Karsten Bormann
bormann@nbi.dk
Computer Graphics Group
Institute for Mathematical Modelling
Denmark Technical University
DK-2800 Lyngby, Denmark

Occlusion Culling in Large Virtual Environments

Abstract

In this paper, occlusion testing in very large virtual environments is discussed from two perspectives: a theoretical one, discussing occlusion culling in relation to detail elision (level of detail), and a practical one, relating to an adaptive occlusion-culling algorithm developed in this paper.

The theoretical perspective formally demonstrates the efficiency of detail elision in the face of real-world-like environments. It further shows the utility of using detail elision not only to lower the triangle count of individual objects but also to lower the load on the scene graph traversal algorithm. Finally, the results indicate that, even in open environments, one needs occlusion culling and that approaches based on choosing only a few good (convex) occluders are insufficient in this case.

Practically, the theoretical perspective is reflected in the development of an occlusion-culling algorithm based on a marriage of the frustum-slicing approach to view frustum culling and hierarchical occlusion maps (HOMs). The resulting algorithm is much simpler than the original HOM algorithm, requires little pre-processing, and integrates detail elision with the occlusion-culling algorithm in a natural way. The approach is well suited for use with large, complex models with a long mean free line of sight ("the great outdoors"), models for which it is not feasible to construct, or search, a database of occluders to be rendered in each frame. The algorithm is tested for such large models, and results show that frame rates tend to converge as the geometrical complexity of the model increases.

1 Introduction

Even as hardware ability continues to grow at a brisk pace, the complexity of the models that one attempts to render grows even faster. This growth explains the attempts to create output-sensitive visibility-culling algorithms, that is, visibility-culling algorithms whose runtime depends upon the amount of geometry actually visible rather than the complexity of the model. The two prime examples of such attempts are detail elision (multiresolution, level of detail) and occlusion culling (view frustum culling being taken for granted).

Detail elision is accomplished by providing multiple levels of detail (Clark, 1976), which may be continuously close, calculated at runtime (Lindstrom et al., 1996; Schmalstieg & Schaufler, 1997; Luebke & Erikson, 1997), and preferably chosen with iterative refinement possible (Howell, Chrysanthou, Steed, & Slater, 1999; Funkhouser & Sequin, 1993).

Detail elision vastly reduces the amount of geometry residing in distant ob-

Presence, *Vol. 10, No. 5, October 2001,* 477–494

jects. However, for highly complex models, the amount of geometry still overwhelms the hardware, and one of the ways to further reduce the amount of geometry is to avoid trying to render objects that are not seen in the final picture because some intermediate object(s) occludes the view.

Of course, the need for such occlusion culling is most obvious for models in which large depth complexity is paired with heavy occlusion, such as the inside of architectural models. Indeed, for architectural models, it is obvious to use the segregation of buildings into rooms, building a conceptual model consisting of cells (roughly corresponding to rooms) connected by portals (doors, windows, and so on). One can then proceed to determine cell-to-cell, eye-to-cell, and eye-to-object visibility to bound exact visibility tightly from above (Teller & Sequin, 1991; Luebke & Georges, 1995).

Cell portal-based algorithms fail to be useful to models without obvious cell structure, and so more general procedures are necessary. Because checking whether an object is occluded generally isn't cheap—for instance, Hudson et al. (1997) and Saona-Vazquez, Navazo, and Brunet (1999) construct a shadow frustum for each occluder against which geometry is culled before rendering—one often restricts the type of occluders to convex objects and severely limits the number of occluders that can be considered for use in a given frame. This in turn makes it important to find, at runtime, good occluders, which is generally a hard task implying a fair amount of preprocessing as well (Coorg & Teller, 1996, 1997; Hudson et al., 1997; Cohen-Or, Fibich, Halperin, & Zadicano, 1998; Saona-Vazquez et al., 1999).

The above algorithms consider occluders individually and consider only a few, good occluders. This may be sufficient for some applications; however, the optimal set of occluders is, in a sense, exactly the set of visible objects/geometry, which may be considerably larger. Using this observation and appealing to temporal coherence, the hierarchical *z*-buffer algorithm (Greene, Kass, & Miller, 1993) maintains a list of polygons that are visible from the current viewpoint. Then, in the following frame, these polygons are the first ones rendered, and subsequently one culls geometry against this initial content of the (hierarchical) *z*-buffer.

The hierarchical *z*-buffer algorithm has all geometry (at the level of individual polygons) placed in an octree. Starting at the root, the octree is traversed by considering child nodes in order of proximity to the viewpoint and visiting only those child nodes for which a check against the hierarchical *z*-buffer reveals them to be visible. The actual test is done by finding the lowest level in the *z*-pyramid where a single pixel covers the entire projection of the primitive or node and checking the *z*-value of this pixel conservatively against the depth of the node or primitive. If visible, the projection of the node or primitive is divided into four and checked against the next lower level in the *z*-pyramid, and so forth.

It should be noted that, although the occlusion-culling algorithms cited above are preoccupied with static models, this restriction can be somewhat alleviated, for instance by using temporal bounding volumes for dynamic objects (Sudarsky & Gotsman, 1996), as one can allow a certain indeterminacy in object positions as long as that does not change the picture rendered.

However, for increasingly interactive and collaborative virtual reality applications, it is important to further do away with the assumption that models be static. Also, as models grow, and generally need to be shared or linked between remote locations, it probably is important to use algorithms that reduce the need for preprocessing, a first step being algorithms that consider geometry at the object level rather than at the level of geometrical primitives.

The hierarchical occlusion map (HOM) is one such algorithm (Zhang, Manocha, Hudson, & Hoff, 1997). Like the hierarchical *z*-buffer, it essentially uses a mipmap of a rendering of occluders to perform hierarchical occlusion checks. (See figure 1.) However, the HOM algorithm does not use the previous frame image for creating the occlusion map, but instead selects a small (sub)set of occluders from a database of potentially good occluders, essentially choosing those that are both large and close to the viewpoint. These occluders are then rendered in monochrome, the corresponding depth values of the pixels conservatively estimated, and

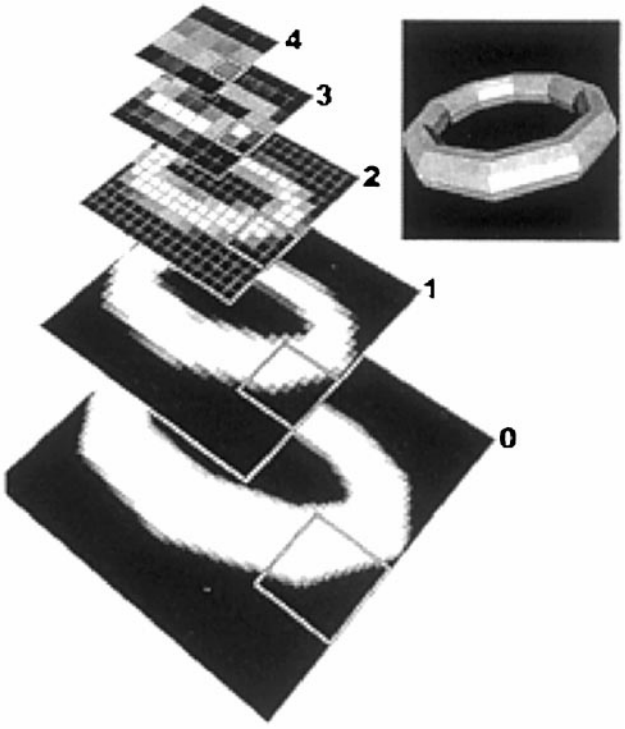

Figure 1. *Hierarchical representation of a torus, from (Zhang et al., 1997).*

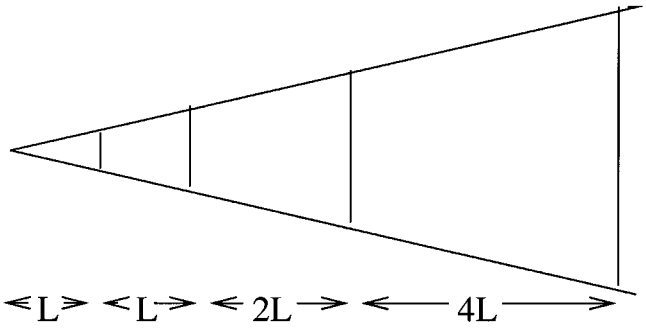

Figure 2. *Slicing the viewing frustum.*

the HOM constructed. To cull potential occludees against the HOM, the algorithm begins at the level of the hierarchical occlusion map at which one pixel is approximately the same size as the projected bounding box and examines each pixel that overlaps this projection. If any of the overlapping pixels are not completely opaque, the algorithm recursively descends to the next level of the map to check all the subpixels covered by the projection of the object's bounding box.

