Conservative Visibility Preprocessing using Extended Projections

Frédo Durand†‡, George Drettakis†, Joëlle Thollot†and Claude Puech †

†iMAGIS*- GRAVIR/IMAG - INRIA ‡Laboratory for Computer Science - MIT

Abstract

Visualization of very complex scenes can be significantly accelerated using *occlusion culling*. In this paper we present a visibility preprocessing method which efficiently computes potentially visible geometry for volumetric viewing cells. We introduce novel *extended projection* operators, which permits efficient and conservative occlusion culling with respect to all viewpoints within a cell, and takes into account the combined occlusion effect of multiple occluders. We use extended projection of occluders onto a set of projection planes to create extended occlusion maps; we show how to efficiently test occludees against these occlusion maps to determine occlusion with respect to the entire cell. We also present an improved projection operator for certain specific but important configurations. An important advantage of our approach is that we can re-project extended projections onto a series of projection planes (via an *occlusion sweep*), and accumulate occlusion information from multiple blockers. This new approach allows the creation of effective occlusion maps for previously hard-to-treat scenes such as leaves of trees in a forest. Graphics hardware is used to accelerate both the extended projection and reprojection operations. We present a complete implementation demonstrating significant speedup with respect to view-frustum culling only, without the computational overhead of on-line occlusion culling.

KEYWORDS: Occlusion culling, visibility determination, PVS

1 Introduction

Visualization of very complex geometric environments (millions of polygons) is now a common requirement in many applications, such as games, virtual reality for urban planning and landscaping etc. Efficient algorithms for determining visible geometry are a key to achieving interactive or real-time display of such complex scenes; much research in computer graphics is dedicated to this domain. Object simplification using different levels of detail (LOD) (*e.g.*, [Cla76, FS93]) or image-based approaches (*e.g.*, [SLSD96]) have also been used to accelerate display, either as an alternative or in tandem with visibility algorithms.

Visibility *culling* algorithms try to reduce the number of primitives sent to the graphics pipeline based on occlusion with respect to the current view. In *view frustum culling* only the objects contained in the current view frustum are sent to the graphics pipeline.

*iMAGIS is a joint research project of CNRS/INRIA/UJF/INPG. E-mail: {Fredo.Durand|George.Drettakis|Joelle.Thollot|Claude.Puech}@imag.fr http://www-imagis.imag.fr/

Occlusion culling attempts to identify the visible parts of a scene, thus reducing the number of primitives rendered. We can distinguish two classes of occlusion culling: *point-based* methods which perform occlusion culling on-the-fly for the current viewpoint, and *preprocessing* approaches which perform this calculation beforehand, typically for given regions (volumetric cells).

Point-based methods are very effective, and in particular can treat the case of occluder fusion, *i.e.* the compound effect of multiple occluders. This is important, for example in a forest: each individual leaf hides very little, but all the trees together obscure everything behind them. However, point-based methods have significant computational overhead during display, and cannot be simply adapted for use with pre-fetching if the model cannot fit in memory.

No previous preprocessing method exists which can handle occluder fusion. In addition, the success of such preprocessing methods is often tied to the particular type of scene they treat (*e.g.*, architectural environments [TS91, Tel92]).

In this paper we present a visibility preprocessing algorithm based on a novel *extended projection* operator, which generalizes the idea of occlusion maps to volumetric viewing cells. These operators take occluder fusion into account, and result in an occlusion culling algorithm which is efficient both in memory and computation time. Our algorithm results in a speedup of up to 18 compared to optimized view-frustum culling only. In addition, using repeated re-projection onto several projection planes, we can treat particularly difficult scenes (*e.g.,* forests), for which we obtain speedups of 24, again compared to view-frustum culling. An example is shown in Fig. 1.

1.1 Previous work

It is beyond the scope of this paper to review all previous work on visibility. Comprehensive surveys can be found in *e.g.*, [Dur99]. In particular we do not review analytical 3D visibility methods (*e.g.*, [PD90, Dur99]) because their algorithmic complexity and their robustness problems currently prevent their practical use for large scenes. In what follows, we first briefly overview preprocessing occlusion culling algorithms and then discuss point-based approaches.

Occlusion culling techniques were first proposed by Jones [Jon71] and Clark [Cla76]. Airey *et al.* [ARB90] and Teller *et al.* [TS91, Tel92] were the first to actually perform visibility preprocessing in architectural environments. They exploit the fact that other rooms are visible only through sequences of *portals* (doors, windows). These methods have proven very efficient in the context of walkthroughs where they can be used in conjunction with LOD [FS93] for faster frame-rates. The problem of data size and the consequent treatment of disk pre-fetching and network bandwidth has been addressed, notably by Funkhouser (*e.g.*, [Fun95, Fun96]). Applications to global lighting simulation have also been demonstrated, *e.g.*, [TH93]. Unfortunately these methods rely heavily on the properties of indoor scenes, and no direct generalization has been presented. Visibility for terrain models has also been treated (*e.g.,* [Ste97]).

Preprocessing algorithms capable of treating more general scenes have recently begun to emerge (*e.g.*, [COFHZ98, COZ98, WBP98]). Nonetheless, they are currently restricted to occlusions caused by a single convex occluder at a time. Since they cannot

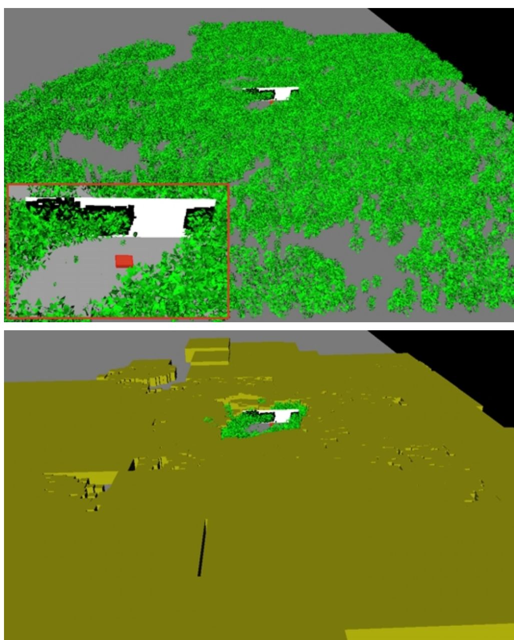


Figure 1: Top: A 7.8 M poly forest scene (only a tenth of the leaves are rendered here). The inset shows a close-up on the viewing cell (in red) and one of the projection planes. Bottom: Visibility computation from the view cell. In yellow we show the nodes of the hierarchy of bounding boxes culled by our method. We also show one of the projection planes used.

handle the general case of occluder fusion they compute potentially visible sets which are often very large. However, the technique by Schaufler *et al.* [SDDS00] in this volume can also handle occluder fusion for volumetric visibility.

A novel approach using conservative occluder simplification has also recently been proposed [LT99] to decrease the cost of occlusion preprocessing.

On the other hand, point-based methods have been proposed which perform an occlusion culling operation for each frame on-the-fly. Greene *et al.* [GKM93] create a 2D hierarchical z-buffer, used in conjunction with a 3D octree hierarchy to accelerate the occlusion of hidden objects. A related algorithm using hierarchical occlusion maps was introduced by Zhang *et al.* [ZMHH97], which uses existing graphics hardware. This approach includes "important" blocker selection and approximate culling for regions almost occluded. It is important to note that these approaches only work for a single given viewpoint and are thus unsuitable for precomputing visibility information; In a different vein, Luebke and George [LG95] presented an algorithm which extends view-frustum culling by restricting views through convex portals. Wonka and Schmalstieg [WS99] use a z-buffer from above the scene to perform occlusion culling in terrains.

Other point-based methods include the work by Coorg and Teller [CT96, CT97] which use spatial subdivision and the maintenance of separating planes with the viewpoint to achieve rapid culling of hidden surfaces. A similar technique is presented by Hudson *et al.* [HMC$^+$97].

