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

An Efficient Hybrid Shadow Rendering Algorithm

Eric Chan Frédo Durand

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology†

Abstract

We present a hybrid algorithm for rendering hard shadows accurately and efficiently. Our method combines the strengths of shadow maps and shadow volumes. We first use a shadow map to identify the pixels in the image that lie near shadow discontinuities. Then, we perform the shadow-volume computation only at these pixels to ensure accurate shadow edges. This approach simultaneously avoids the edge aliasing artifacts of standard shadow maps and avoids the high fillrate consumption of standard shadow volumes. The algorithm relies on a hardware mechanism for rapidly rejecting non-silhouette pixels during rasterization. Since current graphics hardware does not directly provide this mechanism, we simulate it using available features related to occlusion culling and show that dedicated hardware support requires minimal changes to existing technology.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Graphics processors, I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture

Keywords: shadow algorithms, graphics hardware

1. Introduction

Shadow maps and shadow volumes are two popular means for the real-time rendering of shadows. Shadow maps are efficient and flexible, but they are prone to aliasing. Shadow volumes are accurate, but they have large fillrate requirements and thus do not scale well to complex scenes. Achieving both accuracy and scalability is challenging for real-time shadow algorithms.

Sen et al. [SCH03] observed that shadow-map aliasing is only noticeable at the discontinuities between shadowed and lit regions, i.e. at the shadow silhouettes. On the other hand, shadow volumes compute shadows accurately at every pixel, but this accuracy is needed only at the silhouettes. This observation suggests a hybrid algorithm that uses a slower but accurate algorithm near the shadow discontinuities and a faster, less exact algorithm everywhere else.

In this paper, we describe a hybrid shadow rendering algorithm (see Figure 1). We first use a shadow map to find quickly pixels in the image that lie near shadow silhouettes, then apply the shadow volume algorithm only at these pixels; the shadow map determines shadows for the remaining non-silhouette pixels. This approach greatly reduces the fillrate needed for drawing shadow volumes, because the number of silhouette pixels is often a small fraction of the shadow polygons' total screen area (see Figure 2). We show that our method produces accurate hard shadows and has substantially lower fillrate requirements than the original shadow volume algorithm.

To avoid processing non-silhouette pixels during shadow volume rasterization, we propose an extension to graphics hardware called a *computation mask*. Computation masks are useful in general for accelerating multipass rendering algorithms. Although they are not directly exposed in current hardware, we show how to simulate them efficiently using available features related to early z occlusion culling. Since computation masks exploit existing culling hardware, adding native hardware support requires minimal changes to modern graphics chips.

2. Related Work

Researchers have developed many shadow algorithms over the years, several of them based on the classic shadow map and shadow volume methods. Recent work has focused on extending these methods to handle soft shadows, which we do not discuss in this paper. Many efforts have also been made to address the shadow-map aliasing and the shadow-volume fillrate issues. We focus our discussion below on

† email: {ericchan|fredo}@graphics.csail.mit.edu

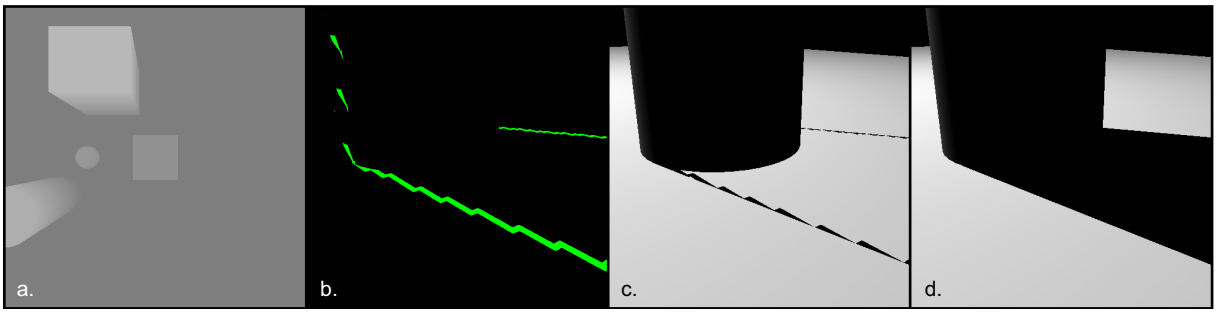

Figure 1: *Overview. We first use a shadow map (a) to identify pixels in the image that lie close to shadow silhouettes. These pixels, seen from the observer's view, are shaded green in (b). Next, we render shadow volumes only at these pixels to obtain accurate shadow edges (c). We use the shadow map to compute shadows everywhere else, and the final result appears in (d).*

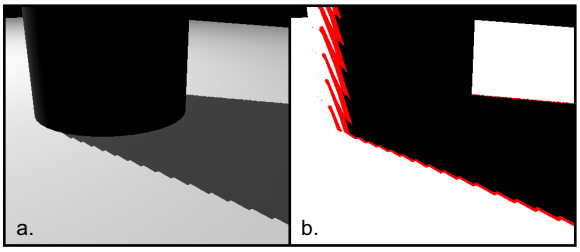

Figure 2: *(a) The cylinder's shadow exhibits aliasing due to shadow-map undersampling. However, aliasing is only apparent at the shadow edges. (b) Pixels that lie near shadow edges (shown in red) account for only a small fraction of the total image size.*

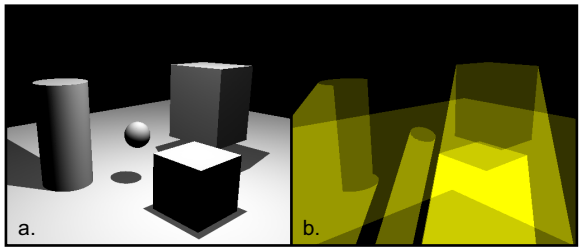

Figure 3: *A simple scene with a few objects (a) can lead to high fillrate consumption when using shadow volumes. (b) The shadow polygons (shown in yellow) occupy substantial screen area and overlap in screen space. Lighter shades of yellow correspond to regions with higher overdraw.*

this recent body of work and refer the reader to Woo et al.'s paper [WPF90] for a broader survey of shadow algorithms. Akenine-Möller and Haines's book [AMH02] also provides a good discussion of real-time shadow algorithms.

Shadow maps were introduced by Williams in 1978 [Wil78]. The algorithm works in image space: it first renders a depth map of the scene from the light's viewpoint; the depth map is then used to determine which samples in the final image are visible to the light. Shadow maps are efficient and are supported in graphics hardware, but they are prone to sampling artifacts such as aliasing.

Researchers have developed many techniques for addressing shadow-map aliasing. Approaches based on filtering and stochastic sampling [LV00, RSC87] produce nice antialiased shadows. Unfortunately, effective filtering requires a large number of samples per pixel, which is expensive for real-time applications. Furthermore, using a large filter width leads to self-shadowing artifacts that are scene-dependent and hard to avoid.

