Eurographics Symposium on Rendering (2004)
A. Keller, H. W. Jensen (Editors)

CC Shadow Volumes

D. Brandon Lloyd, Jeremy Wendt, Naga K. Govindaraju, and Dinesh Manocha

University of North Carolina at Chapel Hill
http://gamma.cs.unc.edu/ccsv

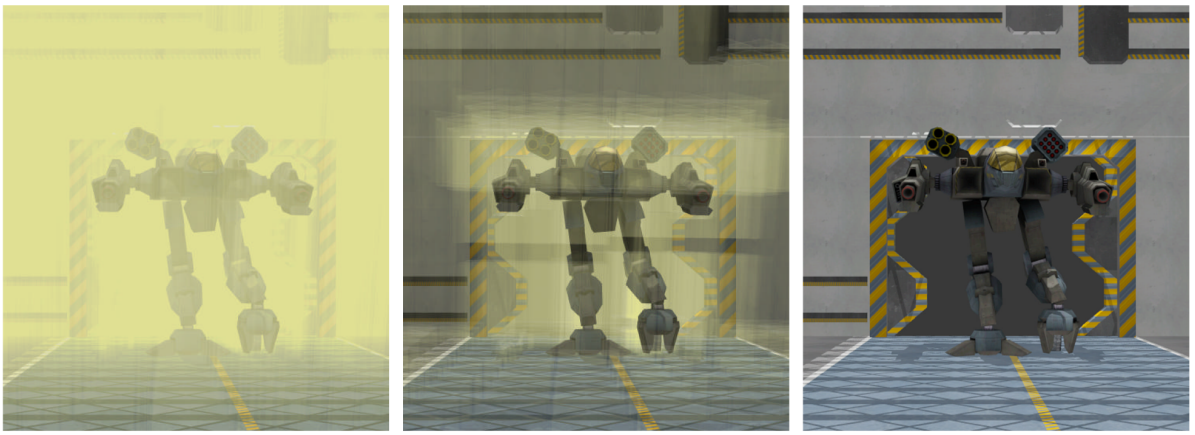

Figure 1: *These images demonstrate the benefits of CC shadow volumes on a scene with* $96K$ *polygons. Standard shadow volumes are shown in the left image and CC shadow volumes in the middle. Shadow volumes are shown in transparent yellow. The right image shows the shadows generated by CC shadow volumes at interactive rates. CC shadow volumes generate up to* 7 *times less fill than standard shadow volumes in this scene.*

Abstract

We present a technique that uses culling and clamping (CC) for accelerating the performance of stencil-based shadow volume computation. Our algorithm reduces the fill requirements and rasterization cost of shadow volumes by reducing unnecessary rendering. A culling step removes shadow volumes that are themselves in shadow or do not contribute to the final image. Our novel clamping algorithms restrict shadow volumes to those regions actually containing shadow receivers. In this way, we avoid rasterizing shadow volumes over large regions of empty space. We utilize temporal coherence between successive frames to speed up clamping computations. Even with fairly coarse clamping we obtain substantial reduction in fill requirements and shadow rendering time in dynamic environments composed of up to a 100K triangles.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Color, Shading, Shadowing and Texture

1. Introduction

Shadows are important in computer graphics because they add realism to a scene and can aid in understanding spatial relationships between objects. Shadows have been an active area of research in computer graphics for more than two decades. Advances in graphics hardware have made it possible to accurately render shadows from point light sources in interactive applications including games and walkthroughs.

Shadow volumes [Cro77] are a popular technique for shadow generation. A shadow volume is the region of space behind a shadow caster containing points that lie in shadow. Shadow volumes compute shadow boundaries implicitly, which makes them attractive for computing shadows on complex geometry. Moreover, the asymptotic complexity of shadow generation is linear in the number of shadow caster polygons.

Shadow volumes can be implemented using the stencil buffer on current graphics systems. The algorithm proceeds in three stages. First, the scene is rendered with only ambient lighting. Second, the shadow volumes are rendered to the stencil buffer, which sets the stencil for shadowed pixels. Finally, the scene is rendered again with full lighting using the stencil test to prevent shadowed pixels from changing.

A major drawback of the algorithm is that the rasterization of shadow volumes can be expensive. Since shadow volumes extend away from shadow casters toward infinity, they sometimes cover much of the screen, leading to high bandwidth and fill consumption. Often large portions of the rendered shadow volumes make no contribution to the final image. The three main sources of unnecessary shadow volume rendering are: large regions of empty space, shadow casters completely enclosed in other shadow volumes, and shadow generation on parts of the scene not visible to the viewer.

Main Results: We present methods for accelerating the performance of shadow volume computations. Our algorithms target scenarios where shadow volume rasterization is the major bottleneck. We decrease the rasterization cost by using two techniques:

Shadow Volume Culling: Using a simpler variation of the shadow culling algorithm presented in Govindaraju et al. [2003], we eliminate the shadow casters that are themselves completely in shadow. We also eliminate the shadow casters whose shadows are not visible to the eye. Shadow volume culling is shown in Fig. 2(b).