1.2 Overview

Our visibility preprocessing algorithm adaptively subdivides the scene into *viewing cells*, which are the regions of observer movement. For each such cell, we compute the set of objects which are potentially visible from all the points inside the cell. This set is called the *PVS* or potentially visible set [ARB90, Tel92, TS91].

To compute these sets efficiently, we introduce a novel *extended projection* operator. If we consider each individual viewpoint in a cell, the extended projections are an *underestimate* of the projection of occluders, and an *overestimate* for the occludees. By defining these operators carefully, we can check if an occludee is hidden with respect to the *entire* cell by simply comparing the extended projections of the occludee to the extended projections of the occluder. Extended projections can handle occluder fusion and are efficient in the general case. We also present an improved extended projection for occludees for specific, but not uncommon configurations. Once occluders have been projected into a given projection plane, we can *re-project* them onto other planes, aggregating the effect of many small occluding objects. This *occlusion sweep* allows us to create occlusion maps for difficult cases such as the leaves in a forest.

The rest of this paper is organized as follows. In the next section we define the *extended projection* operators, and show how to compute them (section 3). An improvement for specific cases is presented in section 4. We then introduce a reprojection operator and the occlusion sweep in section 5. In section 6 we describe the preprocess to compute PVS's and discuss the interactive viewing algorithms which use the result of the preprocess. In section 7 we present the results of our implementation, together with a discussion and future work. The interested reader will find more details in the extended version present in the proceedings CD-ROM or on the authors' web page, as well as in the first author's thesis [Dur99].

2 Extended projections

To compute the potentially visible geometry in every direction for a given viewing cell, we perform a conservative occlusion test for each object (occludee) of the scene, with respect to every viewpoint in the cell. To do this efficiently, we use a representation of occlusion caused by occluders which is based on *extended projections* onto a plane.

2.1 Principle

In point-based occlusion culling algorithms [GKM93, ZMHH97], occluders and occludees are projected onto the image plane. Occlusion is detected by testing if the projection of an occludee is contained in the projection of the occluders (overlap test) and if this occludee is behind (depth test) for the given viewpoint.

Our approach can be seen as an extension of these single viewpoint methods to volumetric viewing cells. This requires the definition of *extended projection operators* for occludees and occluders. To determine whether an occludee is hidden with respect to all viewpoints within the viewing cell the new projection operators need to satisfy the following conditions: (i) the extended projection of the occludee must be contained in the extended projection of the occluders and (ii) the occludee must be behind the occluders.

Even though we describe our method for a single plane, six planes will actually be necessary to test occlusion in all directions. The position of the projection plane is an important issue and will be discussed in section 6.1.

We define a *view* as the perspective projection from a point onto a projection plane. However, in what follows, the projection plane will be shared by all viewpoints inside a given cell, resulting in sheared viewing frusta.

2.2 Extended projections

We next define extended projection operators for both occluders and occludees using *views* as defined above.

Definition 1 *We define the extended projection (or* $\mathcal{P}$*rojection) of an* **occluder** *onto a plane with respect to a cell to be the* **intersection** *of the views from any point within the cell.*

Definition 2 *The extended projection (or* $\mathcal{P}$*rojection) of an* **occludee** *is defined as the* **union** *of all views from any point of the cell.*

In what follows, we will simply use *Projection* to refer to an extended projection. The standard projection from a point will still be named *view*.

Fig. 2 illustrates the principle of our extended projection. The *Projection* of the occluder is the intersection of all views onto the projection plane (dark gray), while the *Projection* of the triangular occludee is the union of views, shown in light green.

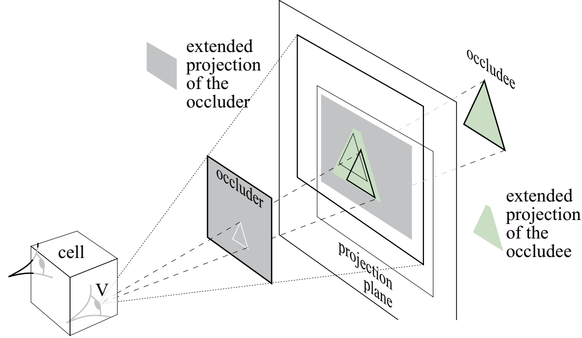


Figure 2: Extended projection of an occluder and an occludee. The view from point V is shown in bold on the projection plane. A view from another viewpoint is also shown with thinner lines. The extended projection of an occluder on a plane is the *intersection* of its views. For an occludee, it is the *union* of the views.

This definition of *Projection* yields conservative occlusion tests. To show this consider the case of a single occluder (Fig. 2). Assume (for the purposes of this example) that an occludee is behind the occluder. It is declared hidden if its *Projection* is contained in the *Projection* of the occluder. This means that the union of the occludee views is contained in the intersection of the views of the occluder. From any viewpoint V inside the cell, the view of the occludee is contained in the view of the occluder.

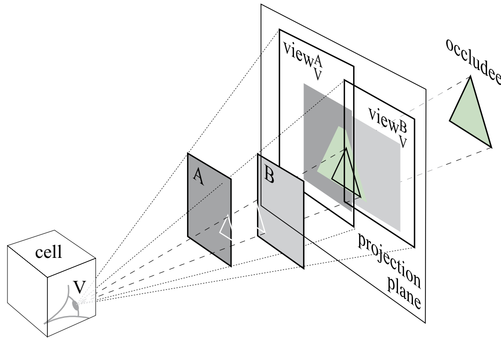


Figure 3: *Projections* handle *occluder fusion* of two occluders A and B. We show the example of a view from point V. The view of the occludee is contained in the cumulative view of the two occluders, as determined by the *Projection* occlusion test.

Consider now the case of an occludee whose *Projection* is contained in the cumulative *Projection* of two (or more) occluders. This means that from any viewpoint V in the cell, the view of the occludee is contained in the cumulative view of the occluders. To summarize, we have:

$$view_V(occludee) \subset \bigcup_{V \in cell} view_V(occludee)$$

$$\subset \bigcup_{occluders} \bigcap_{V \in cell} view_V(occluder)$$

$$\subset \bigcup_{occluders} view_V(occluder)$$

The occludee is thus also hidden in this case (see Fig. 3). Our *Projection* operators handle *occluder fusion*. We do not however claim that they always find *all* occluder fusions; as will be discussed in section 7.3, the position of the projection plane is central. Note that convexity is not required in any of our definitions, just as for point-based occlusion-culling.

2.3 Depth

Unfortunately there is no one-to-one correspondence between a point in a *Projection* and a projected point of an object, as with a standard perspective view (see Fig. 4(a)). We can see that many depth values correspond to a single point in the *Projection*, depending on which point of the viewing cell is considered.

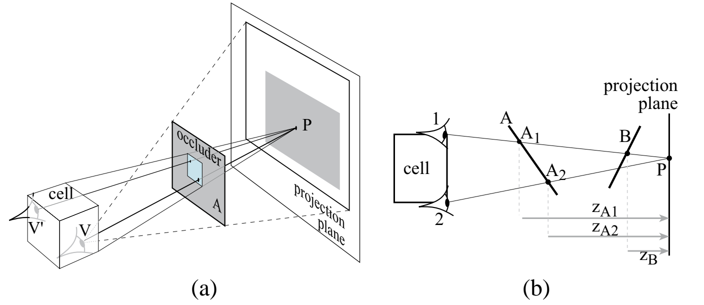


Figure 4: (a) We show the points of the occluder corresponding to P in the view from V and V'. The light blue region of the occluder corresponds to the set of points which project on P. (b) The *Depth* of a point P in the *Projection* of an occluder A is the maximum of the depth of the corresponding points (z_{A2} here). In the case of multiple occluders, the occluder closest to the cell is considered (occluder A here). Note that *Depths* are negative here.