This approach elegantly solves, in image space, the problem of fusion of occluders and the usage of irregular occluders. Also, it allows for approximate visibility culling and for early termination if the degree of occlusion is too low.

It is, however, not integrated with detail elision (LOD), has not really done away with the assumption that the model is static (due to the heavy cost of the preprocessing step), and, for use with very large, general VEs (notably large outdoor environments), it clearly is insufficient to consider only a few nearby occluders because the amount of distant geometry is large, in turn leading distant occluders to be of significant value.

The algorithm in section 2 attempts to address these issues by combining a modified version of the hierarchical occlusion map algorithm (Zhang et al., 1997) with the frustum-slicing approach to detail elision and view frustum culling (Bormann, 2000). Beginning from a calculation of the efficiency of the frustum-slicing approach in models of real-world complexity (the actual calculation is found in appendix A), section 3 theoretically discusses the role of detail elision and the relation to occlusion culling. Then, sections 4, 5, and 6 describe and discuss the experimental setup and the generated test results. Finally, section 7 provides a conclusion.

2 Algorithm Overview

Slice the frustum as shown in figure 2 and assume that full use is made of detail elision and that the level of detail of any given object is decided by which frustum slice the object resides in.

Because the minimal eye-object distance for objects residing in slice $n + 1$ is the double of that of objects residing in slice n, the level of detail should be such that the smallest details rendered in slice $n + 1$ are twice as big as the smallest details rendered in slice n, as this translates into equal angular resolution.

Thus, because small triangles are much more prevalent than large ones (Deering, 1993), detail elision culls a significant amount of geometry. But then, when detail elision is fully utilized, reducing the load on the scene graph traversal and occlusion-culling algorithms is asymptotically more important than reducing the load on the (rest of the) rendering pipeline.

The frustum-slicing algorithm addresses this problem by placing objects in an octree. If the minimal objects, resident in slice n, that the eye can resolve are at depth d in the octree, render the octree('s overlap with slice n) only to this depth. When rendering objects residing in slice $n + 1$, render the octree('s overlap with slice $n + 1$) only to depth $d - 1$, because objects at depth $d - 1$ are approximately twice as large as those placed at depth d, and, thus, in slice $n + 1$, the eye is not able to resolve objects smaller than those residing at depth $d - 1$ in the octree.

As small objects are far more prevalent than large ones, this approach, in an efficient manner, vastly reduces the load on the octree (or "scene graph") traversal and occlusion-culling algorithms.

The frustum-slicing method makes for both efficient culling of the world octree and for efficient control of level of detail, if provided by the objects, giving good frame rates when mean observer-object distances are large (Bormann, 2000). However, occlusion culling is needed for congested models as—when the mean distance to objects drop and, as a consequence, one can actually see only a few objects, although close by—the attempt is made to render many occluded objects at high detail. Consequently, frame rates drop unacceptably. Also of importance for very large models is the fact that occlusion culling provides for earlier termination of the octree traversal.

For this reason, one may integrate the hierarchical occlusion map (HOM) algorithm of Zhang et al. (1997) with the frustum-slicing approach by creating a HOM of the alpha layer of the (accumulated) frame buffer every time one finishes rendering the objects that reside in a given frustum slice. Because slices are traversed in front-to-back order, one can then cull nodes and/or objects resident in the next slice against this HOM. This approach eliminates the depth estimation buffer as well as the preprocessing step and the runtime occluder selection process of the original HOM algorithm at the cost that objects are not culled against occluders residing in the same slice as themselves.

Because most modern hardware provides neither an alpha-buffer nor hardware support for mipmapping, one needs modifications to the simple scheme above. The lack of an alpha-buffer necessitates that one renders the HOMs, in monochrome, on a separate pass of the frustum-slicing algorithm. Also, in this case, one can render to a smaller viewport, and one can choose to render only a simplified "occluder representation" of the objects (an LOD that has little geometrical complexity and that provides a reasonably tight lower bound on the occlusion properties of the object[1]).

Further, one can choose a simplified way of picking good occluders, namely by rendering the octree less deeply than is done when rendering the scene. That is, if in slice n the octree is rendered to depth d when rendering the scene, then, when one renders the occluder representation, one should render the octree to only depth $d_{\text{occl}} = d - d_{\text{corr}}$, where d_{corr} is the depth correction.

Note that, because one thus renders fewer objects, to a lower detail, and to a smaller viewport, then the actual rendering of occluders is quite fast. The major part of the cost of the algorithm then is in actually making the HOMs, that is, in creating and storing mipmaps. On systems with more than one graphics pipeline, one may then render the occluder representation in a separate pipeline because one then gains the difference between the time taken to render the scene and the time taken to render the occluder representation—time that can be used creating the mipmap. As scalability is a main concern of the algorithm, it is further important to note that the cost of creating HOMs is more or less independent of the geometrical complexity of the virtual environment and also the cost will fall as hardware capability increases.

Further, one may choose not to create a HOM if the opacity of the frame buffer is very low (as is often the

1. One could even imagine the occluder representation having different LODs for use at different distances.

case for the first few frustum slices) because checking nodes and objects against the HOM is not worth the cost in this case. This notion may be made adaptive by appealing to frame-to-frame coherence in the following way. Begin by making an HOM at some predetermined distance, $dist_{max}$, being the maximal distance from the viewpoint at which one is prepared to make occlusion maps. Now, if at the topmost level of the HOM (where 1×1 pixel corresponds to the accumulated picture) the opacity is less than some threshold, don't construct HOMs for distances less than $dist_{max}$. However, if the opacity is larger than this threshold, then in the next frame one makes an occlusion map at distance $dist_{max}/2$ (that is, one frustum slice closer to the viewpoint) as well. During successive frames, one continues making HOMs one frustum slice closer to the viewpoint until the opacity drops below the threshold.[2]

HOMs at the far end of the frustum are shed if, in the previous frame, the opacity is above the occlusion threshold for some HOM closer to the viewpoint. Likewise, the HOM closest to the eye is not re-created if, in the previous frame, the opacity was too low to justify the cost of creating it and of culling objects against it. Finally, one can also avoid creating intermediate HOMs if in the previous frame the opacity of the HOM was not much different from an HOM closer to the viewpoint (as, in this case, one is essentially building the same HOM twice).

The actual values of $dist_{max}$, d_{corr}, and the opacity thresholds may be chosen adaptively. The maximal distance, $dist_{max}$, at which to make HOMs is decided in part by the cost of creating HOMs and in part by the fact that HOM opacities, while growing at a brisk pace close to the viewpoint, tend to converge relatively fast towards some asymptotic value. The opacity thresholds for culling octree nodes and objects need to be large enough to justify the cost of culling them against the HOM. Thus, it will generally be higher for nodes than for objects of similar size, and it will be higher for small objects than for large. And the opacity threshold for building HOMs at the near end at the frustum likewise must justify the cost of creating one even closer to the viewpoint; that is, one must expect its opacity to be such that it is useful for culling objects and/or nodes.

Finally, if the depth correction, d_{corr}, when rendering the occluder representation of the scene is chosen too small, rendering the occluders becomes too expensive, whereas choosing it too large will lead to too few occluders being utilized, leading to low-opacity HOMs that are of little use, but still expensive to create. The actual value of course depends upon the peculiarities of the model that is being viewed.

Now, if one has good runtime knowledge of the cost of rendering the objects in the model and of the price of creating HOMs and of culling against them, one could in principle determine the above parameters. However, for practical purposes, one can instead adapt to the VE viewed by keeping track of the frame time: the values of the algorithm parameters can be changed incrementally at random time intervals, and, if the next frame time is lower, one keeps the new value.

The actual culling against the HOM is done somewhat differently from the procedure chosen by Greene et al. (1993) and Zhang et al. (1997), wherein one starts at the level of the *z*-pyramid (respectively, the hierarchical occlusion map) at which a single pixel is approximately the same size as the projected primitive (respectively, object). If the primitive or object is visible, one recursively descends to the next level of the *z*-pyramid or hierarchical occlusion map to check all the subpixels covered by the bounding box projection.