Shadow Volume Clamping: By clamping the extents of each shadow volume to the intervals containing shadow receivers we avoid unnecessary rendering in large regions of empty space. To compute the occupied intervals of a shadow volume we use two techniques. The first technique employs bounding volumes to identify continuous intervals along the shadow volume that contain objects. (Fig. 2(c)). Temporal coherence is used to accelerate the computations by performing incremental computations between successive frames. Our second technique divides a shadow volume into discrete intervals and utilizes the graphics hardware to test for objects within each interval (Fig. 2(d)).

The culling and clamping (CC) algorithms often work well together. Culling eliminates completely shadowed objects, creating empty space in the shadow volumes. The size of the shadow volumes is reduced by the clamping algorithms, leading to lower rasterization costs.

We have tested our algorithms on a PC with an NVIDIA GeForce FX 5950 Ultra graphics card. In a dynamic environment composed of 100K triangles, we have observed up to a 7 times reduction in fill and a 4 times speed-up in shadow volume rendering time by using CC shadow volumes over standard shadow volumes.

Organization: This paper is organized as follows: Section 2 reviews previous research in the area of interactive shadow generation. Section 3 provides the details of shadow volume culling and clamping. We describe our implementation of the algorithms in Section 4 and highlight their performance. We analyze our techniques in Section 5, discuss some of their limitations, and compare them with other methods.

2. Related Work

Shadow volumes were introduced by Crow [Cro77]. Bergeron [Ber85] generalized shadow volumes for non-manifold objects and non-planar polygons. BSP trees have been used to accelerate shadow volume computation [CF89, CS95, BJ99], but they do not work well with dynamic lights or many moving objects.

One of the first hardware implementations of shadow volumes was demonstrated in Pixel-Planes 4 [FGH*85]. Heidmann [Hei91] implemented Crow's algorithm on graphics hardware using the stencil buffer. This approach, known as the *z-pass* method, can produce incorrect results when the viewport cuts through a shadow volume. Diefenbach [Die96] presented capping methods, but these were not completely robust. To overcome these problems several researchers have proposed *z-fail* testing for shadow volume computation [Car00, EK02]. Brabec and Seidel [BS03] described an algorithm for fast shadow volume computation using the graphics hardware for silhouette edge computation.

To deal with the fill-consumption problem Lengyel [Len02] proposed using the scissor test to restrict shadow volume rendering to within the light bounds. McGuire et al. [MHE*03] improved upon Lengyel's algorithm by adding culling and using the depth bounds test to further restrict shadow volume rendering. Chan and Durand [CD04] use a hybrid of shadow maps and shadow volumes to reduce fill. They render a shadow map to identify shadow boundaries and render shadow volumes only in these areas. Aila and Möller [AAM04] perform shadow volume calculations on coarse tiles in screen space to determine which tiles contain shadow boundaries, then render shadow volumes only in these tiles. These approaches rely on existing or proposed culling hardware to avoid unnecessary rendering. Our algorithm is orthogonal in that it reduces the size and complexity of the shadow volumes that are rendered in the first place.

3. Shadow Volume Acceleration

In this section, we present our algorithms for accelerating shadow volumes. We represent a scene as a hierarchical scene graph. Each object in the scene is represented as a leaf node in the hierarchy. We decompose spatially large objects into smaller sub-objects using a k-D tree to provide better localization. The sub-objects can largely be treated as independent objects except when rendering shadow volumes as explained later in Section 3.3.

3.1. Shadow Volume Culling

Each object in the scene can be a potential shadow caster as well as a potential shadow receiver. The purpose of shadow

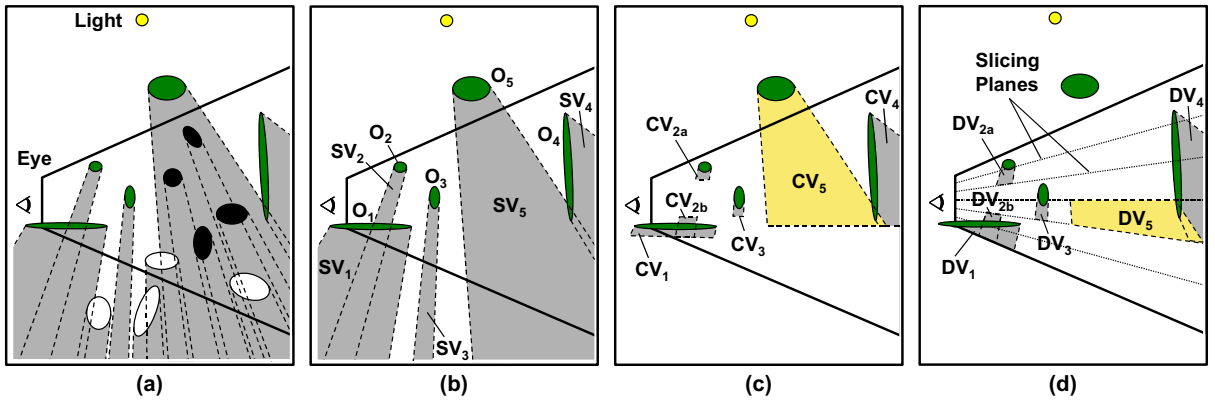

Figure 2: ***Shadow Volume Acceleration****: (a) Every object in the scene is a potential shadow caster and receiver. (b)* ***Shadow Volume Culling****: The shadow casters visible to the light are PSC* $= \{O_1, \ldots, O_5\}$ *and shadow receivers visible to the eye are PSR* $= \{O_1, O_2, O_3, O_5\}$*. (c)* ***Continuous Shadow Clamping:*** *Shadow volumes* SV_i *are clamped by using AABBs around the shadow receivers to compute clamped volumes* CV_i*. (d)* ***Discrete Shadow Clamping:*** SV_i *are clamped to intervals defined by slicing planes. Testing for actual object containment results in a smaller* DV_5*. Discrete clamping may be combined with continuous clamping to refine the poorly clamped* CV_5*, producing a optimal set of shadow volumes* $\{CV_1 \ldots CV_4, DV_5\}$ *which significantly reduces fill requirements.*

volume culling is to eliminate those shadow casters that are themselves in shadow or that cast shadows not contributing to the final image [GLY*03]. The computation proceeds in two steps:

1. Compute potential shadow receivers (PSR): PSR consists of the set of objects that may be visible from the viewpoint of the eye.

2. Compute potential shadow casters (PSC): PSC consists of objects that may be visible from the viewpoint of the light (see Fig. 2(b)).

We use occlusion queries and stencil tests for computing *PSC*. From the viewpoint of the light, we render the scene, creating a representation of the visible surface in the depth buffer. Next, with depth buffer writes disabled, we render the bounding box of each object using an occlusion query. The occlusion query indicates whether any pixels pass completely through the pipeline. If all pixels fail the depth test then the object is completely occluded, i.e. completely in shadow. Objects with visible bounding boxes are added to the *PSC*. A similar algorithm can be used to compute *PSR*, except that the scene is rendered from the viewpoint of the eye. If a scene has little occlusion, we use only view-frustum culling to compute *PSC* and *PSR*.

PSC may contain objects that cast shadows on areas that are not visible to the eye. Step 2 can be modified slightly to remove these shadow casters. After rendering the occlusion representation, we render *PSR*, setting the stencil when the depth test fails. This identifies the shadowed regions of *PSR*. When performing occlusion queries, we also enable the stencil test so that only shadow casters covering shadowed regions are included in *PSC*.

Figure 3: *Standard shadow volumes (left) vs. CC shadow volumes (right).*

3.2. Shadow Volume Clamping

Each shadow volume extends to infinity, thus its projection can cover a large number of pixels in the stencil buffer. To reduce fill consumption we compute *clamped volumes* that more tightly enclose the objects in *PSR* (see Fig. 2(c)).

We use two different clamping techniques that are in some ways complimentary. Each technique uses a different method for determining where interactions occur between shadow receivers and a particular shadow volume. The continuous algorithm clamps precisely to the bounds of the shadow receivers, but can overestimate the size of a shadow volume when only a small part of the receiver lies in shadow. The discrete algorithm clamps only to truly occupied regions, but the bounds on the region are only as accurate as the region discretization. Both algorithms can be used independently or together. Poorly clamped volumes result-

ing from continuous clamping can be further refined using discrete clamping.

3.2.1. Continuous Shadow Clamping

Continuous shadow clamping proceeds in two steps (Fig. 4). First, we use overlap tests in the light view to identify the shadow receivers that lie in each shadow volume. Then we compute the occupied intervals along a shadow volume by merging the extents of the shadow receivers it contains. These computations are performed entirely on the CPU.

Overlap Tests: We calculate overlaps using the axis-aligned bounding boxes (AABBs) of the objects' projection on the light's image plane. We also compute the depth interval (z_{min}, z_{max}) of each object in light-space. An object S possibly lies within the shadow volume of an object T when the AABBs of S and T overlap and when the z_{max} of S is greater than the z_{min} of T. Note that S and T may be mutually contained within the others' shadow volume.

Occupied Intervals Computation: Computation of occupied intervals can be performed efficiently by processing all the shadow casters simultaneously. Each shadow caster stores a list of the intervals occupied by the receivers within its shadow volume. The lists are initialized with the depth intervals of the shadow casters themselves to account for self-shadowing. Then we process the shadow receivers according to their z_{min} value, from smallest to largest, updating the occupied intervals of the shadow volumes in which each receiver lies. Since the shadow receivers are processed in order, only the last occupied interval in each list needs to be updated. There are three possible ways to update the occupied interval:

1. The occupied interval completely contains the receiver. Nothing is done in this case.

2. The occupied interval partially contains the receiver. The interval is extended to include the receiver.

3. The object lies completely outside the occupied interval. In this case the depth interval of the receiver is added to the occupied interval list.

Whenever a new occupied interval is created in case 3, two sets of caps and an extra set of edges must be drawn. If the size in screen-space of the gap between successive intervals is small, the cost of rendering the new shadow volume interval may exceed any savings in rasterization cost. A simple heuristic can be used for determining whether or not to create a new interval. Let V be the vertex processing cost of the new interval and R be the rasterization cost of the gap. If $V > R$, then a new interval is not created. Instead, the receiver depth interval is merged as in case 2.

