is considered completely opaque, the culling algorithm does not go into the descendent pixels for further opacity checking. If the opacity a pixel in a low-resolution map is not 1.0 (because some of the pixel's descendents have low opacities), but is still higher than the opacity threshold assigned to that map, the culling algorithm does not descend to the sub-pixels to find low opacities. In effect this means that some small holes in higher-resolution maps are ignored.

Approximate visibility is useful because we don't expect to see many meaningful parts of the model through small holes in or among the occluders. In practice, culling such portions of the models does not create noticeable visual artifacts. Omitting such holes can significantly increase the culling rate. To have similar degrees of approximations at different levels of maps, the opacity threshold should go up as map resolution goes up. In Figure 6, we show the impact of varying the opacity threshold on an environment with a "forest" of small occluders.

5.5 Dynamic Environments

The algorithm easily extends to dynamic environments. As no static bounding volume hierarchy may be available, the algorithm uses oriented bounding boxes around each object. The occluder selection algorithm involves no pre-processing, so the occluder database is exactly the model database. The oriented bounding boxes are used to construct the depth estimation buffer as well as to perform the overlap test with the occlusion map hierarchy.

6 Implementation and Performance

We have implemented the algorithm as part of a walkthrough system, which is based on OpenGL and currently runs on SGI platforms. Significant speed-ups in frame rates have been observed on different models. In this section, we discuss several implementation issues and discuss its performance on SGI Max Impacts and Infinite Reality platforms.

6.1 Implementation

As the first step in creating the occlusion map hierarchy, occluders are rendered in a 256×256 viewport in the back framebuffer, in full white color with lighting and texture mapping turned off. Anyone of the three color channels of the resulting image can serve as the highest-resolution occlusion map on which the hierarchy is based. An alternate method could be rendering the occluders with the original color and shading parameters and use the α channel of the rendered image to construct the initial map. However, for constructing occlusion maps we do not need a "realistic" rendering of the occluders, which may be more expensive. In most cases the resolution of 256×256 is smaller than that of the final rendering of the model. As a result, it is possible to have artifacts in occlusion. In practice, if the final image is rendered at a resolution of 1024×1024, rendering occluders at 256×256 is a good trade-off between accuracy and time required to filter down the image in building the hierarchy.

To construct the occlusion map hierarchy, we recursively average 2×2 blocks of pixels using the texture mapping hardware as well as the host CPU. The resolution of the lowest-resolution map is typically 4×4. The break-even point between hardware and software hierarchy construction (as described in Section 4.2) varies with different graphics systems. For SGI Max Impacts, we observed the shortest construction time when the algorithm filters from 256 to 128×128 using texture-mapping hardware, and from 128×128 to 64×64 and finally down to 4×4 on the host CPU. For Infinite Reality, which has faster pixel transfer rates, the best performance is obtained by filtering from 256×256 to 64×64 using the hardware and using the host CPU thereafter. Hierarchy construction time is about 9 milliseconds for the Max Impacts and 4 milliseconds for the Infinite Reality, with a small variance (around 0.5 milliseconds) between frames.

The implementation of depth estimation buffer is optimized for block-oriented query and updates. The hierarchical overlap test is straight-forward to implement. It is relatively harder to optimize as it is recursive in nature.

6.2 Performance

We demonstrate the performance of the model on three environments. These are:

- **City Model:** It is composed of different models and has 312,524 polygons. A bird's eye view of the model has been shown in Figure 9.
- **Dynamic Environment:** It is composed of dinosaurs and teapots each undergoing independent random motion. The total polygon count is 986,800. It has been shown in Figure 6 and Fig. 10.
- **Submarine Auxiliary Machine Room (AMR):** It is a *real-world* CAD model obtained from industrial sources. The model has 632,252 polygons. Different views of the model are shown in Figure 1 and Figure 11.

As mentioned earlier, our algorithm uses a bounding volume hierarchy (i.e. a scene graph) for both the original model database as well as the occluder database. Each model we used is originally a collection of polygons with no structure information. We construct an axis-aligned bounding box hierarchy for each database.

