Computing Conservative Multi-Focal Visibility using the z-Buffer

James Gregory Marsden Leonidas J. Guibas

Stanford University*

Abstract

Despite advancements in computer graphics technology, algorithmic model simplification plays a crucial role in manipulating and rendering complex polygonal models.

In this paper, we describe the theoretical justification for a new technique that combines nearby viewpoints into a volume, and uses these viewpoint volumes to simplify complex models by removing occluded polygons. Existing techniques calculate occlusion relationships between polygons based on geometric proximity measurements; this new technique determines hidden geometry based on the inability of occluded polygons to intercept visibility rays.

This technique can take advantage of hardware acceleration in the z-buffer to further accelerate removal of hidden surfaces.

1 Introduction

In complex geometric models only a fraction of the polygons are visible from any given viewpoint. Many of the polygons are invisible because they are occluded by larger polygons which lie closer to the viewpoint. While many applications rely on the graphics hardware to ascertain which polygons are visible, sending occluded geometry to the graphics pipeline slows the rendering process by burdening the pipeline with unnecessary computations that are not reflected in the scene. Because occluded geometry are not visible from the camera position, the integrity of the generated image can be preserved without rendering them.

Most occlusion culling techniques incorporate prior knowledge of the model to compute the set of relevant polygons. The few general solutions require large amounts of computation each time the viewpoint changes, and are unable to cache occlusion relationships between different viewpoints.

We attempt to address these concerns through the use of a *conservative multifocal z-buffer*, which provides a general solution to occlusion culling.

By exploiting proximity between viewpoints, the multifocal technique combines viewpoints into a volume for which occlusion relationships are computed. This computation remains valid as long as the viewpoint remains within the volume. To guarantee that only invisible polygons will be removed, we use conservative techniques which maintain that we strictly underestimate the effects of occluders and overestimate the size of occluded polygons.

The multifocal technique computes viewpoint correspondence by fixing a plane in space and calculating the viewing rays passing through the plane at every point. Summaries of the viewing ray bundles provide a metric for representing the occlusion relationships between polygons in the scene, and allow occluded polygons to be removed conservatively.

The multifocal z-buffer, which does not replace the conventional z-buffer, does not generate an absolute depth ordering for the polygons in the scene. Rather, it uses the fast memory of the graphics card to accelerate computation of complex occluder relationships within the scene. This flexible approach can operate off-line as a preprocessor, or provide incrementally refined sets of visible geometry as the application runs. No prior distinction between occluder and occluded geometry is necessary to the correct function of the multifocal z-buffer.

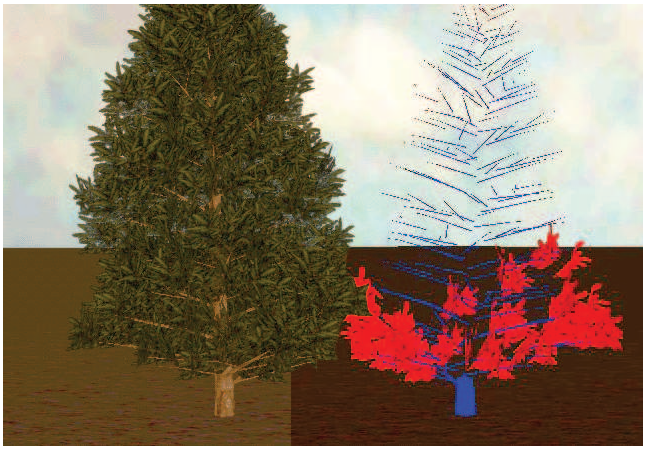

Figure 1: Multifocal z-Buffer
Self occlusion in a densely tesselated model. About 22% of the twelve thousand polygons are obscured (on right, in red).

This paper first reviews earlier efforts to remove hidden surfaces, specifically focusing on related occlusion culling algorithms. It then presents the theoretical justification for viewpoint correspondence, building upon that justification to develop a general, scalable, and extensible occlusion culling algorithm for volumes of viewpoints. In addition, we present techniques from computational geometry to further accelerate the application of the multifocal z-buffer. After summarizing our implementation protocol and experimental results, we briefly consider possible extensions of this work.

2 Background and Previous Work

At any viewpoint, few polygons in the actual model are visible. Even those polygons which are oriented toward the viewer are often blocked from line-of-sight by other polygons within the scene. If a hidden surface removal algorithm is not conservative, it can result in an incorrect image. However, without hidden surface removal model generation depends upon the brute-force capability of the hardware graphics card. The multifocal z-buffer removes occluded polygons from the graphics pipeline by computing complex occlusion relationships within large models and amortizing the computation cost over multiple viewpoints. A brief survey of hidden surface removal algorithms follows.