Depth comparison is more involved than in the single viewpoint case since a simple distance between the viewpoint and a projected point cannot be defined (see Fig 4(a)). Our definition of depth must be consistent with the properties of occlusion; For each ray emanating from a viewpoint inside the cell and going through the projection plane, depth must be a monotonic function of the distance to the viewpoint. We define depth along the direction orthogonal to the projection plane. We chose the positive direction leaving the cell and placed zero at the projection plane.

Definition 3 *We define the extended depth (or Depth) of a point in the Projection of an* **occluder** *as the* **maximum** *of the depth of all the corresponding projected points.*
Similarly, the extended depth (Depth) of a point in the Projection of an **occludee** *is the* **minimum** *depth of its projected points.*

See Fig. 4(b) for an illustration where for point P and occluder A, depth z_{A2} is maximum and is thus used as *Depth*.

If the *Depth* of a point in the *Projection* of an occluder is smaller than the *Depth* of the point in the *Projection* of an occludee, all the corresponding points of the occludee are *behind* the occluder from any viewpoint inside the cell. As a result, this definition of *Depth* satisfies our conservative depth test requirement and yields valid occlusion computation.

We construct a *Depth Map* as follows: For each point of the projection plane, we define the value of the *Depth Map* as the *Depth* of the occluder closest to the cell which projects onto it (that is the occluder with the *minimum Depth*). In the example of Fig. 4(b), occluder A is closest to the cell, and thus chosen.

2.4 Implementation choices

Until now, our definitions have been quite general, and do not depend on the cell, plane, occludee nor on the way that we test for containment of an occludee *Projection* in an occluder *Projection*, or the way that *Depth*s are compared. We now present the choices we have made for our implementation.

The viewing cells are non-axis-aligned bounding boxes. The projection planes will be restricted to the three directions of the cell

(note that these three directions depend on the cell). The occludees are organized in a hierarchy of axis-aligned bounding boxes.

The projection planes we use are actually not infinite but are finite rectangles. We use a pixel-map representation of the *Depth* Map. This may at first seem like a concession to conservatism, but we will see in section 3.2 that a conservative rasterization is used. This allows the use of the graphics hardware simplifying most of the computation, and it avoids the robustness issues inherent to geometrical computations.

We store a *Depth* value for each pixel of the *Depth* Map. As described above, for each pixel we consider the closest occluder, *i.e.* the minimum *Depth*. Occluder fusion is handled by the natural aggregation in the *Depth* Map. Following [GKM93] we organize the *Depth* Map into a pyramid for efficient testing. We call it the Hierarchical *Depth* Map.

3 Computation of extended projections

Using the extended projections we defined, we can efficiently test occludee *Projections* against occluder *Projections* in a preprocess to find the potentially visible geometry for viewing cells. We next describe how to compute *Projections* for occludees and then for occluders (both convex and concave).

3.1 Occludee *Projection*

Recall that the *Projection* of an occludee is the union of its views. Our cells are convex as is the bounding box of an occludee. The *Projection* of such a box reduces by convexity to the 2D convex hull of its views from the vertices of the cell.

To simplify computation, we use the bounding rectangle of the *Projection* on the projection plane as an *overestimate* of the *Projection* (see Fig. 5). We then split the problem into two simpler 2D cases. We project the cell and the occludee bounding box onto two planes orthogonal to the projection plane and parallel to the sides of the cell. The 2D projection of the cell is a rectangle, while the 2D projection of the occludee bounding box is a hexagon in general (Fig. 5 shows a special case of a quadrilateral for simplicity).

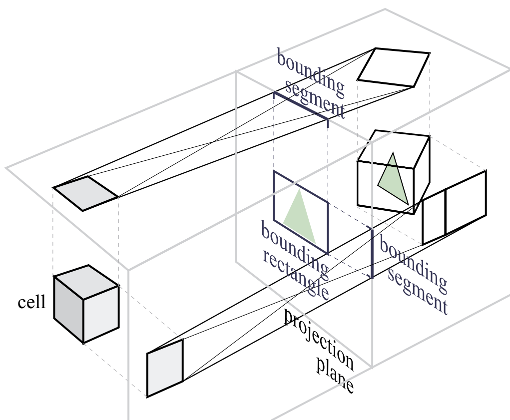


Figure 5: Occludee *Projection* is reduced to two 2D problems.

We then compute the separating and supporting lines [CT97] of the rectangle and hexagon. The intersections of these lines with the projection plane define a 2D bounding segment. A bounding rectangle on the projection plane is defined by the Cartesian product of the two 2D segments as illustrated in Fig. 5. Separating lines are used when the occludee is between the cell and the projection plane, while supporting lines are used if the occludee lies behind the plane.

This method to compute an occludee *Projection* is general and always valid, but can be overly conservative in certain cases. In section 4 we will present an improvement of this *Projection* for some particular, but not uncommon, configurations.

3.2 *Projection* of convex occluders using intersections

By convexity of the cell and occluder, the intersection of all possible views from inside the cell is the intersection of the views from the vertices of the cell. This *Projection* can be computed using standard geometric intersection computation.

We have nevertheless developed an efficient method which takes advantage of the graphics hardware. It is a multipass method using the *stencil buffer*. The stencil buffer can be written, and compared to a test value for conditional rendering.

The basic idea is to project the occluder from each vertex of the cell, and increment the stencil buffer of the projected pixels without writing to the frame-buffer. The pixels in the intersection are then those with a stencil value equal to the number of vertices.

The consistent treatment of *Depth* values (as described in section 2.3) in the context of such an approach requires some care. More details on the hardware implementation are given in appendix A.

If standard OpenGL rasterization is used, the *Projections* computed are not conservative since partially covered pixels on the edges may be drawn. We use a technique proposed by Wonka *et al.* [WS99] which "shrinks" polygons to ensure that only completely covered pixels will be drawn. Each edge of a displayed polygon is translated in the 2D projection plane by a vector of (+/-1 pixel, +/-1 pixel) towards the interior of the polygon (the sign is determined by the normal). The 2D lines corresponding to the edges are translated and the vertices are computed by intersection. Note that only silhouette edges need be translated. If the polygons to be displayed are too small, the shrinking results in a void *Projection*.

3.3 Concave occluder slicing

Concave polygonal meshes can be treated in our method, by computing the *Projection* of each individual triangle of the mesh. However, some gaps will appear between the *Projections*, resulting in the loss of the connectivity of occluders. To overcome this problem, we use the following simple observation: the *Projection* of a closed manifold lying in the projection plane is the object itself. We thus consider the intersection of concave objects with the projection plane, which we call a *slice* of the object.

If the projection plane cuts the object, we compute the intersection and a 2D contour found. The normals of the faces are used to accurately treat contours with holes. The contour is then conservatively scan-converted with the value of the *Depth* Map set to zero (*i.e.* the *Depth* value of the projection plane).

4 Improved *Projection* of occludees

In this section we present an improvement to the extended projection calculation for the case of *convex* or *planar* occluders for configurations in which our initial *Projection* yields results which are too conservative. In what follows, we discuss only the case where the occludee is between the projection plane and the cell. If the occludee is behind the plane, the *Projection* which we have presented in section 2.2 yields satisfying results.

In Fig. 6(a) we show a 2D situation in which our *Projection* is too restrictive. The *Projection* of the occludee is not contained in the *Projection* of the occluder, even though the occludee is evidently hidden from any viewpoint in the cell. As illustrated in Fig. 8(a), we will show that in this case we can use the supporting lines instead of the separating lines in the computation of the occludee *Projection*.

To improve our occlusion test, we will first discuss conditions required to prove that an occludee is hidden by an occluder. We will then deduce a sufficient condition on the occludee *Projection* to

yield valid tests. In 2D, this condition can be simply translated into the definition of an *improved Projection*. Based on this 2D construction, we develop an improved 3D occludee *Projection*, using a projection approach similar to that of section 3.1.