For the dynamic environment and the city model, we use the model database itself as the occluder database, without any pre-processing for static occluder selection. For the AMR model, the pre-processing yields an occluder database of 217,636 polygons. The algorithm removes many objects that has little potential of being a good occluder (like the bolts on the diesel engine, thin pipes etc.) from the original model. Further, most of these parts are densely tessellated, making them to expensive to be directly used as occluders. We use the simplified version of the parts which are produced by algorithms in [Cohen96]. Although many algorithms of this kind give good error bounds on the simplified model, they do not guarantee that the projection of the simplified object lies within that of the original. Therefore, visibility artifacts may be introduced by the simplified occluders. We use very tight error bounds so that so that artifacts are rarely noticeable.

The performance of the algorithms has been highlighted in Figure 7. The graphs on the left show the frame rate improvement, while the graphs on the right highlights the percentage of the model culled at every frame. The performance of the city model was generated on the Max Impact while the other two were rendered on the Infinite Reality. The actual performance varies due to two reasons:

1. Different models have varying depth complexities. Furthermore, the percentage of occlusion varies with the viewpoint.

2. The ability of the occluder selection algorithm to select the "right" subset of occluder. The performance of the greedy algorithm, e.g. distance based criterion, varies with the model distribution and the viewpoint.

The occluder polygon count budget ($\mathcal{L}$) per frame is important for the performance of the overall algorithm. If too few occluders are rendered, most of the pixels in the occlusion map have low opacities and the algorithm is not able to cull much. On the other hand, if too many occluder polygons are rendered, they may take a significant percentage of the total frame time and slow down the rendering algorithm. The algorithm starts with an initial guess on the polygon count and adaptively modifies it based on the percentage of the model culled and frame rate. If the percentage of the model culled is low, it increases the count. If the percentage is high and the frame rate is low, it decreases the count.

Average time spent in different stages of the algorithm, occluder selection and rendering, hierarchy generation, occlusion culling and final rendering, have been shown in Figure 8. The average time to render the model without occlusion culling is normalized to 100%. In these cases, the average time in occluder rendering varies between $10-25\%$.

7 Analysis and Comparison

In this section we analyze some of the main features of our algorithm and compare it with other approaches.

Our algorithm is generally applicable to all models and obtains significant culling when there is high depth complexity. This is mainly due to its use of occlusion maps to combine the occluders in the image space. The extensive use of screen space bounding rectangles as an approximation of the object's screen space projection makes the overlap tests and depth tests fast and cheap.

In terms of hardware assumptions, the algorithm requires only the ability to read back the framebuffer. Texture mapping with bilinear interpolations, when available, can be directly used to accelerate the construction of the occlusion map hierarchy.

In general, if the algorithm is spending a certain percentage of the total frame time in occluder rendering, HOM generation and culling (depth test and overlap test), it should at least cull away a similar percentage of the model so as to justify the overhead of occlusion culling. If a model under some the viewing conditions does *not* have sufficient occlusion, the overall frame rate may decrease due to the overhead, in which case occlusion culling should be turned off.

7.1 Comparison to Object Space Algorithms

Work on cells and portals[ARB90, TS91, LG95] addresses a special class of densely occluded environments where there are plenty of cell and portal structures, as in an in-door architectural model. [ARB90, TS91] pre-precesses the model to identify potentially visible set of primitives for each cell. [LG95] developed a dynamic version which eliminates the pre-processing. These methods works very well for the particular environment, but are not applicable to models without cell/portal structures.

Our algorithm works without modification for environments with cells and portals, but occluder selection can be optimized for these environments. The cell boundaries can be used to form the occluder database. As an alternative, we can fill a viewport with white pixels and then render the portals in black to form the occlusion map. In general however, we do not expect to outperform the specialized algorithms in cell/port environments.

