Interactive Shadow Generation in Complex Environments

Naga K. Govindaraju Brandon Lloyd Sung-Eui Yoon Avneesh Sud Dinesh Manocha
University of North Carolina at Chapel Hill
{naga,blloyd,sungeui,sud,dm}@cs.unc.edu
http://gamma.cs.unc.edu/Shadow

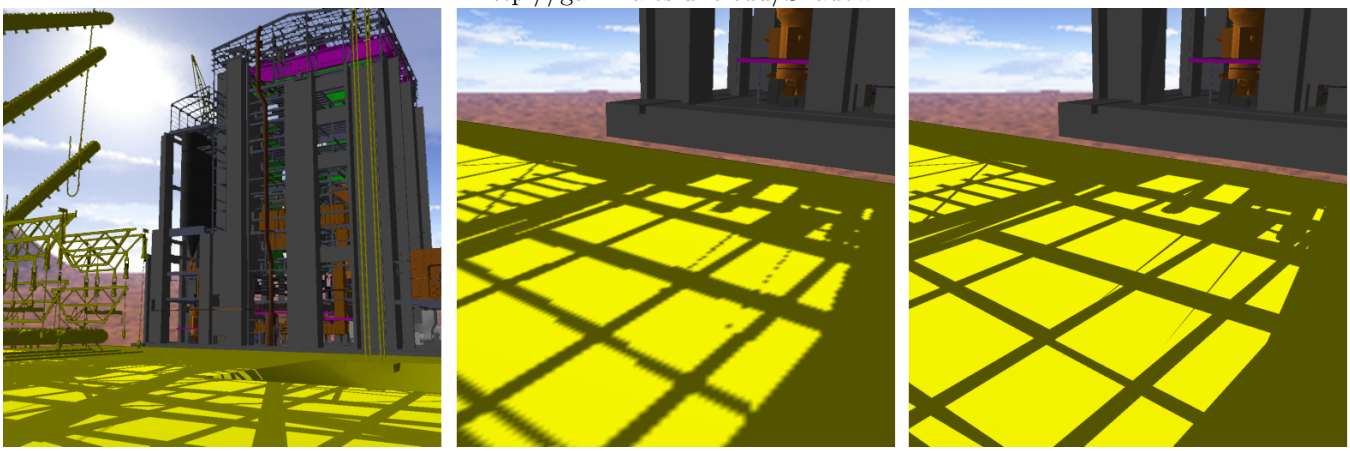


Figure 1: *The left image shows a snapshot generated from the application of our hybrid shadow generation algorithm to the powerplant model (12.7M triangles). The middle image shows a different viewpoint generated using perspective shadow maps. Notice the aliasing artifacts. The right image highlights the shadows generated by our interactive algorithm from the same viewpoint with sharper boundaries.*

Abstract: *We present a new algorithm for interactive generation of hard-edged, umbral shadows in complex environments with a moving light source. Our algorithm is based on a hybrid approach that combines some of the efficiencies of image-precision techniques along with the image quality of object-precision methods. We present interactive algorithms based on levels-of-detail (LODs) and visibility culling to compute the potential shadow-casters and shadow-receivers. We further reduce their size based on a novel cross-visibility culling algorithm. Finally, we use a combination of shadow polygons and shadow maps to generate shadows. We also present techniques for LOD-selection that eliminate the artifacts in self-shadows. Our algorithm can generate sharp shadow edges and reduce aliasing. We have implemented the algorithm on a three PC system with NVIDIA GeForce-4 cards and applied it to three complex environments composed of millions of triangles. It can render the scene and generate shadows at* $7-25$ *frames per second on these models.*

CR Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]:Picture/Image Generation–Bitmap and framebuffer operations; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism-Color, shading, shadowing, and texture

Keywords: Shadow generation, object space, image space, visibility, level-of-detail, interactive display, parallel rendering

1 Introduction

The generation of shadows is a classic problem in computer graphics. Shadows provide important spatial cues and can greatly enhance the visual realism of computer-generated images. In this paper, we address the problem of interactive and accurate calculation of hard-edged, umbral shadows cast by a moving point or directional source in complex static environments. These environments may be architectural models, urban datasets, or CAD models of a large structure such as an airplane, oil tanker, or a power plant. These types of scenes may consist of thousands of objects or millions of polygons and can have a wide depth range.

The design review and evaluation of complex environments benefits greatly from the ability to generate interactive walkthroughs. Shadows are helpful in walkthroughs because they provide additional information about an object's shape and its relative placement in the environment. [Wanger 1992]. Walkthroughs can further benefit from the use of dynamic light sources. The shifting of shadows caused by a moving light source amplifies the viewer's understanding of the 3D environment.

The problem of shadow generation is well-studied in computer graphics. Two classes of algorithms have been popular for generation of real-time shadows: shadow maps [Williams 1978] and shadow volumes [Crow 1977]. Shadow mapping is an image-based approach that is easy to use. The transformations and depth comparisons necessary to do shadow mapping are available on current graphics hardware. The main drawback of shadow maps is aliased shadow edges due to their limited resolution. Aliasing is especially problematic in walkthroughs of large models because the viewer is often very close to the scene geometry while the light source is placed over the head at a distance so as to cover the entire scene (as shown in Fig. 1). Shadow volumes avoid the aliasing problem by computing object-based shadow boundaries. Shadow volume calculations can be accelerated on graphics hardware using the stencil buffer. Unfortunately, the approach does not scale well for very large models due to the large number of shadow-casters.

Main Contributions: We present a new algorithm for interactive shadow generation in complex environments. Our algorithm is based on a hybrid approach that combines the image quality of object-precision shadow generation techniques with the simplicity and efficiencies of image-precision

methods. We use a combination of hierarchical representations, LODs and visibility culling algorithms to compute the potentially visible set (PVS) from the light-view and the eye-view. The resulting sets are classified as potential *shadow-casters* and *shadow-receivers*, respectively. We reduce the size of these sets by performing additional cross-visibility culling computations. Finally, we compute the shadow-polygons using an object space clipping algorithm and use them along with shadow maps. The shadows are rendered in two passes using a one-bit stencil buffer.

We use image-space occlusion queries along with hierarchical representations of the objects in the scene graph to perform significant culling. As a result, the sizes of shadow-casters and shadow-receivers are small and we are able to compute the shadow-polygons efficiently. We have implemented the algorithm on three PCs, each with a NVIDIA GeForce 4 graphics card. Our process-parallel implementation introduces a frame of latency in the pipeline, in addition to double-buffering. Our system has been applied to three complex environments with a moving light source: a power plant model composed of 12.7 million triangles, a tanker model of more than 82M triangles, and a house model containing more than 1.3 million triangles. The power plant and the tanker models contain long, thin structures which cause considerable aliasing in shadow maps. Long and narrow triangles are challenging for pure object-precision approaches as they result in a very large number of shadow-polygons. Our system runs at $7-25$ frames per second, depending on the light and eye positions.

New Results: Some novel aspects of our work include:

1. An improved PVS computation algorithm for complex environments that can be used to accelerate both rendering and shadow generation.
2. An LOD-selection algorithm designed to reduce the errors in shadow generation, self-shadowing artifacts, and popping.
3. A cross-culling visibility algorithm, between shadow-casters and shadow-receivers, that can accelerate the performance of object-precision shadow generation algorithms.
4. A hybrid shadow computation algorithm that uses a combination of shadow maps and shadow polygons to generate shadows at interactive rates.

Compared to earlier approaches, our hybrid approach offers many advantages. It makes no assumptions about the input model or connectivity information. It can generate sharp shadow edges and greatly reduces aliasing. Finally, it can compute sharp shadows from a moving light source at interactive rates in complex environments.

Organization: The rest of the paper is organized in the following manner. We give a brief overview of related work in Section 2. Section 3 presents the PVS computation algorithm and techniques to bound the error due to LODs in shadow computation. We present the hybrid shadow generation algorithm in Section 4. Section 5 describes our implementation and highlights its performance on different models. We analyze performance of the system and discuss some of its limitations in Section 6.