Other methods reduce aliasing by increasing the effective shadow-map resolution. Adaptive shadow maps [FFBG01] detect and redraw undersampled regions of the shadow map at higher resolutions. Unfortunately, the required data structures and host-based calculations preclude real-time performance for dynamic scenes. Perspective shadow maps [SD02] are simpler: they just require an additional perspective transformation that effectively provides more resolution in the shadow map to samples located close to the viewer. This method is simple and fast, but it does not reduce aliasing in all cases. For instance, when the light source and the viewer face each other, the perspective transforms mutually cancel and the result is a standard uniform shadow map.

Sen et al. [SCH03] observed that shadow-map aliasing is only problematic near the shadow silhouettes, i.e. discontinuities between shadowed and lit regions. They propose the silhouette map, a 2D data structure that provides a piecewise-linear approximation to the true geometric silhouette seen from the point of view of the light source. The silhouette map provides an excellent reconstruction of the shadow silhouette and eliminates shadow-map aliasing in many cases. Since only one silhouette point may be stored per texel in the silhouette map, however, artifacts may appear when multiple shadow boundaries meet. Our edge reconstruction uses shadow volumes and avoids these artifacts.

Unlike the shadow map, which works in image space, Crow's shadow volume algorithm [Cro77] works in object space by drawing polygons to represent the boundary between illuminated and shadowed regions of 3D space. The shadow volume itself is the volume bounded by these polygons, and a shadow query involves checking if a point in the

image lies within the volume. Bergeron [Ber86] generalized the method to handle open models and non-planar surfaces.

Heidmann [Hei91] showed how to accelerate shadow-volume calculations using a hardware stencil buffer. Unfortunately, the original algorithm for stencil-based shadow volumes suffered from numerous robustness issues, some of which were addressed by Diefenbach [Die96] and Carmack [Car00]. Recently, Everitt and Kilgard [EK02] described a robust implementation using modern graphics hardware.

Unfortunately, the shadow volume algorithm does not scale well to scenes with high shadow complexity. The method involves drawing extra geometry, but the main problem is the large *fillrate* required. There are two reasons for this. First, shadow-volume polygons occupy substantial screen area, as shown in Figure 3, especially in heavily-shadowed scenes or when the observer is in shadow. Second, the shadow polygons overlap in screen space, and every rasterized pixel of every shadow polygon potentially updates the stencil buffer. This degree of overlap, combined with the large screen coverage of shadow polygons, grows rapidly as shadow complexity increases, leading to an explosion in fillrate consumption.

Lengyel [Len02], Everitt and Kilgard [EK03], and McGuire et al. [MHE*03] describe a number of culling techniques for optimizing stencil shadow volumes to reduce fillrate. One method estimates the shadow extent on a per-blocker basis and uses scissoring to discard shadow-polygon pixels that lie outside the computed scissor rectangle. A related method also considers the depth range of the shadows cast by individual blockers and discards pixels from shadow-volume polygons that lie outside of this range. The key idea here is to discard pixels early in the pipeline without performing stencil updates, thereby accelerating rasterization and saving valuable memory bandwidth.

The above techniques are useful, but they are less effective for heavily-shadowed scenes. This is because pixels that lie in shadow are precisely those that lie within the depth ranges and scissor rectangles computed in the above optimizations; thus such pixels do not benefit from these optimizations. Scenes in which shadow polygons have large depth ranges are also problematic. In contrast, our method performs rasterization and stencil updates only for pixels that lie near shadow silhouettes. Thus even for scenes with many shadows, the fillrate consumed by shadow-volume polygons remains small. Note that our method is complementary to the aforementioned optimizations.

Researchers have recently proposed several new methods for tackling the fillrate problem of shadow volumes. Aila and Akenine-Möller [AAMar] describe a two-level hierarchical shadow volume algorithm. Their approach is similar to ours in that they identify shadow-boundary pixels and rasterize shadow volumes accurately only at those pixels. There are two important differences, however. First, their method detects tiles of boundary pixels in *object space* by checking for triangle intersections against the tiles' 3D bounding boxes, whereas our method identifies the boundaries in *image space* using a shadow map. Second, an efficient implementation of their method requires numerous changes to hardware, including a modified rasterizer, logic for managing tile updates, and the addition of a delay stream [AMN03]. In contrast, our method relies on existing culling hardware to reduce shadow-volume fillrate.

Lloyd et al. [LWGMar] take a different approach to reducing shadow-volume rasterization costs. One of their techniques is to use image-space occlusion queries to identify blockers that lie entirely in shadow; shadow-volume polygons for such blockers are redundant and may be culled. A similar method is used to cull blockers that cast shadows only on receivers lying outside the observer's view frustum.

Furthermore, Lloyd et al. limit the screen-space extent of shadow-volume polygons in two ways. In the first method, they compute overlaps of the blockers' axis-aligned bounding boxes to estimate the depth intervals from the light source that contain shadow receivers; they clamp shadow polygons to non-empty depth intervals. In the second method, they partition the observer's view frustum into discrete slices and use occlusion queries to determine which slices contain receivers; as in the first approach, shadow polygons are clamped to non-empty slices. The second approach provides greater culling but incurs additional overhead. The key difference between all of the above culling strategies and our method is that Lloyd et al. reduce fillrate by reducing the number and size of the shadow-volume polygons, whereas we draw the entire polygons and rely on the hardware to perform culling of non-silhouette pixels. These approaches are fully complementary.

McCool [McC00] was the first to propose a hybrid algorithm that combines shadow maps and shadow volumes. His method first renders a depth map and runs an edge-detection algorithm to find the blockers' silhouette edges. Next, the method reconstructs shadow volumes from these edges and uses them to compute shadows in the final image. A strength of McCool's approach is that shadow-volume polygons are generated only for silhouette edges that are visible to the light source. Unfortunately, an expensive depth-buffer readback is required, shadow polygons are fully rasterized, and artifacts can occur due to aliasing in the shadow-volume reconstruction.

Govindaraju et al. [GLY*03] propose a different hybrid shadow algorithm. First, they use level-of-detail and PVS culling techniques to reduce the number of potential blockers and receivers; these techniques are implemented using hardware occlusion queries. Next, they compute shadows using a mix of object-space clipping and shadow maps. Exact clipping of receiver polygons against the blockers' shadow frusta is performed for receivers that would otherwise exhibit shadow-map aliasing artifacts. To identify these receivers, they use a formula derived by Stamminger and Drettakis [SD02] that relates the size of a pixel in shadow-map space to its size when projected into the observer's image space. Shadow maps are used for the remaining receiver polygons. This hybrid approach improves the accuracy of

shadow silhouettes without requiring an excessive number of clipping operations.

The approach of Govindaraju et al. is similar to ours in that both methods limit the amount of computation required to render accurate shadow silhouettes. The two methods perform culling at different stages, however. Whereas their method minimizes the number of *objects* that are processed by their clipping algorithm, our method minimizes the number of *pixels* in the image that are treated by shadow volumes. The two methods could be combined by replacing their software-based polygon clipping with our optimized, hardware-accelerated shadow volume rasterization.