Two different object space solutions for more general models have been proposed by [CT96b, CT96a] and [Hu96]. They dynamically choose polygons and convex objects (or simple convex combination of polygons) as occluders and use them to cull away invisible portions of the model. However, many models do not have single big convex occluders. In such cases, merging small, irregular occluders is critical for significant culling, which is an difficult task in the object space. Our algorithm lies between the object space and the image space and the occluder merging problem is solved in image space.

7.2 Comparison with Hierarchical Z-buffer Algorithm

In many ways, we present an *alternative* approach to hierarchical Z-buffer visibility [GKM93]. The main algorithm presented in [GKM93] performs updates of the Z-buffer hierarchy as geometry is rendered. It assumes special-purpose hardware for fast depth updating and querying to obtain interactive performance. In terms of potential, we believe it is perhaps the most powerful and effective algorithm for visibility culling. However, we are not aware of any hardware implementation.

There is a possible variation of hierarchical Z-buffer algorithm which selects occluders, renders them, reads back the depth buffer once per frame, builds the Z-pyramid, and use the screen-space bounding boxes for fast culling. (The algorithm proposed in [GKM93] uses the exact projection of octree nodes, which requires software scan-conversion.) In this case, the main difference between our approach and the possible variation of hierarchical Z-buffer reduces to that between the properties of the occlusion map hierarchy and the Z pyramid.

The major advantage of the Z hierarchy is that it has the depth values, which we need to manage separately for hierarchical occlusion maps (Section 5.2). On the other hand, the occlusion map hierarchy has several unique features. These are based on the fact HOM use an average operator, whereas the Z hierarchy use a minimum or maximum operator.

1. The construction of HOM has readily-available hardware support on many graphics systems. Further, if filtering is performed in software, cost of the average operator is smaller than the minimum/maximum operator (due to no branching instructions).

2. HOM supports early termination in the hierarchical test by using a transparency threshold (Section 5.1) and approximate occlusion culling at different resolutions by using an opacity threshold (Section 5.4).

3. A very low opacity pixel in the higher-level (low-resolution) occlusion map implies that there is insufficient occlusion in the corresponding lower-level map. Not only is this information used for early termination of the hierarchical test, but also used to adaptively adjust the selection criterion used in the occluder selection algorithm.

7.3 Comparison with Hierarchical Tiling with Coverage Masks

Hierarchical polygon tiling [Gre96] tiles polygons in front-to-back order and uses a "coverage" pyramid for visibility culling. The coverage pyramid and hierarchical occlusion maps serve the same purpose in that they both record the aggregate projections of objects. (In this sense, our method has more resemblance to hierarchical tiling than hierarchical Z-buffer.) However, a pixel in a mask in the coverage pyramid has only three values (covered, vacant or active), while a pixel in an occlusion map has a continuous opacity value. This has lead to desirable features, as discussed above. Like HOM, the coverage masks do not contain depth information and the algorithm in [Gre96] uses a BSP-tree for depth-ordering of polygons. Our algorithm has no restriction in terms of rendering the polygons front to back. Rather it only needs a conservatively estimated boundary between the occluders and potential occludees, which is represented by the depth estimation buffer. Hierarchical tiling is tightly coupled with polygon scan-conversion and has to be significantly modified to deal with non-polygonal objects, such as curved surfaces or textured billboards. Our algorithm does not directly deal with low-level rendering but utilizes existing graphics systems. Thus it is readily applicable to different types of objects so long as the graphics system can render them. Hierarchical tiling requires special-purpose hardware for real-time performance.

8 Future Work and Conclusion

In this paper we have presented a visibility culling algorithm for general models that achieves significant speedups for interactive walkthroughs on current graphics systems. It is based on hierarchical occlusion maps, which represent an image space fusion of all the occluders. The overall algorithm is relatively simple, robust and easy to implement. We have demonstrated its performance on a number of large models.

There are still several areas to be explored in this research. We believe the most important of these to be occlusion preserving simplification algorithms, integration with levels-of-detail modeling, and parallelization.