2 Related Work

In this section, we give a brief overview of previous work on shadow generation and interactive display of complex environments. Woo et al. [1990] give a survey of some of the basic shadowing techniques. We limit ourselves to algorithms that compute hard-edged umbral shadows cast. In general, shadowing algorithms can be classified as either image-precision or object-precision. A few hybrid combinations have also been proposed.

2.1 Image Precision Methods

Shadow maps were introduced by Williams [1978] as an image-precision solution for generating shadows. A shadow map is simply a depth map generated from the light-view. To determine whether a point lies in shadow, its light-space depth is compared to the depth value stored in the shadow map. Shadow maps can be implemented with standard hardware [Segal et al. 1992; Heidrich and Seidel 1999] and recent graphics cards have improved support for handling shadow mapping efficiently. By using parabolic projections, shadow maps can be used for hemispherical and omni-directional light sources. [Brabec et al. 2002].

One of the main drawbacks of shadow maps is aliasing. Aliasing can occur when a shadow map pixel projected on the scene subtends more than one pixel in the eye view. There are two main types of aliasing - perspective aliasing and projective aliasing [Stamminger and Drettakis 2002]. Perspective aliasing occurs when a point is much closer to the eye than to the light source. Projective aliasing occurs when the angle formed with a surface normal is greater for the light direction than the view direction. These situations arise often in walkthroughs of complex models with curved objects and wide depth range.

Many techniques have been proposed to handle aliasing of shadow edges. Reeves et al. [1978] introduced percentage closer filtering which improves the appearance of aliased edges by blurring them. In some situations the blurring may be excessive or even undesirable. Brabec et al. [2001] applied this filtering for hardware-based shadow map rendering. Fernando et al. [2001] presented adaptive shadow maps which are used to increase the effective shadow map resolution in areas where edge aliasing occurs. Unfortunately, adaptive shadow maps require software rendering, which is too slow for interactive rendering of large models. Adaptive shadow maps also use progressive refinement, which may not work well for scenes with a moving light source. Another approach similar to adaptive shadow maps uses multiple shadow maps of varying resolution [Tadamura et al. 2001]. Perspective shadow maps [Stamminger and Drettakis 2002] ameliorate aliasing by warping the depth buffer in order to allocate more samples near the viewer. Though perspective shadow maps can often reduce perspective aliasing, their performance is highly view-dependent and they do not reduce projection aliasing.

Other image-precision methods for shadow generation are based on ray-tracing. Many algorithms for fast ray-tracing have been proposed on shared-memory multiprocessor systems [Parker et al. 1999] as well on a cluster of PCs [Wald et al. 2001].

2.2 Object-Precision Approaches

Object-precision approaches avoid the edge aliasing problem by computing exact shadow boundaries. These approaches include projection techniques that calculate shadow boundaries on the scene polygons. Atherton et al. [1978] clipped the scene polygons against each other from the light-view. The resulting clipped polygons, representing the lit surfaces, are attached to the original polygons as surface detail. Blinn [1988] rendered shadows by projecting the vertices of an occluder object into the plane of a receiver polygon and used the resulting polygons to modulate the surface color. These techniques do not scale well to large models in practice.

One of the most popular object-precision techniques is the shadow volume algorithm introduced by Crow [1977]. A shadow volume is the set of points that lie in shadow behind a shadow-caster. For a polygonal shadow-caster, the shadow volume is a semi-infinite frustum extending away from the edges of the polygon to infinity. The shadow volume algorithm checks if a particular point is in shadow by counting the crossings with shadow frusta polygons on any ray ex-

tending away from the point. Bergeron [1985] generalized shadow volumes for non-manifold objects and non-planar polygons. BSP trees have been used to represent shadow volumes [Chin and Feiner 1989; Chrysanthou and Slater 1995]. These techniques do not work well with dynamic lights because the entire tree has to be rebuilt when the light source moves. Heidmann [1991] showed that shadow volumes can be implemented in hardware by using the stencil buffer to count crossings. Recently, techniques have been proposed to ensure that hardware shadow volumes are not clipped "open" by the near and far clipping planes [Everitt and Kilgard 2002]. The enhanced robustness of the algorithm has led to shadow volumes to become increasingly more popular in games, e.g. Doom-3.

Shadow volumes do not scale well for complex models. The number of shadow polygons can be extremely large. A common configuration in walkthroughs is a light source overhead with geometry such as beams or trusses above the viewer. The shadow frustum polygons created by the overhead geometry may fill the whole screen, yet the shadows they define are very small. In the presence of many large shadow volumes, the application will quickly become fill-bound.

2.3 Hybrid Approaches

Some combinations of object-space and image-space techniques have been proposed for shadow generation and related computations. Brotman and Badler [1984] combined shadow volumes with a software-based, depth-buffered, tiled renderer to generate soft shadows. McCool [2000] extracts edges from a shadow map to create shadow volumes. While the technique replaces aliased edges with sharp edges, it does not replace the details lost due to the limited resolution of the shadow map. Udeshi and Hansen [1999] presented an improved shadow volume algorithm using multiple CPUs and graphics processors on a shared memory architecture, but they only rendered relatively small indoor scenes.

2.4 Interactive Display Of Complex Environments

The problem of interactive display of complex environments has been well-studied in computer graphics and related disciplines. Many rendering acceleration techniques based on visibility culling, levels-of-detail (LODs) and image-based representations have been proposed. An excellent survey of visibility algorithms has been given in [Cohen-Or et al. 2001] and of LOD methods in [Luebke et al. 2002]. Many hybrid algorithms that combine LODs methods with occlusion culling have been proposed as well [Andujar et al. 2000; El-Sana et al. 2001; Baxter et al. 2002; Govindaraju et al. 2002].

3 LOD-based Interactive PVS Computation

Visibility computation is an integral part of any shadow generation algorithm. Given a point source, the hard-edged umbral shadows can be determined by partitioning the visible-surface of the eye-view with respect to visibility to the light-view. Regions not visible to the light lie in shadow. However, exact computation of the visible-surface is too slow for interactive applications and is prone to geometric robustness problems.

In this section, we present an interactive potentially visible set (PVS) computation algorithm. We use this algorithm to compute the PVS from the eye-view (PVS_E) and the PVS from the light-view (PVS_L). We use levels-of-detail (LODs) to accelerate the algorithm and also present techniques to select appropriate LODs from each view.

Our PVS computation algorithm is based on recent work of [Govindaraju et al. 2002] that uses a pair of graphics processors to perform occlusion culling. One graphics processor computes the occlusion representation, while the other performs culling at an object level using image-space occlusion queries. The algorithm described in [Govindaraju et al. 2002] has two limitations. First, a process-parallel implementation of the algorithm on two separate PCs introduces one frame of additional latency in the overall pipeline. Second, the culling is performed only at the object level. As a result, the size of the PVS can be overly conservative, ranging anywhere from $200K - 450K$ triangles. The *size* of a PVS, or an object, refers to the total number of triangles it contains. Although we can render the resulting PVS at interactive rates on current graphics processors, the PVS is too large for our hybrid shadow generation algorithm. We present an improved algorithm that lowers the latency in the pipeline and reduces the size of PVS by almost one order of magnitude.