During initial testing, using the rather large "outdoor type" models described in section 4, this procedure turned out to be too expensive, and so a simpler one, severely limiting the amount of recursion allowed, was devised. This procedure, starting at level 0 (the 1×1 pixel representation of the viewport to which the occluders were rendered), checks whether the HOM opacity is above the occlusion threshold. If it isn't, one descends to the next level only if the on-screen projection of the object or node is completely within the confines

2. An alternative to this procedure is to rely on a qualified guess at the HOM opacity for the present frame utilizing the angular extension of the occluders rendered as calculated from the object size and the eye-object distance represented by the level in the octree at which the object resides and the slice number, respectively—or equivalently, by the difference between the object's level in the octree and the lowest level in the octree rendered for the given slice.

of one of the "child pixels" of the HOM, and so on, until one reaches the lowest level at which one is prepared to cull against the HOM. This level may be different for objects and nodes, be dependent upon their sizes, and may also be set adaptively.

The efficiency of the frustum-slicing approach is formally investigated in appendix A, and section 3 discusses the relevance to occlusion culling in general and to the above algorithm in particular.

3 Occlusion Culling in Large, Natural-World-Type VEs

Eventually, one would like to create virtual environments matching their physical counterparts in complexity. In the physical world, it is generally found that the number of objects (and of features within objects) follows a fractal distribution, written formally as

$$N(l) \sim \left(\frac{l}{l_0}\right)^{-D_H}, \tag{1}$$

where l is the size of the object or feature, l_0 is a reference size, and D_H is the Hausdorff dimension (Mandelbrot, 1982). This in turn means that small objects and features are much more prevalent than large ones. It is a noteworthy fact that approximately such a relationship is also found by Deering (1993) to hold for a large number of high-quality CAD models. Thus, although simplistic models created with real-time rendering capability in mind probably do not conform with fractal scaling laws, this type of more complex model is of theoretical—and eventually of practical—importance because it represents what one would ultimately like to render and describes it in a generic manner with only three parameters per type of object, allowing for theoretical analysis.

In appendix A, the efficiency of the frustum-slicing approach is calculated for the case of a large VE with fractal scaling properties. The analysis, which does not take occlusion into account (that is, objects are de facto assumed rendered whether occluded or not) yields the following result for the density of objects, ρ, and the polygon count, ρ_Δ, as a function of the frustum slice, n, in which the objects reside:[3]

$$\rho(n) \sim \sum_{d=0}^{d_{\lambda_{\min}}-(n-1)} 2^{(D_H-1)d} \tag{2}$$

$$\rho_\Delta(n) \sim 2^{-(D_{H_\Delta}-1)n} \sum_{d=0}^{d_{\lambda_{\min}}-(n-1)} 2^{(D_H-D_{H_\Delta}-1)d} \tag{3}$$

Here, D_H and D_{H_Δ} are the Hausdorff dimensions for, respectively, the objects and the object features (polygons), and $d_{\lambda_{\min}}$ is the depth, in the octree, of the smallest octree nodes. In equation (2), the object density is a sum over the number of objects residing at different levels in the octree. The object density thus decreases with slice number, with the rate being determined by the fractal dimension. The polygon density, equation (2), consists of two factors: an exponential in the slice number that is due to detail elision at the object level, and a modified sum over objects that is due to frustum slicing. The modification ensures that one avoids double-counting triangles that disappear both due to frustum slicing (because the host object is eliminated) and due to detail elision.

Obviously, the geometry and object elimination due to detail elision and frustum slicing is dependent upon the exact values of the Hausdorff dimensions D_H and D_{H_Δ} and on the value of $d_{\lambda_{\min}} - (n-1)$. However, for large worlds, the latter will be much larger than one most of the time, and one expects that $D_H \sim D_{H_\Delta}$ and also that the Hausdorff dimensions normally approach a value of two. (Because triangles tile object surfaces and because objects tend to cluster close to the ground, one expects that D_H isn't much larger, normally.) Thus, one expects that, in most conditions, equations (2) and (3) approximately imply that

$$\rho(n+1) \sim \frac{\rho(n)}{2^{D_H-1}} \text{ for } d_{\lambda_{\min}} - (n-1) \gg 1 \tag{4}$$

and

$$\rho_\Delta(n+1) \sim \frac{\rho_\Delta(n)}{2^{D_{H_\Delta}-1}} \text{ for } d_{\lambda_{\min}} - (n-1) \gg 1 \text{ and } D_H \sim D_{H_\Delta}. \tag{5}$$

3. The corrective term of equation (15) in appendix A has been omitted.

This is perhaps of little surprise because frustum slicing essentially does on the object level what detail elision does on the geometry level. However, a few things are worth noting. For infinite worlds, frustum slicing doesn't contribute much to the elimination of geometry. For finite worlds this is no longer true, but even in this case most of the geometry will be hooked up in large objects, and, furthermore, objects being eliminated due to their insignificant size normally would have their triangles eliminated anyway because they are of small size as well.

Also, although the densities of triangles and objects are forcefully reduced by detail elision and frustum slicing, the absolute numbers are not. Indeed, the size of the frustum increases with the square of the distance to the eye. Going from slice n to slice $n + 1$, the volume of the frustum slice thus grows by approximately a factor of eight (or, if taken on a per unit depth basis, a factor of four).[4] This in turn means that the absolute amount of objects and geometry grows with distance: There is always enough large, distant objects for the load on the rendering pipeline to rise. The implication is that even in large, open environments, a measure of occlusion culling is needed. Further, this occlusion-culling algorithm needs to be able to utilize distant occluders because a significant part of the geometry is hooked up in (large) distant objects. The algorithm presented in section 2 meets this criterion.

This in turn leads to the significance of frustum slicing: reducing the load on the scene graph traversal and the occlusion-culling algorithms. Frustum slicing doesn't much change the amount of rendered geometry, but detail elision does. However, even if detail elision is used to its fullest, the amount of geometry still grows with distance, and one needs occlusion culling. But, without frustum slicing, the load on occlusion culling and scene graph traversal are asymptotically much worse than that of geometry rendering. The need for frustum slicing, or some similar approach, is only strengthened when one starts considering objects having behaviors that must be updated, rather than just passive geometrical objects.

Finally, we must dwell a little on the significance of the calculation in appendix A. There, the only assumption (besides that of a fractal model) was that the angular resolution of the eye (or HMD) determines the detail to which one desires to render the model. In absolute values, the minimal detail size thus increases by a factor of two from one frustum slice to the next, and this minimal detail size was used as a cutoff for the fractal distributions. This implicitly implies an optimal triangulation of the object, and it does so without making any assumptions about the object simplification scheme. Also, the fact that the calculation de facto considers only discrete LODs (because the frustum-slicing algorithm does so) is of minor importance, as it also can be seen as an approximation to any scheme using continuous LODs. Thus, the calculation not only gives the efficiency of the frustum-slicing approach but also a good approximation of the efficiency of any (near) optimal utilization of detail elision.

4 Experimental Setup

This section first outlines the (fractal) reference model used during testing of the algorithm, then describes the setup in some detail, and finally gives some notes regarding the implementation.

4.1 A Fractal Reference Model

For testing the algorithm of section 2, a reference model was created that is generic in the sense that it reflects on real-world complexity (Mandelbrot, 1982) as well as that of "typical" computer models (Deering, 1993) by populating the models with objects and object features that exhibit fractal scaling laws.

The Hausdorff dimension for objects was fixed at a modest $D_H = 3/2$ for all object types. However, because the different object types were sampled at different length intervals (and also because the minimal and

4. For the case where objects cluster close to the ground, the effective volume per unit length grows by just a factor of two, which is only a tad larger than the expected Hausdorff dimension, giving a rather slow increase in the amount of geometry and the number of objects on a per unit depth basis. However, as the world is assumed very large, the conclusion below remains unaltered in this case.

maximal sizes are different for different types of objects), the resulting model does not scale according to a fractal law with $D_H = 3/2$ (which is as it should be because the fractal scaling applies only to objects of the same kind, not to their combined scaling properties).

To capture the fractal scaling of object features (triangle sizes), a reference object was created that consisted of a tetrahedron, the sides of which are recursively subdivided[5] until the minimal detail of the object reflects the minimal detail size to be shown on screen. Thus, $D_{H_\Delta} = 2$. Note in passing that the crudest LOD is completely contained in any finer LOD version of the object and thus can be utilized as an "occluder representation" as well. Two other types of visual objects, both with a fixed number of polygons (no LOD) were introduced as well: some large boxes with holes ("windows") cut out of their faces, their numbers reflecting a fractal scaling law as well, and a few large icosahedra.

Thus, the model constructed is a reference model with approximately fractal scaling properties and with geometric complexity growing approximately in accordance with Deering (1993). Further, the object's footpoints are placed, at random positions, within a circular disk of height 16 m. The disk diameter is either 0.5 km, or 1 km. This model has some resemblance to a suburban neighborhood (figure 3).