Occlusion Preserving Simplification: Many models are densely tessellated. For fast generation of occlusion maps, we do not want to spend considerable time in rendering the occluders. As a result, we are interested in simplifying objects under the constraint of occlusion preservation. This implies that the screen space projection of the simplified object should be a subset of that of the original object. Current polygon simplification algorithms can reduce the polygon count while giving tight error bounds, but none of them guarantees an occlusion preserving simplification.

Integration with Level-of-Detail Modeling: To display large models at interactive frame rates, our visibility culling algorithm needs to be integrated with level-of-detail modeling. The latter involves polygon simplification, texture-based simplification and dynamic tessellation of higher order primitives.

Parallelization: Our algorithm can be easily parallelized on multi-processor configurations. Different processors can be used for view frustum culling, overlap tests and depth tests. These tasks take a small fraction of the frame time for our current applications. However, they can take significant time on even larger models.

9 Acknowledgements

We are grateful to Fred Brooks, Jon Cohen, Anselmo Lastra, Ming Lin, Turner Whitted and members of UNC Walkthrough project for productive discussions. The auxiliary machine room model was provided by Greg Angelini, Jim Boudreaux, and Ken Fast at Electric Boat, a subsidiary of General Dynamics.

References

[ARB90] J. Airey, J. Rohlf, and F. Brooks. Towards image realism with interactive update rates in complex virtual building environments. In *Symposium on Interactive 3D Graphics*, pages 41–50, 1990.

[BM96] D. Blythe and T. McReynolds Programming with OpenGL: Advanced *Siggraph'96 course notes*, 1996.

[Br96] R. Brechner et al. Interactive walkthrough of large geometric databases. *Siggraph'96 course notes*, 1996.

[Car84] L. Carpenter. The A-buffer, an antialiased hidden surface method. *Proc. of ACM Siggraph*, pages 103–108, 1984.

[Cat74] E. Catmull. *A subdivision algorithm for computer display of curved surfaces.* PhD thesis, University of Utah, 1974.

[Cla76] J.H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[Co96] J. Cohen et al. Simplification envelopes. In *Proc. of ACM Siggraph'96*, pages 119–128, 1996.

[CT96a] S. Coorg and S. Teller. A spatially and temproally coherent object space visibility algorithm. Technical Report TM 546, Laboratory for Computer Science, Massachusetts Institute of Technology, 1996.

[CT96b] S. Coorg and S. Teller. Temporally coherent conservative visibility. In *Proc. of 12th ACM Symposium on Computational Geometry*, 1996.

[Dor94] S. E. Dorward. A survey of object-space hidden surface removal. *Internat. J. Comput. Geom. Appl.*, 4:325–362, 1994.

[FDHF90] J. Foley, A. Van Dam, J. Hughes, and S. Feiner. *Computer Graphics: Principles and Practice.* Addison Wesley, Reading, Mass., 1990.

[FKN80] H. Fuchs, Z. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Proc. of ACM Siggraph*, 14(3):124–133, 1980.

[GBR91] R. Coifman G. Beylkin and V. Rokhlin. Fast wavelet transforms and numerical algorithms: I. *Communications of Pure and Applied Mathematics*, 44(2):141–183, 1991.

[GBW90] B. Garlick, D. Baum, and J. Winget. Interactive viewing of large geometric databases using multiprocessor graphics workstations. *Siggraph'90 course notes: Parallel Algorithms and Architectures for 3D Image Generation*, 1990.

[GCS91] Z. Gigus, J. Canny, and R. Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):542–551, 1991.

[GK94] N. Greene and M. Kass. Error-bounded antialiased rendering of complex environments. In *Proc. of ACM Siggraph*, pages 59–66, 1994.

[GKM93] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. In *Proc. of ACM Siggraph*, pages 231–238, 1993.

[Gre95] N. Greene. *Hierarchical Rendering of Complex Environments.* PhD thesis, University of California at Santa Cruz, 1995.

[Gre96] N. Greene. Hierarchical polygon tiling with coverage masks. In *Proc. of ACM Siggraph*, pages 65–74, 1996.