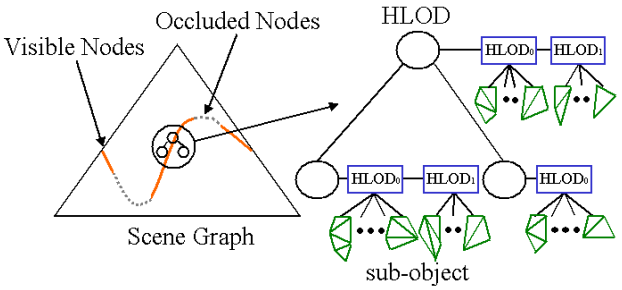


Figure 2: *Scene graph hierarchy used for PVS computation and shadow generation. This figure also highlights the sub-object hierarchy associated with a HLOD. Each sub-object is shown in green.*

3.1 Scene Graph Representation

We use the scene graph representation presented in [Govindaraju et al. 2002] and augment it with a sub-object hierarchy. A scene is described as a collection of objects composed of triangles. We use a combination of partitioning and clustering algorithms to ensure that all objects in the scene have roughly the same size. Each node in the scene graph stores references to its children and to the bounding box enclosing them. The algorithm pre-computes LODs for each object along with hierarchical levels-of-detail (HLODs) for intermediate nodes [Govindaraju et al. 2002]. A HLOD associated with an intermediate node represents a simplification of all objects contained in the tree rooted at that node. Each LOD and HLOD is associated with an object-space Hausdorff error deviation metric.

We subdivide each LOD and HLOD into a group of sub-objects and represent them using a sub-object hierarchy (as shown in Fig. 2). A sub-object is composed of k triangles, where k is typically a small number (say $1 - 10$). The sub-object hierarchy is typically $1 - 2$ levels deep and each node consists of multiple children. A deeper hierarchy can lead to stalls, which can happen when performing image-space occlusion queries at multiple levels. This is explained further in Section 5.1.

3.2 Scene Traversal and Occlusion Culling

Given the eye-view or light-view, the algorithm traverses the scene graph and performs view frustum culling and occlusion culling. It also checks whether any LOD or HLOD associated with a node satisfies the screen-space error threshold. The set of LODs and HLODs of visible objects selected by the algorithm form a cut across the scene graph representing the entire model (as shown in Fig. 2).

The occlusion culling algorithm proceeds in two steps. It first creates an occlusion representation (OR) and then performs scene graph culling (SGC) to cull away objects and sub-objects that are not visible from the viewpoint.

Computing occlusion representation (OR): The algorithm uses the PVS from the previous frame as an approximation of occluders for the current frame [Greene et al. 1993]. It renders the objects and sub-objects in the PVS

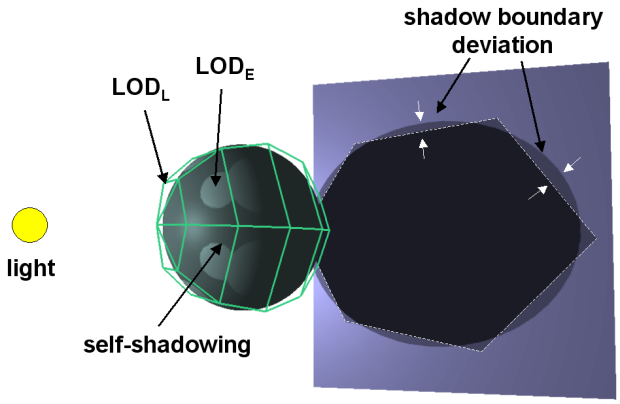


Figure 3: *Artifacts in self-shadows generated due to a naive LOD selection algorithm. We correct this problem by using LOD_E during PVS_L computation and not LOD_L.*

based on the current camera location and generates a depth map. Note that the depth map resulting from the computation of PVS_L is the same as the shadow map.

Scene graph culling (SGC): Given the occlusion representation, the algorithm traverses the scene graph in a top-down manner using image-space occlusion queries to check whether the bounding box of each node is visible. These hardware-supported queries rasterize the bounding box and check if any fragment passes the depth test. We traverse the scene graph and perform these queries at multiple levels of the hierarchy.

- **Object Culling:** We render the bounding box of the object and check whether it is occluded by the occlusion representation. If the object is visible, we check whether any LOD or HLOD associated with that node satisfies the user-specified screen-space error bound. If it is within the error bound, we separately check each sub-object associated with the LOD or HLOD for visibility. Otherwise, we apply the algorithm recursively to its children nodes in the scene graph. Note that we perform sub-object level culling after determining all the visible objects.

- **Sub-object culling:** We render all the triangles contained in a sub-object and check whether they have been occluded by the occlusion representation. The visible sub-objects at the leaf nodes of the sub-object hierarchy are added to the PVS.

Two factors affect the overall performance of the occlusion culling algorithm. The first factor is whether sub-object culling is performed at all. Sub-object culling takes more time because of additional occlusion queries, but results in a smaller PVS. The second factor is the number of triangles used per primitive (k). A higher value of k reduces the number of sub-objects per object, thereby reducing the number of occlusion queries to be performed. On the other hand, a lower value of k results in a much smaller PVS. If $k = 1$, the algorithm computes the smallest PVS for a given LOD error threshold.

A smaller PVS improves the overall performance of our shadow generation algorithm. It results in a smaller set of occluders rendered during OR generation. As a result, we compute the occlusion representation and perform SGC on the same graphics processor. This is different than computing them on separate graphics processors, as proposed in [Govindaraju et al. 2002], . This results in reduced latency in the overall pipeline. Moreover, a process-parallel implementation lowers the load in terms of network bandwidth and latency.

3.3 LODs and Shadow Generation

The use of LODs and HLODs in PVS computation introduces inaccuracies in shadow boundaries and can cause artifacts in self-shadows. In this section, we highlight the artifacts and present our LOD selection algorithm to minimize their affect.

For a visible node, the rendering algorithm projects the object-space error bound associated with the node's LODs or HLODs into screen-space. The coarsest LOD or HLOD with a screen-space error less than the user-specified error-threshold (δ) is selected for rendering. This ensures maximum deviation of δ pixels in an object's screen-space silhouette.

Inaccuracy in Shadow Boundaries: Our algorithm uses LODs to compute PVS_L as well as PVS_E. They correspond to potential shadow-casters and shadow-receivers, respectively. The projection of a shadow-caster's silhouette onto objects in the scene corresponds to the shadow boundary. The deviation in the silhouette due to LODs causes a deviation in the shadow boundary. There are two factors leading to error in the shadow boundaries due to the use of LODs. First, the LOD deviation is magnified as the distance between a shadow-caster and a shadow-receiver is increased. Second, the error increases as the angle between the shadow-receiver's normal and the light-source direction approaches 90 degrees. The second factor can be seen on the right side of the shadow in Fig. 3. While we cannot bound error introduced by the orientation factor, we can bound the error caused by the distance factor. Our LOD selection scheme computes this bound implicitly because distance is included in the projection of the object-space error bound onto the image plane. The absolute error bound will be different at each point due to its distance and orientation with respect to the light, but all shadow boundary deviations are guaranteed to meet that bound.

Self-Shadows: To avoid self-shadowing artifacts, we need to select the LODs properly. Lets consider an object in the scene. Let LOD_L and LOD_E be the LODs of the object selected from the light-view and the eye-view, respectively. Depending on the position of this object in the scene, there are three possibilities:

1. The object is visible from both the views. As a result, it is a potential shadow-caster and a potential shadow-receiver. If LOD_L and LOD_E are not the same, self-shadowing artifacts can occur (as shown in Fig. 3).
2. If the object is not seen by the light-view, it lies completely in shadow.
3. If the object is not seen by the eye-view, no shadows are computed on it.

In the second and third case, the LODs for the objects can be chosen independently, from the light-view and the eye-view. However, we still need to ensure that LOD_L is the same as LOD_E for objects visible in both the views. This can cause a sudden switch in the LOD selection if LOD_L and LOD_E vary independently, which results in visible pop in the object or its shadow. To avoid this popping we use the same LOD selection for the light-view as is used for the eye-view.

Recomputing PVS_L: Whenever the user moves, LOD_E for any visible object in the scene can change. As a result, we need to recompute PVS_L, even though the light source may be static. We do this to avoid any artifacts in self-shadows and popping in the final image. This implies that there is no additional overhead of using a moving light source, as compared to a static light source.