The 0.5 km model was populated with either roughly 9,400 residing objects (16 icosahedra of size 24 m, 224 boxes of sizes 10 m to 24 m and 9,190 tetrahedra of sizes 0.2 m to 10 m), or with an object density four times higher (that is, with approximately 37,600 objects). The 1 km model came in only one variety, namely with approximately 37,600 objects (that is, with the same object density as the low density 0.5 km model).

The size of finest detail (of the recursive tetrahedra) was chosen to be 8 mm, and so the model rendered to full detail would consist of roughly 20,000,000 (respectively, 80,000,000) triangles and take approximately

5. By placing, on each face, a three-sided pyramid of dimensions half of that of the face, then, on each face of the pyramid, placing a pyramid of half the dimensions of the face, and so on.

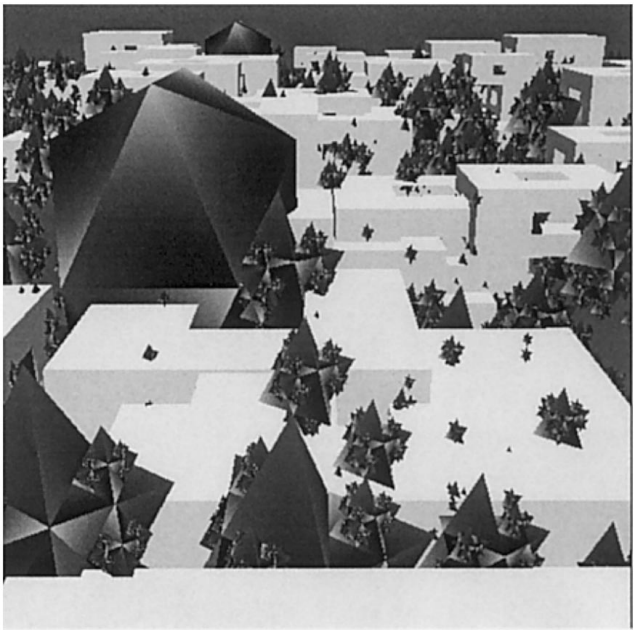

Figure 3. *A view of the model used for the experiments.*

200 sec. (respectively, approximately 800 sec.) to render to full detail.

4.2 The Observer

Denoting the bottom of the disk in which the objects reside the *ground plane,* the viewpoint is placed at the center of the disk at heights of 10 m, 24 m, and 40 m above the ground plane (corresponding to "in" the disk, slightly above the objects (like seeing a town from the roof of a house), and well above the objects). For the two highest positions, the view direction is inclined 20 deg. towards the ground plane, whereas, in the lowest position, the viewing direction is parallel to the ground plane. In all three cases, the view direction is rotating about the vertical direction for a total of 1,000 frames and one full turn. Furthermore, for each viewpoint-model combination, a number of such 1,000 frame runs were made. Because the objects were randomly (re)distributed throughout the disk, this has the same effect as changing the viewpoint while keeping the model fixed. The advantage of redistributing the model around the viewpoint, placed in the center of the disk

(and rotating the view direction), is that one keeps the number of objects and the amount of geometry within the frustum roughly fixed for all frames while at the same time simulating the effect of viewpoint motion. The frame rates will of course still fluctuate, due to the fluctuations in the distances to geometry-heavy objects as well as to good occluders, but these fluctuations are inherent in the problem. The size, L, of the first two frustum slices (figure 2) is set to 0.1 m corresponding to the fact that the human eye is able to resolve most detail if the eye-object distance is in the interval from 0.1 m to 0.2 m, approximately.[6] The minimal detail size when rendering the scene was chosen to be (the equivalent of) 1 mm for objects viewed at a distance of 10 cm to 20 cm as, even though the human eye can resolve features of approximately 0.1 mm at this distance, such resolutions are not provided by current HMDs (Gossweiler, 1997). From figure 3, it is apparent that the minimal size details rendered are comparable to the pixel size. Finally, the field of view is 60×60 deg. square, and the scene was rendered to a 512×512 pixel viewport, and occluders were rendered to a viewport of 128×128 pixels. The algorithm was implemented on a 180 MHz R10000-based O2 from Silicon Graphics.

4.3 Algorithm Implementation Notes

The main purpose of the testing of the algorithm presented in the next section was one of exploration. For this reason, the dynamics of the algorithm was restricted, in that the depth correction and the maximal depth to which an HOM is traversed when culling an object or octree node against it were kept at fixed values, as were the opacity thresholds. Also, all HOMs with an opacity above the threshold α_0^{build} were created, even when the increases in opacity between neighboring HOMs were insignificant. The opacity thresholds used were $\alpha_0^{\text{obj}} = 0.125$ (and the threshold for building HOMs was $\alpha_0^{\text{build}} \equiv \alpha_0^{\text{obj}}$), $\alpha_0^{\text{node}} = 0.8$, and $\alpha_0^{\text{occl}} = 0.98$. From the results in section 5, it is apparent that the value of α_0^{obj} was on the low side, whereas the value of α_0^{build} was definitely too small to be an optimal choice.

The depth correction, which represents the minimal angular size of occluders to be rendered, was set at $d_{\text{corr}} = 3$, corresponding to occluders being at least eight times the size of the minimal objects to be rendered. Initial testing showed a value of $d_{\text{corr}} = 4$ to be somewhat better for low-congestion models, but, as the main concern is the behavior of the algorithm in the face of increased geometrical congestion, a value of $d_{\text{corr}} = 3$ was chosen. For the chosen distribution of number of objects as a function of size (a fractal scaling law with Hausdorff dimension 3/2, see section 4.1), this means that the part of the octree traversed is only about 10% of that traversed when $d_{\text{corr}} = 0$.

As for the depth to which the HOM is traversed when performing the actual culling, this corresponded to 8×8 (respectively 64×64) pixels in the viewport when culling objects (respectively octree nodes).

One might wonder why the opacity threshold for culling nodes, α_0^{node}, is (much) higher than the culling threshold, α_0^{obj}, for culling objects for occlusion and why the depth to which the HOM is traversed is larger when culling objects than when culling nodes. This rests on preliminary tests indicating that the ratio of tested nodes to occluded nodes is considerably higher than that of tested objects to occluded objects. (A fairly representative test had one in three tested objects being occluded, whereas only one node in nine tested was found occluded, when using the same threshold). Add to this that nodes are rather more common than objects (typically, for the models used, more common by a factor of four) and add further that nodes are larger (up to a factor of eight, in the implementation) than their resident objects (meaning that they have a smaller chance of being occluded than do their resident objects). Taking detail elision into account, as most nodes are small and their resident objects are smaller still, the resident objects that a typical node tested will not contain much geometry anyway. Thus, the amount of geometry removed and time saved by the occlusion test is likely to be insignificant unless the node has a reasonable size in

6. As far as the (test of the) algorithm is concerned, L should just be smaller than the distance to the nearest good occluder(s), which for the current model is normally at least an order of magnitude larger.

turn requiring a rather occluded frame buffer. The main rationale for culling nodes, then, is to provide early termination in the case of heavily congested, and/or very large models, and the reason for the large opacity threshold and low traversal depth when culling nodes is to avoid large occlusion-culling costs for noncongested worlds.

Of course, had the algorithm been implemented with adaptive values for the above constants, one had been spared these considerations; however, the results of the experimental tests would have been harder to interpret.

5 Results

This section provides a number of experiments exploring the occlusion-culling algorithm. In all these experiments, frustum slicing was used. For an exploration of the effect of frustum slicing in and by itself, refer to Bormann (2000).

5.1 The Cost of HOM Construction

The first experiment was made exclusively to determine the price of making the hierarchical occlusion maps. It was done using a 0.5 km world with density 1 and an observer height of 24 m and, on average, showed a penalty of ~0.02 sec./*frame* for opening another viewport—even when no rendering was done in this "occlusion viewport." Rendering occluders with a depth correction, d_{corr}, equal to 2, in monochrome and to least detail, added an extra ~0.02 sec./*frame*. Finally, actually building ~4 occlusion maps per frame added another ~0.06 sec./*frame*. This sums up to 0.1 sec. for making four occlusion maps each frame.

It is important to note that only about 20% of the HOM construction cost had to do with occluder rendering.[7] Thus, most of the cost is detached from the geometrical complexity of the scene, which is good from the point of view of scalability of the algorithm.

Finally, the HOM construction could presumably be optimized. For instance, although occluders were rendered in monochrome, the use of standard software libraries (OpenGL) ensures that the HOM is effectively in true color, making mipmap construction as well as moving the mipmaps in memory more expensive than necessary.

7. Actually, even this low value is on the pessimistic side, as a depth correction of only $d_{\text{corr}} = 2$ was used for this initial experiment.