[Hu96] T. Hudson et al. Accelerated occlusion culling using shadow frusta. Technical Report TR96-052, Department of Computer Science, University of North Carolina, 1996.

[LG95] D. Luebke and C. Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In *ACM Interactive 3D Graphics Conference*, Monterey, CA, 1995.

[McK87] M. McKenna. Worst-case optimal hidden-surface removal. *ACM Trans. Graph.*, 6:19–28, 1987.

[Mul89] K. Mulmuley. An efficient algorithm for hidden surface removal. *Computer Graphics*, 23(3):379–388, 1989.

[Nay92] B. Naylor. Partitioning tree imge representation and generation from 3d geometric models. In *Proc. of Graphics Interface*, pages 201–12, 1992.

[RH94] J. Rohlf and J. Helman. Iris performer: A high performance multiprocessor toolkit for realtime 3d graphics. In *Proc. of ACM Siggraph*, pages 381–394, 1994.

[SBM+94] B. Schneider, P. Borrel, J. Menon, J. Mittleman, and J. Rossignac. Brush as a walkthrough system for architectural models. In *Fifth Eurographics Workshop on Rendering*, pages 389–399, July 1994.

[SDS96] E. Stollnitz, T. Derose, and D. Salesin. *Wavelets for Computer Graphics.* Morgan Kaufmann Publishers, 1996.

[SG96] O. Sudarsky and C. Gotsman. Output sensitive visibility algorithms for dynamic scenes with applications to virtual reality. *Computer Graphics Forum*, 15(3):249–58, 1996. Proc. of Eurographics'96.

[Tel92] S. J. Teller. *Visibility Computations in Densely Occluded Polyheral Environments.* PhD thesis, CS Division, UC Berkeley, 1992.

[TH93] S. Teller and P. Hanrahan. Global visibility algorithms for illumination computations. In *Proc. of ACM Siggraph*, pages 239–246, 1993.

[TP75] S. Tanimoto and T. Pavlidis. A hierarchical data structure for picture processing. *Computer Graphics and Image Processing*, 4(2):104–119, 1975.

[TS91] S. Teller and C.H. Sequin. Visibility preprocessing for interactive walkthroughs. In *Proc. of ACM Siggraph*, pages 61–69, 1991.

[War69] J. Warnock. A hidden-surface algorithm for computer generated half-tone pictures. Technical Report TR 4-15, NTIS AD-753 671, Department of Computer Science, University of Utah, 1969.

[Wil83] L. Williams. Pyramidal parametrics. *ACM Computer Graphics*, pages 1–11, 1983.

[YR96] R. Yagel and W. Ray. Visibility computations for efficient walkthrough of complex environments. *Presence*, 5(1):1–16, 1996.