4 Hybrid Shadow Generation Algorithm

In this section, we give an overview of our hybrid shadow generation algorithm. This algorithm uses a combination of

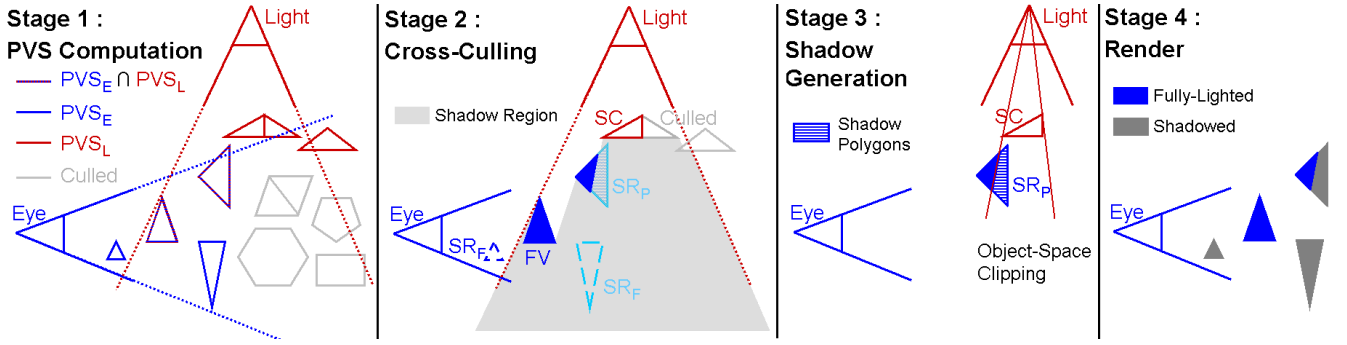


Figure 4: *Overview of our hybrid approach. This left-to-right sequence shows the four stages of our algorithm and the intermediate computations.*

object-precision and image-precision techniques to compute the set of triangles that are fully or partially in shadow and renders them using the stencil buffer (as shown in Fig. 4). We also present a process-parallel architecture, which uses three graphics cards and takes into account network latency and bandwidth to improve the overall performance.

Given the eye-view and the light-view, our algorithm computes PVS_E and PVS_L, using the algorithm described above. Next it performs cross-culling followed by shadow generation.

4.1 Cross-Culling

Every triangle in PVS_L is a potential shadow-caster and every triangle in PVS_E is a potential shadow-receiver. Cross-culling performs visibility computations between PVS_E and PVS_L and prunes the number of potential shadow-casters and shadow-receivers. In particular, cross-culling checks the visibility of triangles in PVS_E with respect to PVS_L and partitions the triangles in PVS_E into three subsets (as shown in Fig. 4):

- **Fully-lighted ($\mathcal{FV}$):** These triangles are fully visible from the light-view and are not in shadow at all.
- **Fully-shadowed receivers ($\mathcal{SR}_F$):** These triangles are totally occluded from the light-view and therefore, are fully in shadow.
- **Partially-shadowed receivers ($\mathcal{SR}_P$):** These triangles are partially visible from the light-view.

Moreover, we compute a subset of PVS_L that casts shadows on $\mathcal{SR}_P$. We refer to the resulting triangles as the *shadow-casters*, $\mathcal{SC}$. The cross-culling algorithm proceeds in two steps:

1. Consider all the triangles belonging to PVS_E and check whether they are occluded by PVS_L. Based on occlusion queries, our algorithm partitions PVS_E into $\mathcal{FV}$, $\mathcal{SR}_F$ and $\mathcal{SR}_P$.
2. The algorithm evaluates each triangles in PVS_L and culls away the triangles that do not cast a shadow on $\mathcal{SR}_P$. The remaining triangles, $\mathcal{SC}$, are used for shadow generation.

In the first step, we render the PVS_L and generate the depth map from the light-view. We render the triangles in PVS_E and perform occlusion queries. Based on the outcome of the query, we classify the triangles as either occluded ($\mathcal{SR}_F$) or non-occluded. The non-occluded subset computed by these queries is further partitioned into triangles that are fully visible ($\mathcal{FV}$) from the light-view or only partially visible ($\mathcal{SR}_P$). To compute $\mathcal{FV}$ and $\mathcal{SR}_P$, we set the depth function to GL_GREATER and perform occlusion queries by rendering the resulting triangles. Note that we do not modify the depth buffer while performing occlusion queries. Also, the use of shadow maps for occlusion culling provides the advantages for shadow-caster fusion.

In the second step, we compute a subset of PVS_L that corresponds to the triangles that cast shadows on $\mathcal{SR}_P$. We compute the shadowed regions by enabling the stencil and rendering $\mathcal{SR}_P$ from the light-view. The depth mask is disabled and the stencil is set to 1 in regions where the triangles in $\mathcal{SR}_P$ project and fail the depth test. We set the stencil test to pass in regions where the stencil is 1 and use occlusion queries while rendering triangles of PVS_L. The resulting triangles ($\mathcal{SC}$) that pass both the occlusion and the stencil tests form the potential shadow-casters and are used for shadow generation.

4.2 Shadow Generation

We compute the object-precision shadows either in software or use a hybrid approach. The sizes of the resulting $\mathcal{SR}_P$ and $\mathcal{SC}$ are often quite small (e.g. a few thousand triangles). Moreover, the triangles in $\mathcal{SR}_P$ and $\mathcal{SC}$ tend to have a fairly low depth complexity, We compute the shadow-polygons explicitly on the CPU rather than implicitly using GPU-based shadow volumes. Even with a small $\mathcal{SC}$ of a few thousand triangles, rendering shadow volumes tends to be fill bound on the latest graphics cards because the shadow volumes are large. On the other hand, the shadow-polygons computed by our algorithm tend to be quite small. Our shadow generation algorithm uses a variation of the classic Atherton-Weiler-Greenberg algorithm [Atherton et al. 1978]. The triangles in $\mathcal{SR}_P$ are clipped against the shadow frusta formed by each of the triangles in $\mathcal{SC}$. The resulting shadow-polygons are calculated by repeatedly clipping the scene triangles against the planes of the shadow frusta. We chose this technique because clipping a triangle against a plane is quick, simple, and robust.

We must compute the shadow-polygons efficiently. Initially, we subdivide the light's screen-space into a 2D grid of bins. Each bin contains a list of triangles from the shadow-casters, $\mathcal{SC}$, that overlap with the bin. We clip each triangle in $\mathcal{SR}_P$ against the shadow-casters contained in the bins that the triangle overlaps. This procedure results in a collection of convex shadow-polygons. To avoid duplicate tests with shadow-casters that may appear in more than one bin, we employ a simple mail-boxing scheme, where each shadow caster stores the last shadow receiver triangles it was tested against. If the triangles are uniformly distributed and are proportional in size to the bins, the algorithm's expected behavior is $O(N)$, where N is the number of triangles in $\mathcal{SC}$ and $\mathcal{SR}_P$. If most of the triangles fall in the same bin or the scene consists of a high number of long and skinny triangles, the number of intersections can grow to $O(N^2)$ in the worst case.

Hybrid Scheme: If the number of shadow-polygons is very high (e.g. more than 50K), it may not be possible to calculate all the intersections at interactive rates on the CPU. It turns out that the shadow-polygons on many surfaces are so small or so far away from the eye-view that they make

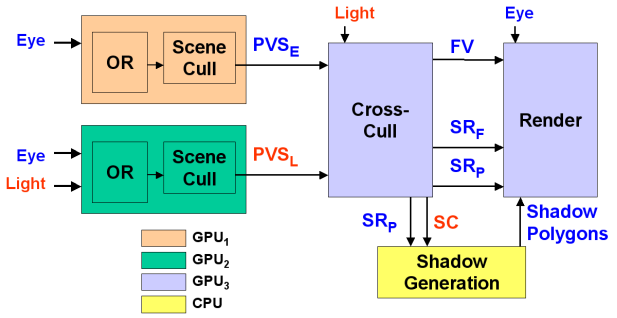


Figure 5: *Architecture of the Process-Parallel Algorithm: This figure highlights different components of our hybrid shadow generation algorithm. Each color represents a separate graphics processor or CPU.*

little or no contribution to the final image. These shadows can easily be rendered with a shadow map. Therefore, we use a hybrid approach for generating shadows. We compute shadow-polygons only where it will make a significant difference in image quality and use a shadow map everywhere else.