Table 1 *Runtimes with No Occlusion Culling (See Section 5 for Notation)*

ID	Λ (km)	ρ	h_{eye} (m)	p_{eye}	n	t (s)	σ_t
1A	0.5	1	40	−20	3	227	3
1B	0.5	1	24	−20	5	377	106
1C	0.5	1	10	0	5	870	1271
1D	0.5	4	40	−20	3	836	70
1E	0.5	4	24	−20	3	1150	183
1F	0.5	4	10	0	5	2306	1497
1G	1	1	40	−20	3	456	6
1H	1	1	24	−20	3	577	59
1I	1	1	10	0	8	651	145

5.2 Reference Experiment: No Occlusion Culling

Reference data was gathered for the case in which no occlusion culling was performed. The runtime is given in Table 1. The notation of the table is such that Λ is the size of the largest octree node (the world size) measured in kilometers; ρ is the (arbitrarily) normalized object density; h_{eye} is the viewpoint's elevation measured in meters above ground level, p_{eye} is the inclination of the view direction measured in degrees; n is the number of (1,000 frame) runs made with these model parameters; t is the time, in seconds, that it took to render the 1,000 frames; and σ_t is the standard deviation of this rendering time, also in seconds. As the objects of the model are redistributed between runs, the runtime standard deviation is a measure of the variation of frame rate with viewpoint.

Runtime grows both as the object density grows and as the viewpoint approaches the ground-plane disk, as then the average distance to objects shrinks. An impor-

tant observation is that not only the runtime but also its standard deviation grows with object density and with diminishing viewpoint-to-object distance.

To provide a few numbers, the highly congested models group 1F runs and the less congested models of group 1G contained some 37,000 objects and approximately four times as many octree nodes. In both cases, the octree traversal algorithm visited approximately 19,000 nodes. In the congested 1F case, for a typical run, approximately 1,600 objects were rendered, whereas, for a low-congestion 1G run, one had approximately 1,000 objects rendered. (The difference being due to frustum slicing and the increased eye-object distance for the larger 1G model.)

The vast majority of objects rendered (>85%) reside in the farthest two slices (which comprise the larger part of the frustum volume), stressing the need for culling distant objects as well as for utilizing distant occluders. Further, for distant objects, most occlusion is by (aggregates of many) distant occluders which in turn requires a simple algorithm for picking and making use of distant occluders.

For the above-mentioned models, 1,000 (respectively, 1,600) objects were rendered. A back-of-the envelope calculation[8] has approximately 6,000 objects within the frustum, indicating the efficiency of frustum slicing. It, however, also indicates that reported efficiency of occlusion-culling algorithms tend to be on the optimistic side as none of the algorithms reviewed in section 1 were tested in environments utilizing detail elision.

5.3 Culling Both Octree Nodes and Objects Against the HOMs

In table 2, runtime and its standard deviation are given when occlusion culling is done for both octree nodes and for objects.

The cost of HOM creation and of culling, for most runs, makes rendering slower than if no occlusion culling is used. However, for the heavily congested models (runs 2D, 2E, and 2F), the use of occlusion culling tends to stabilize the frame rate both in the sense that the standard deviation is relatively small and that the change of frame rate with decreasing viewpoint-to-object distance influences the frame rate less. Run 3I is a notable exception to this pattern showing that, whereas occlusion culling may tend to stabilize frame rates, frame rate is still highly susceptible to viewpoint and view direction changes.

Table 2 *Results When Checking Both Objects and Nodes for Occlusion*

ID	Λ (km)	ρ	h_{eye} (m)	p_{eye}	n	t (s)	σ_t
2A	0.5	1	40	−20	3	314	24
2B	0.5	1	24	−20	3	379	29
2C	0.5	1	10	0	4	601	380
2D	0.5	4	40	−20	3	954	59
2E	0.5	4	24	−20	3	1282	154
2F	0.5	4	10	0	4	1356	138
2G	1	1	40	−20	3	591	28
2H	1	1	24	−20	3	647	34
2I	1	1	10	0	4	1121	539

For low-congestion models (and, as an example, choose a typical 2G run (world size 1 km, object density 1), on average 4.5 occlusion maps were created (for frustum slices 8 through 12) whose opacity averaged 0.00, 0.03, 0.22, 0.31, and 0.38. Because this is below the threshold for node culling, α_0^{node}, the number of nodes visited was approximately 19,000, as in the reference case. The average number of occluders rendered, for this representative run, was 84. On average, 783 objects were culled against the HOMs, and of these 143 were found occluded. Not surprisingly, most of those resided in the farthest slices. The average number of objects rendered was 913.

A representative run for the case of a congested model (a 2F run, with a world size of 0.5 km, and an object density of 4) on average rendered 151 occluders, resulting in 5.6 HOMs (for frustum slices 6 through 11) whose average opacities were 0.02, 0.28, 0.84,

8. The objects reside in a disk. Thus, in the vertical direction, almost no objects are outside the frustum and the proportion of objects within the frustum equals the horizontal field of view divided by 360 deg.

0.94, and 0.98, and 0.98. A total of 3,900 octree nodes were culled against HOMs, and of these 950 were found occluded. Only 3,700 nodes were visited. Likewise, 704 objects were culled against the HOMs resulting in 464 objects being termed occluded. Together with the objects whose host nodes were found occluded, this resulted in a mere 285 objects being rendered.

Now, let's compare the time consumption spent by the CPU, as found from profiler data, on occlusion culling to that spent on octree traversal and object rendering. For type 2G runs, in the range of 40% of the time used is for octree traversal, whereas occlusion culling consumes approximately 5% of the time spent (not including the time it takes to build the HOMs). Finally, 10% to 15% is spent calculating objects on the fly, and the rest (approximately 40%) almost exclusively is used for actual rendering.

For type 2F runs, octree traversal, slicing, and frustum culling accounts for 10% to 15% of the time consumption, and occlusion culling takes 5% to 10% (not including map building). Calculating objects on the fly accounts for 25% to 30% of time spent, and rendering accounts for the rest (approximately 55%). For both type F and type G runs, profiler data indicate that a little less than 10% of rendering time is spent rendering occluders.

Detail elision generally causes the runtime of type G runs to be less than that of type F runs. It is perhaps worth remarking that it does so despite the fact that the time spent on octree traversal increases significantly, both relatively and numerically. This is a manifestation of the fact that, asymptotically, scene graph traversal is a harder problem than rendering, even after frustum slicing has been applied.

5.4 The Cost of Suboptimal Usage of Nearby Occluders

For the testing presented in subsection 5.3, occlusion culling is mostly well used for congested models. However, by observing the viewport as well as the occlusion viewport, it was apparent that, in some situations, adequate use wasn't made of large nearby occluders. That is, in some cases, one might have rather high rendering times even with rather few objects visible, and, also, in some cases, a relatively large object would not be used as an occluder. The major reason for this suboptimal behavior (see figure 4(a)) can be traced to the fact that, when generating the results in subsection 5.3, occluders were not rendered if overlapping the far end of the frustum slice (the far-end slice separator) because, in doing so, one risks deeming some nonoccluded objects occluded. (See figure 4(b).) By rendering occluders immediately even if they overlap the far end of the frustum slice, one will be able to utilize most large occluders (almost) immediately at the cost of some artifacts, thus giving some insight into the cost of suboptimal utilization of nearby occluders.

