Shadow Volume Reconstruction from Depth Maps

MICHAEL D. McCOOL
University of Waterloo

Current graphics hardware can be used to generate shadows using either the shadow volume or shadow map techniques. However, the shadow volume technique requires access to a representation of the scene as a polygonal model, and handling the near plane clip correctly and efficiently is difficult; conversely, accurate shadow maps require high-precision texture map data representations, but these are not widely supported.

We present a hybrid of the shadow map and shadow volume approaches which does not have these difficulties and leverages high-performance polygon rendering. The scene is rendered from the point of view of the light source and a sampled depth map is recovered. Edge detection and a template-based reconstruction technique are used to generate a global shadow volume boundary surface, after which the pixels in shadow can be marked using only a one-bit stencil buffer and a single-pass rendering of the shadow volume boundary polygons. The simple form of our template-based reconstruction scheme simplifies capping the shadow volume after the near plane clip.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Color, shading, shadowing, and texture*; I.4.8 [**Image Processing**]: Scene Analysis—*Range data*

General Terms: Algorithms, Human Factors, Performance

Additional Key Words and Phrases: Hardware accelerated image synthesis, illumination, image processing, shadows

1. INTRODUCTION

Shadows are a very important spatial cue. They help determine the relative positions of objects, particularly depth order and the height of objects above the ground plane. Since a shadow is basically a projection of the scene from

This research was sponsored by a grant from the National Science and Engineering Research Council of Canada.
Author's address: Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada; email: mmccool@cgl.uwaterloo.ca; http://www.cgl.uwaterloo.ca/~mmccool/.

an alternative viewpoint, cast shadows can also elucidate the shape of an object [Wanger 1992] and the positions of light sources.

Generating shadows is a classic computer graphics problem, and considerable research effort has been devoted to it [Crow 1977; Woo et al. 1990]. In this paper we focus on the simplest case: hard-edged umbral shadows cast by point sources. Given a fast umbral shadow algorithm, soft shadows can be approximated by summing the contributions of several such sources. However, there are severe limitations in existing algorithms for interactive generation of even hard-edged shadows using the current generation of hardware.

1.1 Taxonomy and Tradeoffs

Existing hard shadow algorithms can be divided into four broad categories: ray casting, projection, shadow volumes, and shadow maps. Of these, only the last three are (currently) applicable to real-time rendering. There are also several hybrid algorithms that combine features from different categories.

A distinction can also be drawn between object-precision and image-precision algorithms. Object-precision algorithms result in more precise shadows, but require access to a polygonal representation of the scene.

Polygonal representations are not available in systems that use "alternative" modeling and rendering techniques such as selective raycasting of objects, run-based rendering of enumerated volumes, layered depth images, depth buffer implementation of CSG [Rossignac and Requicha 1986; Wiegand 1996]; distance-volume primitives [Westermann et al. 1999]; direct rasterization of biquadratic patches, depth shaders, adaptive isocontour tracing of parametric patches [Elber and Cohen 1996], etc.

Image-precision algorithms are less accurate, but also place fewer constraints on the scene representation. In practice, even if the scene is represented with polygons, they can be easier to use and more flexible. For instance, many applications use polygonal primitives but may generate them on-the-fly from other representations, and do not maintain a polygonal database. Even if polygonal primitives are used and can be intercepted in a noninvasive way (for example, via OpenGL feedback [Kilgard 1997]), adjacency information (required for an important optimization in shadow volumes, for instance) may not be available. Fortunately, for many applications shadows do not have to be very accurate to be useful, and image-precision algorithms can be used.

In the following sections the three classes of shadowing algorithms applicable to real-time rendering are surveyed and compared. Our new technique is a hybrid of the shadow volume technique and the shadow map technique.

1.2 Projection Algorithms

Projection techniques project primitives away from the light source onto the surface of other primitives. Blinn's "fake shadows" algorithm [Blinn

1988] squashes all polygons in the object casting a shadow down onto the plane of another polygon, where they can be drawn in black or composited over previously rendered pixels to approximate a shadow of the object onto that plane. This technique is simple, and is suitable for hardware acceleration, but does not scale or generalize well.