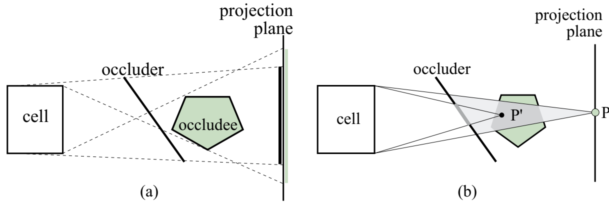


Figure 6: (a) Configuration where the initial *Projection* is too restrictive. The *Projection* of the occludee is not contained in the *Projection* of the occluder, even though it is obviously hidden. (b) Any point P' inside the cone defined by P and the cell and behind the occluder is hidden.

Property 1 *For a given point P in the occluder umbra region with respect to a cell, all points P′ in space behind the occluder, which are also contained in the cone defined by P and the cell, are hidden with respect to the cell.*

The *occluder umbra region* with respect to a cell is the umbra (totally hidden) volume which results if the cell is considered as an area light source. This property is illustrated in Fig. 7. The proof is evident for convex occluders since the cone defined by P' and the cell is contained in the cone defined by P. The section of the occluder which occludes P is a superset of the section which occludes P'.

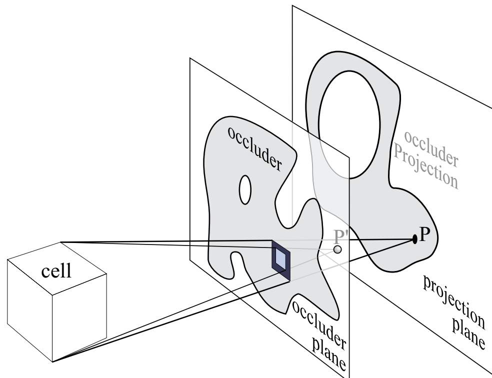


Figure 7: The intersection of the occluder and cone (defined by point P and the cell) is shown in dark blue.

The 3D case of concave planar occluders is similar to the convex case. Consider a point P in the umbra of a concave occluder (Fig. 7). Since P is in the umbra, the cone defined by P and the cell is "occluded": the intersection of this cone and the occluder is equal to the intersection of the cone and the plane of the occluder. The intersection of the cone defined by P' (the light blue inner square in Fig. 7) is a subset of this intersection. P' is thus also hidden.

Planarity of the occluder is required to ensure that the intersection of the cone and the occluder is convex. The planar occluder can be concave and have a hole (as in Fig. 7), but if P is in the umbra, the intersection is convex.

To yield valid occlusion tests, our *improved Projection* must have the following property:

Property 2 *The union of cones defined by each point of the improved* Projection *of the occludee and the visibility cell must contain the occludee.*

To see why this property is important, consider the cone defined by P and the cell in Fig 6(b). The points of the occludee contained in this cone are occluded by Property 1. Consider the union of cones defined by all the points of a hypothetical improved *Projection* and the cell. If the occludee is contained in this union of cones, any point of the occludee is in one of these cones, and is thus hidden by Property 1. Note that occluder fusion is still taken into account: all points P defining the cones need not be hidden by the same convex or planar occluder.

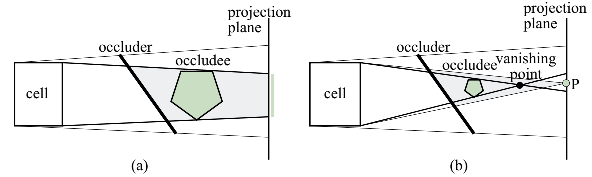


Figure 8: Improved *Projection* in 2D.

An improved *Projection* respecting property 2 is defined in 2D by considering the supporting lines of the cell and the occludee as illustrated in Fig. 8(a). However, if the occludee is too small, the two supporting lines intersect in front of the projection plane at the *vanishing point*. In this case, any point P between the intersections of the two supporting lines and the projection plane satisfies property 2, (Fig. 8(b)). In practice, we use the mid-point in our calculations.

Unfortunately, supporting planes cannot be used in 3D as simply as supporting lines, and the vanishing point is ill-defined. We thus project onto two planes orthogonal to the projection plane and parallel to faces of the cell, as illustrated in Fig. 9(a). On each plane we use our 2D improved *Projection*. The Cartesian product of these two 2D improved *Projections* defines our 3D improved *Projection*.

The 3D improved *Projection* is the Cartesian product of 2D improved *Projections* which are points or segments. It is a rectangle (segment $\times$ segment), a segment (segment $\times$ point) or a point (point $\times$ point).

5 Occluder reprojection and occlusion sweep

In the previous sections, we have limited the discussion to a single projection plane for a given cell. However, it can be desirable to use multiple parallel projection planes to take into account the occlusion due to multiple groups of occluders (Fig. 9(b)).

An important property of our method is that we can *re-project* occluders onto subsequent planes. The aggregated *Projections* are reprojected to compute a *Depth* Map on new projection planes (Fig. 9(b)). Occluder fusion occurring on the initial plane is thus also taken into account on the new planes. We next describe how we perform reprojection and its generalization which we call the *occlusion sweep*.

5.1 Reprojection

The *Projections* of occluders can be reprojected only if the initial projection plane is behind them. In this case, the initial *Projections* are inside the umbra of the occluders and can thus be used as a single conservative equivalent occluder (see appendix B for a proof).

The *Projections* onto the initial projection plane (and the conservative bit-map encoding) define a new planar occluder. The reprojection scheme we are about to present is in fact an extended projection operator for the special case of planar blockers parallel to the projection plane.

We base our reprojection technique on the work by Soler and Sillion [Max91, SS98] on soft shadow computation, even though we are interested only in the umbra region. They show that in the

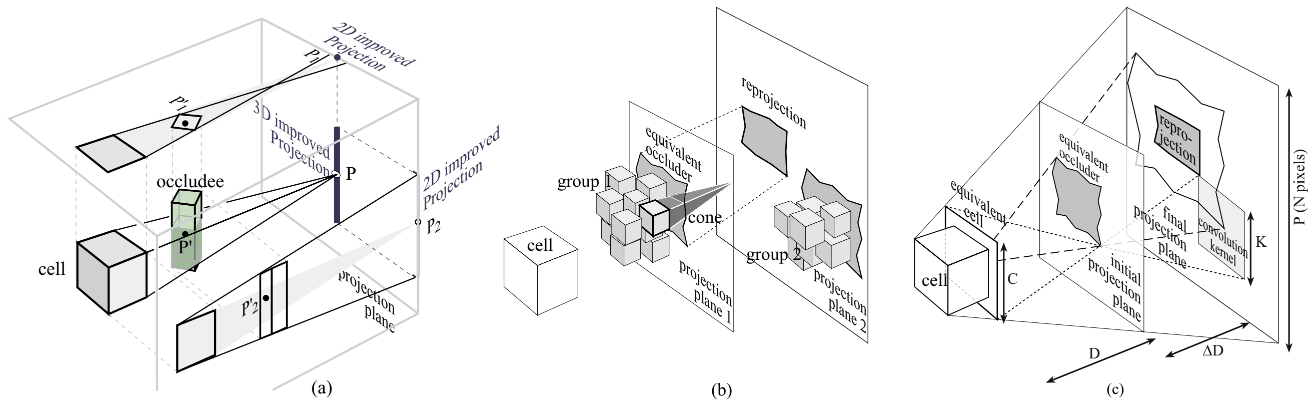


Figure 9: (a) The 3D improved *Projection* is the Cartesian product of two 2D improved *Projections*. Any point P' of the occludee is contained in a cone defined by one point P of the 3D improved *Projection* and the cell. This cone can be constructed by considering the two corresponding 2D projections. (b) If projection plane 2 is used for re-projection, the occlusion of group 1 of occluders is not taken into account. The shadow cone of one cube shows that its *Projection* would be void since it vanishes in front of plane 2. The same constraints apply for group 2 and plane 1. It is thus desirable to project group 1 onto plane 1, and *re-project* the aggregate equivalent occluder onto plane 2. (c) Occluder reprojection. The extended occlusion map of plane 1 is re-projected onto plane 2 from the center of the equivalent "cell". It is then convolved with the inverse image of the equivalent "cell" (dark region on plane 2).

case of planar blockers parallel to the source and to the receiver, the computation of soft shadows is equivalent to the convolution of the projection of the blockers with the inverse image of the source. Their method is an approximation in the general case, but we will use it in the particular case of parallel source, blockers and receiver, where it is exact.