3. Algorithm

We assume that blockers are polygonal, well-behaved, closed, and manifold; these properties ensure a robust implementation of shadow volumes [EK02].

An overview of our approach is shown in Figure 1. We first create an ordinary shadow map, which serves to identify shadow silhouette pixels and compute shadows for non-silhouette pixels. Then, we use shadow volumes to compute accurate shadows only at silhouette pixels. The underlying assumptions are that the silhouette pixels account for a small fraction of the total number of shadow-polygon pixels, and that the hardware supports a mechanism for efficiently discarding pixels that do not lie on the silhouette.

We now explain these concepts in more detail; implementation and hardware issues are discussed in the next section. The algorithm's steps are:

1. Create a shadow map. We place the camera at the light source and render the nearest depth values to a buffer, as shown in Figure 1a. Since we only need the shadow map to approximate the shadow silhouette, we can use a low-resolution shadow map to conserve texture memory and speed up shadow-map rendering. The tradeoff is that low-resolution shadow maps can miss small features and usually increase the number of pixels classified as silhouette pixels. We will discuss this issue further in Section 5.1.

2. Identify shadow silhouette pixels in the final image. We render the scene from the observer's viewpoint and use a technique suggested by Sen et al. [SCH03] to find silhouette pixels. We transform each sample to light space and compare its depth against the four nearest depth samples from the shadow map. If the comparison results disagree, then we classify the sample as a silhouette pixel (shown in green in Figure 1b). Otherwise, the sample is a non-silhouette pixel and is shaded according to the depth comparison result.

Reducing the number of silhouette pixels is desirable because it limits the amount of stencil fillrate consumed when drawing shadow volumes. For example, pixels that are back-facing with respect to the light are always in shadow, so we never tag them as silhouette pixels. Additional methods for reducing the number of silhouette pixels are discussed in Section 6.3.

During this step, we also perform standard z-buffering,

```
// Find discontinuities: shadow silhouette pixels.
void main (out half4 color    : COLOR,
           half diffuse       : COL0,
           float4 uvProj      : TEXCOORD0,
           uniform sampler2D shadowMap)
{
    // Use hardware's 2x2 filter: 0 <= v <= 1.
    fixed v = tex2Dproj(shadowMap, uvProj).x;

    // Requirements for sil pixel: front-facing and
    // depth comparison results disagree.
    color = (v > 0 && v < 1 && diffuse > 0) ? 1 : 0;
}
```

Figure 4: *Cg pixel shader for finding shadow silhouettes.*

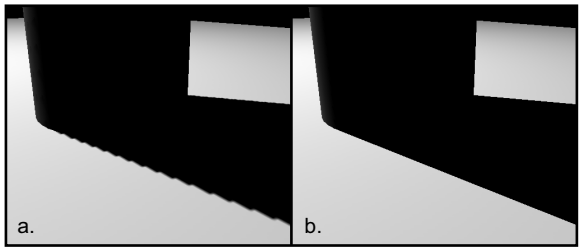

Figure 5: *Comparison: a cylinder's shadow is rendered using (a) a 512×512 shadow map and (b) our hybrid algorithm with a 256×256 shadow map. Using a lower-resolution shadow map in our case is acceptable because shadow volumes are responsible for reconstructing the shadow silhouette.*

which leaves the nearest depth values seen from the observer's viewpoint in the depth buffer. This prepares the depth buffer for drawing shadow volumes in the next step.

3. Draw shadow volumes. The stencil shadow volume algorithm works by incrementing or decrementing the stencil buffer based on whether pixels of shadow-volume polygons pass or fail the depth test. We follow the z-fail setup described by Everitt and Kilgard [EK02] because of its robustness. The key difference in our approach is that we rasterize shadow-polygon pixels and update the stencil buffer only at framebuffer addresses containing silhouette pixels.

At the end of this step, the stencil buffer contains non-zero for pixels that lie in shadow; it contains zero for pixels that either are not shadowed or are not silhouette pixels. For example, the black shadow edges in Figure 1c show the regions where the stencil-buffer contains non-zero.

4. Compute shadows. We draw and shade the scene only at pixels with stencil values equal to zero, thereby avoiding the shadowed regions of the image.

4. Implementation Issues

Our shadow algorithm performs rasterization and stencil updates of the shadow-volume polygons only at the silhouette pixels. To find these silhouette pixels, we compare the depth of each image sample with the four nearest depth samples in the shadow map and check if the results agree. We use

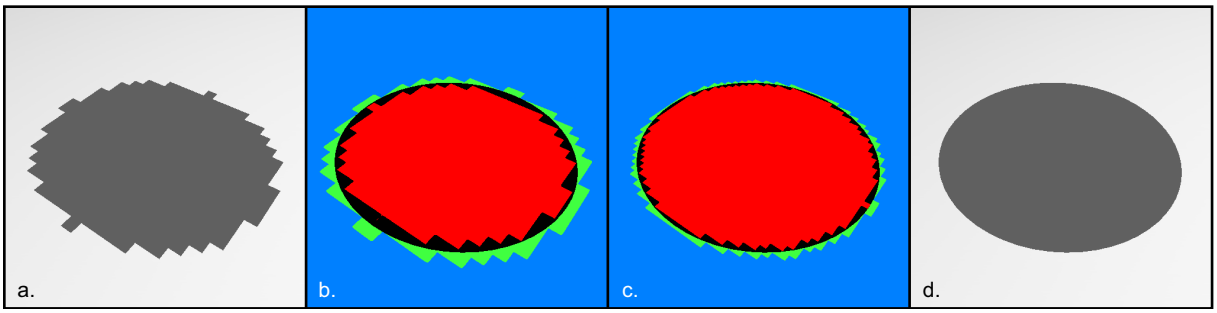

Figure 6: *Visualization of mixing shadow maps and shadow volumes. We see the shadow of a ball cast onto the ground plane. (a) Aliasing is evident when the ball's shadow is rendered using a 256×256 shadow map. The rest of the images illustrate how our method minimizes aliasing. In (b) and (c), non-silhouette pixels are shaded red and blue; for these pixels, the shadow map determines which ones are in shadow (red) and which ones are lit (blue). Silhouette pixels are shaded black and green; shadow volumes determine which ones are in shadow (black) and which ones are lit (green). Shadow-map resolutions of 256×256 and 512×512 were used for (b) and (c), respectively. The final shadow is shown in (d).*

hardware-assisted percentage closer filtering [RSC87] to accelerate this step. If the shadow query returns exactly 0 or 1, the depth comparison results agree and the pixel is not a silhouette pixel; otherwise the results disagree and the pixel lies on a silhouette. This optimization allows us to issue a single texture-fetch instruction in a pixel shader, shown in Figure 4. We tag silhouette pixels by writing 1 to the color buffer.

Rasterization and stencil updates of shadow-volume polygons are limited to silhouette pixels. We accomplish this task using a *computation mask*, a device that lets us pick specific framebuffer addresses to mask off so that the hardware can avoid processing pixels at those locations. Computation masks are useful for accelerating multipass rendering algorithms. For instance, Purcell et al. [PDC*03, Pur04] found that a computation mask with coarse granularity improved the performance of their hardware-based photon-mapping algorithm by factors of two to ten.