* *gmarsden@cs.stanford.edu,* guibas@cs.stanford.edu

2.1 Hidden Surface Removal

Hidden surface algorithms are often grouped into two categories, *image space* and *object space* methods.[de Berg 1993] Image-space algorithms operate at output level precision, meaning that the resolution of the output image and the final pixel resolution are the same. Increasing the resolution of the output image after it has been generated is not possible. Image space algorithms work in screen space, calculating the color of each pixel on the screen based on the color of the visible polygon closes to that pixel. Two basic methods are encountered for calculating the closest visible polygon—the first, *ray tracing*, calculates the closest polygon along a ray sent out by the viewpoint through each pixel. Ray tracing depends upon geometric information about the scene, which makes it difficult to accelerate in hardware. For this reason, it is rarely used for real time applications. The second technique, the *z-buffer*, depends solely on hardware acceleration and does not involve geometric information in its calculations. The z-buffer performs a brute-force search for the closest polygon at every pixel in the screen. The z-buffer stores a depth value for every pixel in the screen, and updates this value every time a pixel is drawn on the screen. Pixels are drawn if their depth value is less than the current depth value in the z-buffer. Even in the worst case, hardware acceleration makes this approach very efficient and makes the z-buffer the most commonly used hidden surface removal technique in computer graphics. More recently, researchers in computer graphics have looked for an image space technique to simplify very complex models. One example, a stochastic image space technique developed by Wand et al., performs random point sampling at pixel resolution to determine occlusion relationships and cull invisible geometry.[Wand et al. August 2001] Because it relies on stochastic sampling the resulting images are probabilistically accurate, but not geometrically conservative. As a result, the randomized z-buffer requires multiple passes to eliminate flicker between frames.

Object space algorithms operate on the geometry of the scene, and provide results which have provable accuracy. The most common object space visibility solution sorts objects in the scene by their depth from the camera and renders objects in order from furthest to nearest. This technique is commonly known as the *painter's algorithm*, because it paints objects on the screen from back to front. An absolute depth ordering for scene geometry does not always exist if polygons can overlap; to get around this problem, the painter's algorithm is often applied in conjunction with the z-buffer to provide the fastest possible rendering. Another common object space technique, the *binary space partitioning* (BSP) tree, accelerates computation by dividing the scene into cells with a fixed ordering relation. The BSP tree can be used to obtain a rough depth ordering for the scene.

Taking the occlusion relationships stored in a depth order, and deriving the set of occluded geometry in the scene is a difficult problem.

2.2 Occlusion culling

Neither the image space nor the object space techniques presented above do more than ensure the correctness of rendered scenes. When dealing with large models, we also want to discard polygons which are hidden from a given viewpoint—*occlusion culling*. In some cases, large polygons close to the viewpoint will occlude a large amounts of scene geometry. More often, many small polygons combine to form large occlusions. Full computation of the occluded geometry in a scene is equivalent to computing what polygons are visible from a given viewpoint, and a hard problem.

Occlusion culling techniques are common for rooms within a building, and these techniques have more recently expanded to include visible polygon determination in caves, tunnels, and on general terrain features.[Jones 1971; Stewart 1997]

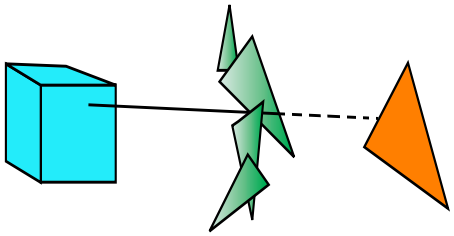

Figure 2: Complex occlusion relationships
Often, occlusion results not from a single large polygon, but from the interaction multiple smaller ones. Computing this interaction is a hard problem.

The specificity of the results of occlusion culling make it difficult to justify in practice. Small shifts in viewpoint can dramatically alter the set of visible polygons, and computation cannot be cached reliably from one viewpoint to the next. Nonetheless, viewpoint locality is the most attractive option for optimizing computation of occluded geometry in a scene. One method developed for computer vision applications divides the space of viewpoints into cells based on boundary events in the scene. Boundary events (edge events) occur when changes in viewpoint cause polygons to disappear or appear behind other occluders.[Plantinga and Dyer 1990] These viewpoint space partitions are useful in determining the stable viewable contours of an object, but become overly complicated in densely tessellated scenes where edge effects occur for every shift in viewpoint.

Bern et al. describe a technique for computing the visibility problem for an observer flying along a straight flight path, and determining the points at which the topology of the viewed scene changes.[Bern et al. 1994] While the search for topological discontinuities and edge events breaks down in densely tessellated scenes with many small triangles, the flight path and aspect graph techniques demonstrate two viable solutions to exploiting viewpoint locality within a scene in a method sensitive to scene geometry.

2.3 Volume visibility algorithms

Volume visibility algorithms exploit the spatial locality nearby of viewpoints to optimize occlusion culling. Unlike the previous techniques which relied on scene geometry to determine viewpoint bounding regions, recent work focuses on calculating the visibility for arbitrary volumes of viewpoints. Volume visibility can be seen as a three-dimensional extension of point-visibility which provide a potentially visible subset (*PVS*) of the original model. In a volume visibility application, the set of visible polygons for any viewpoint within the volume is always a subset of the *PVS* for the volume (See Equation 1 on the following page).

As a three-dimensional extension of point-visibility, volume visibility suffers additional constraints. The nonzero-width viewing volume can be larger than potential occluders within the scene, so the volume visibility method must be able to handle finite as well as infinite umbras cast by such objects.[Durand et al. July 2000] The *conservative weak visibility* constraint stipulates that polygons must be considered visible if any part is visible from any viewpoint within the volume.

Two recent techniques implement volume visibility, the first using scene discretization, and the second using extended occluder masks. In an application of scene discretization, Schauffler et al. propose to abandon the concept of polygons as occluders altogether and discretize the scene into a set of volume elements (voxels).[Schaufler et al. July 2000] They discretize the scene

into voxels which they classify as completely visible, completely occluded, or boundary cases. Working outward from the voxel containing the viewing volume, they determine complex occlusion relationships with respect to the viewing volume, and introduce a technique for treating nearby occluded voxels as occluders themselves. The latter extension allows occlusion to propagate away from the viewpoint throughout the scene. Scene discretization works best if the objects in the scene lend themselves to being divided into cubes, but is difficult to apply in scenes with very complex inter-occlusion relationships.