To guarantee an acceptable frame rate we establish a fixed budget of intersection tests. The triangles in $\mathcal{SR}_P$ are processed one at a time, computing each triangle's intersections with the shadow-casters until the budget is exceeded. We prioritize the triangles using a heuristic to quantify the visual impact of aliasing on each triangle. Aliasing occurs wherever the pixels in the shadow map project to larger than a pixel on the screen. Using the formulation in [Stamminger and Drettakis 2002], we define the resolution mismatch factor, m, as the ratio of the projected area of a shadow map pixel on a surface to that of an image pixel:

$$m = \frac{d_s r_s / cos(\alpha)}{d_i r_i / cos(\beta)},$$

where d_s and d_i are the sizes of the pixels on the shadow and image planes, α and β are the angles formed by the surface normal and the light-view and eye-view directions, and r_s and r_i are the distances from the point to the light and eye, respectively. We assign a priority to a triangle as the maximum of the resolution mismatch value computed at its vertices and centroid. A triangle with mismatch values less than one can be rendered without aliasing by the shadow map and is not considered further. The priorities of the remaining triangles are weighted according to their projected area based on the observation that aliasing is perceptually most apparent on large flat surfaces and is somewhat masked on small, thin structures.

Rendering: The shadows are rendered in two passes using a one-bit stencil buffer. In the first pass the triangles of $\mathcal{SR}_F$ and $\mathcal{SR}_P$ are rendered with only ambient lighting. Then the shadow-polygons computed by the clipping algorithm are rendered to the stencil buffer with the depth test enabled. At this point the stencil is set wherever there is shadow. In the second pass the triangles in $\mathcal{SR}_P$ and $\mathcal{FV}$ are rendered with full lighting using the stencil test to prevent writing in the shadowed regions. A small amount of triangle offset may be required for the shadow-polygons to account for errors due to limited depth precision.

4.3 Process-Parallel Algorithm

It may not be possible to compute the PVS_E and PVS_L, and perform cross-culling and shadow generation on a single graphics processor at interactive rates. As a result, we use a process-parallel algorithm that uses three graphics cards (on three different PCs) and pipelines the computation. However, our algorithm introduces one frame of additional latency in the pipeline. An architecture of the resulting system is shown in Fig. 5. The three graphics cards and the CPUs perform the following operations:

Figure 6: *A snapshot generated from an application of our interactive shadow generation algorithm to the house model. The model has about 1.3M triangles. No LODs were used.*

- GPU_1: Computes PVS_E.
- GPU_2: Computes PVS_L.
- GPU_3 & CPU: Perform cross-culling, shadow generation and render the final scene.

At the beginning of each frame, the PC with GPU_3 transmits the current light-view and eye-view to the other GPUs and receives the corresponding PVSs for the previous frame. All the communication between the PCs is synchronized using acknowledgements.

4.3.1 Network Transmission

Our algorithm assumes that the scene graph is replicated on each PC. Instead of sending a list of visible triangles computed by sub-object culling, our algorithm uses id's for each sub-object in the scene graph and transmits those id's to GPU_3. Each id is represented by 4 integers and requires 16 bytes. In most cases, a small change occurs in the PVS computed between successive frames. Instead of sending a list of all the visible triangles in the sub-objects, the algorithm keeps track of changes between successive frames and only transmits those changes. Overall, frame-to-frame coherence results in lower network traffic between the PCs.

5 Implementation and Performance

In this section, we describe the implementation of our algorithm and highlight its performance on three complex environments.

5.1 Implementation

We have implemented our hybrid algorithm on 3 Dell Precision workstations, with dual 1.8 GHz pentium CPUs, 2 GB of main memory and a NVIDIA GeForce-4 Ti 4600 GPU. Typically, we are able to render 2M triangles in immediate mode and about 14M triangles in retained mode. We replicate our scene database across each PC.

For higher performance, we allocate 72MB out of 128MB on each GPU to store the vertices of objects, sub-objects, and bounding boxes. The memory allocated in the graphics card is sufficient to hold 6 million vertices. We use memory management if we exceed this limit. We also use NVIDIA vertex arrays in video memory to accelerate rendering. Our algorithm keeps track of the starting location of the vertices of each object in the video memory. Vertex arrays in NVIDIA GeForce-4 card have an index limit of 1 million, each object. Objects must store their position in video memory so that if they are not within this limit, the vertex array pointer may be changed.

We use the NVIDIA OpenGL extension *GL_NV_occlusion_query* to perform image-based occlusion queries. In order to avoid stalls in the graphics

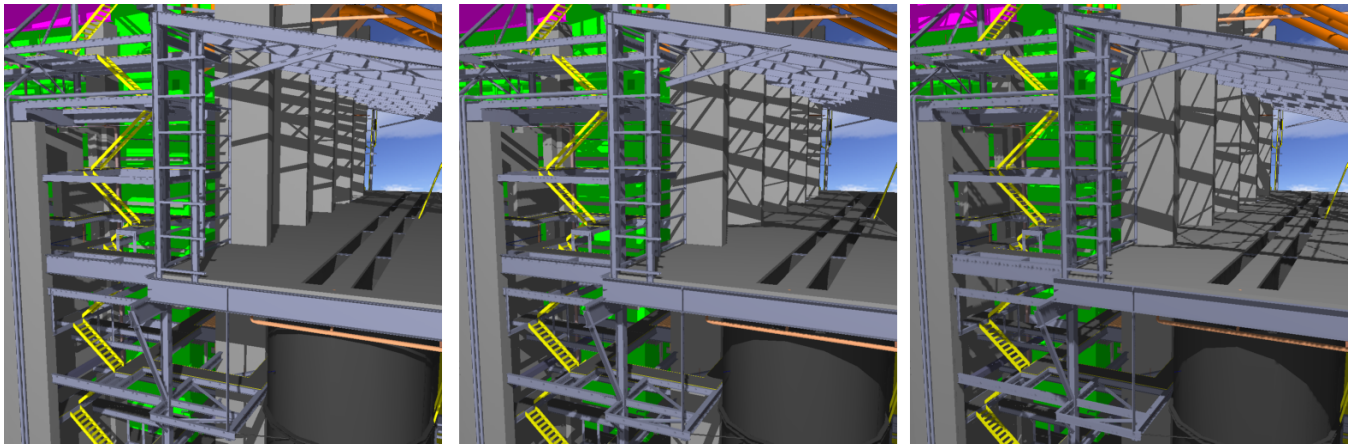


Figure 7: *A sequence generated by a light source moving over the power plant away from the viewer. Our algorithm can generate shadows at* 10 *frames per second on average.*

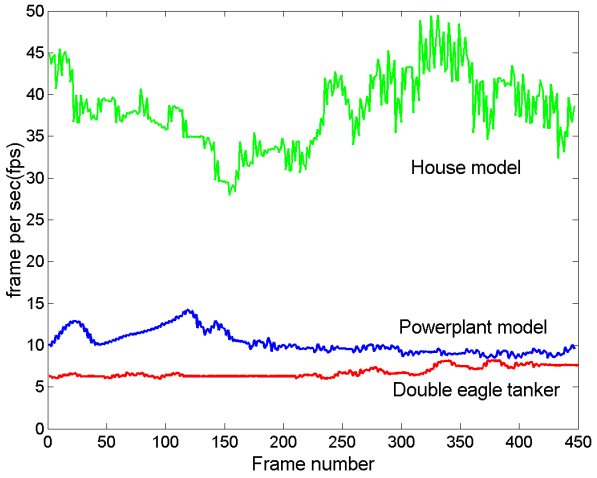


Figure 8: *This graph highlights the frame rates obtained for each model. The house model was rendered without any LODs. The frame rate is lower on the power plant and the tanker model because of long and thin objects. This results in a higher number of primitives in* $\mathcal{SR}_P$ *and* $\mathcal{SC}$.

pipeline, we perform all the queries at once and then obtain the results at the end. In theory, current graphics processors can perform these queries at the rate of rasterization. However, we have observed a considerable overhead due to implementation of current drivers, which limit the number of queries performed. The current driver for NVIDIA GeForce 4 performs about $240K$ queries per second on the Linux OS. In order to work around this limitation we introduce sub-objects to the hierarchy instead of performing occlusion culling directly at the triangle level. We allocate 8 triangles per sub-object (i.e. $k = 8$). We perform around 20K occlusion queries per frame (on average) to do cross-culling. We have utilized stencil tests along with depth tests in these occlusion queries to compute the shadow-casters. In order to maintain interactive frame rates we use a a budget of 50K shadow-polygon intersections in the hybrid shadow generation algorithm.