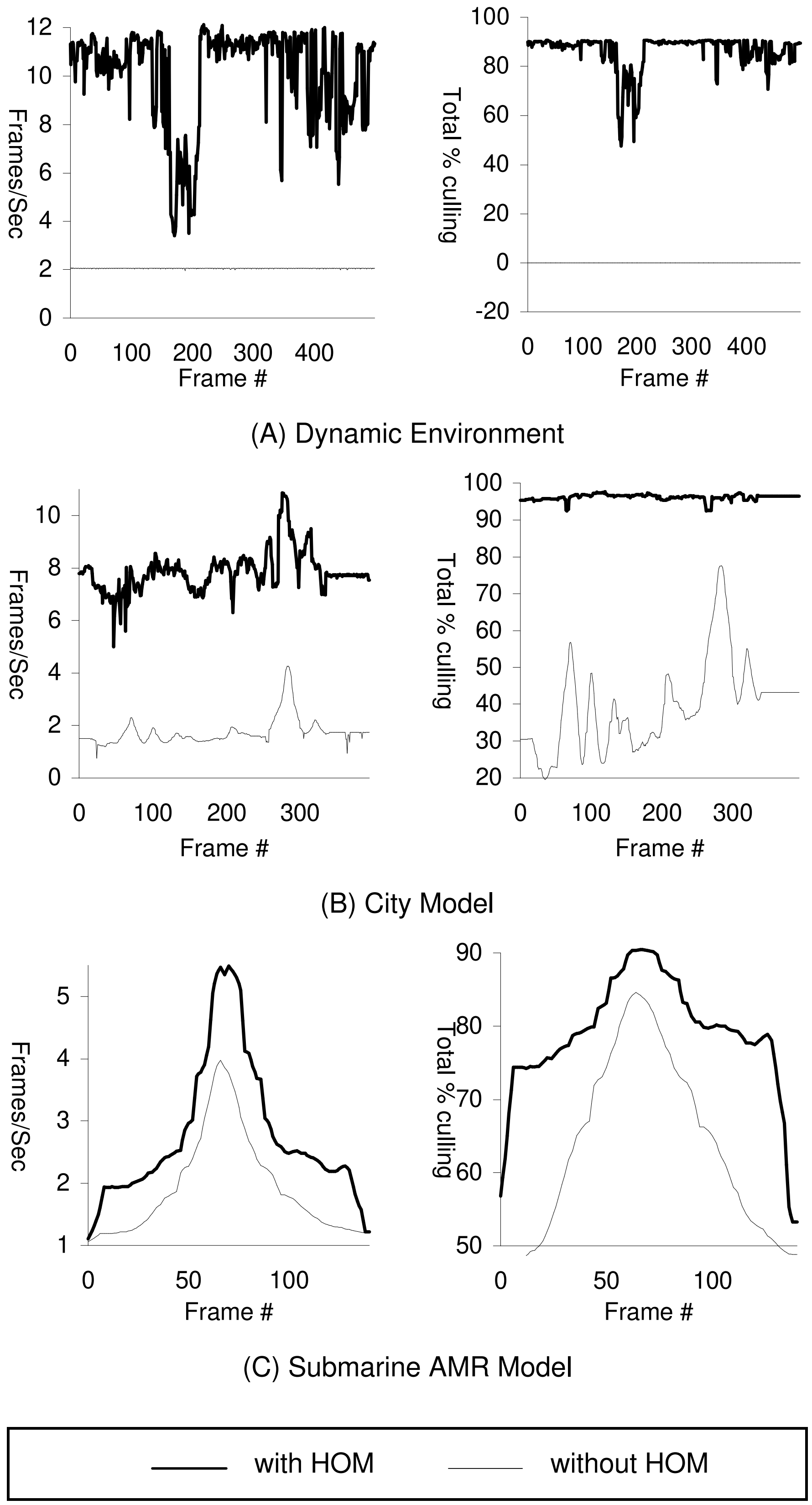

Figure 7: The speed-up obtained due to HOM on different models. The left graphs show the improvement in frame rate and the right graphs show the percentage of model culled. The statistics were gathered over a path for each model.