Using geometric proximity to compute occlusion from within a viewpoint volume, Durand et al. introduce a method for creating occlusion maps which can be placed arbitrarily through the scene and represent the *extended projection* of scene elements onto occluder planes.[Durand et al. July 2000] The extended projections can also carry occlusion information over between occlusion planes, and can compute both finite and infinite umbras within the scene. The extended projections can be implemented in the stencil buffer or expanded to compute projection volumes behind large occluders within the scene. H. L. Lim proposed a fuzzy visibility system which uses a projection plane at infinity to track viewing volume occlusion.[Lim 1992] He places a plane at infinity with each point storing the projected distance between scene geometry and the viewing volume, thereby capturing the geometric proximity between the volume and the polygon and generating a limited extended projection. Both Durand and Lim project geometry onto a distant plane and computing the volume visibility occlusion mask of the distance between the viewpoint and the projection planes.

In contrast to previous methods, the multifocal z-buffer technique introduced in this paper treats polygons as the basic occluder in the scene, but does not rely on geometric proximity measurements to calculate occlusion relationships. Instead, we develop a novel method for computing and storing viewpoint correspondence between distinct viewpoints within the viewing volume. The many techniques from computational geometry used to keep track of viewpoint correspondence are discussed in Section 5 on page 6.

3 Multifocal Volume Visibility

Multifocal volume visibility is a form of occlusion culling which simplifies large models based on inter-occlusion between polygons in the scene. The correctness of occlusion culling is easily proven because those polygons which are occluded do not participate in the final scene. Occlusion relationships even exist in scenes in which no single polygon obscures any other. Finding all occluded polygons in a scene, a hard problem itself, is equivalent to computing the exact set of visible geometry. The multifocal z-buffer solves this problem by exploiting what locality does exist between nearby viewpoints to compute a potentially visible set of polygons for a volume of viewpoints.

This technique is referred to as *multifocal* because it exploits the viewpoint correspondence between nearby viewpoints within the viewing volume by exploring their interaction with a coherence plane fixed in space. Rather than project scene polygons onto an occlusion plane, the multifocal z-buffer defines a ray-space perspectivity to compute which polygons can intercept visibility rays.

Computing a multifocal potentially visible set has immediate and obvious advantages. If occlusion culling can be applied to multiple viewpoints at once, the cost of computing the *PVS* would be spread across every viewpoint within the region. Though individual viewpoints have little coherence with respect to their *PVS*, the proximity of viewpoints in a volume can be exploited to generate an accurate *PVS*.

In practice, the multifocal z-buffer is efficient because the union of the *PVS*s for all the viewpoints in the volume remains smaller than the actual size of the model.

$$\forall v \in V : visible(v) \subseteq PVS \subseteq M \tag{1}$$

$$\bigcup_{v \in V} visible(v) \subseteq PVS \tag{2}$$

The worst case behavior for the multifocal z-buffer would be to send a *PVS* to the graphics pipeline which is the same size as M. If the visibility information were preprocessed, this would not affect the rendering time of the scene. In most cases the *PVS* will be much smaller than the size of the original model.

Equation 2 highlights the *conservative* property of the multifocal z-buffer: the *PVS* of the volume contains all triangles visible at any individual point within the volume. The correctness of the multifocal z-buffer depends upon the $\subseteq$ relation, and is maintained as long as no polygon is incorrectly removed from the *PVS*.

3.1 Multifocal occlusion masks

The occlusion mask was mentioned briefly as an *extended projection* in Durand et al. To create an occlusion mask, project scene geometry from the viewing volume onto a plane and use the umbra of the occlusion mask as a stencil to determine visible geometry. The occluder mask requires a known depth ordering or set of occluders to perform optimally.

The viewing volume occlusion mask is a volume of projective viewpoints, and the volume occlusion mask must summarize the combined interaction of all viewpoints. The standard solution uses the extended projection operator, which calculates which regions in the plane are completely obscured from the viewing volume. The penumbral portions of the plane are treated as unobscured portions of the mask when applied to the scene. The volume occlusion mask behaves like a projection surface onto which a light source the size of the viewing volume projects; those regions of the plane used as the occluder mask are completely dark. Use of extended projections as occluders is addressed in Section 2.3.

The multifocal occlusion mask does not use the extended projection operator, but attempts to define viewpoint correspondences between distinct viewpoints within the volume. Instead of creating an occlusion mask plane, this technique generates a viewing ray correspondence plane by defining a perspectivity between viewpoints.

3.2 The Viewpoint Correspondence Problem

To create a multifocal z-buffer, we must define a viewpoint perspectivity between distinct viewpoints which can then be applied in place of standard geometric proximity measurements to determine occlusion relationships within a scene. Defining viewpoint correspondence requires summarizing the effects of every viewing ray emerging from every viewpoint within the viewing volume. Unlike a conventional z-buffer, which deals only with a two-dimensional ray space at each pixel, the multifocal z-buffer must address all possible degrees of freedom for viewing rays emerging from the viewing volume.

