A Spatially and Temporally Coherent Object Space Visibility Algorithm†

Satyan R. Coorg
Synthetic Imagery Group
MIT Laboratory for Computer Science
satyan@graphics.lcs.mit.edu

1 Introduction

Identifying visible polygons or eliminating hidden polygons is an important component of efficient scene rendering algorithms. Despite the availability of high performance z-buffer hardware, a significant fraction of graphics machines have lesser or no z-buffering capabilities. For a machine with a software z-buffer (e.g. personal computers), this forms a significant bottleneck in rendering. Even if a graphics machine contains a hardware z-buffer, the z-test occurs *after* other graphics processing (shading, texture mapping), wasting hardware resources on invisible portions of the model.

One way to address this problem is by developing algorithms that efficiently identify, then render, only the visible portions of the model. In many models, such as a building or a city, much of the occlusion is caused by the presence of a small number of large occluders near the viewpoint. This sketch describes a novel algorithm that maintains a set of large occluders near the viewpoint, and uses the set to cull away occluded portions of the model. Full details of the algorithm are given in [1].

2 The Visibility Algorithm

Our algorithm is based on several ideas. First, we propose a simple (and fast) visibility test that identifies whether some region of the model (typically a kD-tree node) is completely/partially occluded by a set of occluders. This test is used in a hierarchical visibility algorithm in which it is repeatedly applied to determine the visibility status of tree nodes. Finally, the hierarchical visibility algorithm is combined with a cheap preprocessing step that identifies large occluders near all viewpoints.

2.1 Conservative Visibility Testing

In this section, we address the following problem: given a viewpoint, a set of convex occluders and a convex occludee, is the occludee visible? That is, is there a line segment from the viewpoint to some point on the occludee that meets no occluder? Our methods use the notion of *supporting* and *separating* planes. Separating planes of two convex polyhedra are planes formed by an edge of one polyhedron and a vertex of the other such that the polyhedra lie on opposite sides of the plane. Supporting planes are analogous, except that both polyhedra lie on the same side of the plane.

Consider the interaction between a single occluder and an occludee (typically a kD-tree node), which can be completely described in terms of the plane of the occluder, and the supporting and separating planes of the occluder and occludee (Figure 1-a). First, the occluder A can occlude T only if the viewpoint and T are on opposite sides of the plane of polygon A. This region can be divided into three qualitatively distinct regions as shown in the figure. In region 1, T is not occluded. In region 2, T is partially occluded, and in region 3, T is completely occluded. Thus, checking for occlusion is equivalent to determining which region (if any) contains the viewpoint and this can be accomplished by computing the signed distance of the viewpoint from each plane.

Figure 1-b shows occlusion caused by two connected occluders, i.e., two occluders that share a vertex (an edge in 3D). If the viewpoint is in the shaded region, T is occluded by the combined effect of A and B, even though neither occludes it alone. This case is handled by *ignoring* the supporting planes through a non-silhouette shared edge (labeled e), if both polygons adjacent to e partially occlude T.

The tests described in this section are *conservative*, i.e., they never classify a visible kD-tree node as occluded, though they may fail to detect some cases of occlusion (those caused by disconnected occluders, and those caused by connected occluders with non-convex silhouettes). However, we have found that these tests are sufficient to cull away significant fractions of models encountered in practice.

2.2 Hierarchical Visibility Algorithm

Applying the visibility test for every possible visual interaction (which could consume $O(n^2)$, for n polygons) is expensive. Instead, we use a hierarchical data-structure (a kD-tree) to organize the polygons in a model. At a given kD-tree node, the algorithm applies the visibility test to determine whether if the node is invisible. If the test succeeds, the algorithm returns without performing any further work. If it fails, the algorithm recurses with those occluders that partially occlude the kD-tree node, since only these occluders can occlude any descendant kD-tree node.

For a moving observer, the algorithm *caches* the occlusion relations – a list of supporting and separating planes – at each visited kD-tree node. When the viewpoint changes, the algorithm needs only *check* existing occlusion relations, and update only those kD-tree nodes whose visibility status has changed. Note that frustum culling can be incorporated

†The research described in this paper was supervised by Prof. Seth Teller.