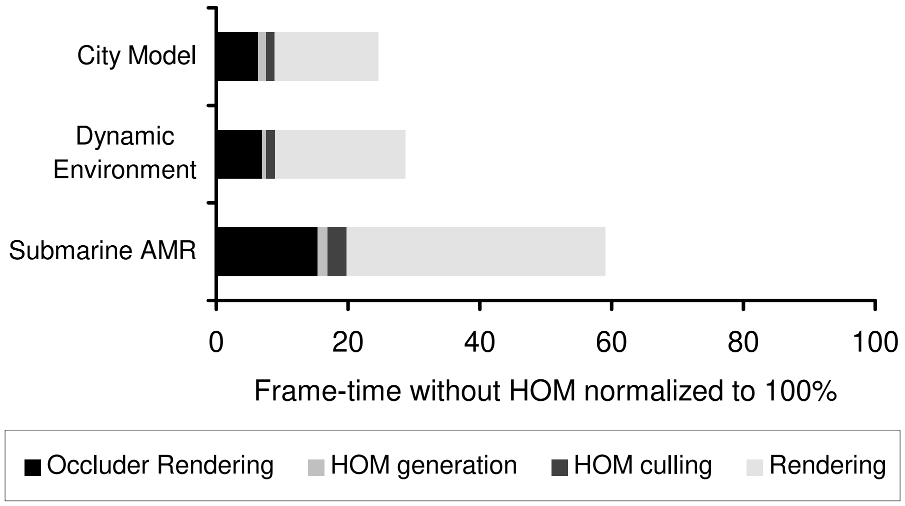

Figure 8: Average speed-up obtained due to HOM culling on different models. The total time to render each model without HOM culling is normalized to 100%. Each bar shows the percentage of time spent in different stages of our algorithm.

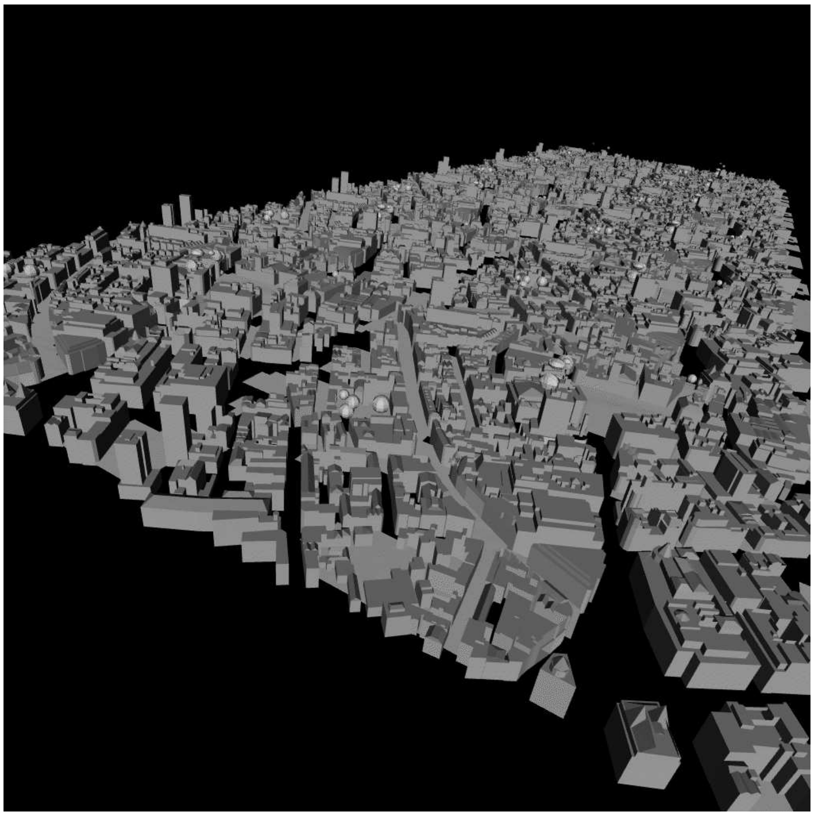

Figure 9: City model with 312,524 polygons. Average speed-up obtained by our visibility culling algorithm is about five.