More general object-precision projection algorithms clip polygons into shadowed and unshadowed parts [Atherton et al. 1978; Weiler and Atherton 1977] in a prepass. Since they use polygon clipping, these algorithms require access to a polygonal representation of the scene.

Data structures such as BSP trees may be used to accelerate the polygon-clipping process [Chin and Feiner 1989] and manage the selective reclipping of polygons when objects are moved [Chrysanthou and Slater 1995]. The splitting planes of the BSP trees in these algorithms are given by the scene polygons and the shadow volume boundary, so these techniques are in fact hybrids of the shadow volume and projection approaches.

The projection approach can be applied to volumetric primitives by projecting semitransparent slices onto each other recursively [Behrens and Ratering 1998]. Projection techniques also exist for area light sources and soft shadows, in both exact object-precision [Dretakkis and Fiume 1994; Stewart and Ghali 1994] and approximate image-precision [Soler and Sillion 1998] forms.

1.3 Shadow Volumes

The classical shadow volume algorithm [Bergeron 1985; 1986; Crow 1977] requires a representation of the scene as a polygonal database, and represents the boundary between illuminated and shadowed space with object-precision polygons. The intersection between the boundary of the shadow volume and the scene can be computed at image precision using per-pixel depth comparison and counting operations.

A *shadow volume boundary polygon*, or *shadow polygon* for short, is constructed for each edge of every object polygon in the scene. Each shadow polygon is a semi-infinite quadrilateral with two finite vertices corresponding to each pair of edge endpoints and two infinite vertices. The infinite vertices are placed at the limit of rays emanating from the light source and passing through each finite vertex. A clipper operating with homogeneous coordinates will reduce this semi-infinite quadrilateral to a finite size when it is clipped against the viewing frustum.

The shadow polygons together with their generating object polygon enclose a semi-infinite volume of space that is shadowed by that particular object polygon. The union of all the shadow volumes generated by all object polygons is the shadow volume for the scene.

Computing the union of the per-polygon shadow volumes can be done by counting “in” and “out” events along rays from the eye. Orientation of shadow polygons must be maintained so that the front faces of the shadow polygons point towards illuminated space. This is necessary so “in” events can be distinguished from “out” events. If the eye is in fully illuminated

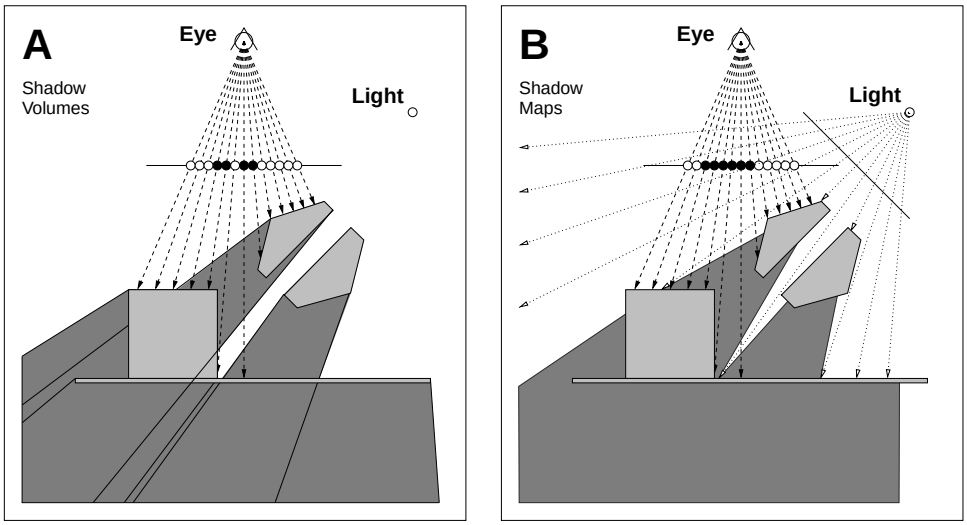

Fig. 1. (a) The shadow volume technique; (b) the shadow map technique.

space, a ray from the eye to a point on a surface will pierce an equal number of front-facing and back-facing shadow polygons if and only if the surface point at the ray terminus is not inside the union of the shadow volumes, i.e., is illuminated.