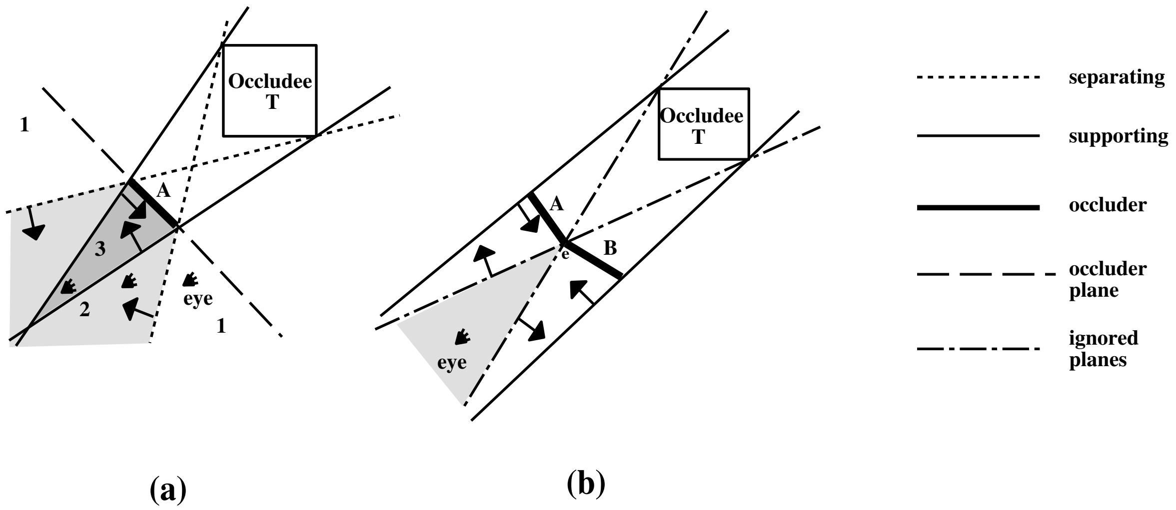

Figure 1: Figure (a) and (b) show occlusion in two dimensions (a planar cross section of 3D). Figure (a) shows separating and supporting planes of a single occluder. Figure (b) shows supporting planes that are ignored for two connected occluders.

into the algorithm in a straightforward way, by first checking whether the kD-tree node is inside the viewing frustum before invoking the visibility algorithm. Finally, the algorithm can be parallelized, yielding significant speedups [1].

If there are small "detail" objects present in the model, they are associated with the kD-tree node in which they are contained. If the node is visible, each object in the node may be tested for occlusion before it is drawn.

2.3 Dynamic Occluder Selection

The algorithm as described maintains the state of the kD-tree with respect to a fixed set of occluders. As the viewpoint moves, it is crucial to update this occluder set to contain those polygons that are "large" in the image, and therefore likely to occlude substantial portions of the model. Likewise, polygons that become small in apparent size could be ejected from the set.

A simple metric for the usefulness of a polygon as an occluder is its subtended solid angle. A reasonable estimate of this is the quantity $\frac{\vec{A}\cdot\vec{V}}{|\vec{D}|^2}$ (called *area-angle*) where A represents the (directed) area of the occluder, $\vec{V}$ represents the viewing direction,and $\vec{D}$ represents the vector from the viewpoint to the center of the occluder.

Using this metric, each kD-tree leaf (and a viewing direction) is associated with those k occluders that are likely to be most effective from viewpoints in the leaf. The number of occluders k is a parameter of the algorithm.

As interactive model viewing begins, the algorithm locates the kD-tree leaf which contains the initial viewpoint. Then, the algorithm uses the set of k occluders associated with the kD-tree leaf for the current viewing direction. When the viewpoint moves outside the kD-tree cell or the viewing direction changes substantially, the set of occluders is modified to correspond to the current viewing position and direction. Note that if the observer is moving smoothly, the set of occluders that are large from the current viewpoint is likely to be almost equal to the set of occluders that are large from the next viewpoint, and only a few changes need to be made to the occluder set.

3 Results

We have implemented the algorithm described in this sketch, as well as tools for visualizing its operation. For models consisting of a few thousand large occluders (and additional detail objects), the algorithm is able to maintain interactive rates on an SGI Indigo2$^{\mathrm{TM}}$ workstation (with a 100 MHz R4000 processor and 48 MB of main memory and a hardware z-buffer) and cull a significant fraction of the models.

We studied the performance of the algorithm on the following models: the fifth floor of Berkeley's Soda Hall building (Soda); a city from Viewpoint Datalabs (City); a number of tetrahedra on a plane (Tetra); and Soda with furniture (SodaFur). The results are averaged over the viewpoints visited during the walkthroughs. The kD-tree is of height 8 (around 250 nodes) and the number of occluders is fixed at 32 per direction (for 8 directions) in these experiments. Initialization and kD-tree construction used around 5 seconds CPU time. The results show that the algorithm can significantly reduce the rendering load, especially for models with many detail objects.

Scene	Polygons	Frustum	Occlusion
Soda	1685	0.31	0.17
City	2857	0.32	0.18
Tetra	3601	0.30	0.20
SodaFur	134832	0.30	0.02

Table 1: The column Frustum shows the fraction of polygons drawn with only view frustum culling. The column Occlusion shows the fraction of polygons drawn with frustum *and* occlusion culling.

References

[1] Coorg, S., and Teller, S. A Spatially and Temporally Coherent Object Space Visibility Algorithm. Tech. Rep. TM-546, Laboratory for Computer Science, MIT (available at `http://graphics.lcs.mit.edu/`), 1996.