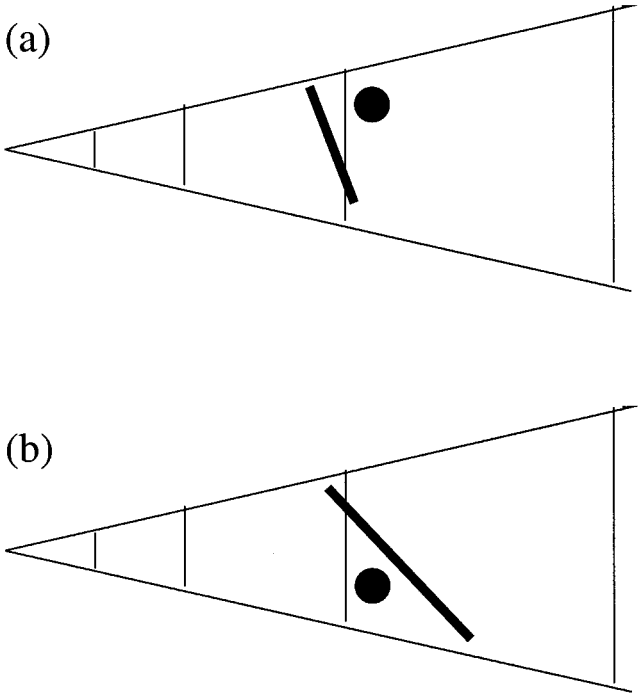

Figure 4. *In figure (a), suboptimal use is made of the large occluder. It will not be utilized in a slice that it overlaps (which is where it is most urgently needed, as detail elision eliminates much distant geometry). If, on the other hand, the algorithm fails to check whether an occluder cuts the far-end slice separator, the situation in (b) will have the small object erroneously deemed occluded.*

From table 3, it is seen that, in the case of highly congested models (3F), one gets a much better frame rate

Table 3 *Runtimes When Checking Both Objects and Nodes for Occlusion for the Case When Occluders are Rendered in the First Frustum Slice They Occur, Even if They Intersect the Far-End Slice Separator*

ID	Λ (km)	ρ	h_{eye} (m)	p_{eye}	n	t (s)	σ_t
3A	0.5	1	40	−20	3	331	12
3B	0.5	1	24	−20	3	385	64
3C	0.5	1	10	0	4	685	428
3D	0.5	4	40	−20	3	1001	40
3E	0.5	4	24	−20	4	1271	145
3F	0.5	4	10	0	5	650	677
3G	1	1	40	−20	3	586	12
3H	1	1	24	−20	3	665	48
3I	1	1	10	0	3	919	146

Table 4 *Runtimes When Checking Only Objects for Occlusion*

ID	Λ (km)	ρ	h_{eye} (m)	p_{eye}	n	t (s)	σ_t
4A	0.5	1	40	−20	5	324	12
4B	0.5	1	24	−20	6	407	82
4C	0.5	1	10	0	13	551	462
4D	0.5	4	40	−20	7	936	33
4E	0.5	4	24	−20	8	1132	173
4F	0.5	4	10	0	11	1138	930
4G	1	1	40	−20	5	582	11
4H	1	1	24	−20	7	683	125
4I	1	1	10	0	9	797	334

Occluders were rendered in the first frustum slice they occurred, even if intersecting the far-end slice separator.

when utilizing large occluders immediately as opposed to only for the next frustum slice or the frustum slice beyond that. However, the guaranteed frame rate doesn't get any better as the standard deviation gets much larger, almost offsetting the advantage. Indeed, while local occluders have large effect if available, the guaranteed frame rate is instead determined at larger distances where one sees the combined effect of many occluders which of course does not vary as much.

5.5 The Utility—and Cost—of Culling Octree Nodes

Although, in the previous section, octree nodes were culled against the HOMs for only highly occluded models (as $\alpha^0_{node} = 0.8$), it still might be too expensive. Table 4 summarizes the results of repeating the setup of previous section (using occluders even if cut by the far end of the previous frustum slice) but culling only objects, not octree nodes.

Comparing to the reference case in which no occlusion testing is done, it is worth noting that, although runtimes are generally higher, for congested models it is lower, as is its standard deviation. Comparing with the results of the previous subsection one sees that, although runtimes generally are lower, for the highly congested view of type 4F runs, one has a high runtime as well as a rather high standard deviation. Comparing finally to subsection 5.3 and run 2F, one has comparable runtimes but a much higher standard deviation. Thus, for highly congested VEs, it is indeed important to use occlusion culling at the node level to cut off octree traversal.

6 Discussion

The implementation discussed in the previous sections was a little lavish when it came to HOMs. Indeed, in open environments, HOM opacities will follow a typical growth curve,[9] growing briskly in the beginning, then converging to some value, α_∞. Obviously, when opacities start converging, one needn't build a new HOM at every frustum slice. How much the HOM opacity needs to increase over the previously built HOM to be worth its creation depends upon the potential benefit, that is, of the detail level of the VE. For the models discussed in the pre-

9. In the simple case of a random distribution, this growth function is (disregarding fluctuations) $\alpha_\infty(1 - e^{-\gamma l})$, where α_∞ is the opacity at "infinity," which is typically less than one in open environments, l is the distance from the eye, and γ is a quantization of the amount of congestion.

vious sections, three HOMs (opacities of approximately 0.4, 0.6, and 0.8) probably would be more appropriate (for congested models). Although even creating just three HOMs is an expensive proposition at present, it should be noted that at most one-fifth of the cost is due to rendering the occluders; that is, the cost is largely independent of the complexity of the model. Instead, the cost is tied tightly to hardware capability, which is growing at a brisk pace. (Indeed, when mipmapping becomes hardware supported, the issue probably will go away.)

For general-purpose rendering algorithms to be a viable proposition, their cost must be output sensitive (that is, tied to screen resolution, lighting models, and the like), rather than dependent on scene complexity. That is, one would like rendering time to converge as model complexity increases. However, one cannot make models arbitrarily large and complex because the limit to the number of objects that can be allocated by the operating system is reached long before one reaches real-world complexity. Furthermore, in the experiment reported in the previous sections, the model maximal complexity was such that no swapping between memory and hard disk was needed, as that would have complicated interpretation of the results. Still, by varying the object density as well as the number of objects, one could get an indication of the asymptotic behavior, and it suggested that, for the proposed algorithm, frame rates do indeed tend to converge in the face of increased scene complexity.

For the low-congestion type G runs and the high-congestion type F runs, the model consisted of approximately 37,000 objects (and some 80,000,000 triangles, if rendered to full detail). Of these, approximately 6,000 objects were within the viewing frustum at any one time. The frustum-slicing algorithm saw that only approximately 1,000 (1G), respectively approximately 1,600 (1F), of these objects were rendered. For the creation of the approximately five HOMs, approximately 100 occluders were rendered (in their occluder representation). This cut the number of objects rendered to approximately 900 (2G), respectively approximately 300 (2F). Still, despite rendering three times as many objects in the low-congestion case (2G), due to detail elision, the rendering time is only half of that of the highly congested (2F) model. Thus, in sparsely populated areas, detail elision is very effective, whereas, in congested ones, occlusion culling can be used to large effect.

As implemented, the algorithm makes suboptimal use of large nearby occluders, as they are utilized only in the frustum slices farther away from the eye than the farthest slice that the occluder overlaps. One could remedy this problem by combining the algorithm (which has the great outdoors as its main concern) with an algorithm suited for culling against a few occluders only. (These occluders could still be found by the algorithm proposed in this paper.) Or one could introduce fractional frustum slices when a large occluder is detected, in order to be able to utilize it sooner.

Whatever the remedy, it is important to note, however, that the guaranteed frame rate depends not upon the accidental presence of a few large nearby occluders but rather upon the asymptotic properties of the model. After all, even if inside a house (situated in a large VE), one can look out the windows. In the previous section, this was reflected in the fact that if, for heavily congested models, one chose to utilize occluders immediately, this resulted in a much better frame rate, but with a much higher standard deviation. Note that the fact that the worst case is determined by the statistical properties of the model at long distances from the eye dictates that a general-purpose occlusion-culling algorithm must be able to efficiently choose and utilize distant occluders in a simple manner, something seldom seen (the hierarchical *z*-buffer being an exception).

7 Conclusion

Zhang et al. (1997) noted the desirability of integrating the hierarchical occlusion map algorithm with level of detail. The algorithm proposed in this paper, combining the hierarchical occlusion map with the frustum-slicing method of Bormann (2000) does just that, and, at the same time, the resulting algorithm is much simpler than the original hierarchical occlusion map algorithm, potentially making it more robust.

Another feature of this combined algorithm is that all objects of sufficient angular size are used as occluders. In large, open virtual environments, this is an important

enhancement over the original HOM algorithm because, theoretically, the worst-case rendering time depends upon the asymptotic properties of the model (even when detail elision is fully utilized). Many objects will be far away and so will many good occluders—occluders likely to be missed by the original HOM algorithm. Thus, for large outdoor-type environments, one is faced with providing a simple means of utilizing (congregations of) a large number of distant occluders, even if detail elision vastly reduces the value of any single occluder with distance. Other notable features of the proposed algorithm are the minimal use made of frame-to-frame coherence and the virtually nonexistent preprocessing, making the algorithm as suitable for dynamic scenes as for static ones.

Testing the algorithm using very large models led to the construction of approximately five HOMs, which perhaps was overkill because the opacity increase between neighboring HOMs occasionally was too small to justify their creation. The cost of creating those HOMs was very significant. (A limit of three HOMs seems reasonable.) Importantly, however, this cost is tied much stronger to hardware capability than to the geometrical complexity of the model and thus is expected to fall in the future, even as the models considered grow more complex.