Current graphics hardware does not directly expose computation masks, but it turns out that the EXT_depth_bounds_test OpenGL extension [Ope02] can be treated as one; see the appendix for a brief explanation of this extension. The idea is to use a pixel shader to mask off pixels by setting their depth values to a constant *outside* the depth bounds. Then we enable depth-bounds testing so that in subsequent rendering passes, rasterized pixels at these masked-off framebuffer addresses can be discarded early in the pipeline. Similar culling mechanisms are commonly used for early z occlusion culling [Mor00].

In our implementation, we set up a computation mask as follows. We draw a screen-aligned quad and use a pixel shader to set the depth values of all non-silhouette pixels to $z = 0$; depth values of silhouette pixels are unmodified. Next, we enable depth-bounds testing and set the depth bounds to $[\epsilon, 1]$ for some small constant $\epsilon \approx 0.001$. Finally, we apply the robust z-fail variation of stencil shadow volumes [EK02]. Since the hardware discards all rasterized pixels whose depth values in the framebuffer are equal to $z = 0$, only silhouette pixels will be rendered.

Note that our implementation depends on the hardware's ability to preserve early culling behavior when using a pixel shader to compute depth values. This feature is available on the NVIDIA GeForce 6 (NV40) but not on earlier NVIDIA architectures such as the GeForce FX (NV30). ATI's Radeon 9700 (R300) and newer architectures also support this feature, but unfortunately those chips do not support the EXT_depth_bounds_test extension.

5. Results

All of the images presented in this section and in the accompanying video were generated at a resolution of 1024 × 768 on a 2.6 GHz Pentium 4 system with a NVIDIA GeForce 6800 (NV40) graphics card.

Examples. Figure 5 shows a cylinder casting a shadow onto the ground plane. We used an ordinary 512×512 shadow map and 2×2 bilinear percentage closer filtering [RSC87] for the image in Figure 5a. We used our hybrid method with a 256×256 shadow map for the image in Figure 5b. The lower-resolution shadow map is acceptable because the shadow-volume portion of our algorithm reconstructs the shadow edges accurately.

Figure 6 shows a closeup of the ball's shadow from Figure 3a and illustrates how our method operates near shadow silhouettes. Figure 6a shows the aliasing artifacts that result from using an ordinary 256×256 shadow map. In the middle two images, non-silhouette pixels are shown in red and blue; red pixels are fully shadowed, and dark gray pixels are fully lit. Visibility for these pixels is determined using the shadow map. Silhouette pixels are shown in black and green; black pixels are in shadow and green pixels are lit. Shadow determination in this case is performed by shadow volumes. Figures 6b and 6c use 256×256 and 512×512 shadow maps, respectively. Figure 6d shows the final shadow computed by our method.

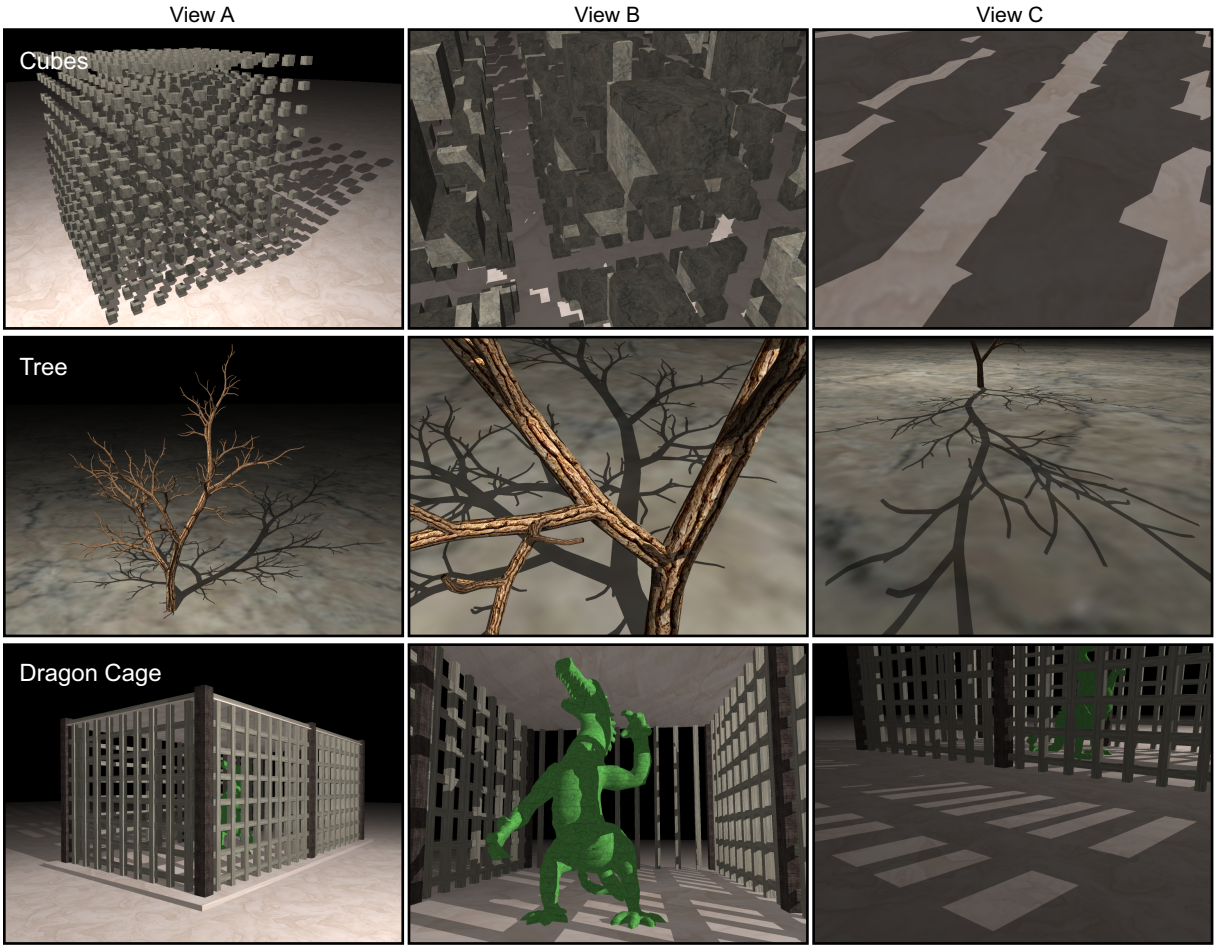

Figure 7: *Three test scenes with high shadow complexity. Rows contain different scenes, and columns show different views. Each scene is illuminated using a single point light source.*

Methodology. We evaluated our method using the scenes shown in Figure 7. The Cubes and Dragon Cage scenes contain 12,000 triangles each, and the Tree scene has 40,000 triangles. We chose these scenes and viewpoints for several reasons. First, they have high shadow complexity and require enormous fillrate when using ordinary shadow volumes. Second, these scenes have many overlapping shadow edges, which can lead to temporal artifacts when using shadow silhouette maps [SCH03]. Third, we have chosen some camera views (see View C) that are on the opposite side of the blockers as the light source; these cases are difficult to handle using perspective shadow maps [SD02]. Finally, these scenes are heavily-shadowed, and the depth range of shadow-volume polygons is large, making it difficult to apply the scissor and depth-bounds optimizations described in Section 2 [Len02, EK03, MHE*03]. In summary, real-time shadow rendering is a challenging task in all of these scenes.