In an important optimization, shadow polygons originating in nonsilhouette shared object polygon edges can be removed, with the silhouette defined relative to each light source. A flatland example of a shadow volume after such editing is given in Figure 1(a).

Shared-edge shadow polygons can be removed because the orientations of such polygons generated from shared edges between adjacent nonsilhouette polygons will be opposed and will cancel. If both open and closed (nonmanifold) objects can appear in the scene, weights must be added to the shadow volume boundary polygons. Nonmanifold, unshared edges should always generate shadow polygons with a weight of 1. Shadow polygons generated from shared silhouette edges should be assigned a weight of 2 [Bergeron 1986].

Shadow volumes can be rendered in conjunction with scanline [Bergeron 1985; 1986; Bouknight and Kelly 1970; Crow 1977]; and with depth-buffer and BSP hidden-surface algorithms [Chrysanthou and Slater 1995]. Jansen and van der Zalm [1991] show how to derive shadow volumes for objects constructed with CSG set operators from the shadow volumes of their arguments.

To render shadows with hardware acceleration, a *shadow count* needs to be maintained per pixel by the hardware. After the eye view of the scene has been rasterized to initialize the depth buffer, the shadow polygons are rasterized. Shadow polygon rasterization fragments do not modify the color or depth buffer. However, if a shadow polygon fragment passes the depth test, the corresponding shadow count is incremented by the weight of the

front-facing shadow polygons and decremented by the weight of the back-facing shadow polygons. Pixels with a shadow count of 0 at the end of this process are illuminated; the rest are in shadow.

The Pixel Planes architecture [Fuchs et al. 1985] includes support for shadow volumes. One interesting feature of this architecture is that polygon size does not affect performance. This is fortunate, since shadow polygons tend to be large. In most architectures larger polygons take longer to rasterize. If the fill rate is high relative to the vertex transformation rate, this dependency may not be significant.

Under OpenGL the (fixed-precision) buffer for maintaining the shadow count is called a *stencil buffer*.

The shadow count for an illuminated surface, 0, is only one out of 2^s possible counts in an s-bit stencil buffer. If modular arithmetic is used, it is possible that parts of the scene could be falsely illuminated, although improbable: a nonzero shadow count would have to alias to 0 mod 2^s.

However, the OpenGL stencil buffer *clamps* at both 0 and its maximum value, 2^s-1. To avoid underflow, at least two passes over the shadow volume are needed: the front-facing (positive count) shadow polygons must be rendered first, followed by back-facing shadow polygons. More passes are necessary if weights are used. If overflow occurs during rendering of the front faces, illuminated parts of the scene would be rendered correctly, since the magnitude of the negative count will be equal to the positive count, and therefore larger than 2^s-1. However, some parts of shadow may be falsely illuminated, as some portion of the positive count might be lost.

1.4 Shadow Maps

The shadow map technique uses a depth map generated from the point of view of the light source. As each pixel is rendered in the eye view, its three-dimensional location is transformed back into the lighting coordinate system with a projective transformation. The depth of this transformed point in the light source coordinate system is compared with the stored depth. If the stored depth is smaller, there is an occluder between the point being rendered and the light source, and the point is in shadow.

To avoid false self-shadowing of surfaces and "surface acne," small biases are added to the depth comparisons. The shadow depths may also be dithered and a soft transition to shadow may be used [Reeves et al. 1987] to simulate penumbra.

A version of this algorithm has been implemented in hardware [Segal et al. 1992]. The result of the depth comparison sets an alpha value in each pixel that can then be used to composite illuminated and unilluminated renderings. On standard hardware, a similar technique can be implemented using the alpha test and projective texturing [Heidrich 1999].