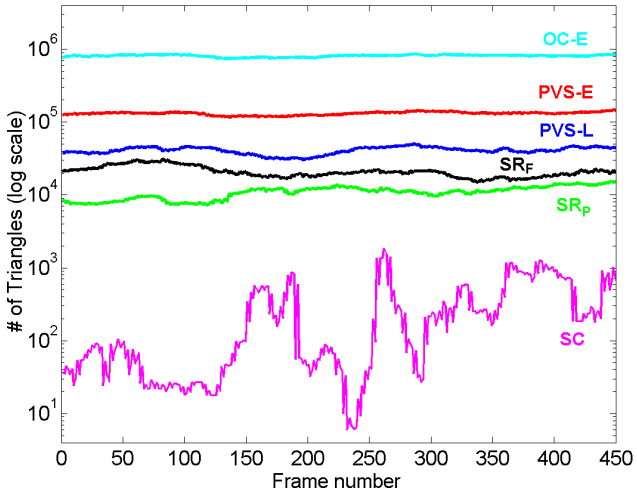
5.2 Performance

In this section, we highlight the performance of our algorithm on three models. We generated multiple paths through each model. The graphs show the performance of different culling algorithms used for shadow generation (as shown in Fig. 9) in a portion of the path through the powerplant and the tanker. In these paths, PVS_E is more than PVS_L. Moreover, cross-culling is able to reduce the size of potential shadow-casters and shadow-receivers by almost one order of magnitude.

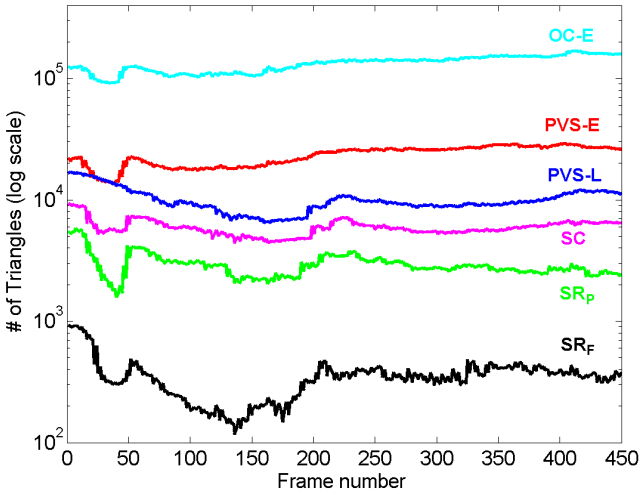
We tested our system on three complex models with moving light sources:

- A power plant model (shown in Fig. 1) composed of more than $1,200$ objects and 12.7M triangles. It is a very challenging environment for real-time rendering and shadow generation algorithms. It consists of many long, thin pipes, which cause considerable aliasing in shadow maps. The pipes are made up of narrow triangles that increase the number of shadow-polygons generated by the clipping algorithm. Therefore, it would be very difficult for a purely object-based approach to generate shadows at interactive rates on this model. We generated multiple paths inside and outside the power plant (as shown in the video). Fig. 7 shows a sequence of images generated by the moving light source in the power plant model.
- A Double Eagle tanker model (shown in Fig. 10) composed of more than 82 million triangles. Like the power plant, it has long and thin objects as well. We use a point light source and moved it on top of a deck the tanker, as the eye-view follows it. We also generate a path in the engine room. We also generate a path in the engine room.
- An architectural model (shown in Fig. 6) of a replicated house composed of more than 1.3 million triangles. It consists of a number of rooms with furniture. We generated a path inside the house such that a pure shadow map based approach will result in projective aliasing.

We highlight the frame rates obtained over the different paths in each model in Fig. 8. The upper bound on the number of shadow-polygons is $50,000$. The objects in the house model are not over-tessellated. As a result, we do not use LODs in the PVS computation algorithm. The culling algorithm works quite well and the size of $\mathcal{SR}_P$ and $\mathcal{SC}$ is relatively small. The power plant and the tanker model are much more challenging. They have about $38K$ and $54K$ objects, obtained after partitioning and clustering. As a result, the PVS computation algorithm spends more time in traversing the tree and performing object culling and sub-object culling. These structures consist of many long and thin triangles. As a result, the size of $\mathcal{SR}_P$ and $\mathcal{SC}$ is relatively big (as compared to the house model). The average



(a) Double Eagle Tanker model at 20 pixels of error



(b) Powerplant model at 10 pixels of error

Figure 9: *These graphs highlights the performance of our culling algorithms. The OC_E refers to the number of triangles after object culling. The PVS's are obtained by performing sub-object culling. Notice almost one order of magnitude reduction in PVS_E as compared to OC_E. The cross-culling algorithms perform significant culling and the size of potential shadow-casters ($\mathcal{SC}$) and shadow-receivers ($\mathcal{SR}_P$ and $\mathcal{SR}_F$) is of the order of few thousands (three orders of magnitude less than the original model size).*

frame-rate in the power plant and tanker model is 10 and 7, respectively.

6 Analysis and Limitations

In this section, we analyze the performance of our algorithm and highlight some of its limitations.

6.1 Analysis

The overall performance of our algorithm is governed by a number of factors. These include the model complexity, scene graph representation, the relative positions of the light-view and the eye-view, and the rate at which we can perform the occlusion queries. We have already highlighted the issues that arise from the use of LODs in Section 3.3. We address some other factors in this section.

Image-space Occlusion Queries: Our PVS computation and shadow generation algorithms use image-space occlusion queries to accelerate the visibility computations. These are performed at image-precision. It is possible that the projection of some object is smaller than a pixel or that it covers only a portion of a pixel. The object's visibility may therefore be mis-classified. A mis-classification results in missing shadows or polygons that are incorrectly deemed to be fully shadowed. The seriousness of this problem depends on the resolution used for occlusion culling. In practice, we have seen very few of these artifacts.

Cross-culling: The purpose of cross-culling is to limit the number of potential shadow-casters and shadow-receivers. It has additional overhead due to the occlusion queries (about $20-40$ milli-seconds in our current benchmarks). Its benefit depends on the relative size of $\mathcal{SR}_P$with respect to PVS_E and that of $\mathcal{SC}$with respect to PVS_L.