5.1. Image Quality

Figure 8 compares the image quality of shadow maps, our hybrid method, and shadow volumes; we used a 1024×1024 shadow map for the first two techniques. These images show that our method minimizes the edge-aliasing artifacts of shadow maps.

A more subtle improvement is that our method reduces self-shadowing artifacts. With regular shadow maps, incorrect self-shadowing may occur due to limited depth-buffer resolution and precision. These artifacts are visible, for example, in the Dragon Cage scene in Figure 8 (see the lower-left image). The problem is usually addressed by adding a small bias to the depth values when rendering the shadow map [AMH02, RSC87, Wil78]. Unfortunately, the amount of bias required depends on the scene configuration and is hard to set automatically for dynamic scenes.

In our approach, however, incorrectly-shadowed pixels are often classified as silhouette pixels and thus are rendered correctly by the shadow-volume portion of the algorithm.

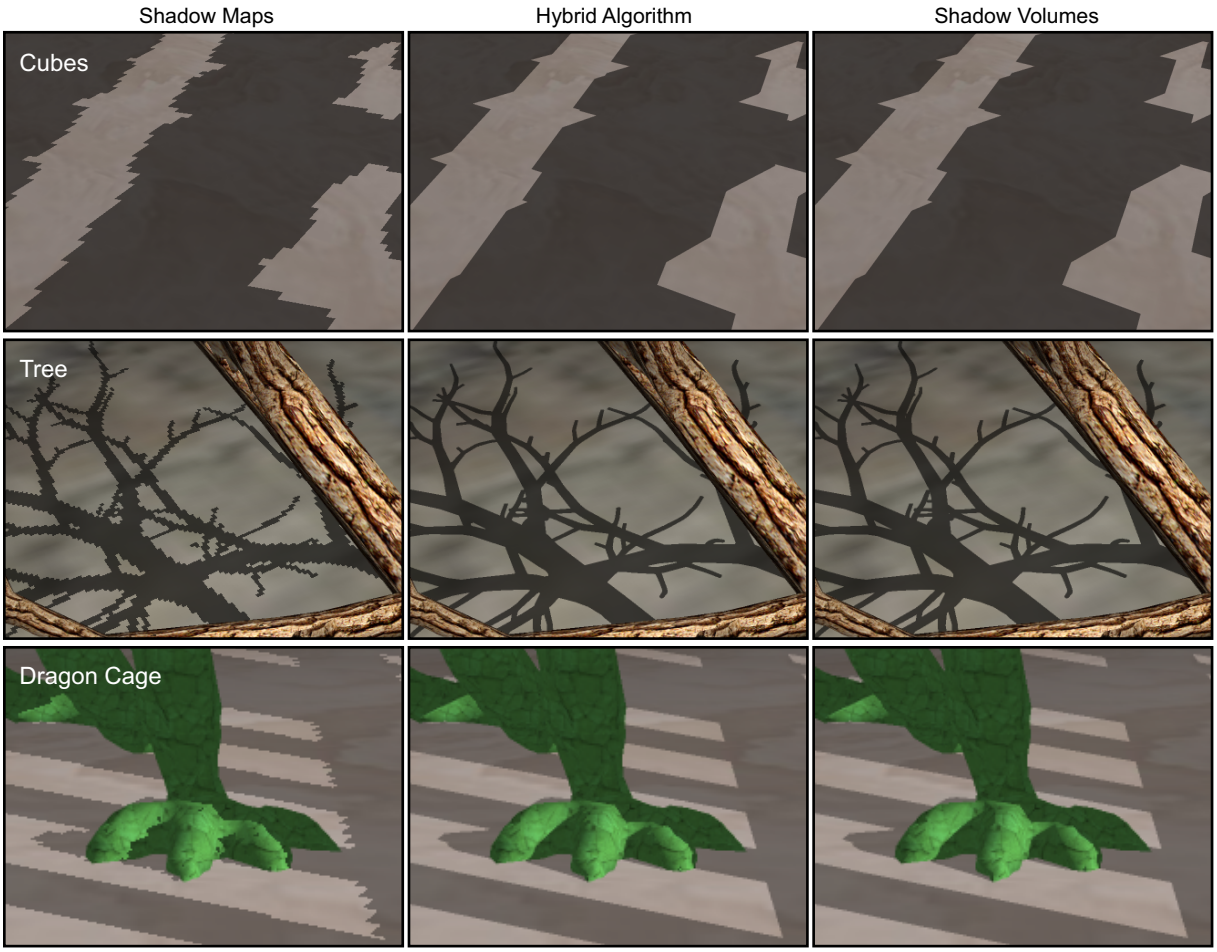

Figure 8: *Comparison of image quality using shadow maps (left column), our hybrid algorithm (center column), and shadow volumes (right column). A shadow-map resolution of 1024×1024 was used for the shadow map and hybrid algorithms.*

The reason is that the depth value of an affected pixel usually lies between the depth values of two adjacent samples in the shadow map. As a result, the depth comparisons disagree and the pixel is tagged as a silhouette pixel. One implication is that we can choose the shadow bias conservatively, erring on the side of applying too little bias and relying on the shadow volumes to avoid self-shadowing artifacts. If we apply too little bias, however, then most of the pixels in the image will be classified as silhouette pixels.

Although shadows computed using our approach are often similar to those computed using shadow volumes, small differences may occur due to sampling errors in the shadow map. To understand these differences better, we studied several images produced using the hybrid algorithm at different shadow-map resolutions. Figure 9 shows an example of one such set of images; we chose the Tree scene as our example because its thin branches are difficult to represent accurately in a discrete buffer. The indicated regions in red and green show missing or incorrect shadows due to undersampling.

More generally, a limitation of using lower shadow-map resolutions is that small blockers may be poorly represented in the shadow map. This form of aliasing manifests itself in vanishing and popping shadows. Existing shadow algorithms that rely on discrete buffers for visibility queries, such as the work of McCool [McC00], Govindaraju et al. [GLY*03], and Sen et al. [SCH03], also exhibit similar artifacts. The difficulty is that arbitrarily small objects may cast arbitrarily large shadows, depending on the scene configuration, and low-resolution shadow maps are more likely to miss such objects. This problem could be addressed by combining our method with a perspective shadow map [Koz04, SD02], which optimizes the depth buffer's sample distribution to maximize resolution near the viewer.