Unfortunately, the precision of values stored in texture maps is usually low. To be generally useful, the shadow map approach requires not only texture map support but also a high precision depth representation supported by the texture mapping hardware. (However, Mark Kilgard recently

demonstrated extended-precision shadow map comparisons under OpenGL.) Texture mapping datapaths are generally designed for processing color, each component of which needs at most 12 bits; color can be adequately represented for most purposes using only 8 bits. In contrast, the precision of a hardware depth buffer is generally between 16 and 32 bits.

Depth precision is especially important for shadows; the shadow volume boundary naturally grazes the surfaces in the scene. If the depth values are imprecise, a large bias must be used to avoid surface acne, and this can cause serious inaccuracies elsewhere. For instance, with a large bias, shadows cast on ground planes by objects sitting on these planes can become detached from the objects casting the shadows. An "adaptive bias" can be used to correct this problem, for instance by storing in the depth map the depth halfway between the frontmost and the secondmost depth from the lightsource, but this complicates the acquisition of the shadow map.

A flatland example of the shadow map algorithm is given in Figure 1(b), using linear interpolation of depth values. Bias is not shown, but would in general be needed to avoid surface self-shadowing in the presence of quantization error.

Conservative variants of shadow maps have been used in ray-tracing acceleration [Haines and Greenberg 1986; Woo 1993] for direct shadows. In interactive applications, the shadow maps can be incrementally reprojected, as can the eye view, to increase the frame rate [Chen and Williams 1993].

1.5 Shadow Volume Reconstruction

The *shadow volume reconstruction* algorithm is a hybrid of the shadow map and shadow volume algorithms that does not require a polygonal representation of the scene. Like the shadow map algorithm, it requires a depth map rendered from the point of view of the light source instead. This light-view depth map information is used to reconstruct a polygonal shadow volume boundary that can be combined with an eye-view depth map using only a one-bit stencil buffer.

The shadow volume reconstruction algorithm imposes no limitations on the use of the hardware during the rendering of the light-view depth map or the eye-view and its depth map; hence, it can be used with *any* hardware-assisted rendering algorithm that generates a correct depth map.

The reconstruction of the shadow volume is based on computer vision techniques. We detect silhouette edges and build a polygonal mesh representing the important shadow volume boundaries.

The next section presents the basics of the hybrid algorithm. Section 3 discusses the important optimization of silhouette edge detection, and Section 4 presents details on how the polygon mesh for the shadow volume boundary is constructed. In Sections 5 and 6 the practical problems of dealing with multiple shadow maps, edge effects, and eye-view near plane clipping are covered. In Section 7 several shadow rendering modes are

discussed. Section 8 discusses some useful extensions resulting from the hybrid nature of the algorithm, and presents some preliminary results from a vectorizing reconstruction process. Finally, in Section 9 per-phase timings are given.

2. HYBRID ALGORITHM

Observe that the $z[x, y]$ depth samples in the shadow map, in conjunction with their pixel coordinates, describe points (x, y, z) in the scene relative to the light source in an orthogonal device coordinate system. We can join these points into a polygonal mesh and transform them into world space, using the inverse of the shadow map projection. The surface thus constructed defines the boundary between shadow and light in the scene; it is the boundary of the shadow volume.

Unlike Crow's algorithm, a shadow volume boundary generated this way consists of a single star-shaped surface, with no nesting or overlapping. In this case, there is no need to implement a full CSG union operation, summing front-facing polygons and subtracting back-facing polygons. We just need to keep track of the *parity* of the number of shadow polygons that are in front of the surface at each pixel.

If there are an odd number of shadow volume boundary intersections along a ray from the eye to the first surface point at a pixel, then that surface point is in shadow. Otherwise, it is illuminated. To keep track of shadow parity, a single bit stencil buffer is needed. The stencil bit for a pixel is simply toggled whenever a shadow polygon fragment is drawn on top of it. Practically speaking, this simplification is a major advantage for hardware implementations. The extra frame buffer storage required is minimal and finite. It is impossible to overflow the shadow depth count, as is possible with Crow's algorithm.

The details of the hybrid algorithm are as follows:

(1) **Render the shadow map**. The scene is rendered from the point of view of the light source and the depth buffer is read back. Multiple shadow maps may be required for omnidirectional illumination and shadowing.