6.2 Interactive Performance and Load Balancing

The performance of the overall algorithm is governed by the performance of each stage (as shown in Fig. 4) as well as the network latency between the PCs. Because we utilize frame-to-frame coherence, the relative change in the size of PVS is typically low; and therefore, network bandwidth or latency is typically not a significant factor in the overall performance. The main factors that affect the system performance are the LOD error threshold and the capabilities of the graphics processors. A higher LOD threshold improves

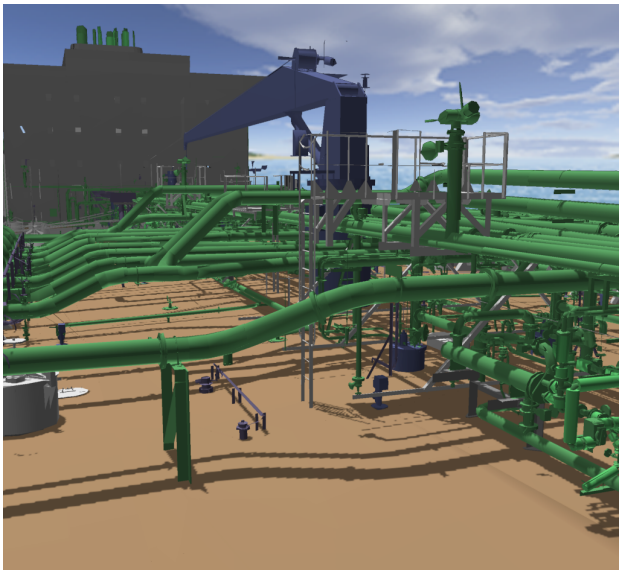


Figure 10: *A snapshot of the tanker model rendered using our system. The tanker has more than 82 million triangles. This view highlights the shadows generated by the long and thin pipes on the deck. The average frame rate is 7 frames per second.*

the performance of the overall algorithm, at the cost of image quality. A faster GPU will speed up both rendering and occlusion culling.

The first stage of the algorithm computes the PVS from the eye-view and the light-view. If a scene does not have high depth complexity, we need not perform occlusion culling using separate graphics processors. Otherwise, we compute each PVS in parallel on separate PCs. Both of them traverse the scene graph using an identical LOD selection criterion (based on the eye-view). The difference in their performance depends on the extent of culling obtained via view-frustum and occlusion culling. Within each PVS computation, the sub-object culling step typically requires more occlusion queries than the object-culling step. Therefore,

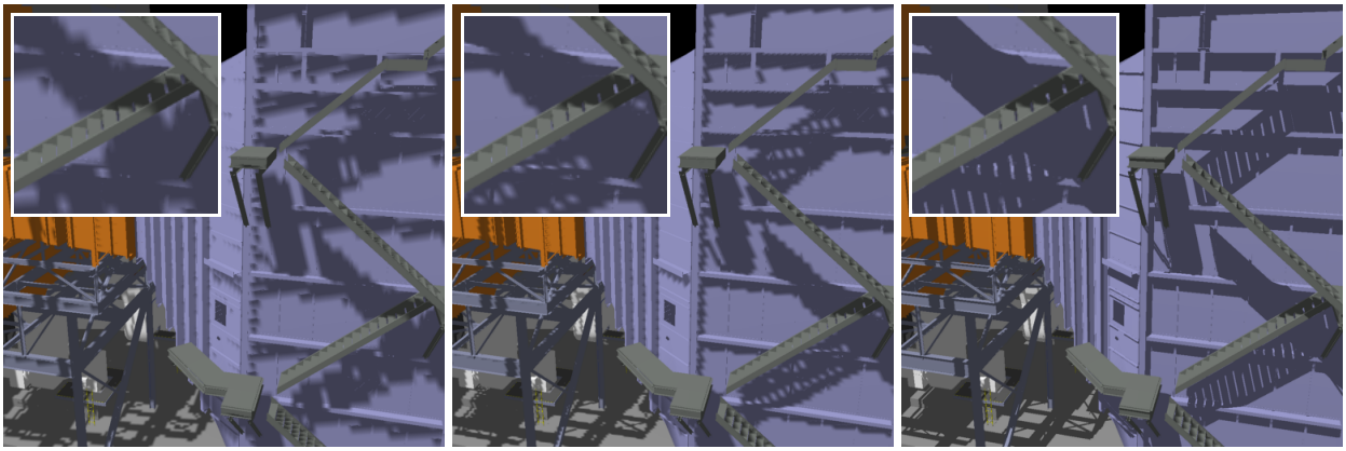


Figure 11: *Comparison of shadows generated by uniform shadow maps (left), perspective shadow maps (middle) and our hybrid algorithm (right). Each image also includes a zoomed view of the shadow boundaries on the top-left corner. Perspective shadow reduce some of the aliasing artifacts as compared to uniform shadow maps. However, they are unable to generate sharp shadows in many scenarios.*

more time is spent in SGC than computing the occlusion representation.

The performance of cross-culling depends on the size of PVS_L and PVS_E. The size of $\mathcal{FV}$, $\mathcal{SR}_F$ and $\mathcal{SR}_P$ is governed by the position of the light-view relative to the eye-view. The performance of shadow generation routine is mostly determined by the size of $\mathcal{SC}$ and $\mathcal{SR}_P$. In general, the smaller the size of these sets, the more shadow polygons can be generated, leading to higher image quality. The number of shadow-polygons that can be computed by our algorithm is bounded by the speed of the CPU.

Load Balancing: It is possible that the algorithm spends more time in PVS computation than the other stages. In this case we can either increase the LOD threshold use a higher value of k, the number of triangles per sub-object, to reduce the number of occlusion queries that are performed. If cross-culling or shadow generation becomes the bottle neck we use a smaller k. This lowers the size of PVS_E and PVS_L, as well as the number of potential shadow-casters and shadow-receivers.

6.3 Comparison with Other Approaches

In this section, we briefly compare our algorithm with earlier approaches.

Shadow Maps: Our 3-PC based hybrid shadowing algorithm can greatly reduce the aliasing artifacts that are present in shadow map based approaches (shown in Fig. 11). Uniform shadow maps [Williams 1978] are simple to use but suffer from the aliasing. Perspective shadow maps greatly reduce the aliasing problem for many view configurations [Stamminger and Drettakis 2002], but cannot always eliminate it completely. This is especially true when the field-of-view is narrow or when the near plane must be kept close to the viewpoint. Our hybrid algorithm improves the shadow quality in parts of the scene where the aliasing artifacts are the worst. Overall, it yields higher quality and sharper shadows, as compared to pure image-precision approaches.

Shadow Volumes: Shadow volumes are too slow for large models because the current graphics systems cannot handle the sheer number of shadow-casters. Asymptotically, the running time for shadow volumes is linear in the number of shadow-casters, while our shadow-polygon computation algorithm is super-linear. However, the cost of shadow volumes is dominated by the fill-rate. It is likely that future graphics systems will have sufficiently high fill rates that shadow volumes can be used in place of our current shadow generation algorithm (in the third stage). Note that our algorithms for PVS computation and cross-culling can be used to improve the performance of a pure shadow volume based approach.

Shadow Volume Reconstruction from Depth Maps: Our hybrid algorithm is different from that of [McCool 2000] in that our shadow boundaries can have object-precision. McCool [2000] uses only the information in the depth buffer to construct shadow boundaries. We use the depth buffer as part of the culling step to determine which objects are casting shadows. Ultimately the object-space precision of the shadow-casters is used to compute shadow boundaries where shadow maps are not sufficient. As a result, we get sub-pixel accuracy and sharp edges. Moreover, McCool's algorithm requires depth-buffer readback during each frame, which can be slow on current graphics systems (e.g. 50 milliseconds at $1K \times 1K$ resolution from a high-end PC with NVIDIA GeForce 4 card). This overhead limits the usefulness of the algorithm for interactive performance in complex environments.

6.4 Limitations

Our current algorithm can only generate hard-shadows from point-light sources. In theory, we can use more than one light-source, but this requires additional graphics processors (one per each light source). Moreover, we will have to perform more visibility computations during cross-culling and shadow generation. One of the main limitations of our current system is the additional latency in the pipeline. This latency is mainly introduced by the PVS computation algorithm, which is performed on separate graphics processors. Typically, this latency is slightly less than one frame. This is in addition to double-buffering and can be a major issue for latency-sensitive applications.

Our algorithm expects high coherence between successive locations of the eye-view and the light-view. If either view undergoes drastic motion, the set of visible primitives from the previous frame may not approximate well to the potential occluders for the current frame. As a result, the sizes of the PVS's, $\mathcal{SC}$ and $\mathcal{SR}_P$, can become large and this can increase the frame time.