V and R can be computed with the following equation:

$$V = (2C + 2S)v,$$
$$R = Ar$$

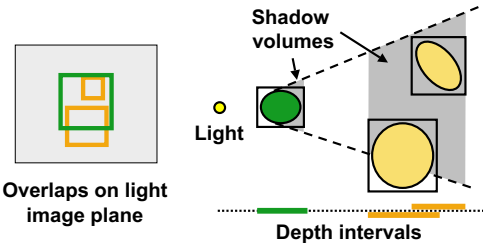

Figure 4: ***Continuous Clamping****: Bounding box overlap tests on the light's image plane are used to determine potential shadow receivers. Depth intervals of receivers are merged to determine shadow volume intervals.*

where C and S are the number of vertices in the shadow volume cap and silhouette respectively, v is cost per vertex, A is an estimate of the area of the gap in pixels, and r is the rasterization cost per pixel. The values of v and r can be determined empirically by rendering a "typical" set of shadow volumes at different resolutions. Rendering at a low resolution, e.g. 10×10, gives an estimate of v. Rendering at high resolution and subtracting v gives an estimate of r. Though this heuristic assumes an overly simplified model of graphics processing, it gives acceptable results.

Temporal Coherence: We accelerate the overlap tests by performing incremental computations. We employ a variation of the sweep-and-prune algorithm [CLMP95] used to perform bounding box overlap tests in large environments composed of multiple moving objects. We project the intervals of the AABBs along the X and Y axis in the light's image plane and sort the projected values along each axis. When the objects or the light move we update the AABBs and re-sort the list using an insertion sort with local interchanges. The list of depth intervals is maintained similarly. We compute the overlapping AABBs from the sorted lists.

3.2.2. Discrete Shadow Clamping

The continuous shadow clamping algorithm computes clamped volumes CV_i for each shadow caster based on the AABBs of shadow receivers. In many cases, AABBs can generate tight fitting clamped volumes (e.g. $CV_1 \ldots CV_3$ in Fig. 2(c)). Since the entire extents of the shadow receivers are used to determine occupied intervals, continuous clamped volumes may fit poorly when only a small part of a receiver lies within a shadow volume (e.g. CV_5 in Fig. 2(c)).

Discrete shadow clamping uses the GPU to test for shadow receivers within discrete shadow volume intervals. A set of similarly-oriented planes partition the view frustum into slices. The discrete intervals are determined by the intersection of the shadow volumes with the slices(Fig. 5). A convenient choice for slicing planes are those which face towards the light source and pass through the viewpoint, splitting the image plane into strips of equal width. The intervals created by these slicing planes cover an approximately equal area on the image plane, regardless of how far away

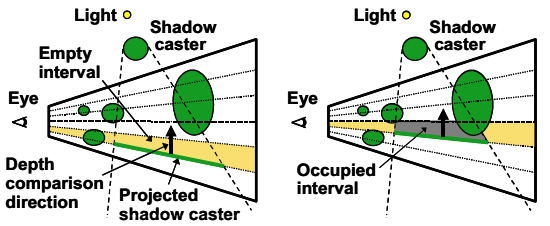

Figure 5: ***Discrete Clamping****: A single shadow caster is tested against an empty interval (left). In the next slice (right), the interval is occupied so a shadow volume is rendered for this interval.*

a shadow volume is from the viewpoint. An additional benefit is that the caps between intervals need not be drawn. Since they lie on planes that pass through the eye, the caps do not affect any pixels. Note that this choice of planes is most useful if the light source lies outside the view frustum. This corresponds to situations where the empty space problem is most common.

The discrete clamping computations are performed in the light's view. Each slice is rendered in back-to-front order using a pair of clipping planes. We test each shadow caster against a given slice with an occlusion query. We set depth test to *GREATER* and project the shadow caster onto the bottom plane of the slice. If the occlusion query indicates that no pixels passed the depth test, then the shadow volume interval corresponding to the slice is empty.

Projecting a shadow caster onto a plane does not change its shape in the light view, only the depth of the pixels rasterized. The projection is accomplished with a simple transformation matrix. A plane $\mathbf{p}$ is represented in homogeneous 4D coordinates as $\mathbf{p} = (a\ b\ c\ d)$ with vector components corresponding to the coefficients of the plane equation:

$$ax + by + cz + d = 0$$

The 4×4 matrix $\mathbf{M}$ that projects onto the plane $\mathbf{p}$ through a center of projection $\mathbf{c} = (c_x\ c_y\ c_z\ 1)$ is given by:

$$\mathbf{M} = \mathbf{c} \otimes \mathbf{p} - (\mathbf{p} \cdot \mathbf{c})\mathbf{I}$$

where $\otimes$ denotes outer product and $\mathbf{I}$ is the identity matrix. $\mathbf{M}$ should be negated if the center of projection is below the plane. Though this has no effect on the final 3D coordinates of the projected points, it does ensure that the fourth coordinate is positive, which is necessary to prevent the point from being clipped by the graphics hardware. If a portion of an object is farther away from a slicing plane than the light, then it will not be projected correctly. To handle this problem, the edges of the object's shadow volume should be extruded to infinity from the last plane onto which the object can be correctly projected.

3.2.3. Conservative Discrete Shadow Clamping

The use of image-precision occlusion queries in discrete clamping may lead to sampling errors due to limited screen resolution and z-buffer precision. The errors can cause occupied intervals to be incorrectly classified as empty leading to small areas of missing shadows. Recently Govindaraj et al. [GLM04] described a technique for performing robust interference detection using graphics hardware. This technique "fattens" the geometric primitives sufficiently to avoid sampling errors. Discrete clamping is an interference computation problem between shadow casters and the objects within the slices. It is possible to adapt the conservative overlap tests [GLM04] to perform reliable clamping. First, the bounding representations of the shadow volumes are constructed and fattened. Using these representations, we can then perform overlap tests with the fattened objects enclosed in the slices. The resulting algorithm is slower but eliminates the image-precision artifacts.