To begin with a reduced form of the problem, we will define the correspondence between just two viewpoints v_1 and v_2. Fix a plane P in space. Then at any point $p \in P$, we define the perspectivity between the two viewpoints as the viewing rays $\overline{v_1 p}$ and $\overline{v_2 p}$ which pass through point p in the plane. There are three possible relationships between the viewpoints, P, and a potential occluder polygon R. If these rays intersect R, we can consider this ray bundle to completely summarize the occluding effect of R on v_1 and v_2. If the rays do not intersect the polygon, then the result is equally strong in the negative. Finally, if the polygon intersects only portion of the ray bundle, it must be considered visible. To

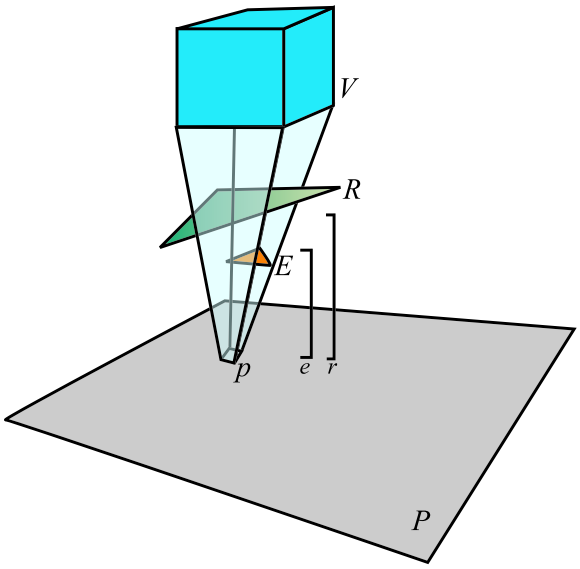

Figure 3: Multifocal Occlusion
$e < r$, therefore R occludes E from the viewing volume at p.

formally state this conclusion, a polygon R is visible if and only if it intersects a viewing ray between the viewpoint and some point p in the fixed correspondence plane.

This conclusion can be extended to the viewing volume problem. Consider a bundle of rays which emerges from viewing volume V and passes through a point p in the correspondence plane. If we define visibility as the ability to intersect visibility rays, then if no vector within the ray bundle intersects a given polygon R, R must be invisible from V because no visibility ray can ever intersect it. This definition of visibility is both weak, because if any portion of R is visible then R is visible, and conservative, because it will never incorrectly assume R is invisible.

Storing the entire vector bundle passing through point p is impractical, so it remains necessary to summarize the interaction between volume V, polygon R and plane P. This summary must also be conservative. For the sake of simplicity, we will consider occluders as triangles without loss of generality. Then the interaction summary between V, R, and p is the minimum distance along a viewing ray $|Rp|$. If this distance is defined, then the plane is visible. For any polygon with a lesser minimum distance along a visibility ray, that polygon must be obscured. The summary $|Rp|$ is conservative because no distance measurement can be less than the minimum distance, thus preserving the interaction summary as conservative as well.

The summary distance between the polygon and the correspondence plane further provides a summary of the occlusion properties of R at p (see Figure 3). A polygon E is occluded between R and p if and only if the maximum distance $|Ep|$ is less than the minimum distance $|Rp|$; in that case, R intersects all visibility rays at p and E is invisible. (See Figure 3)

There are no restrictions on the placement of the correspondence plane in the scene. Correspondence planes which are skew to the visibility volume may more accurately cull complicated geometry, and multiple planes can be utilized in a given scene. The correspondence plane can be placed anywhere within the scene, and multiple of such planes can be applied.

3.3 Assumptions

To simplify the discussion in this paper, the viewing volume is treated as an axis-aligned cube. In practice, this restriction is not necessary, and one could use a bounding rectangular parallelepiped without theoretical difficulty. A spherical viewpoint volume is also possible and would simplify the equations for the viewing-ray volume computation. However, spheres do not tessellate, as required by the next section.

The size of the viewing volume is wholly scene-dependent. If the viewing volume is too large, then the *PVS* will be the same as the model. If the viewing volume is too small, a moving viewpoint will violate the boundaries of the viewing volume more often, requiring more computation. It should be possible to create an adaptive viewing volume based on kinetic data structures to preserve geometric relationships, but such possibilities are not addressed in this paper.[Guibas and Stolfi 1985] We will assume that the size of the viewing volume is tuned to the specific application and model.

4 The Multifocal z-Buffer

The multifocal z-buffer uses the correspondence plane from the previous section to formulate a novel method for computing occluded scene geometry in a quick and incremental method. By formulating the problem in a manner which can be implemented in the z-buffer, this occlusion culling technique has immediate potential. Correspondence planes can be discretized without loss of generality or their conservative properties, and these summary results capture their interactions of potentially occluded polygons.

4.1 The Object-space z-Buffer

As described in Section 3.2, calculation of occlusion culling requires computation involving every point in the correspondence plane to ensure that occlusion culling operates accurately. This section outlines a method for compactly summarizing the correspondence data in a manner which can be implemented in existing hardware: the multifocal z-buffer.

The z-buffer is an inherently image space construct, and care must be taken to ensure that the multifocal z-buffer remains a conservative technique when using the z-buffer. The first step to transforming the correspondence plane into a z-buffer friendly abstraction is to discretize the plane. This discretization creates a grid of square pixels—not a grid of points—at a user-specified granularity. From each square pixel p, one calculates the minimum distance from a triangle to the pixel along a ray originating in the viewing volume. By storing this value in the corresponding z-buffer entry, one creates an easily accessible, conservative summary of the interaction between the square pixel and potential occluders.

It is not hard to show that this summary is conservative. A single square pixel represents a set of correspondence points contained in its area, and each one is a minimum distance summary. The absolute minimum distance ray is therefore a conservative estimate of any single distance stored for a given point. Although these smaller summary distances loose the upper bound on occlusion and may allow truly invisible polygons to be needlessly rendered, essential polygons will not be removed. The conservative claim is valid only if the volume represented between V and square pixel p does not contain any polygon edges: otherwise a conservative coverage map is necessary to retain the conservativeness of the algorithm. Conservative coverage maps are explained in the next section.