For the models considered, frustum slicing reduced the number of objects considered for rendering by a factor of four or more. For heavily congested models, occlusion culling reduced the number of objects rendered by another factor of five, while having almost no impact for low-congestion models. Consequently, as visual congestion increased, the mean rendering time tended to converge, and, also, the fluctuations in frame rate tended to decrease, as compared to the case in which no occlusion culling is used, indicating that the proposed algorithm adequately addresses the issues of occlusion culling in large, open environments.

Appendix A

On the Efficiency of Detail Elision and of the Frustum-Slicing Approach

In this appendix, the theoretical efficiency of detail elision (LOD) and of the frustum-slicing approach to scene graph traversal is calculated. The premise of the calculation is that one eventually will want to render models of real-world complexity. In nature, this complexity is generally found to conform to fractal scaling laws (Mandelbrot, 1982). Two parallel computations will be performed, calculating the number of objects and the amount of geometry (triangles) to be rendered in frustum slice n. Both these calculations assume that optimal use is made of detail elision.

Denote by $N_i(l)$ the number of objects of type i and size, l, where size can be taken to be the diameter of a bounding sphere. Further denote the maximal size of type i objects by $l_{\max_i}$ and the corresponding number of type i objects of maximal size by $N_{l_{\max_i}}$. Then, the assumption that the number of objects, as a function of size, follows a fractal scaling law can be written as

$$N_i(l) = N_{l_{\max_i}} \left(\frac{l}{l_{\max_i}} \right)^{-D_{H_i}} \tag{A1}$$

where D_{H_i} is the Hausdorff dimension for objects of type i.

Not only the sizes of objects but also the sizes of features within objects are distributed according to fractal scaling laws; consequently, so are the triangle sizes. Denoting by $\tilde{N}_{l_{\max_{\Delta i}}}$ the number of triangles of maximal size $l_{\max_{\Delta i}}$ (with $l_{\max_{\Delta i}} < 1$, of course), the number of triangles of size l_Δ residing in an object of size l scales according to

$$\Delta_i(l_\Delta) = \tilde{N}_{l_{\max_{\Delta i}}} \left(\frac{l_\Delta}{l_{\max_{\Delta i}}} \right)^{-D_{H_{\Delta i}}} = N_{l_{\max_{\Delta i}}} \left(\frac{l_\Delta}{l} \right)^{-D_{H_{\Delta i}}} \tag{A2}$$

where, in the second equality, the scaling law has been expressed in terms of the host object size l (instead of $l_{\max_{\Delta i}}$) by simultaneously redefining the constant of proportionality.[10] If needed, the constant of proportionality can be determined from the constraint that the sum of the triangle surface areas equal that of the object.

Now, choose the minimal detail size, $\delta_{\min}$, to be the size of the smallest features useful to a human operator, and thus the smallest features to be rendered on screen. (Practically, the hardware determines $\delta_{\min}$). According to the

10. Note, by the way, that one could also express the fractal scaling law in terms of the triangle area rather than in terms of the bounding circle radius of the triangle as normally done (and as done in this appendix) because this would merely lead to a redefinition (a constant scaling) of the Hausdorf dimension.

frustum-slicing scheme, one is not going to render objects to detail $\delta_{\min}$, except when rendering objects residing in slice 1. Denoting the minimal size of features to actually be rendered in slice n by δ_n and noting that the minimal eye-object distance doubles from one slice to the next one then has the following relation between the minimal detail sizes:

$$\delta_n = 2\delta_{n-1} = \cdots = 2^{n-1}\delta_1 \equiv 2^{n-1}\delta_{\min} \tag{A3}$$

Below, the discussion regards what is actually rendered; thus, δ_n is used, rather than $\delta_{\min}$.

Denote the minimal octree node size by $\lambda_{\min}$. The number of objects of sizes in the interval $[2^k\lambda_{\min}, 2^{k+1}\lambda_{\min}]$ is (assuming $D_{H_i} > 1$ in the following):

$$\begin{aligned} N_i(l \in [2^k\lambda_{\min}, 2^{k+1}\lambda_{\min}]) &= \int_{2^k\lambda_{\min}}^{2^{k+1}\lambda_{\min}} N_{l_{\max_i}} \left(\frac{l}{l_{\max_i}}\right)^{-D_{H_i}} \mathrm{d}l \\ &= \frac{N_{l_{\max_i}}}{D_{H_i} - 1}\, l_{\max_i}^{D_{H_i}} \lambda_{\min}^{1-D_{H_i}} (2^k)^{1-D_{H_i}} [1 - 2^{1-D_{H_i}}] \end{aligned} \tag{A4}$$

The number of triangles making up a type i object of size l is

$$\begin{aligned} \Delta_i(l) &= N_{l_{\max_{\Delta i}}} \int_{\delta_n}^{l} \left(\frac{l_\Delta}{l}\right)^{-D_{H_{\Delta i}}} \mathrm{d}l_\Delta \\ &= \frac{1}{\delta_n^{D_{H_{\Delta i}}-1}} \frac{N_{l_{\max_{\Delta i}}} l_{H_{\Delta i}}^D}{D_{H_{\Delta i}} - 1} \left[1 - \left(\frac{\delta_n^{D_{H_{\Delta i}}-1}}{l}\right)^{D_{H_{\Delta i}}-1}\right] \end{aligned} \tag{A5}$$

and, thus, the number of triangles residing in objects of sizes in the interval $[2^k\lambda_{\min}, 2^{k+1}\lambda_{\min}]$ is

$$\begin{aligned} &\Delta_i(l \in [2^k\lambda_{\min}, 2^{k+1}\lambda_{\min}]) \\ &= \int_{2^k\lambda_{\min}}^{2^{k+1}\lambda_{\min}} N_{\Delta i} \left(\frac{l}{l_{\max_i}}\right)^{-D_{H_i}} \frac{1}{\delta_n^{D_{H_{\Delta i}}-1}} \frac{l_{H_{\Delta i}}^D}{D_{H_{\Delta i}} - 1} \\ &\quad \times \left[1 - \left(\frac{\delta_n^{D_{H_{\Delta i}}-1}}{l}\right)^{D_{H_{\Delta i}}-1}\right] = \frac{N_{\Delta i}}{D_{H_{\Delta i}} - 1} \frac{l_{\max_i}^{D_{H_i}}}{\delta_n^{D_{H_{\Delta i}}-1}} \\ &\quad \times \left[\frac{2^{(D_{H_{\Delta i}} - D_{H_i}+1)k} \lambda_{\min}^{D_{H_{\Delta i}} - D_{H_i}+1}}{D_{H_{\Delta i}} - D_{H_i} + 1} (2^{(D_{H_{\Delta i}} - D_{H_i}+1)} - 1)\right. \\ &\quad \left. - \frac{2^{(2-D_{H_i})k} \lambda_{\min}^{2-D_{H_i}} \delta_n^{D_{H_{\Delta i}}-1}}{2 - D_{H_i}} (2^{2-D_{H_i}} - 1)\right] \end{aligned} \tag{A6}$$

with $N_{\Delta i} \equiv N_{l_{\max_i}} N_{l_{\max_{\Delta i}}}$.

Denote the depth (level) of a node in the octree by d and its corresponding node size by λ. By convention, the root of the octree has $d = 0$, and, also, the level of nodes of minimal size is denoted by $d_{\lambda_{\min}}$. Now, to simplify the calculation, assume that all objects reside in an octree node of approximately the same size. Then, the objects of type i at level d in the octree are those with sizes such that $NodeSize > ObjectSize > NodeSize/2$ (that is, $\lambda > l > \lambda/2$) and thus, from equation (A4), the number of objects residing at depth d in the octree is

$$\begin{aligned} N_i(d) &= N_i(l \in [2^{d_{\lambda_{\min}} - d - 1}\lambda_{\min}, 2^{d_{\lambda_{\min}} - d}\lambda_{\min}]) \\ &= \frac{N_{l_{\max_i}}}{D_{H_i} - 1}\, l_{\max_i}^{D_{H_i}} \lambda_{\min}^{1-D_{H_i}} [1 - 2^{1-D_{H_i}}] (2^{d_{\lambda_{\min}} - d - 1})^{1-D_{H_i}} \end{aligned} \tag{A7}$$

except when $d = d_{\lambda_{\min}}$ where the relation breaks down because, in principle, one has an ever increasing number of objects but has stopped subdividing the nodes that thus contain an infinity of objects. For practical purposes, of course, one may think of the smallest nodes as being the size of the smallest objects useful to a human operator in the VE, as then, for the purposes of the present discussion, the breakdown of equation (A7) is of little consequence. From equation (A6), the corresponding number of triangles contained in the objects residing in octree nodes at depth d is

$$\begin{aligned} \Delta_i(d) &= \Delta_i(l \in [2^{d_{\lambda_{\min}} - d - 1}\lambda_{\min}, 2^{d_{\lambda_{\min}} - d}\lambda_{\min}]) \\ &= \frac{N_{\Delta i}}{D_{H_{\Delta i}} - 1} \frac{l_{\max_i}^{D_{H_i}}}{\delta_n^{D_{H_{\Delta i}}-1}} \left[\frac{(2^{(D_{H_{\Delta i}} - D_{H_i}+1)} - 1)\lambda_{\min}^{D_{H_{\Delta i}} - D_{H_i}+1}}{D_{H_{\Delta i}} - D_{H_i} + 1}\right. \\ &\quad \times 2^{(D_{H_{\Delta i}} - D_{H_i}+1)(d_{\lambda_{\min}} - d - 1)} \\ &\quad \left. - \frac{(2^{2-D_{H_i}} - 1)\lambda_{\min}^{2-D_{H_i}} \delta_n^{D_{H_{\Delta i}}-1}}{2 - D_{H_i}} 2^{(2-D_{H_i})(d_{\lambda_{\min}} - d - 1)}\right] \end{aligned} \tag{A8}$$