3.3. Rendering Clamped Volumes

Both continuous and discrete clamped shadow volumes are rendered in a similar manner. In the continuous case, silhouette edges of a shadow caster are extruded to form quads that extend across the occupied intervals. Caps are drawn at interval boundaries. In the discrete case, the quads are formed by projecting the silhouette edges onto the slicing planes. No caps need to be drawn because they lie in the slicing planes which pass through the eye, so they have zero area in screen-space.

Two shadow casters may have edges in common if they are sub-objects of the same parent object. To avoid redundant shadow volume sides arising from shared edges, the list of occupied intervals for the triangles on either side of a shared edge are merged. Portions occupied on both sides are not drawn.

3.4. Omnidirectional Light Sources

The culling and clamping algorithms depend on planar projections, which implies that they can only be used within a restricted frustum. For omnidirectional light sources, CC shadow volumes can be used for a portion of the space around the light and standard shadow volumes can be used for the rest. This works best for lights that are located near the edge of the viewport or close to a wall. In these cases,

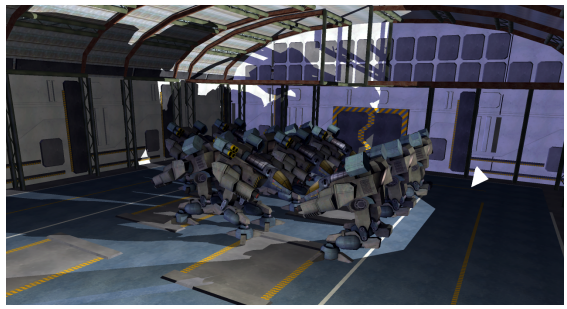

Figure 6: *CC shadow volumes in a 96K polygon scene composed of multiple robots.*

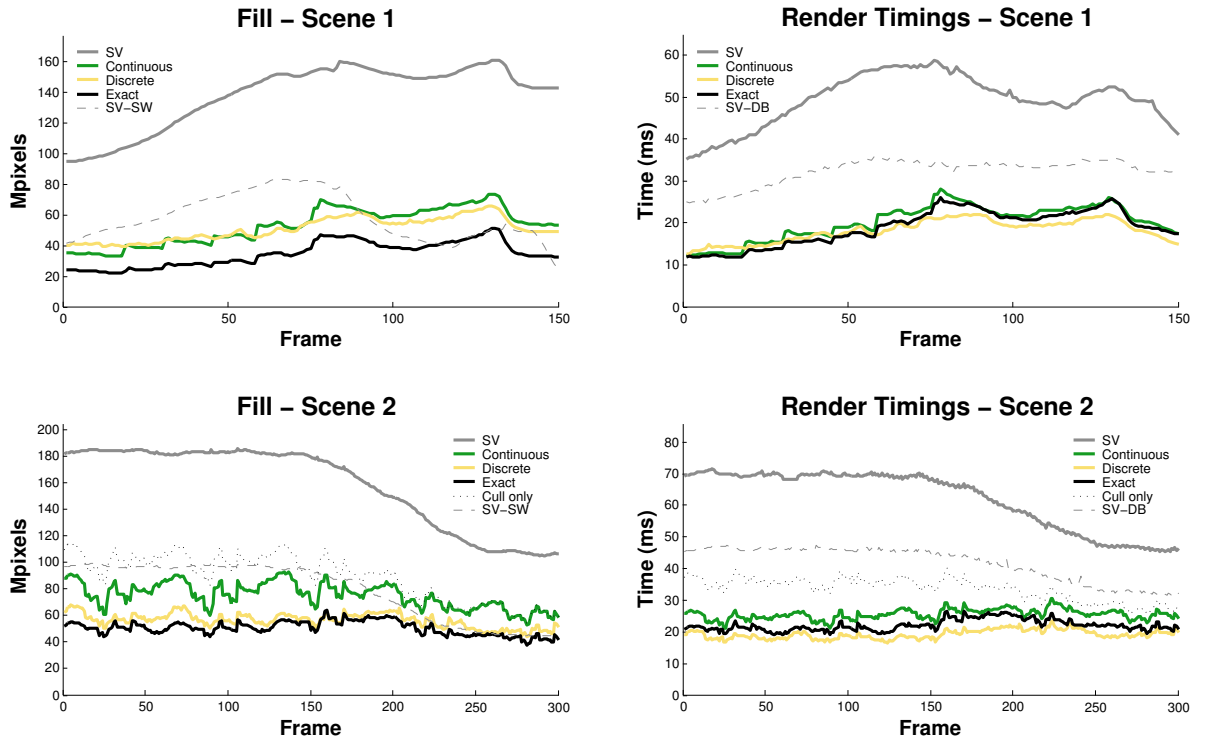

Figure 7: ***CC Shadow Volumes vs. Standard Shadow Volumes (SV)****: These graphs show the reduction in fill and shadow volume rendering time obtained with CCSVs in two test scenes with continuous, discrete, and exact clamping (described in Section 4). Also shown are the number of stencil writes (SV-SW) and the time to render shadow volumes with an empty depth bounds range (SV-DB).*

the large shadow volumes which would benefit most from culling and clamping lie predominantly in one direction.