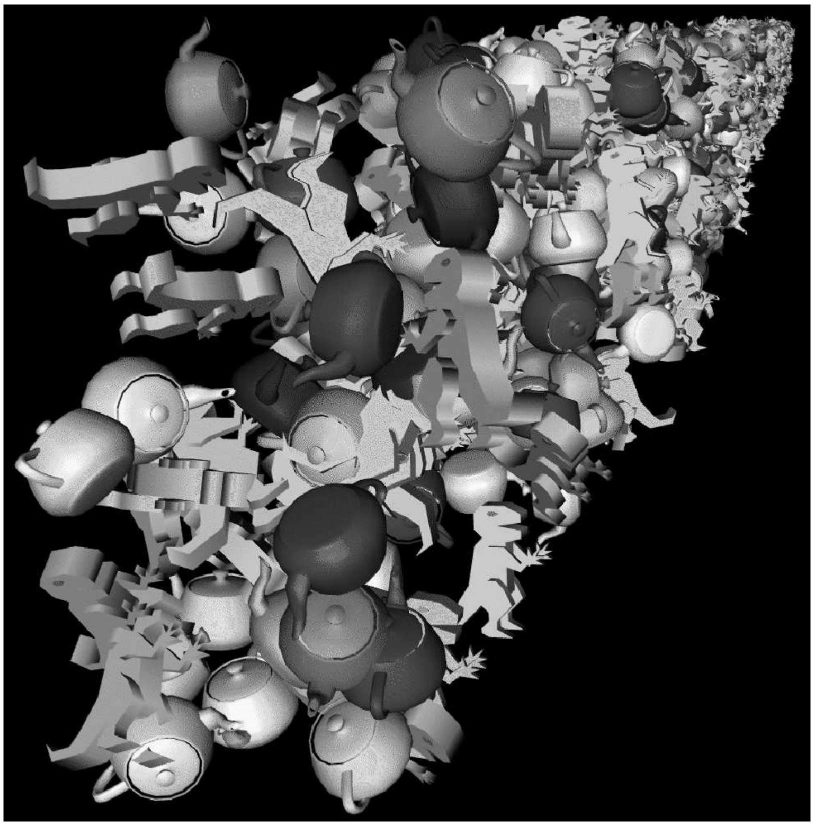

Figure 10: Dynamic environment composed of dinosaurs and teapots. The total polygon count is 986,800. The HOM algorithm achieves 4-5 times speed-up.

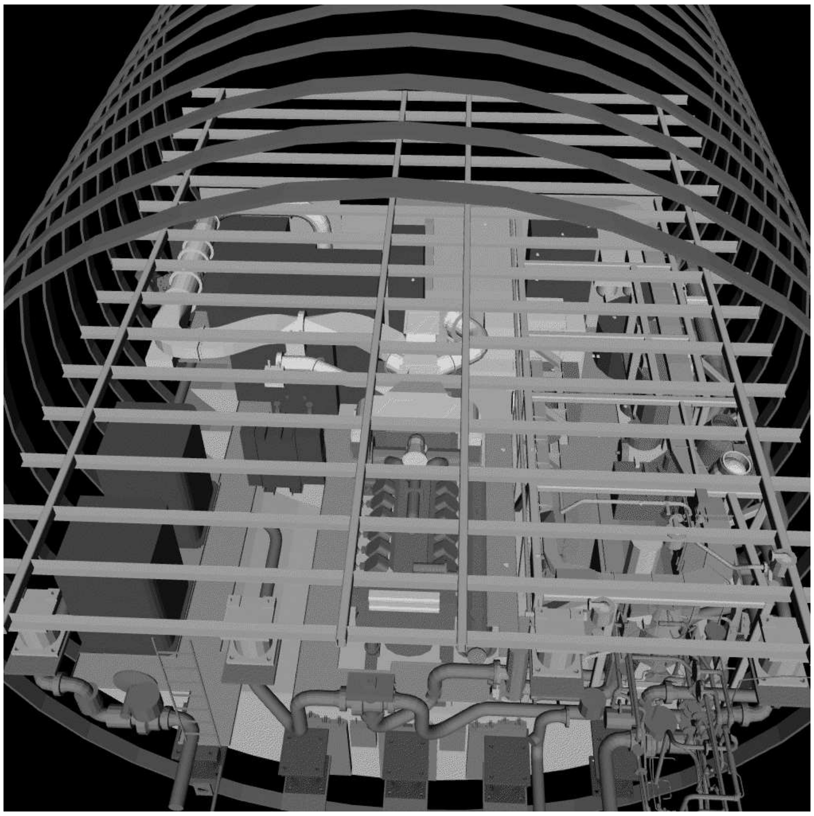

Figure 11: A top view of the auxiliary machine room of a submarine composed of 632,252 polygons. Average speed-up is about two due to occlusion culling.