5.2. Performance

Fillrate consumption is significant for the shadow volume algorithm in all of our test scenes. Figure 10a shows an example in which shadow-volume polygons cover the entire

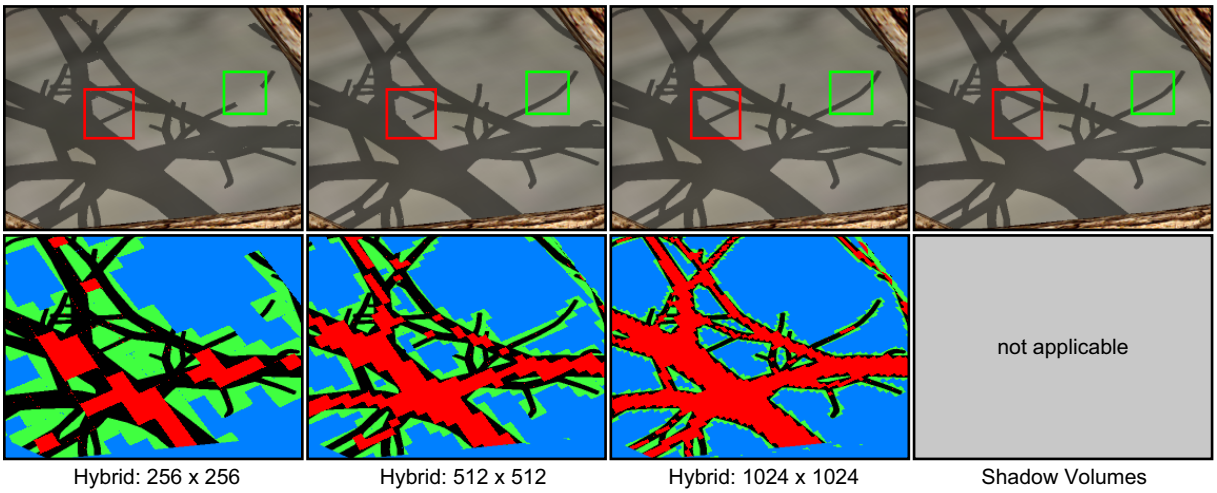

Figure 9: *Artifacts. These images are crops from the Tree scene, View B. The images in the left three columns were generated using our hybrid algorithm with varying shadow-map resolutions. In the top row, the regions indicated in red and green show missing or incorrect shadows due to undersampling in the shadow map. The corresponding images in the bottom row visualize the reconstruction errors using the same color scheme as in Figures 6b and 6c. The reference image in the far-right column was obtained using shadow volumes.*

image and have an overdraw factor of 79, meaning that every pixel is processed 79 times on average. We reduce this huge fillrate consumption by limiting shadow-polygon rasterization to silhouette pixels, shaded green in Figure 10b; silhouette pixels in this scene cover just 5% of the image. Performing rasterization and stencil updates only at these pixels leads to the stencil buffer shown in Figure 10c.

Figure 11 compares the number of pixels rasterized by shadow volumes and our hybrid method. It also shows the percentage of pixels classified as silhouette pixels as a function of shadow-map resolution. Even with a 256×256 shadow map, the fraction of silhouette pixels is much smaller than the fraction of all shadow-volume pixels.

Figure 12 compares the performance of the shadow map, shadow volume, and hybrid algorithms. Performance is measured using the time required for each algorithm to render one frame; all times are reported in milliseconds. The plot gives a performance breakdown for each part of each algorithm. Not surprisingly, the cost of the hybrid and shadow volume algorithms is dominated by the rasterization and stencil updates of the shadow-volume polygons. Our method is significantly faster, however: we observed speedups of 30% to over 100%, measured in frames per second. Keep in mind that these performance numbers are highly scene-dependent and view-dependent; hardware culling implementations (see Section 6.2) also play a large role in determining the actual speedup. We provide performance numbers simply to demonstrate that substantial acceleration is attainable across a number of different scenes and viewpoints.

6. Discussion

6.1. Algorithm Tradeoffs

The key performance tradeoff in our work is the reduction of fillrate at the cost of an extra rendering pass. We emphasize that our method is designed to handle dynamic scenes with high shadow complexity, which would ordinarily give rise to many overlapping shadow volumes and quickly saturate the hardware's fillrate. Thus our method is most relevant to fillrate-limited applications, such as many of the current real-time game engines. Ordinary shadow volumes will clearly run faster for scenes of sufficiently low shadow complexity.

Since our method combines both shadow maps and shadow volumes, it inherits some of the limitations from both. In contrast to the shadow map algorithm, which handles any geometry that can be represented in a depth buffer, our method requires watertight polygonal models for robust shadow volume rendering. In contrast to the shadow volume algorithm, our method is restricted to directional light sources because we use shadow maps to find silhouette pixels; omnidirectional lights require additional rendering passes. Finally, our method requires one more rendering pass than ordinary shadow volumes because we must first create the shadow map. Fortunately, this extra pass is inexpensive because it can be done at lower resolutions and requires no shading.

6.2. Computation Masks

We described at length in Section 4 how to treat the depth bounds test as a computation mask, but this trick is only necessary because current hardware lacks a dedicated computation mask. We believe that adding a true computation mask to graphics hardware is worthwhile for several reasons. First,

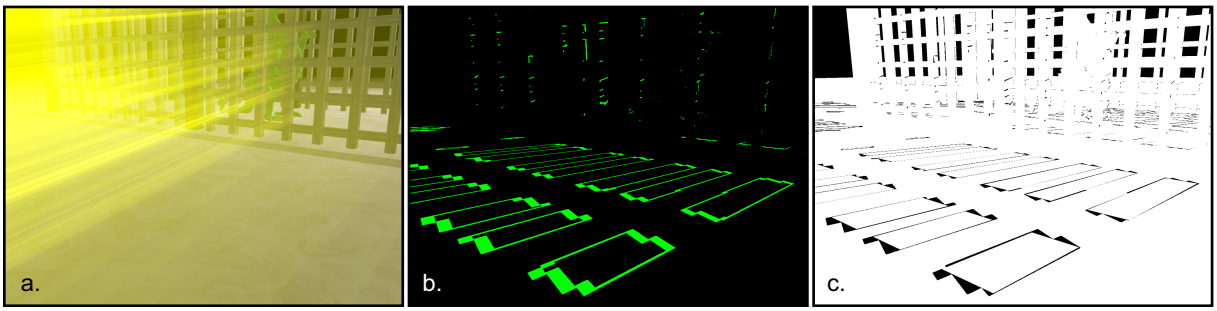

Figure 10: *Fillrate consumption and overdraw in the Dragon Cage scene. The shadow volume polygons, shaded yellow in (a), cover the entire image and have an overdraw factor of 79; brighter yellow corresponds to higher overdraw. Our hybrid method restricts drawing shadow polygons to the silhouette pixels, shaded green in (b); these pixels cover just 5% of the image. The image on the right (c) illustrates the resulting stencil buffer: black pixels on the floor and walls are in shadow and represent non-zero stencil values.*

as we pointed out earlier, computation masks are closely related to early z occlusion culling, and thus most of the required technology is already present in current hardware. In particular, computation masks can take advantage of the early tile-based rejection mechanisms already used for occlusion culling and depth bounds testing. Furthermore, the representation for a computation mask is much more compact: only a single bit per pixel is needed, as compared to 16 or 24 bits per pixel for an uncompressed depth buffer.