(2) **Render the eye view**. The scene should be rendered as usual from the point of view of the eye.

(3) **Reconfigure the frame buffer.** Clear the stencil buffer. Disable writing to the color and depth buffers, but enable the depth test and set the stencil to toggle when a shadow polygon fragment passes the depth test.

(4) **Render the shadow volume**. Reconstruct the shadow volume from the $z[x, y]$ coordinates in the shadow map, transform it back into world space, project it through the same viewing transformation as the rest of the scene, and rasterize it. Both front and back faces can be rendered simultaneously; ordering or distinguishing them is not necessary.

(5) **Cap the shadow volume**. Whenever the shadow volume boundary is clipped by the near plane, cap polygons need to be generated to ensure that the shadow volume is properly enclosed.

(6) **Darken the shadowed pixels.** The pixels where the stencil bit is set to 1 are in shadow. Render the shadow using one of the techniques in Section 7.

Since the texture-mapping hardware is not busy rendering shadows (as in Segal et al. [1992] or with Kilgard's technique), the scene can contain textured polygons without an extra rendering pass. All other hardware facilities are also available during rendering of the eye view and of the shadow map, including the stencil planes. The advantages of the shadow map algorithm are inherited: In particular, since the shadows depend only on the values left in the depth buffer by the scene renderer, it is nonintrusive. Adding shadows to a scene will not typically require access to or modification of the base renderer.

Another advantage of the hybrid approach is that shadow polygons are minimal in size. They extend to the shadowed surface from the shadowing surface, but extend no further. A practical problem with Crow's algorithm is the size of the shadow polygons. All shadow polygons extend outwards to the edge of the viewing frustum after clipping, since intersection with the surfaces being shadowed is not checked. If the speed of the rasterization engine depends on the image-space size of polygons, this can degrade performance.

The hybrid method also inherits some of the disadvantages of the shadow map approach. The quantization of light buffer coordinates, particularly x and y, causes aliasing of detail in the shadow volume. The problem is literally magnified when the shadow is projected. Precision in depth is usually adequate, but appropriate bias translations need to be added to both the light-relative z values and, in our experience, the depth values from the eye-view [Reeves et al. 1987].

The other disadvantage is that in this most basic variant of the hybrid algorithm, as in the basic version of Crow's algorithm, a large number of shadow polygons are generated. In the following section we show how this deficiency can be partially overcome through silhouette edge detection.

3. EDGE DETECTION

The shadow volume only needs to contain polygons originating in the silhouette edges of objects as seen from the light source. However, if all the points defined by the shadow map are used, many shadow polygons will be rendered that lie on the surface of objects. These shadow polygons will be culled by the depth test, so transforming and rasterizing them is a source of inefficiency that should be eliminated.

To generate only the shadow polygons corresponding to the silhouette edges, an edge-detection process can be used. Edge detectors for computer vision are generally designed to be robust in the face of noisy, low-precision