We are nearly in this ideal case: our blocker (the *Projections* on the initial projection plane) and the receiver (the new projection plane) are parallel. However our light source (the cell) is a volume. We define an equivalent "cell" which is parallel to the projection planes and which yields conservative *Projection* on the new projection plane. Its construction is simple and illustrated in Fig. 9(c). We use the fact that our projection planes are actually finite rectangles. Our equivalent "cell" is the planar rectangle defined by the face of the cell closest to the plane, the supporting planes of the cell and the final projection rectangle.

Any ray going through the cell and the projection plane also intersects our equivalent "cell". Thus if an object is hidden from the equivalent "cell", it is also hidden from the cell.

To obtain a conservative umbra region, the inverse image of our equivalent cell (*i.e.* the convolution kernel) is conservatively rasterized (overestimated as opposed to the occluder underestimated conservative rasterization), which is straightforward since it is a 2D rectangle.

The convolution method computes continuous grey levels. To obtain a binary Occlusion Map, we only keep the black pixels (*i.e.* the umbra region). A *Depth* map can be used on the final plane. The *Depth* of the re-projected equivalent occluder is the depth of the initial plane.

5.2 Occlusion sweep

To handle the case where multiple concave or small occluders have to be considered, we generalize re-*Projection* to the *occlusion sweep*. This is a sweep of the scene by parallel projection planes leaving the cell. Occlusion is aggregated on these planes using re-*Projection*.

We project the occluders which lie in front of the current projection plane P onto P and also compute the slices of the concave objects which intersect the plane. We then advance to the following projection plane, by re-projecting the *Projections* of the previous plane. We compute the new slices of concave objects, and project the occluders which lie between the two planes. This defines the *Depth* Map of the new projection plane.

The distance ΔD between two projection planes is chosen to make optimal use of the discrete convolution. The size of the convolution kernel (the inverted image of the equivalent cell) must be an integer number K of pixels (there is then no loss in the conservative rasterization). Let D be the distance between the initial plane and the equivalent "cell", and C the size of the equivalent "cell". N is the resolution of the *Depth* Map, and P is the size of the projection plane. Applying Thales theorem gives: $\Delta D = \frac{D(K-1)P}{CN}$.

In practice we use $K = 5$ pixels. Note that this formula results in planes which are not equidistant. This naturally translates the fact that occlusion varies more quickly near the viewing cell.

6 Occlusion culling algorithm

The elements presented so far (*Projection* and re-*Projection* of occluders and occludees) can now be put together to form a complete occlusion culling algorithm. The algorithm has two main steps: pre-processing and interactive viewing.

6.1 Preprocess

The preprocess creates the viewing cells and the PVS's corresponding to the original input scene. Viewing cells are organized in a spatial hierarchical data-structure, potentially related to the specific application (*e.g.*, the streets of a city). The geometry of the scene itself is organized into a separate spatial hierarchy (*e.g.*, a hierarchy of bounding boxes).

Two versions of the preprocess have been implemented, one which simply uses the *Projection* onto 6 planes for each view cell (used in the city example) and one which uses the occlusion sweep (used for the forest scene).

Adaptive preprocess

We start by performing an occlusion computation for each viewing cell. First, we choose the appropriate occluders and projection planes (see the following two sections). We then *Project* the occluders and build a *Hierarchical Depth Map*. Finally, the occludees are tested recursively. If a node is identified as hidden or fully visible, the recursion stops. By fully visible, we mean that its *Projection* intersects no occluder *Projection*, in which case no child of this node can be identified as hidden. The occludees declared visible are inserted in the PVS of the cell.

If we are satisfied with the size of the PVS, we proceed to the next cell. Otherwise, the cell is subdivided and we recurse on the sub-cells. Nonetheless, occlusion culling is only performed on the

remaining visible objects, *i.e.* those contained in the PVS of the parent.

Performing computation on smaller viewing cells improves occlusion detection because the viewpoints are closer to each other. The views from all these viewpoints are thus more similar, resulting in larger occluder *Projections*, and smaller occludee *Projections*.

The termination criterion we use is a polygon budget: if the PVS has more than a certain number of polygons, we subdivide the cell (up to a minimum size threshold). A more elaborate criterion would be to compare the PVS to sample views from within the cell. Note that our adaptive process naturally subdivides cells more in zones of larger visibility changes. However, more elaborate discontinuity-meshing-like strategies could be explored.

PVS data can become overwhelmingly large in the memory required. To avoid this we use a *delta-PVS* storage mechanism. We store the entire PVS for a single arbitrary initial cell (or for a small number of seed or "key" cells). Adjacencies are stored with each cell; a cell simply contains the *difference* with respect to the PVS of the neighboring cells. Our storage scheme is thus not very sensitive to the number of viewing cells, but to the actual complexity of the visibility changes. Other compression schemes could also be implemented [vdPS99].

Occluder selection

For each viewing cell we choose the set of relevant occluders using a solid angle heuristic similar to those presented previously [CT97, ZMHH97, HMC$^+$97]. To improve the efficiency of occlusion tests, we have also implemented an adaptive scheme which selects more occluders in the direction where many occludees are still identified as visible, in a manner similar to Zhang [ZMHH97].

Since our preprocess is recursive, we also use the PVS of the parent cell to cull hidden occluders. Since, as we shall see, the *Projection* of the occluders is the bottleneck of the method, this results in large savings.

Choice of the projection plane

The heuristic we use is simple, based on the optimization of the number of pixels filled in our *Depth* Map. We place a candidate plane just behind each occluder. We evaluate the size of the *Projection* on each such plane for each occluder. This method is brute force, but remains very fast.

Moreover, since we discard occluders which are hidden from the parent cell, the heuristic is not biased towards regions where many redundant occluders are present.

Six projection planes are used to cover all directions. Unlike *e.g.*, the hemicube [CG85] methods, our six planes do not define a box. The planes are extended (*e.g.*, by 1.5 used in our tests) to improve occlusion detection in the corner directions, as shown in Fig. 13(c).

6.2 Interactive Viewing

For on-line rendering we use the SGI Performer library, which maintains a standard scene-graph structure [RH94]. A simple flag for each node determines whether it is active for display. Each time the observer enters a new cell, the visibility status of the nodes of the scene-graph are updated. This is very efficient thanks to our delta-PVS encoding. Nodes which where previously hidden are restored as visible, while newly hidden ones are marked as inactive. The viewer process adds very low CPU overhead, since it only performs simple scene-graph flag updates.

Dynamic objects with static occluders can be treated using our extended projection approach. However, dynamic occluders cannot be handled. A hierarchy of bounding boxes is constructed in the regions of space for which dynamic objects can move [SC96]. During preprocess, these bounding boxes are also tested for occlusion. In the viewing process the dynamic object is displayed, if the bounding box containing the dynamic object is in the current PVS (see the moving cars in the video).