4. Implementation and Performance

We have implemented our system on a PC with a NVIDIA GeForce FX 5950 Ultra graphics card and dual 2.8GHz Xeon processors, although only one processor was utilized for our computations. The system uses the OpenGL API. We use the *vertex array range* extension for accelerating shadow volume rendering by storing the vertex information in video memory on the graphics card. To perform occlusion queries, we use the *GL_NV_occlusion_query* extension.

The computations on the CPU and GPU are organized so as to maximize parallelism. Occlusion queries for culling and clamping require the CPU to wait for the GPU, so they are performed first. Then using data precomputed in the previous frame, the CPU rapidly issues the commands for the ambient pass, shadow volume rendering, and the lit pass. While the GPU processes the commands, the CPU is used for continuous clamping and constructing the shadow volumes for the next frame. The CPU computations usually finish before the GPU, so they have no affect on the frame rate.

We tested the performance of our system on two different scenes:

1. Corridor Scene: The corridor shown in Fig. 1 is part of a model consisting of 96K triangles. Most of the primitives are contained in the complex trusses overhead which cast shadows on the floor and walls below. This scene demonstrates the savings achieved by shadow volume clamping. Most of the shadow volumes traverse empty space. Since there is very little occlusion in the scene, we perform only view-frustum culling to compute *PSC* and *PSR*.

2. Robot Scene: This part of the scene is a large room with a crowd of moving robots (Fig. 6) which creates a high degree of occlusion. When the light is placed low in the scene, the shadow culling algorithm removes many shadow casters.

Fig. 7 shows the fill and shadow volume rendering times for the paths shown in the accompanying video. The test scenes were rendered at 1280×1024 resolution. Clamping and culling are performed at the same resolution. We measured the performance of standard shadow volumes and CC shadow volumes with continuous and discrete clamping. For comparison we also used a more precise form of continuous clamping that determines the exact intervals of intersection of the oriented bounding boxes (OBBs) of the receivers with

the shadow volume of the OBB of the shadow caster. The fill we measured is the total number of pixels touched when rendering the shadow volumes with the depth test disabled. We also measured the number of stencil writes using standard shadow volumes (SV-SW). For z-pass shadow volumes, this is equivalent to the number of fragments that pass the depth test. For both paths there are portions where z-fail shadow volumes are faster than z-pass shadow volumes, and vice versa. For path 1, the average frame rate with z-pass was slightly faster so we present that data here. Z-fail was used for path 2.

With CC shadow volumes we have observed up to 7 times reductions in fill and up to 4 times speed-up in shadow volume rendering time. For the paths shown in Fig. 7, we see an average of 2–3 times reduction in fill and 2.5 times speed-up in shadow volume rendering time. Figure 8 shows the breakdown of the average timings.

5. Analysis

The cost of shadow volume rendering can be divided into two parts. The *vertex processing cost* includes the time to compute the vertices of the shadow volumes on the CPU and the time to transmit, transform, and setup the geometric primitives on the graphics hardware. This cost is independent of screen resolution. The *rasterization cost* is higher when anti-aliasing is enabled and increases proportionally with screen resolution. Our algorithm is targeted for scenes in which rasterization cost is the major bottleneck. Shadow volume culling reduces both the vertex processing and rasterization cost by eliminating the redundant shadow volumes altogether. Shadow clamping splits the shadow volumes into smaller pieces to significantly reduce rasterization cost in empty space while increasing the vertex processing cost. The

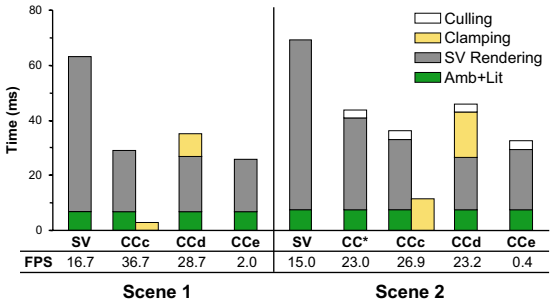

Figure 8: *Timing breakdown for standard shadow volumes (SV) and CC shadow volumes with continuous, discrete, and exact clamping (CCc, CCd, and CCe, respectively). The time for exact clamping (0.5s for scene 1 and 2.5s for scene 2) is not shown. The ambient and lit passes are independent of the method used. Culling was not used in scene 1. The timing of culling alone (CC*) is shown for scene 2. Continuous clamping and other CPU computation is performed in parallel with GPU rendering and does not affect overall frame time.*

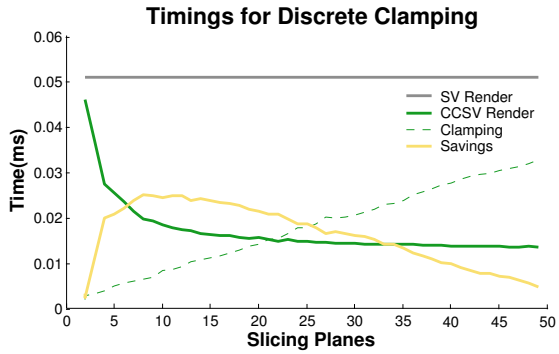

Figure 9: *This graph shows the timings obtained with a varying number of slicing planes using discrete clamping for a typical view. The savings are calculated as the standard shadow volume rendering time minus the combined time for clamping and rendering of CC shadow volumes. Savings peak between 8 and 16 slicing planes.*

graphs in Fig. 7 show the existence of a strong empirical relationship between the area of the shadow volumes and the shadow volume rendering time, though the exact relationship can be hardware dependent.