Now one finally is in a position to calculate the density of objects and the density of geometry to be rendered in frustum slice n, provided that one makes an assumption about the spatial distribution, or clustering, of objects. In nature, objects and features tend to clus-

ter in fractal patterns[11] theoretically described by percolation theory (Andresen, 1991; Essam, 1980). However, this is of little use to us because one would then get highly viewpoint-specific results, except in the asymptotic region. It is better, then, to use a random distribution; then one has a fixed density, except very locally, and thus relatively stable results. Different-density random distribution models can then be used as approximations to different regions of more general spatial distributions. Also, in the asymptotic region, the two types of models effectively coincide.

Further, for simplicity, assume that the minimal node size and the minimal object size equal the minimal detail size, $\delta_{\min}$ (which is equal to δ_1, the minimal detail size of frustum slice 1), so that detail elision starts eliminating objects as well as detail from slice 1 onwards. (If this is not the case, then the results of the following calculations are strictly true only beyond the frustum slice at which detail elision starts reducing the level to which the octree is traversed.)

Finally, remembering that the depth of minimal nodes in the octree is called $d_{\lambda_{\min}}$, note that, except for slice 0, the depth to which the octree is visited when rendering slice n is $d_{\lambda_{\min}} - (n-1)$ (the "-1" stemming from the fact that the most detail is seen in slice 1, not in slice 0[12]). Then, the density, $\rho_i(n)$, of type i objects in slice n simply becomes proportional to the sum of objects residing at the levels of the octree that the rendering algorithm visits, given by equation (A7); that is,

$$\rho_i(n) \sim \sum_{d=0}^{d_{\lambda_{\min}}-(n-1)} N_i(d) \sim \sum_{d=0}^{d_{\lambda_{\min}}-(n-1)} 2^{(D_{H_i}-1)d} \quad \text{(A9)}$$

where the constant of proportionality is determined by the modeled object density (which has been fixed at no point during the calculation) and also, strictly speaking, is only asymptotically a constant due to fluctuations in the object density in the proximity of the viewpoint.

From equations (A3) and (A8), the corresponding density of geometry, $\rho_{\Delta_i}(n)$, residing in type i objects located in slice n, that will be submitted to the (rest of the) graphics pipeline by the frustum-slicing algorithm is

$$\begin{aligned}\rho_{\Delta_i}(n) &\sim \sum_{d=0}^{d_{\lambda_{\min}}-(n-1)} \Delta_i(d) \\ &\sim \beta_1 2^{-(D_{H_{\Delta i}}-1)n} \sum_{d=0}^{d_{\lambda_{\min}}-(n-1)} 2^{(D_{H_i}-D_{H_{\Delta i}}-1)d} \quad \text{(A10)} \\ &\quad - \beta_2 \sum_{k=0}^{d_{\lambda_{\min}}-(n-1)} 2^{(D_{H_i}-2)d}\end{aligned}$$

where β_1 and β_2 are constants. In order to interpret this latter result, note that the two terms in the above equation stem from the factor $[1 - (\delta_n^{D_{H_{\Delta i}}-1}/l)^{D_{H_{\Delta i}}-1}]$ in equation (A6). Because δ_n is the minimal detail size to be rendered and l is the object size, the second term is significant only for objects not much bigger than the size of the smallest triangles being rendered. This term will thus be seen as a corrective term and be omitted in the discussion of the asymptotic behavior given in section 3. Note that, in doing so, the efficiency of the frustum-slicing approach is underestimated rather than overestimated.

Further, it should be remarked that the first term consists of two factors: one exponential in the slice number and one a sum over octree levels. The exponential is due to detail elision, whereas the sum is due to frustum slicing. Actually, the results are a tad more general: any optimal utilization of detail elision at both triangle and object levels would lead to approximately the same result. Had one done away with the discrete slicing of the frustum, and the corresponding placement of objects in an octree, one could go through the same calculation, although with sums replaced by integrals. Presumably, the results wouldn't change much.

Finally, note that, in the above calculation, the geometry residing in very big objects is de facto counted by only the fraction of the node that overlaps the frustum slice which is strictly true only if multiresolution rendering is introduced for big objects. See, for example, Lindstrom et al. (1996), Schmalstieg and Schaufler (1997), and Luebke and Erikson (1997).

11. Think of an object distribution whose density distribution has the spatial layout of a tree root.

12. When using the frustum slice sizes of section 2.

References

Andresen, B. (1991). *Fraktal vaekst*. Course notes, Niels Bohr Institute.

Bormann, K. (2001). *Frustum slicing*. Forthcoming in *Virtual Reality: Research, Development, and Application*.

Clark, J. H. (1976). Hierachical geometric models for visible surface algorithms. *Comm. ACM, 19,* 547–554.

Cohen-Or, D., Fibich, G., Halperin, D., & Zadicano, E. (1998). Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Proc. EUROGRAPHICS, 17*(3), 243–253.

Coorg, S., & Teller, S. (1996). Temporally coherent conservative visibility. *Proc. Comp. Geom.*, 78–87.

———. (1997). Real-time occlusion culling for models with large occluders. *Proc. Int. 3D Graphics,* 83–90.

Deering, M. (1993). Data complexity for virtual reality: Where do all the triangles go? *Proc. VRAIS,* 357–363.

Essam, J. (1980). Percolation theory. *Reports on Progress in Physics, 43*(7), 833–912.

Funkhouser, T., & Sequin, C. (1993). Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Proc. SIGGRAPH,* 247–250.

Gossweiler, R. (1997). Case study synopsis: Visual perception and the design of an interactive system. *SIGGRAPH* course 33, notes 263–288.

Greene, N., Kass, M., & Miller, G. (1993). Hierarchical z-buffer visibility. *Proc. SIGGRAPH,* 231–238.

Howell, J., Chrysanthou, Y., Steed, A., & Slater, M. (1999). A market model for level of detail control. *Proc. VRST,* 96–103.

Hudson, T., Manocha, D., Cohen, J., Lin, M., Hoff, K., & Zhang, H. (1997). Accelerated occlusion culling using shadow frusta. *Proc. Comp. Geom.,* 1–10.

Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L., Faust, N., & Turner, G. (1996). Real-time continuous level of detail rendering of height fields. *Proc. SIGGRAPH,* 109–118.

Luebke, D., & Georges, C. (1995). Portals and mirrors: Simple, fast evaluation of potentially visible sets. *Proc. Symp. Interactive 3D Graphics,* 105–106.

Luebke, D., & Erikson, C. (1997). View-dependent simplification of arbitrary polygonal environments. *Proc. SIGGRAPH,* 199–208.

Mandelbrot, B. (1982). *The fractal geometry of nature*. New York: W. H. Freeman & Company.

Saona-Vazquez, C., Navazo, I., & Brunet, P. (1999). The visibility octree: A data structure for 3D navigation. *Computers & Graphics, 25*(3), 635–644.

Schmalstieg, D., & Schaufler, G. (1997). Smooth levels of detail. *Proc. VRAIS,* 12–19.

Sudarsky, O., & Gotsman, C. (1996). Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality. *Proc. Eurographics, 15,* 249–258.

Teller, S., & Sequin, C. (1991). Visibility preprocessing for interactive walkthroughs. *Proc. SIGGRAPH,* 61–69.

Zhang, H., Manocha, D., Hudson, T., & Hoff, K. (1997). Visibility culling using hierarchical occlusion maps. *Proc. SIGGRAPH,* 77–88.