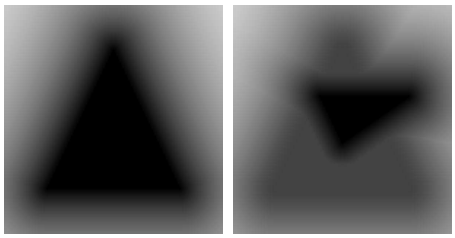

Figure 4: Correspondence maps
Multifocal z-buffer representations of single and overlapping triangles.

4.2 Conservative coverage maps

The multifocal occluder effect summary breaks down at polygon edge boundaries. Unless a polygon intersects the viewing ray volume's complete cross-section, the minimum distance summary does not accurately represent the interaction between the polygon and the correspondence plane. There are two alternatives for handling boundary cases: ignoring boundary cases or handling edge effects specially.

The simplest solution ignores the boundary cases. Although this approach is clearly not conservative, it is not necessarily inaccurate in practice. For a polygon to be marked as invisible, it must be obscured at every square pixel in the correspondence plane. With fine-enough granularity in the square pixel construction, this effect can be minimized. On the other hand, it is possible to construct a scene in which small polygons near the viewpoint are incorrectly discarded because of larger polygons closer to the correspondence plane. An application could proceed in a pseudo-conservative manner by keeping the largest incomplete distance summary. This approach does not qualify as conservative because it fails to underestimate pixels which are not completely occluded by some combination of scene geometry.

The conservative approach special cases the edge effects with conservative coverage maps. Those pixels which do not fall on polygon boundaries are treated as standard, geometrically summarized pixels. For pixels which lie on occluder boundaries, special coverage masks are created to represent pixel coverage information. Even though most of the computation carries over from the geometric transformation process, creation of the coverage maps is significantly more expensive than fully occluded buffer regions. For simplicity, the coverage masks need only consist of linear separators within the pixel to determine the appropriate viewing ray summary for that region of the square pixel. If there is more than one appropriate summary, the largest one is taken. The linear separators within the pixel mark off those regions for which different viewing ray summaries should be applied. By choosing a geometric method over a further pixel discretization, we allow for an adaptive level of geometric detail in the coverage maps without exacerbating the inaccuracies inherent in a discrete approach. The coverage mask need not store the identity of the occluder polygons; the distance ray summaries accurately identify occlusion relationships within the scene.

4.3 Volume-dependent distance

By computing the minimum conservative distance,we calculate the least overestimate of the minimum distance to the triangle. While it would be accurate and conservative to simply compute the smallest distance bound between a pixel and a scene polygon, we can achieve a tighter bound by calculating the minimum distance between the pixel and the polygon along a ray which passes through the viewing volume. This useful additional constraint complicates the distance problem and the solution is presented in Section 5.3 on page 7.

4.4 The occlusion test

The multifocal z-buffer accepts or rejects polygons based on the results of an occlusion test; to not pass the test, the object must be occluded. The occlusion test uses the discretized correspondence plane of the previous section to determine whether polygons can be intersected by visibility rays from the viewing volume. This occlusion test can be applied to individual polygons as well as large objects like object bounding boxes and scene impostors (placeholder polygons which are not rendered). Any object which passes the occlusion test, is placed in the *PVS* of the viewing volume.

The multifocal z-buffer applies the occlusion test to every polygon at every pixel in the correspondence plane. A polygon is only marked invisible if the occlusion test fails for every point in the plane. The actual mechanics of the test compare the minimum distance to the polygon (on the far side of the plane) or the maximum distance to the polygon (on the side of the plane nearest the viewing volume). If the maximum distance from the polygon to the pixel is less than the minimum distance from the occluder to the pixel, then the polygon can conservatively be declared occluded at that point. The minimum distance is stored at that pixel in the z-buffer. If a conservative correspondence map exists for that pixel the multifocal z-buffer will consult it. Alternatively, if the correspondence map has been initialized to a minimum value, the polygon summary being tested can be compared with the largest minimum distance in the pixel to get an accurate summary.

The occlusion test is symmetric for either side of the correspondence plane. If the occluder and the object are on the far side of the correspondence plane, the z-buffer stores the maximum distance to the occluder polygon and the occlusion test compares the maximum distance against the polygon's stored minimum distance. These tests are also valid if the occluder and the polygon are on opposite sides of the correspondence plane, though the special cases are more difficult when the occluder intersects the correspondence plane. With the conservative correspondence plane, the actual occlusion test is easy to apply to scene geometry or other primitives in the scene.

4.5 Dynamic occluder selection

The multifocal z-buffer can accept scene geometry as occluder and testing polygon simultaneously, or alternatively work with a predefined set of occluders and a independent set of potential occludees. Under the first description, the multifocal z-buffer will dynamically select occluders within the scene. Much like the operation of a normal z-buffer, polygons can be tested first for occlusion and then stored as potential occluders for geometry elsewhere in the scene.

A multifocal z-buffer used in this mode exhibits worst case behavior when the major scene occluders are tested last. An expensive workaround would be to apply the multifocal z-buffer twice. A more efficient solution to the problem applies existing occluder-selection algorithms to the scene and feed those polygons into the multifocal z-buffer at the front of the polygon queue. To avoid further problems, the multifocal z-buffer should be designed not to apply very distant polygons as potential occluders; this avoids the creation of many densely populated correspondence maps.