Early pixel rejection (due to either computation masks, occlusion culling, or depth bounds testing) is unlikely to occur with single-pixel granularity. It is more likely that such culling takes place on a per-tile basis, such as a 16-pixel or 8-pixel tile. Fortunately, the images in Figures 1b and 10b suggest that non-silhouette pixels tend to occupy large contiguous regions of screen space and can benefit from conservative tile-based culling.

6.3. Additional Optimizations

One way to reduce further the number of classified silhouette pixels is to consider only the pixels that undersample the shadow map. Stamminger and Drettakis [SD02] derive a simple formula for estimating the ratio r of image resolution to shadow-map resolution; silhouette pixels with $r < 1$ can be omitted because they won't exhibit aliasing artifacts. This culling strategy could be added to the shader in Figure 4 at the cost of additional per-pixel floating-point arithmetic. In our test cases, however, we found that this technique reduced the number of classified silhouette pixels by only 5%, not enough to justify the computational overhead.

We have also considered (but not implemented) an optimization inspired by the work of Lloyd et al. [LWGMar] and the hybrid algorithms of McCool [McC00] and Govindaraju et al. [GLY*03]. As mentioned earlier, an advantage of McCool's work is that shadow volumes are not needed for blockers that are entirely in shadow, because these blockers are absent from the shadow map. Similarly, Lloyd et al. and Govindaraju et al. use image-space occlusion queries to reduce the number of blockers and receivers considered for shadow computation.

These occlusion queries could be combined with our algorithm in the following way. After rendering a shadow map in the first step of the algorithm, render a bounding volume for each blocker from the light's viewpoint and use an occlusion query to check if any of the bounding volume's pixels pass the depth test. If no pixels pass, then drawing the shadow volumes for that blocker may be skipped. Note that this culling strategy must be applied on a per-blocker basis (as opposed to a per-silhouette-edge basis) to ensure the consistency of the stencil-based shadow volume algorithm.

This optimization is useful in situations where complex blockers are often shadowed; a common scenario is a computer game in which monsters hide in the shadows. In these cases, it may be possible to avoid drawing the shadow polygons entirely for a large model. The cost of the optimization includes the occlusion query, which adds latency to the rendering pipeline, and the drawing of bounding boxes, which consumes additional fillrate. On the other hand, most of the latency can be hidden by issuing many queries but only checking the result of the first query after several bounding volumes have been drawn. Drawing a bounding box is also fast because there are no writes to the framebuffer, no shading is needed, and hardware-accelerated occlusion culling is applicable.

7. Conclusions

We have presented a hybrid shadow rendering approach that combines the strengths of shadow maps and shadow volumes. The key idea is to use shadow maps to identify which regions of the image will exhibit edge aliasing, then apply shadow volumes only in those regions to obtain better accuracy.

We have shown that our algorithm benefits from using a computation mask that discards pixels early in the pipeline. More generally, a computation mask allows the programmer to identify a small set of pixels in the scene that are "inter-

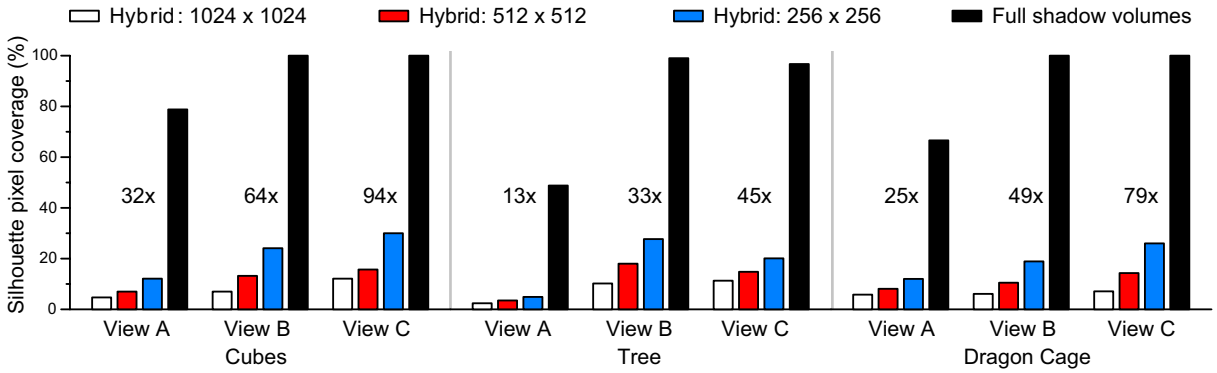

Figure 11: *Shadow volume overdraw and silhouette pixel coverage. The black bars show the percentage of pixels in the image covered by shadow volumes, and the number next to each bar is the overdraw factor (the ratio of shadow-polygon pixels to the total image size). The white, red, and blue bars show the percentage of pixels in the image that are classified as shadow silhouette pixels by our hybrid algorithm; colors correspond to different shadow-map resolutions.*

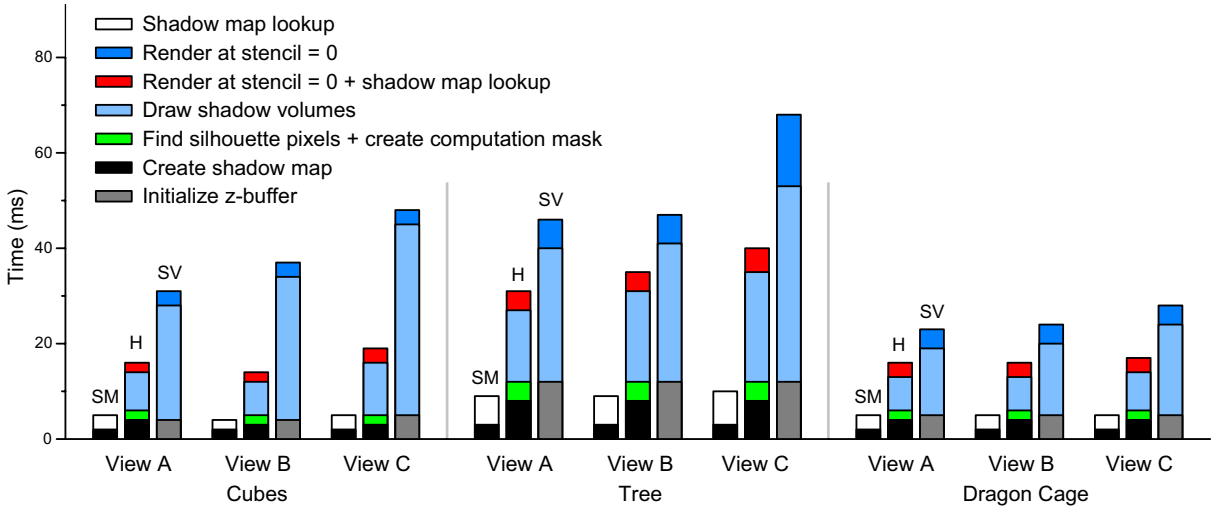

Figure 12: *Performance comparison. The vertical bars measure the per-frame time in milliseconds (ms) for the shadow map algorithm (SM), our hybrid algorithm (H), and the shadow volume algorithm (SV). Colored sections of a bar indicate how much time is spent in each part of the algorithm.*

esting"; all other pixels are masked off. Expensive per-pixel algorithms can then operate on this subset of pixels without incurring the cost over the entire framebuffer. This strategy of decomposing a problem into two parts — a large part that relies on a fast but inexact technique, and a small part that uses a slower but accurate technique — is applicable to a number of multipass rendering algorithms.