One of the advantages of our preprocess is that it could be used for scenes which are too big to fit into main memory, or to be completely loaded on-the-fly from the network. The techniques developed by Funkhouser *et al.* [Fun96] can easily be adapted. A separate process is in charge of the database management. Using the PVS of the neighboring viewing cells, the priority of the objects which are not yet loaded is evaluated. Similarly, a priority order is computed to delete invisible objects from memory. The prediction offered by our method cannot be achieved by previous online occlusion culling methods.

7 Implementation and results

We have implemented two independent systems for the preprocessor and the viewer. The preprocessor uses graphics hardware acceleration wherever possible, notably for the *Projection* and convolution. The *Depth* Maps are read from graphics memory, and the occludee test is performed in software. The delta-PVS's computed are stored to disk and made available to the interactive viewer.

Our current implementation of the viewer is based on SGI Performer [RH94]. Performer implements a powerful scene-graph and view-frustum culling, providing a fair basis for comparison. All timings presented are on an SGI Onyx2 Infinite Reality 200Mhz R10K using one processor for the preprocess and two processors for the viewer.

7.1 *Projection* onto a single projection plane

The test scenes we have used for the single plane method consist of a model of a city district which contains a total number of about 150,000 polygons replicated a variable number of times. The improved *Projection* was used to cull occludees lying between the projection plane and occluders more efficiently.

We first present statistics on the influence of the resolution of the *Depth* maps on running time of the preprocess in Fig. 10. Surprisingly, the curve is not monotonic. If the resolution is below 256x256, the occlusion efficiency (*i.e.* the percentage of geometry declared hidden) of the method is low because of the conservative rasterization. More recursion is thus needed, where more occluders are used because they have not been culled from the parent cell. If the resolution is higher than 256x256, occlusion efficiency is not really improved but the time required to build the Hierarchical *Depth* Map becomes a bottleneck (the huge increase for a resolution of 1,024x1,024 may be explained by cache failure). This part of the algorithm could have been optimized using the graphics hardware, but the very low occlusion efficiency gain made us use a resolution of 256x256 for the rest of our computations. Since we use pyramids, only resolutions which are powers of 2 were used.

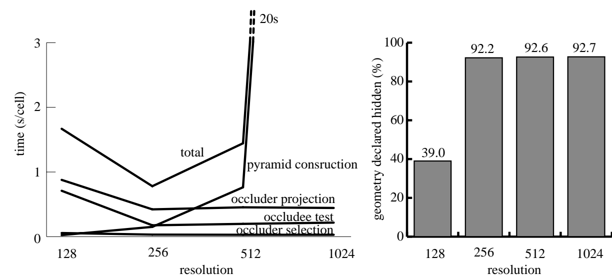


Figure 10: (left) Preprocess running time (sec./cell) versus *Depth* map resolution. (right) Geometry declared hidden vs. resolution of the *Depth*map. All timings for a scene consisting of 600,000 polygons.

We varied the complexity of the input scene by replicating the district. The average preprocessing time per cell is presented in

Fig. 11. The projection of the occluder is the most time-consuming task. A log-log linear fit reveals that the observed growth of the total running time is in $\sqrt{n}$ where n is the number of input polygons. If the total volume of all viewing cells varies proportionally to the number of input polygons, the growth is then $n^{1.5}$

The improved *Projection* presented in section 4 results in PVS 5 to 10% smaller. This is not dramatic, but recall that it comes at no cost. In addition the implementation is simpler, since only supporting lines are considered.

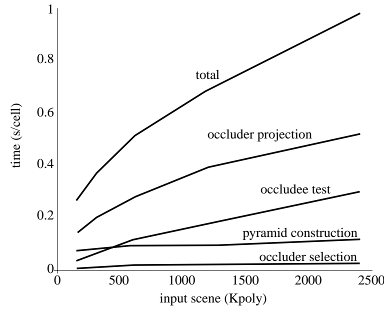


Figure 11: Preprocess running time versus input scene size.

We used a scene consisting of the city district replicated 12 times (1.8M polygons) and 3,000 moving cars of 1.4K polygons each, resulting in a total of 6M polygons. We performed the preprocess for the streets of only one district. The 1,500 initial visibility cells were subdivided into 6,845 leaf cells by our adaptive method (12,166 cells were evaluated, where parent cells are included). The average occlusion efficiency was 96% and the delta-PVS required 60 MBytes of storage. The total preprocess took 165 minutes (0.81s/cell), of which 101 minutes for the occluder *Projection*. We can extrapolate that the preprocess would take 33 hours for the streets of the 12 districts. For an 800 frame walkthrough, an average speed-up of 18 was obtained over SGI Performer view frustum culling (Fig. 12). This is lower than the average geometry ratio of 24 (*i.e.* $\frac{\#\ polys\ after\ frustum\ cull}{\#\ polys\ after\ occlusion\ cull}$) because of constant costs in the walkthrough loop of Performer. Fig. 13 illustrates our results.

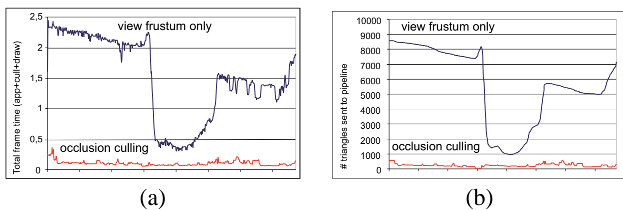


Figure 12: Statistics gathered during interactive walkthrough for a scene of 6M polygons on an Onyx2. (a) Total frame time (app+cull+draw) in seconds. (b) Number of triangles sent to the graphics pipeline.

As an informal comparison, we have implemented the algorithm of Cohen-Or *et al.* [COFHZ98, COZ98]. For the city model, their algorithm declares four times more visible objects on average and the computation time in our implementation is 150 times higher than for extended projection.

7.2 Occlusion sweep

To test the occlusion sweep, we used a model of a forest containing around 7,750 trees with 1,000 leaves each (7.8M triangles). The *Projection* of the leaves close to the projection plane were computed using the convex occluder *Projection* using the stencil buffer. The size of the convolution kernel was fixed to 5 pixels, and we used 15 planes for the sweep. The occlusion sweep took around 23 seconds per cell, 59 minutes for all 158 cells (no adaptive recursion was performed). This is slower than for the city because 15 successive planes are used for the occlusion sweep. 95.5% of the geometry was culled. Fig. 15 shows our sweeping process (we show only one quadrant of the forest for clarity). Observe how the leaves aggregate on the Occlusion Map.

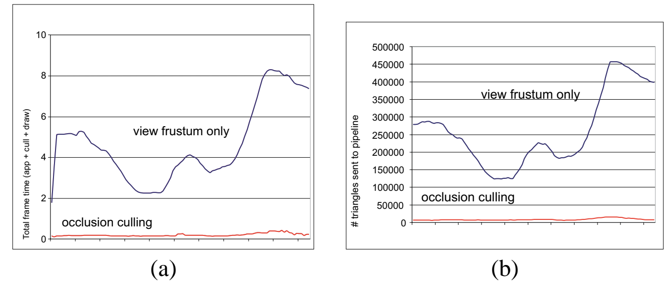


Figure 14: Statistics gathered during interactive walkthrough for a forest scene scene of 7.8M polygons on an Onyx2. (a) total frame time (app+cull+draw) in seconds. (b) Number of triangles sent to the graphics pipeline.

For a walkthrough of 30 sec, we obtained an average speed up of 24 for the interactive display, achieving a framerate between 8.6 and 15 fr/s (Fig.14).

7.3 Discussion

We first have to note that the occlusions our method identifies are a subset of those detected by a point-based method [GKM93, ZMHH97]. Advantages of those methods also include their ability to treat dynamic occluders and the absence of preprocess or PVS storage. However, our method incurs no cost at display time, while in the massive rendering framework implemented at UNC [ACW$^+$99] two processors are sometimes used just to perform occlusion culling. Moreover, for some applications (games, network-based virtual tourism, etc.), the preprocessing time is not really a problem since it is performed once by the designer of the application. Our PVS data then permits an efficient predictive approach to database pre-fetching which is crucial when displaying scenes over a network or which cannot fit into main memory.