In continuous clamping, the overlap tests are relatively inexpensive. Although there are potentially n^2 interactions between shadow casters and shadow receivers, by utilizing temporal coherence the expected running time is reduced to $O(n+k)$, where n is the number of objects and k is the actual number of overlapping bounding boxes. In typical scenes, k is usually fairly small. The main cost is incurred in merging depth intervals.

For the discrete clamping, vertex processing and clamping costs increase linearly with the number of slices. Fig. 9 shows the timings for a varying number of slices. The overall savings introduced by adding more slices are large at first, but then begin to diminish. At some point the cost of additional slicing planes dominates the savings. In our experiments maximum savings were obtained by using $8-16$ slices.

The extra rendering used for performing culling and discrete clamping is fairly inexpensive for several reasons.

1. Whenever possible, we use tight bounding volumes instead of the objects themselves to reduce rendering overhead.

2. There are no pixel writes when performing occlusion queries, so this rasterization is less expensive than that of shadow volumes.

3. There is less overdraw. After culling, the depth complexity in the light view is usually much lower than that of the shadow volumes when viewed from the side.

Thus the extra rendering incurred in reducing the size and

complexity of large shadow volumes often leads to significant savings.

On average, discrete clamping resulted in greater fill savings and lower shadow volume rendering time than continuous clamping in our tests, but it had about twice the overhead on the hardware we used. The much more expensive, exact clamping method was only slightly better than the much simpler method we use for continuous clamping. Since continuous clamping uses AABBs in light space, it performs best when the light direction was aligned well with the orientation of the objects in the scene. Discrete clamping is less sensitive to changes in light direction.

Occlusion queries are currently a bottleneck in our system. We have observed a maximum of only 1.2M queries per second at any resolution, though the graphics hardware can render the objects without occlusion queries at a higher rate. We expect the overhead associated with occlusion queries will be reduced considerably in the future. Since the current performance of occlusion queries is slow we are forced to use fewer of them. We have to use a coarser object subdivision which leads to less fill savings.

5.1. Limitations

CC shadow volumes are appropriate only for those situations where rasterization is a bottleneck. In addition, there must be some degree of *unnecessary* shadow volume rendering. Consider the complex self-shadowing of branches in a tree. Shadow volumes in such a scene may be fill-bound, but since there are few shadow casters that lie completely in shadow and few large regions of empty space, CC shadow volumes would provide very little performance gain. Clamping would probably work well, however, to eliminate the empty space in the shadow volumes between the branches and the ground. In general, culling and clamping work best on complex models, which either have a high degree of occlusion and/or large empty spaces.

Occlusion queries are performed in image-space and may lead to artifacts. These artifacts are similar to those of other hybrid algorithms that combine image- and object-space [McC00, GLY*03, SCH03, CD04]. Section 3.2.3 discusses possible artifacts in discrete clamping. Artifacts may also arise in shadow culling. The pixel sampling on the image plane may miss small holes or small occluders, leading to incorrect shadows. Triangles that are nearly parallel to the view can also be problematic because they are covered by few samples. These sampling problems are reduced by using slightly enlarged bounding boxes around the objects and by ensuring that the light's view is fit tightly to the visible objects in the scene in order to maximize the sampling resolution.

Another drawback of CC shadow volumes is that it requires more CPU time to construct them. Methods that offload shadow volume construction to the GPU [BS03] are difficult to adapt for use with CC shadow volumes since the connectivity can change from frame to frame. Though shadow volume construction on the CPU does not adversely affect frame-rate in our current implementation, less time remains for other computations.

5.2. Comparison with other Methods

BSP-tree based shadow volumes [CF89, CS95, BJ99] can render shadows efficiently in static scenes or scenes with few moving objects. However, when the light source moves or many objects move, large portions of the BSP-tree need to be re-built, which is usually an expensive process with large models. Our algorithms can easily handle dynamic scenes with a moving light source. The only limitation is that continuous clamping algorithm will require more computation if the motion is not coherent. The discrete clamping algorithm is completely insensitive to motion.

McGuire et al. [MHE*03] recently demonstrated an effective algorithm for accelerating shadow volumes. Their method uses a combination of the OpenGL scissor test and depth bounds tests to reduce shadow volume fill. The scissor region is set to the intersection of the viewport with the screen projection of the light's region of influence. In the types of scenes for which our algorithm was designed, the scissor region would usually contain the entire viewport because the whole scene lies within the light's region of influence. Thus the only fill savings come from the depth bounds test. Fig. 7 shows the time to render the shadow volumes with a depth bounds range set to (0,0). This range effectively early-rejects every fragment from the pipeline, establishing an upper-bound on the performance of their algorithm. The timings indicate that with the depth bounds test the rasterization cost is still significant. In fact, even if no pixels were to pass the depth bounds test, these results indicate that on the same hardware, CC shadow volumes would be faster for these scenes. For scenes with little occlusion or without large, mostly empty shadow volumes, our algorithm would not perform as well because the overhead would dominate the potential savings.