The multifocal z-buffer can also be used with multiple correspondence planes, as shown in Figure 5. In this case, an iteratively refined subset of polygons passes from one plane to the next and generating a reduced *PVS* at each step. Note that the actual

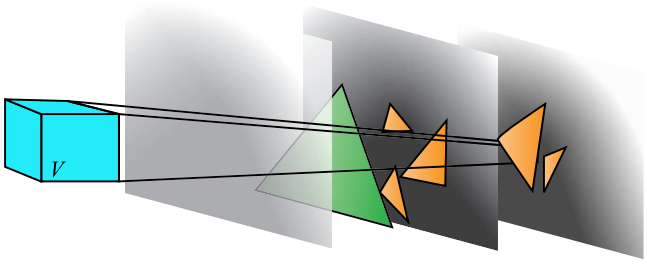

Figure 5: A look inside.
A cutaway showing how multiple correspondence planes can be used for a single viewing volume to share *PVS* data.

occlusion information is retained independently by each plane, and it is only the *PVS* which is communicated.

The multifocal z-buffer can operate with a list of potential occluders, or incrementally like a normal z-buffer, allowing operation incrementally on scene geometry with no prior knowledge of occluders in the scene.

5 Geometry

The multifocal z-buffer engages several techniques from computational geometry to solve the viewpoint coherence problem and accelerate the computation of the correspondence planes. Several interesting geometric challenges arise in the process of calculating the correspondence planes and performing the occlusion tests. The first, computing the minimum distance to a triangle, is greatly accelerated by using Voronoi diagrams to partition the space for quick feature lookups. The second is the computation of the distance between two objects such that the ray along which the distance is calculated passes through a third object in the scene.

The fundamental operation of the multifocal z-buffer is a form of scan conversion. The scan converter compares and selects the most conservative distance summary, carefully accounting for special cases which require conservative coverage maps. The distances computed by the scan converter are not strictly Euclidean—the orientation of the point to the correspondence plane is critical for performing the occlusion test. For simplicity of calculation, negative distances are considered to the far side of the correspondence plane.

5.1 Maximum distance

The scan conversion problem must compute both maximum and minimum distance between a polygon and a point. There are a number of cases in which we must compute a maximum distance. In the first, if a polygon completely covers the cross section of the volume of viewing rays, then the maximum distance between it and the correspondence plane lies along one of the edges of the viewing volume. The potential viewing rays along which the maximum distance will lie must originate from the corners of the viewing volume because the maximum distance between two convex polygons must lie between their vertices. This is the simplest case.

When a polygon does not completely cover the cross section of the viewing rays, the maximum distance must be computed for the portions of the polygon which fall within the viewing-ray volume. If the polygon is clipped to the viewing-ray volume, then because the maximum distance must fall between two vertices, the maximum distance between the clipped polygon and the pixel is easily computed.

If the polygon clipping is too expensive, the tight bound can be replaced with an absolute maximum distance without sacrificing conservatism.

5.2 Voronoi diagrams (Minimum distance)

The computation of the minimum distance from the correspondence plane to the polygon must use a different technique because the minimum distance ray is not restricted to a set of edges. The minimum distance can connect any combination of vertex, edge, and face, between the scene polygon and the square pixel in the correspondence plane. Because we can assume that all scene polygons are triangles, or subdivided into such, we precompute a fast lookup data structure to determine the closest point to a triangle feature, a Voronoi diagram.

A traditional Voronoi diagram locates the locus of points closest to a given point in a field of fixed points. Although, the classical Voronoi subdivision is not helpful for determining the closest polygon feature, the problem of closest point to a polygon was solved in $\Theta(n)$.[Preparata and Shamos 1985; Aggarwal et al. 1989] Several optimizations can be made to reduce the amount of precomputation necessary to generate a Voronoi graph suitable to the triangle-feature problem. Fortunately, this problem does not require the full $\Theta(n)$ solution outlined in Aggarwal et al.[Aggarwal et al. 1989]

Several optimization steps can be taken to reduce the amount of precomputation necessary to generate a polygonal Voronoi graph. The triangle subdivision problem is a subdivision of the general Voronoi polygon problem for two reasons. First, the three-dimensional Voronoi problem can be reduced to a two-dimensional point proximity problem by storing the diagram in the correspondence plane. The mechanics of this projection are nonstandard and will be revisited. Second, the number of edges and vertices are known beforehand, so the quad-edge structure does not need to be flexible.

Projection of the triangle onto the correspondence plane must do more than simply flatten the image—it must preserve the feature invariants. Without preserving these invariants, projecting the triangle onto the correspondence plane would discard orientation information essential to computing the Voronoi diagram. In order to preserve the orientation of the triangle, we first calculate the Voronoi subdivisions in the plane of the triangle and then extend the relevant edges of the diagram into perpendicular planes. Calculate the intersection of the supporting planes with the correspondence plane to recover the intersection information. These intersections are used to find the projected triangle vertex information, which is stored. in a simplified quad-edge structure. The Voronoi structure allows rapid determination of the closest triangle feature to a given point, which greatly accelerates minimum-distance computation.