We now discuss the conditions in which our method succeeds or fails to detect occlusion (infinite resolution of the *Depth* Map is here assumed). In Fig. 16 we represent in grey the volume corresponding to all the possible *Projections* of the occluders. The actual *Projections* corresponds to the intersection of these *Projection volumes* with the projection plane. The occlusion due to a single occluder is completely encoded if the occluder is in front of the plane and if its *Projection* volume intersects the plane (16(a)). If the plane is farther away, the *Projection* of the occluder becomes smaller, but so do the improved *Projections* of the occludees (however, if resolution is taken into account, more distant planes are worse because of our conservative rasterization).

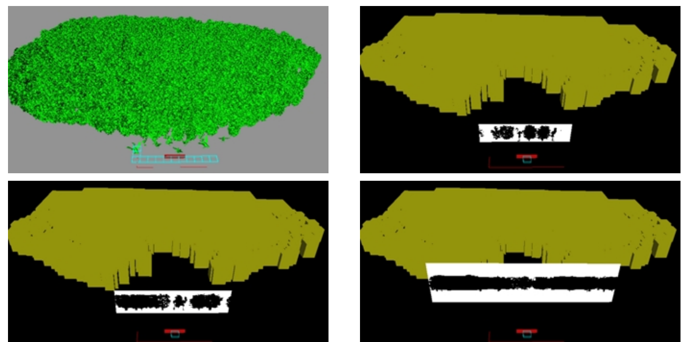
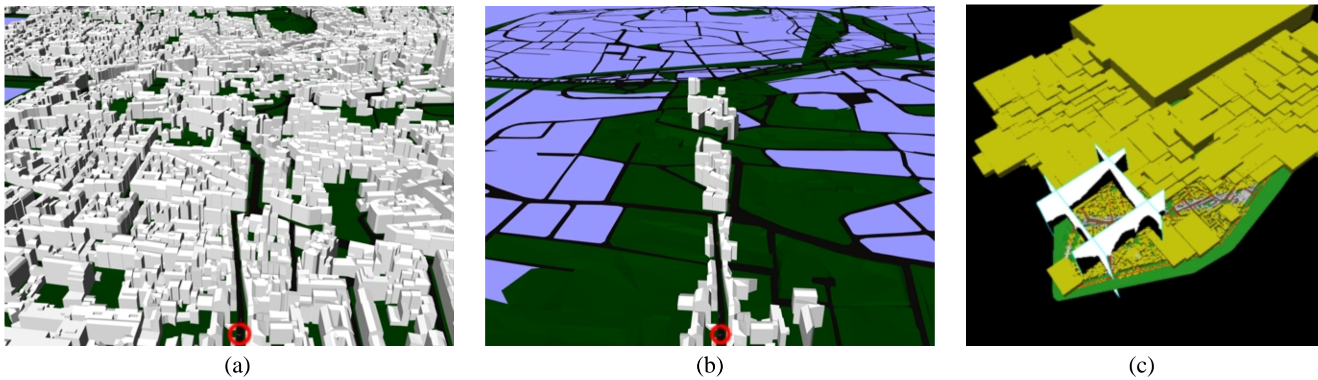


Figure 15: The sweeping process: (a) Part of our 7.8M polygon forest model, (b)-(d) three positions for the sweep projection planes. The yellow bounding boxes are the culled occludees.



(a) (b) (c)

Figure 13: Results of our algorithm. (a) The scene from a bird's-eye view with no culling; the scene contains 600,000 building polygons and 2,000 moving cars containing 1,000 polygons each. (b) The same view using the result of our visibility culling algorithm (the terrain and street are poorly culled because of a poor hierarchical organization of our input scene). (c) Visualization of the occlusion culling approach, where yellow boxes represent the elements of the scene-graph hierarchy which have been occluded.

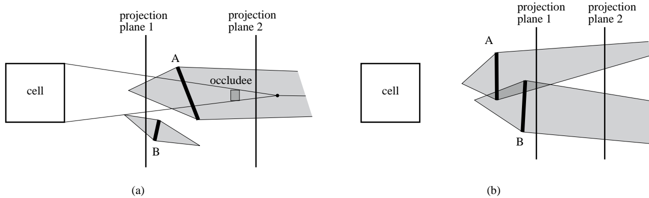


Figure 16: *Projection volumes* are represented in grey (supporting and separating lines are not represented for clarity). (a) Plane 1 does not completely capture the occlusion due to A (*e.g.*, for the occludee). Plane 2 does, but it does not capture the occlusion due to B. (b) Plane 1 capture occlusion fusion between A and B while plane 2 does not.

On the other hand, occluder fusion occurs between two occluders when their *Projection* volumes intersect in the region of the plane (Fig. 16(b)). In this case, the position of the plane is crucial, hence our heuristic which tries to maximize the projected surface on the plane. More elaborate heuristics could search for intersecting *Projection* volumes.

There are situations in which even a perfect occlusion culling method (*i.e.* an exact hidden-part removal) cannot cull enough geometry to achieve real-time rendering. For example, if the observer is at the top of a hill or on the roof of a building, the entire city may be visible. Other display acceleration techniques should thus be used together with our occlusion culling.

The trees used in the forest scene are coarse and made of large triangles. A typical high quality 3D model of tree may require around 50,000 triangles, and leaves are often smaller than our triangles. However, we consider that our results are obtained on a scene which is at least half-way between previous simple architectural examples with large occluders and a high quality forest scene.

8 Conclusions and future work

We have presented *extended projection* operators which permit conservative occlusion tests with respect to volumetric viewing cells. Our operators yield conservative occlusion tests and can handle *occluder fusion*. We have presented an efficient implementation of both operators, as well as an improvement in the case of convex or planar blockers.

We have defined a reprojection operator which allows us to reproject the *Projections* computed on a given projection plane onto another one, allowing us to define an *occlusion sweep*. The results we obtain show a significant speed-up of 15 times compared to a high-end interactive rendering library with view frustum culling only on a 6 million polygon city model.

Results have also been presented showing that our occlusion sweep makes it possible to compute occlusion caused by the cumulative effect of many small objects, such as leaves in a forest, with respect to a volumetric viewing cell.

Future work

Future work includes the computation of *Projections* for portals in architectural environments and the use of unions of convex shapes for the *Projection* of concave occluders. Concave polygonal meshes could also be projected from the center point of the cell, then "shrunk" to compute the *Projection*.

The speed of our method could make its use possible in an on-demand fashion, computing only visibility information for the neighbourhood of the observer. The extended projection concepts can also be used to allow some prediction with on-line occlusion culling methods [GKM93, ZMHH97]. Instead of only testing the bounding box of the objects, their extended projection with respect to a volume centered around the observer could also be tested.

Our method could also be applied to global illumination computation [TH93], LOD for animation, *e.g.,* [CH97], etc. The occlusion sweep could be extended to compute soft shadows.

To be really efficient for complex and cluttered scenes such as forest, our method should be extended to compute semi-quantitative occlusion to drive Level of Detail or image-based acceleration: the more hidden the object, the coarser its display. Human perception and masking effects should then be taken into account.

Acknowledgments

The ideas of this paper were initially developed when the first author was invited to the University of Stanford by Leo Guibas. The discussions with Leo and Mark de Berg have been invaluable for exploring and refining our ideas. Many thanks to Seth Teller, Pierre Poulin, Fabrice Neyret and Cyril Soler. This work was supported in part by a research grant of NSF and INRIA (INT-9724005).

References

[ACW+99] D. Aliaga, J. Cohen, A. Wilson, Eric Baker, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stuerzlinger, R. Bastos, M. Whitton, F. Brooks, and D. Manocha. MMR: An interactive massive model rendering system using geometric and image-based acceleration. In *ACM Symp. on Interactive 3D Graphics*, 1999.