6. Conclusion and Future Work

In this paper, we have presented algorithms for shadow culling and shadow clamping to accelerate the performance of shadow volumes. These include visibility computations for shadow culling and continuous and discrete shadow clamping. These algorithm can be efficiently implemented by utilizing temporal coherence between successive frames. We have demonstrated the performance of these algorithms on two relatively complex environments, reducing fill requirements by up to 7 times.

There are several avenues for future work. Our main focus in this paper has been to reduce rasterization costs. We would like to explore methods to reduce the vertex processing cost of shadow volumes which remains fairly high. Adaptive approaches using slice coverage information from one frame might be used to compute a better set of slicing planes for the next, i.e. use temporal coherence for discrete shadow clamping. We could also improve the continuous clamping by computing better depth bounds with more sophisticated

overlap tests. Finally we would like to extend these ideas to improve the performance of soft shadow generation algorithms, such as the work of [AAM03] which utilizes a variation on shadow volumes to identify penumbra regions.

Acknowledgments

We would like to thank Ben Cloward for modelling the test environments. We would also like to thank Mark Kilgard at NVIDIA and the members of the UNC Walkthrough group for their many useful discussions. This research was supported in part by an NSF Graduate Fellowship, ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards ACI 9876914 and ACR-0118743, ONR Contracts N00014-01-1-0067 and N00014-01-1-0496, DARPA Contract N61339-04-C-0043 and Intel.

References

[AAM03] ASSARSSON U., AKENINE-MÖLLER T.: A geometry-based soft shadow algorithm using graphics hardware. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2003) 22*, 3 (2003), 511–520.

[AAM04] AILA T., AKENINE-MÖLLER T.: A hierarchical shadow volume algorithm. To appear in *Proc. of Graphics Hardware 2004* (2004).

[Ber85] BERGERON P.: Shadow volumes for non-planar polygons. In *Graphics Interface '85 Proceedings* (1985), pp. 417–418.

[BJ99] BATAGELO H. C., JUNIOR I. C.: Real-time shadow generation using bsp trees and stencil buffers. *Proc. SIBGRAPI 12* (1999), 93–102.

[BS03] BRABEC S., SEIDEL H.: Shadow volumes on programmable graphics hardware. *Proc. of Eurographics* vol. 22, (2003), 433–440.

[Car00] CARMACK J.: Email to private list, May 23, 2000. http://developer.nvidia.com/.

[CD04] CHAN E., DURAND F.: An efficient hybrid shadow rendering algorithm. In *Proc. of the Eurographics Symposium on Rendering* (2004).

[CF89] CHIN N., FEINER S.: Near real-time shadow generation using BSP trees. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, vol. 23, (1989), pp. 99–106.

[CLMP95] COHEN J., LIN M., MANOCHA D., PONAMGI M.: I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *Proc. of ACM Interactive 3D Graphics Conference* (1995), pp. 189–196.

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. *ACM Computer Graphics 11*, 3 (1977), 242–248.

[CS95] CHRYSANTHOU Y., SLATER M.: Shadow volume BSP trees for computation of shadows in dynamic scenes. In *1995 Symposium on Interactive 3D Graphics* (1995), pp. 45–50.

[Die96] DIEFENBACH P.: *Multi-pass pipeline rendering: Interaction and realism through hardware provisions*. PhD thesis, University of Pennsylvania, 1996.

[EK02] EVERITT C., KILGARD M.: Practical and robust stenciled shadow volumes for hardware-accelerated rendering. In *SIGGRAPH 2002 Course Notes* (2002), vol. 31.

[FGH*85] FUCHS H., GOLDFEATHER J., HULTQUIST J. P., SPACH S., AUSTIN J. D., BROOKS, JR. F. P., EYLES J. G., POULTON J.: Fast spheres, shadows, textures, transparencies, and image enhancements in Pixel-Planes. In *Computer Graphics (SIGGRAPH '85 Proceedings)* (1985), vol. 19, pp. 111–120.

[GLM04] GOVINDARAJU N., LIN M., MANOCHA D.: *Fast and reliable collision detection using graphics hardware*. Tech. rep., University of North Carolina, Department of Computer Science, 2004.

[GLY*03] GOVINDARAJU N., LLOYD B., YOON S., SUD A., MANOCHA D.: Interactive shadow generation in complex environments. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2003) 22*, 3 (2003), 501–510.

[Hei91] HEIDMANN T.: Real shadows real time. *IRIS Universal*, 18 (1991).

[Len02] LENGYEL E.: The mechanics of robust stencil shadows. *Gamasutra* (October 11 2002). http://www.gamasutra.com/.

[McC00] MCCOOL M.: Shadow volume reconstruction from depth maps. *ACM Trans. on Graphics 19*, 1 (2000), 1–26.

[MHE*03] MCGUIRE M., HUGHES J., EGAN K., KILGARD M., EVERITT C.: *Fast, Practical and Robust Shadows*. Technical report, NVIDIA, 2003. http://developer.nvidia.com/.

[SCH03] SEN P., CAMMARANO M., HANRAHAN P.: Shadow silhouette maps. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2003) 22*, 3 (2003), 521–526.