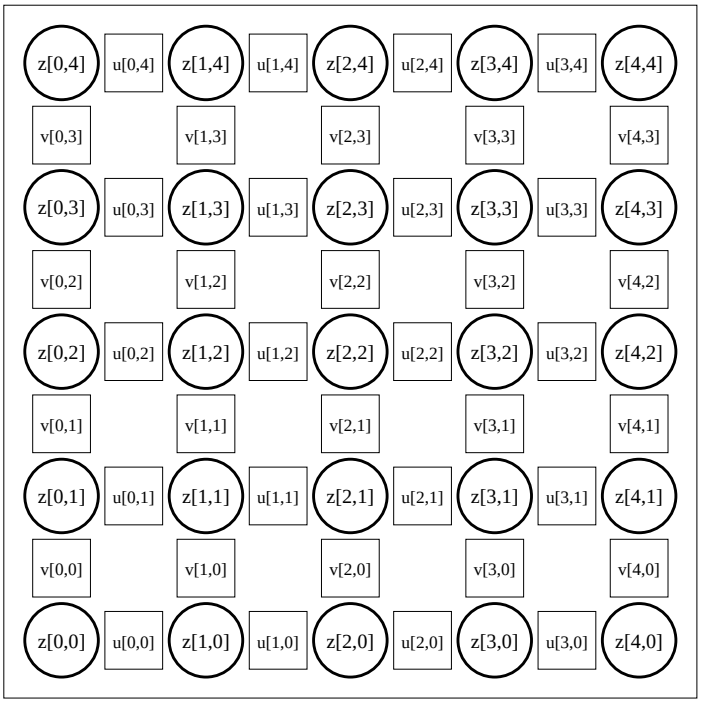

Fig. 2. Two edge flag arrays are interdigitated with the shadow depth map.

sensor data [Canny 1986; Marr and Hildreth 1980]. However, the depth values in the shadow map are available at relatively high precision and the only noise is quantization error. If necessary, we can even modify the shadow map rendering process to produce more information about edges, such as an object ID channel.

Our edge-detection process can also afford to be conservative. If we accidentally generate a shadow polygon that does not correspond to a silhouette edge (false positive), it will be culled by the depth test and will not affect the final image. Unfortunately, a missed edge (false negative) will leave a gap in the shadow volume boundary and result in a visible error. These considerations imply that we should choose the simplest and fastest edge detector available, but should strive to avoid false negatives.

3.1 Difference Magnitude Edges

The simplest discontinuity detector is a thresholded magnitude of the first derivative. The simplest estimate of a derivative is a difference. Using horizontal and vertical first differences, we detect edges in both the horizontal and vertical directions and set flags between neighboring pixels. u is an array of Boolean flags for vertical edges, while the v array flags horizontal edges (see Figure 2). Together we call these flags the *edge map*. Let θ be a threshold value and define the forward differences $\Delta_x z[x, y] =$

$z[x + 1, y] - z[x, y]$ and $\Delta_y z[x, y] = z[x, y + 1] - z[x, y]$. Then the edge flags are set by the following computation:

$$u[x, y] = (|\Delta_x z[x, y]| > \theta),$$

$$v[x, y] = (|\Delta_y z[x, y]| > \theta).$$

We do *not* filter the depth values before applying the differences. Doing so would only blur the edges, and possibly modify their apparent positions (i.e., by rounding corners), and in the absence of significant noise is pointless.

The threshold θ should be chosen conservatively; a value that marks $O(n)$ of the $O(n^2)$ potential edges should be approximately correct. If automatic adaptation is required, a histogram of difference magnitudes can be used, but a constant threshold is adequate if the near and far planes used to render the shadow map are set tightly.

The first difference has a simple geometric interpretation as the distance between surfaces. This simple threshold detector therefore guarantees that a shadow polygon will be generated whenever the distance between surfaces (in the projectively transformed light-relative device coordinate system) exceeds θ. With a little more work we can transform the z values back into world space before applying the threshold. Quantization error, however, will not be constant in world space.

3.2 Reducing False Negatives

False positives merely decrease efficiency. False negatives, however, will cause image artifacts. There are two serious sources of false negatives: aliasing and abutment.

If the shadow map resolution is too low, the rendering of the depth map will be aliased. Certain aliasing artifacts can be guarded against. If small objects are disappearing, an edge rendering can be combined with a standard nonoverlapping polygon fill rendering. In the worst case, edge detection can be disabled altogether to avoid incorrect responses to false structure artifacts such as Moiré patterns. The necessity of such a measure can be estimated from the fraction of potential edge flags that are set with a particular threshold. This fraction will give a measure of scene complexity.

Edge flags can also be omitted at the ends of a silhouette edge defined when two objects abut (as when an object sits on a ground plane), intersect, or where "folds" end in a concave object. The depth contrast in these regions may be too low for differences to exceed the threshold.

By lowering edge-detection thresholds in the vicinity of a dangling edge, we can deal with abutment problems conservatively, but without introducing too many new shadow volume polygons.

Call the square spaces between four adjacent depth samples *cells*. In practice, we store edge flags redundantly in the cells using a packed

representation. Every cell contains a 4-bit code that characterizes the configuration of edge flags around it. Under this encoding, dangling edges are easy to identify: they correspond to cells with the edge codes 0001, 0010, 0100, or 1000.