[ARB90] J. Airey, J. Rohlf, and F. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. In *ACM Symp. on Interactive 3D Graphics*, 1990.

[CG85] M. Cohen and D. Greenberg. The hemicube: A radiosity solution for complex environments. In *Computer Graphics (Proc. Siggraph)*, 1985.

[CH97] D. A. Carlson and J. K. Hodgins. Simulation levels of detail for real-time animation. In *Graphics Interface*, 1997.

[Cla76] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, October 1976.

[COFHZ98] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for visibility partitionning of densely occluded scenes. In *Eurographics*, 1998.

[COZ98] D. Cohen-Or and E. Zadicario. Visibility streaming for network-based walkthroughs. In *Graphics Interface*, 1998.

[CT96] S. Coorg and S. Teller. Temporally coherent conservative visibility. In *ACM Symp. On Computational Geometry*, 1996.

[CT97] S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. In *ACM Symp. on Interactive 3D Graphics*, 1997.

[Dur99] Frédo Durand. *3D Visibility, analysis and applications*. PhD thesis, U. Joseph Fourier, Grenoble, 1999. http://www-imagis.imag.fr.

[FS93] T. Funkhouser and C. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Computer Graphics (Proc. Siggraph)*, 1993.

[Fun95] T. Funkhouser. RING - A client-server system for multi-user virtual environments. *ACM Symp. on Interactive 3D Graphics*, 1995.

[Fun96] T. Funkhouser. Database management for interactive display of large architectural models. In *Graphics Interface*, 1996.

[GKM93] N. Greene, M. Kass, and G. Miller. Hierarchical Z-buffer visibility. In *Computer Graphics, (Proc. Siggraph)*, 1993.

[HMC+97] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frusta. In *ACM Symp. on Computational Geometry*, 1997.

[Jon71] C. B. Jones. A new approach to the 'hidden line' problem. *The Computer Journal*, 14(3):232–237, August 1971.

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

[LT99] F. Law and T. Tan. Preprocessing occlusion for real-time selective refinement. In *ACM Symp. on Interactive 3D Graphics*, 1999.

[Max91] Max. Unified sun and sky illumination for shadows under trees. *Comp. Vision, Graphics, and Image Processing. Graphical Models and Image Processing*, 53(3):223–230, May 1991.

[PD90] H. Plantinga and C. R. Dyer. Visibility, occlusion, and the aspect graph. *Int. J. of Computer Vision*, 5(2), 1990.

[RH94] J. Rohlf and J. Helman. IRIS performer: A high performance multiprocessing toolkit for real–Time 3D graphics. In *Computer Graphics (Proc. Siggraph)*, 1994.

[SC96] O. Sudarsky and C.Gotsman. Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality. In *Proc. Eurographics Conf.*, 1996.

[SDDS00] G. Schaufler, J. Dorsey, X. Decoret, and F. Sillion. Conservative volumetric visibility with occluder fusion. In *Computer Graphics (Proc. Siggraph)*, 2000.

[SLSD96] J. Shade, D. Lischinski, D. Salesin, and T. DeRose. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Computer Graphics (Proc. Siggraph)*, 1996.

[SS98] C. Soler and F. Sillion. Fast calculation of soft shadow textures using convolution. In *Computer Graphics, (Proc. Siggraph)*, 1998.

[Ste97] A. James Stewart. Hierarchical visibility in terrains. *Eurographics Workshop on Rendering 1997*, June 1997.

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

[TH93] S. Teller and P. Hanrahan. Global visibility algorithms for illumination computations. In *Computer Graphics (Proc. Siggraph)*, 1993.

[TS91] S. Teller and C. Séquin. Visibility preprocessing for interactive walkthroughs. In *Computer Graphics (Proc. Siggraph)*, 1991.

[vdPS99] M. van de Panne and J. Stewart. Effective compression techniques for precomputed visibility. In *Eurographics Workshop on Rendering*, 1999.

[WBP98] Y. Wang, H. Bao, and Q. Peng. Accelerated walkthroughs of virtual environments based on visibility processing and simplification. In *Proc. Eurographics Conf.*, 1998.

[WS99] P. Wonka and D. Schmalstieg. Occluder shadows for fast walkthroughs of urban environments. In *Proc. Eurographics Conf.*, 1999.

[ZMHH97] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. Visibility culling using hierarchical occlusion maps. In *Computer Graphics (proc. Siggraph)*, 1997.

A Extended projection using OpenGL

Recall that in Section 3.2 we described how to project convex occluders onto a projection plane as the intersection of the views from the vertices of the viewing cell. Here we present the details of an efficient OpenGL implementation. One of the problems is that during the projection of convex occluders we need to write consistent z-values and also treat the case of multiple blockers. An efficient way to do this in OpenGL is to use the stencil buffer, and a slightly involved z-buffer.

For a perspective projection, depth is considered from the viewpoint. Mapping the z value to our definition of depth requires an addition to set the zero on the projection plane. Unfortunately, OpenGL stores $\frac{1}{z}$ in the z-buffer, preventing a simple addition.

For a given occluder and a given cell, we project (in software) the blocker onto the projection plane, including the calculation of z values. The resulting 2D polygons are then rendered orthographically using a stencil buffer. Z-testing is performed with respect to z-values potentially written by a previously projected blocker, but depth values are not written. The stencil buffer is incremented by one. After all the polygons corresponding to each cell vertex have been rendered, the umbra region is defined by the region of the stencil buffer with the value 8 (*i.e.* blocked with respect to all cell vertices).

The eight 2D polygons are rendered again, using the stencil buffer to restrict writing to the umbra region only. The first polygon is rendered and z-values are written to the z-buffer. The 7 other polygons are then rendered but the z-test is inverted. This results in the *maximum* z-value being written to the z-buffer.

B Validity of the reprojection

We now show that the $\mathcal{P}rojection$ of several occluders can be used as a single conservative equivalent occluder, *i.e.* an occludee hidden by this $\mathcal{P}rojection$ is also hidden by the occluders. We prove the following more general property.

Property 3 *Consider an extended light source, any object A (convex or concave) and U the umbra region of A. Then the shadow of any subset U' of U lies inside U.*

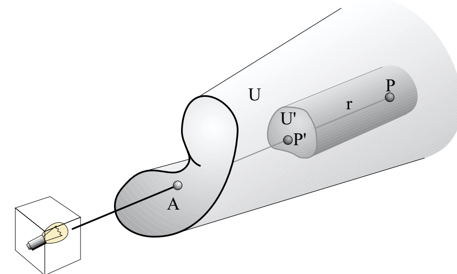


Figure 17: Umbra of the subset of an umbra. Point P is in the umbra of U' which is a subset of the umbra U of A. It is thus also in the umbra of A.

To prove this, consider a point P in the umbra of U' (Fig. 17). Any ray r going through P and the source intersects U'. Consider an intersection point P'. Since $P' \in U' \subset U$, P' is in the umbra of A. Thus any ray (r for example) going through P' and the source intersects A. We have shown that any ray going through P and the source intersects A. P is thus in the umbra of A. Note that property 3 presupposes neither convexity nor planarity of the object A.

If the cell is considered as a light source, this proves that any subset of the umbra of a set of occluders is a conservative version of these occluders. As we have seen, the $\mathcal{P}rojection$ of an occluder which lies in front of the projection plane is its umbra on the plane. This $\mathcal{P}rojection$ can thus be re-Projected as a new occluder. If the occluder lies behind the projection plane, its $\mathcal{P}rojection$ does not lie inside its umbra because the projection plane is closer to the viewing cell than the occluder. Thus property 3 does not apply.

This proof together with property 1 is also an alternative proof of the validity of $\mathcal{P}rojection$ when occluders are in front of the plane.