We believe that the benefits of hardware computation masks will become more significant as pixel shaders increase in complexity. Current hardware already performs aggressive occlusion culling to avoid unnecessary shading. Computation masks represent a logical step in this direction.

Acknowledgments

Special thanks to Nick Triantos and Mark Kilgard from NVIDIA for providing hardware and driver assistance. Our discussions with them have contributed greatly to our understanding of occlusion culling performance issues. Jan Kautz, Addy Ngan, Timo Aila, and our anonymous reviewers provided valuable feedback on this paper. This work is supported by an ASEE National Defense Science and Engineering Graduate fellowship.

Appendix

The EXT_depth_bounds_test OpenGL extension [Ope02] states that a rasterized fragment is discarded if the current depth value stored in the framebuffer at that fragment's ad-

dress lies outside user-specified depth bounds. The key is that an architecture's implementation may discard fragments early in the pipeline without performing buffer updates, such as stencil writes. Thus modern graphics architectures, which often rasterize and shade tiles of fragments in parallel, can aggressively discard an entire tile during rasterization if all the fragments in the tile lie outside of the depth bounds. Similar mechanisms are already used for early z occlusion culling [Mor00].

References

[AAMar] AILA T., AKENINE-MÖLLER T.: A hierarchical shadow volume algorithm. In *Proceedings of the ACM SIGGRAPH / Eurographics Workshop on Graphics Hardware* (2004 (to appear)), ACM Press. 3

[AMH02] AKENINE-MÖLLER T., HAINES E.: *Real-Time Rendering (2nd edition)*. A. K. Peters Ltd., 2002, pp. 248–276. 2, 6

[AMN03] AILA T., MIETTINEN V., NORDLUND P.: Delay streams for graphics hardware. *ACM Transactions on Graphics (TOG) 22*, 3 (2003), 792–800. 3

[Ber86] BERGERON P.: A general version of Crow's shadow volumes. In *IEEE Computer Graphics and Applications* (September 1986), pp. 17–28. 3

[Car00] CARMACK J.: Private email, May 23, 2000. http://developer.nvidia.com/attach/3413. 3

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. In *Proceedings of ACM SIGGRAPH* (1977), ACM Press, pp. 242–248. 2

[Die96] DIEFENBACH P.: *Multi-pass Pipeline Rendering: Interaction and Realism through Hardware Provisions*. Ph. D. thesis MS-CIS-96-26, University of Pennsylvania, 1996. 3

[EK02] EVERITT C., KILGARD M. J.: Practical and robust stenciled shadow volumes for hardware-accelerated rendering, 2002. http://developer.nvidia.com/object/robust_shadow_volumes.html. 3, 4, 5

[EK03] EVERITT C., KILGARD M. J.: Optimized stencil shadow volumes, 2003. http://developer.nvidia.com/docs/IO/8230/GDC2003_ShadowVolumes.pdf. 3, 6

[FFBG01] FERNANDO R., FERNANDEZ S., BALA K., GREENBERG D. P.: Adaptive shadow maps. In *Proceedings of ACM SIGGRAPH* (2001), ACM Press, pp. 387–390. 2

[GLY*03] GOVINDARAJU N. K., LLOYD B., YOON S.-E., SUD A., MANOCHA D.: Interactive shadow generation in complex environments. *ACM Transactions on Graphics (TOG) 22*, 3 (2003), 501–510. 3, 7, 9

[Hei91] HEIDMANN T.: Real shadows, real time. In *Iris Universe, No. 18* (November 1991), Silicon Graphics Inc., pp. 23–31. http://developer.nvidia.com/attach/3414. 3

[Koz04] KOZLOV S.: Perspective shadow maps: Care and feeding. In *GPU Gems* (2004), Addison-Wesley. 7

[Len02] LENGYEL E.: The mechanics of robust stencil shadows. In *Gamasutra* (October 11, 2002). http://www.gamasutra.com/features/20021011/lengyel_01.htm. 3, 6

[LV00] LOKOVIC T., VEACH E.: Deep shadow maps. In *Proceedings of ACM SIGGRAPH* (2000), ACM Press, pp. 385–392. 2

[LWGMar] LLOYD B., WENDT J., GOVINDARAJU N., MANOCHA D.: CC shadow volumes. In *Proceedings of the Eurographics Symposium on Rendering* (2004 (to appear)). 3, 9

[McC00] MCCOOL M. D.: Shadow volume reconstruction from depth maps. *ACM Transactions on Graphics (TOG) 19*, 1 (2000), 1–26. 3, 7, 9

[MHE*03] MCGUIRE M., HUGHES J. F., EGAN K., KILGARD M., EVERITT C.: *Fast, practical and robust shadows*. Technical Report CS03-19, Brown University, October 27, 2003. 3, 6

[Mor00] MOREIN S.: ATI Radeon – HyperZ technology, August 22, 2000. http://www.ibiblio.org/hwws/previous/www_2000/presentations/ATIHot3D.pdf. 5, 11

[Ope02] OPENGL ARCHITECTURAL REVIEW BOARD: `EXT_depth_bounds_test` OpenGL extension specification, 2002. http://www.nvidia.com/dev_content/nvopenglspecs/GL_EXT_depth_bounds_test.txt. 5, 10

[PDC*03] PURCELL T. J., DONNER C., CAMMARANO M., JENSEN H. W., HANRAHAN P.: Photon mapping on programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware* (2003), Eurographics Association, pp. 41–50. 5

[Pur04] PURCELL T. J.: *Ray Tracing On A Stream Processor*. Ph. D. thesis, Stanford University, 2004. 5

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *Proceedings of ACM SIGGRAPH* (1987), ACM Press, pp. 283–291. 2, 5, 6

[SCH03] SEN P., CAMMARANO M., HANRAHAN P.: Shadow silhouette maps. *ACM Transactions on Graphics (TOG) 22*, 3 (2003), 521–526. 1, 2, 4, 6, 7

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. In *Proceedings of ACM SIGGRAPH* (2002), ACM Press, pp. 557–562. 2, 3, 6, 7, 9

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *Proceedings of ACM SIGGRAPH* (1978), ACM Press, pp. 270–274. 2, 6

[WPF90] WOO A., POULIN P., FOURNIER A.: A survey of shadow algorithms. *IEEE Computer Graphics and Applications 10*, 6 (Nov. 1990), 13–32. 2