The use of LODs can introduce visual artifacts as well as inaccurate shadow boundaries. We have highlighted the inaccuracies in the shadow boundaries in Section 3. The use of LODs can result in popping, as the PVS computation algorithm switches between different LODs. Some of these artifacts can be eliminated by performing view-dependent

simplification.

Object-space calculation of shadow polygons may not work well for highly tessellated curved models. If the number of shadow-polygons is very high, the cost of performing intersection tests can increase rapidly to the point that the CPU cannot handle it at interactive rates. In these cases, the intersection test budget may be exceeded long before significant aliasing is removed. Our algorithm works best in environments with large, flat polygons.

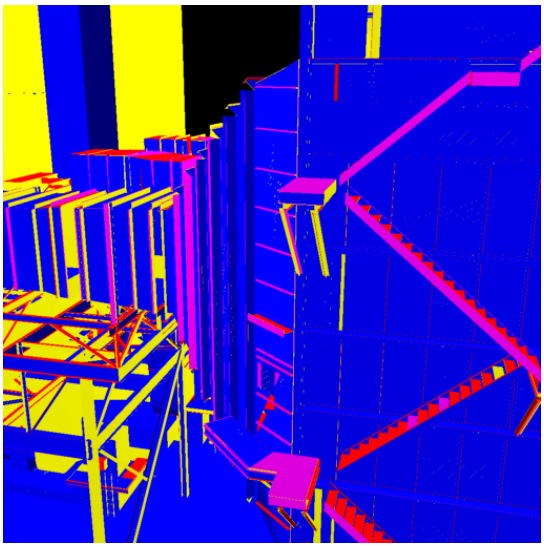


Figure 12: *A color-coded classification of the polygons in Fig. 11 using our hybrid approach. Blue polygons are rendered as shadow-polygons. Red polygons are rendered with the shadow map. Yellow polygons are completely shadowed. Magenta polygons are completely visible.*

7 Summary and Future Work

We have presented a hybrid algorithm for interactive shadow generation in complex environments with a moving light source. Our algorithm can generate shadows with sharp edges and reduces the aliasing artifacts that are present in pure image-precision approaches. We have applied it to three large models and our preliminary results are very encouraging. We have also presented an improved algorithm for PVS computation, which can be useful for other rendering applications. Moreover, our cross-culling algorithm can accelerate the performance of a pure shadow volume based approach.

There are many avenues for future work. Our current approach only handles point and directional light sources. We would like to extend the object-space computation to calculate penumbras from area light sources to produce more natural looking shadows. Our algorithm can be extended to handle omni-directional or multiple light sources by using additional graphics processors. We would like to develop an out-of-core system that doesn't need to replicate the database on 3 PCs and load the entire scene graph in the main memory. Finally, it will be useful to use view-dependent simplification as compared to static LODs, as it can reduce the popping artifacts.

References

Andujar, C., Saona-Vazquez, C., Navazo, I., and Brunet, P. 2000. Integrating occlusion culling and levels of detail through hardly-visibly sets. In *Proceedings of Eurographics*.

Atherton, P., Weiler, K., and Greenberg, D. 1978. Polygon shadow generation. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, vol. 12, 275–281.

Baxter, B., Sud, A., Govindraju, N., and Manocha, D. 2002. Gigawalk: Interactive walkthrough of complex 3d environments. *Proc. of Eurographics Workshop on Rendering*.

Bergeron, P. 1985. Shadow volumes for non-planar polygons. In *Graphics Interface '85 Proceedings*, 417–418.

Blinn, J. 1988. Jim blinn's corner: Me and my (fake) shadow. *IEEE Computer Graphics and Applications 8*, 1 (Jan.), 82–86.

Brabec, S., Annen, T., and Seidel, H. 2001. Hardware-accelerated rendering of antialiased shadows. In *Proc. of Computer Graphics International*.

Brabec, S., Annen, T., and Seidel, H. 2002. Shadow mapping for hemispherical and omnidirectional light sources. In *Proc. of Computer Graphics International*.

Brotman, L. S., and Badler, N. I. 1984. Generating soft shadows with a depth buffer algorithm. *IEEE Computer Graphics and Applications 4*, 10, 71–81.

Chin, N., and Feiner, S. 1989. Near real-time shadow generation using BSP trees. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, vol. 23, 99–106.

Chrysanthou, Y., and Slater, M. 1995. Shadow volume BSP trees for computation of shadows in dynamic scenes. In *1995 Symposium on Interactive 3D Graphics*, 45–50.

Cohen-Or, D., Chrysanthou, Y., and Silva, C. 2001. A survey of visibility for walkthrough applications. *SIGGRAPH Course Notes # 30*.

Crow, F. 1977. Shadow algorithms for computer graphics. vol. 11, 242–248.

El-Sana, J., Sokolovsky, N., and Silva, C. 2001. Integrating occlusion culling with view-dependent rendering. *Proc. of IEEE Visualization*.

Everitt, C., and Kilgard, M. 2002. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. In *SIGGRAPH 2002 Course Notes*, vol. 31.

Fernando, R., Fernandez, S., Bala, K., and Greenberg, D. 2001. Adaptive shadow maps. In *Proceedings of ACM SIGGRAPH 2001*, 387–390.

Govindaraju, N., Sud, A., Yoon, S., and Manocha, D. 2002. Interactive visibility culling with occlusion-switches. Tech. Rep. CS-02-027, University of North Carolina. To appear in Proc. of ACM Symposium on Interactive 3D Graphics.

Greene, N., Kass, M., and Miller, G. 1993. Hierarchical z-buffer visibility. In *Proc. of ACM SIGGRAPH*, 231–238.

Heidmann, T. 1991. Real shadows real time. *IRIS Universal*, 18.

Heidrich, W., and Seidel, H. P. 1999. Realistic hardware-accelerated shading and lighting. In *Proc. of ACM SIGGRAPH*, 171–178.

Luebke, D., Reddy, M., Cohen, J., Varshney, A., Watson, B., and Huebner, R. 2002. *Level of Detail for 3D Graphics*. Morgan-Kaufmann.

McCool, M. 2000. Shadow volume reconstruction from depth maps. *ACM Trans. on Graphics 19*, 1, 1–26.

Parker, S., Martic, W., Sloan, P., Shirley, P., Smits, B., and Hansen, C. 1999. Interactive ray tracing. *Symposium on Interactive 3D Graphics*.

Reeves, W., Salesin, D., and Cook, R. 1987. Rendering antialiased shadows with depth maps. In *Computer Graphics (ACM SIGGRAPH '87 Proceedings)*, vol. 21, 283–291.

Segal, M., Korobkin, C., van Widenfelt, R., Foran, J., and Haeberli, P. 1992. Fast shadows and lighting effects using texture mapping. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, vol. 26, 249–252.

Stamminger, M., and Drettakis, G. 2002. Perspective shadow maps. In *Proceedings of ACM SIGGRAPH 2002*, 557–562.

Tadamura, K., Qin, X., Jiao, G., and Nakamae, E. 2001. Rendering optimal solar shadows with plural sunlight depth buffers. *The Visual Computer 17*, 2.

Udeshi, T., and Hansen, C. 1999. Towards interactive photorealistic rendering of indoor scenes: A hybrid approach. In *Rendering Techniques '99*, D. Lischinski and G. W. Larson, Eds., 63–76.

Wald, I., Slusallek, P., and Benthin, C. 2001. Interactive distributed ray-tracing of highly complex models. In *Rendering Techniques*, 274–285.

Wanger, L. 1992. The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, vol. 25, 39–42.

Williams, L. 1978. Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, vol. 12, 270–274.

Woo, A., Poulin, P., and Fournier, A. 1990. A survey of shadow algorithms. *IEEE Computer Graphics and Applications 10*, 6 (Nov.), 13–32.