In our implementation, we are only concerned with having a fast response for closest triangle feature queries. We use a simplified quad-edge structure, with a fixed number of vertices and edges, to avoid the normal cost of creating an extensible quad-edge structure.[Guibas and Stolfi 1985] A quad-edge structure normally tracks an arbitrary series of vertices and directed edges in a plane by recording the incoming and outgoing edges. This data structure allows fast computation of ‘edge algebra’ like intersections and region subdivision. In our implementation, we are only concerned with having fast response to being queried for closest triangle feature. The quad-edge allows a quick response to queries for triangle features: a closest-edge feature lays to the right of three edges, a closest-vertex feature lies to the left of two particular edges, and the data structure can easily return the relevant triangle feature.

The Voronoi subdivision for the triangle in space is easily achieved. For any two triangle points p_1 and p_2, the first subdivi-

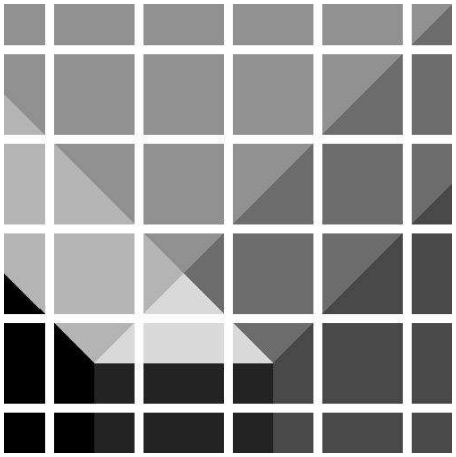

Figure 6: Voronoi subdivision
Gray levels map points in the plane to the nearest triangle feature. The mapping is correct for triangles skew to the correspondence plane.

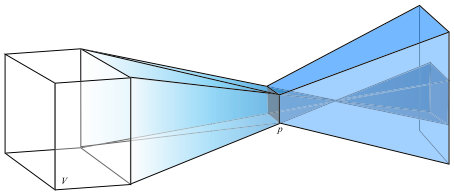

Figure 7: Viewing-ray bundle
The eight planes which define the viewing-ray cone are depicted here. Blue planes originate from the 'inside' corners of the V and cyan planes originate from the 'outside.' They converge at square pixel p in the correspondence plane.

	NumPolys	ProcessingTime	*PVS* size	Reduction
1	12439	90.562s	9491	23.6%
2	74635	93.381s	9491	87.2%
3	385604	2646.030s	296143	23.2%

Figure 8: Experimental Results
Results from running the multifocal z-buffer with bounding box optimizations and dynamic occluder selection.

sion plane has basis p_1 and normal $\overrightarrow{p_1p_2}$, and the second subdivision plane has basis p_2 and normal $\overrightarrow{p_2p_1}$. The vectors in the final projection must be oriented in the proper direction to facilitate the creation of the quad-edge diagram. Fortunately, orientation is reconstructed along with vertex information when the quad-edge diagram is constructed in the correspondence plane.

An example of a projected Voronoi subdivision appears in Figure 6. Shades of gray correspond to closest triangle features. The Voronoi and quad-edge structures allow rapid discovery of the closest triangle features, thereby greatly reducing the number of operations necessary to derive minimum distance calculations.

5.3 Volumetric distance

To preserve the tight bound on occluders, we ensure that the minimum distance between the correspondence plane and a scene triangle is calculated along a viewing ray which originates in the viewing volume. For simplicity, it is easier to think of this problem as a two object intersection in three dimensions. Therefore, we define the viewing-ray bundle as the volume containing all viewing rays which originate in the viewing volume and pass through square pixel p in the correspondence plane. The rectilinear cone which results is illustrated in Figure 7. Within this volume, we can reformulate the problem as a simple minimum distance problem.

By generating the viewing-ray volume, we effectively create a shadow volume for a polygonal light source and determine where the intersection points lie in relation to the shadow volume.[Nishita and Nakamae 1983] Because the object of interest is a square pixel and not the whole of the occlusion plane, we are dealing with a very reduced instance of the shadow volume problem. The viewing-ray volume is delimited by the eight planes visible in Figure 7, four to the left of p and four to the right.

The eight planes define a rectilinear cone whose extents are made up by the viewing volume and the square pixel. Depending on the orientation of the triangle with relation to the correspondence plane, the appropriate set of planes is intersected with the plane of the triangle. The intersection points indicate what portions of the triangle lie within the viewing-ray volume, and the intermediate results facilitate clipping the polygon to the viewing volume to get tighter measurements on the correspondence summaries.

6 Performance and Implementation

An ideal implementation of the multifocal z-buffer would follow one of two models. The first would be to preprocess the model for all possible viewpoints within the scene. Preprocessing every viewpoint within a model is a formidable task, and saving the occlusion relationships would require a great deal of storage. Nonetheless, if the viewpoint is restricted to a reasonable fraction of the model, preprocessing is a viable implementation decision.

A significantly better solution implements the multifocal z-buffer as a parallel process run on a secondary CPU, while the main processor handles graphics and interactivity for the application. The application feeds polygon data to the multifocal z-buffer, which incrementally marks the data in shared memory as visible or occluded. The multifocal z-buffer, because it does not distinguish between potential occluder and occluded geometry, can work to refine the potentially visible set at below frame rates while the application can benefit from the intermediate results as well as the final tight *PVS*. As more polygons are sent to the multifocal z-buffer, the multifocal system incrementally improves the accuracy of the correspondence plane representation. Applications can also send conservative bounding boxes to the pipeline in place of polygons to further reduce polygon burden. Operating below frame rates also allows the use of read-back features from the z-buffer and relaxes the programming constraints on the multifocal z-buffer. Because the z-buffer is optimized for writing and not reading, algorithms which attempt to use the fast memory of the z-buffer often fail if they must read-back from the video memory at frame rates. The multifocal z-buffer requires read-back from the z-buffer, but at significantly slower than frame rates.

The multifocal z-buffer mentioned in the experimental results was implemented in software, with 100 by 100 pixel clipping planes placed parallel to the edges of the axis aligned viewing volumes. These decisions were made solely for precision in implementation, and do not reflect on any inherent limitation in the multifocal z-buffer technique. Because it is an image space algorithm, alpha texturing (transparent polygons) pose a significant problem to the conservative nature of the multifocal z-buffer. We did not explore this problem, but believe it can be solved by an extension of the conservative correspondence maps.

Figure 8 shows the polygon culling ability of the multifocal z-buffer with a viewing extent of 0.02 units with multiple 12,000 polygon trees, preprocessed on a 1 GHz Pentium III with 256 MB RAM. Row 1 is a single tree, demonstrating the ability of the multifocal z-buffer to capture self occlusion within a triangle mesh. Six trees are placed in line with the viewing volume in Row 2. Because the algorithm tests bounding boxes before individual polygons, the running time is very similar to test #1. Bounding boxes can be used to replace polygons for occlusion testing, but because they are conservatively larger than occluding polygons, cannot because used as occluders. The last row shows multifocal occlusion performance on a scene with high depth complexity, showing the effects of both bounding box occlusion culling and polygon occlusion culling.

7 Conclusions and Future Work

In this paper, we presented an algorithm that uses the properties of volumes of viewpoints to render large polygonal models more quickly. This algorithm provides a potentially visible set of polygons which overestimates the visible polygons for any single viewpoint. The occlusion relationships computed by the multifocal z-buffer remain valid for any viewpoint within the viewing volume, so occluded geometry can be discarded conservatively. The multifocal z-buffer operates successfully without prior knowledge of the scene and minimizes the cost of shifting viewpoints. This paper justifies the multifocal technique on theoretical grounds and presents a general solution for volume visibility occlusion culling.

As with any innovation, many enhancements are possible. For example, systems could greatly benefit from adaptively inserting correspondence planes within the scene based on density of geometry. Existing algorithms for calculating potential occluders could be used in the placement of correspondence planes to maximize occluder detection. Because the correspondence planes can function using only relevant subsets of input geometry, a binary space partition (BSP) could significantly reduce the polygon load on the multifocal z-buffer. Additionally, in combination with a BSP, a kinetic data structure could allow occlusion relationships within individual cells of the BSP tree to remain valid even as the viewpoint crosses the boundaries of the volume. The volume visibility computation can allow networked applications to download relevant subsets of a larger polygonal model, making networked interactive walkthroughs much more practical.

Furthermore, the viewpoint correspondence technique developed for the multifocal z-buffer has applications to many other areas of graphics research. The depth summaries can be used to assist computation of effects like fog and the soft shadows produced by volume light sources.

The multifocal z-buffer is able to address volumes of viewpoints by defining a correspondence plane, which summarizes the interaction between scene geometry and the viewing volume. This novel method encodes the correspondence between nearby viewpoints to allow for generalizations about the impact of polygons with all possible viewing rays emerging from the viewpoint volume. By computing the multifocal z-buffer, which establishes the combined occlusion relationships for all viewpoints within the volume, the cost of this computation can be distributed over time as long as the viewpoint remains within the volume.

Acknowledgements

This project was made possible by the *CURIS* program and the Stanford URO/URP. I am especially indebted to the remarkable patience of my readers and the encouragement of my parents.

List of Figures

References

AGGARWAL, A., GUIBAS, L., SAXE, J., AND SHOR, P. 1989. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete and Computational Gemoetry 4*, 591–604.

BERN, M., DOBKIN, D., EPPSTEIN, D., AND GROSSMAN, R. 1994. Visibility with a moving point of view. *Algorithmica 11*, 360–378.

DE BERG, M. 1993. *Ray Shooting, Depth Sorting, and Hidden Surface Removal*. Springer-Verlag Lecture Notes in Computer Science.

DURAND, F., DRETTAKIS, G., THOLLOT, J., AND PUECH, C. July 2000. Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000*. Held in New Orleans, Louisiana.

GUIBAS, L., AND STOLFI, J. 1985. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics 4*, 2, 74–123.

JONES, C. B. 1971. A new approach to the 'hidden line' problem. *The Computer Journal 14*, 3, 232–..

LIM, H. L. 1992. Toward a fuzzy hidden surface algorithm. In *Computer Graphics International*, 621–635.

NISHITA, T., AND NAKAMAE, E. 1983. Half-tone representation of 3-d objects illuminated by area sources or polyhedron sources. *IEEE COMPSAC*, 237–242.

PLANTINGA, H., AND DYER, C. 1990. Visibility, occlusion, and the aspect graph. *International Journal of Computer Vision 5*, 2, 137–160.

PREPARATA, F., AND SHAMOS, M. 1985. *Computational Geometry An Introduction*. Springer-Verlag, New York.

SCHAUFLER, G., DORSEY, J., DECORET, X., AND SILLION, F. July 2000. Conservative volumetric visibility with occluder fusion. *Proceedings of SIGGRAPH 2000*. Held in New Orleans, Louisiana.

STEWART, A. J. 1997. Hierarchical visibility in terrains. In *Eurographics Rendering Workshop*, Eurographics.

WAND, M., PETER, I., STRASSER, W., FISCHER, M., AND HEIDE, F. August 2001. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. *Proceedings of SIGGRAPH 2001*. Held in Los Angeles, California.