As a last resort, we can generate other information during the rendering of the shadow map to help determine the location of silhouette edges. Suppose we render different objects in the scene using different constant colors (IDs). We can then read back the color channel, and if adjacent pixels are different colors, set the edge flag between them. If only convex objects are used in the scene, or if all concave objects are split into convex parts, then objects cannot self-shadow and the depth-based edge detector is not needed. If objects can be concave, the results of the ID test and the depth test can be ORed. This need only be done in the vicinity of a dangling edge.

The usefulness of an ID test will depend on how efficiently frame buffer contents can be read back and scanned, whether disabling writing to the color buffers actually decreases the rendering time on a particular set of graphics hardware, how complex the scene is, and how stringent the image quality requirements are.

3.3 Reducing False Positives

Our basic edge detector thresholds the magnitude of the first difference. Assume that we have run this detector using a relatively low threshold θ_L to obtain a set of candidate edge flags.

A low threshold can result in blocks of false positives over steeply sloped surfaces that are nearly edge-on in the light view. To eliminate these false positives, we can use a second derivative test: If the second derivative does not have a strong zero-crossing of the correct sign across a candidate edge flag, indicating a local minima or maxima in the first derivative, then the candidate edge flag can be cleared. The second derivative can be estimated using a second difference.

The second derivative test thins "thick edges." This can lead to false negatives when, for instance, two silhouette edges at different depths align to within one pixel. The second derivative test should therefore be combined with a second threshold test relative to $\theta_H > \theta_L$ to identify "sure" edges:

$$u[x,y] = \begin{cases} |\Delta_x z[x,y]| > \theta_L \quad \rightarrow \begin{cases} |\Delta_x z[x,y]| > \theta_H & \rightarrow 1 \\ \Delta_x z[x,y] > 0 & \text{and} \\ \Delta_x^2 z[x-1,y] > \theta_S & \text{and} \\ \Delta_x^2 z[x,y] < \theta_S & \rightarrow 1 \\ \Delta_x z[x,y] < 0 & \text{and} \\ \Delta_x^2 z[x-1,y] < \theta_S & \text{and} \\ \Delta_x^2 z[x,y] > \theta_S & \rightarrow 1 \\ \text{otherwise} & \rightarrow 0 \end{cases} \\ \text{otherwise} \quad \rightarrow 0 \end{cases}$$

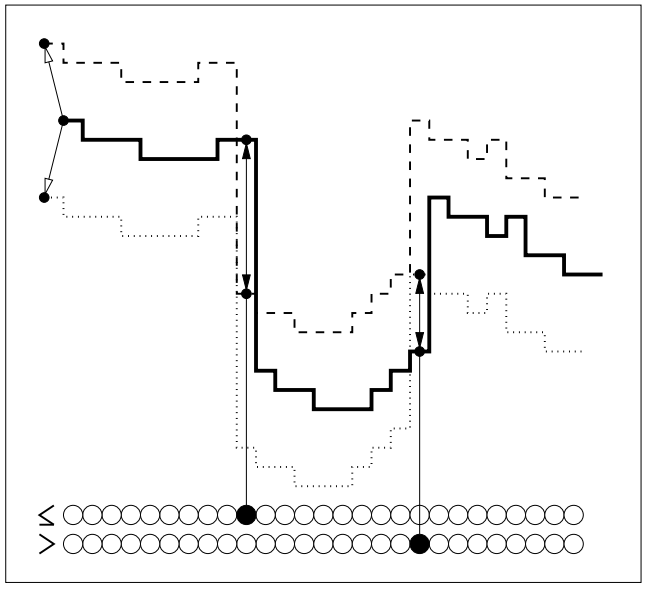

Fig. 3. Difference thresholded edge detection can be performed in the depth buffer hardware.

and likewise for $v[x, y]$.

Additional ideas from computer vision, such as threshold hysteresis [Canny 1986], can also be used to improve the detection of edges in low-contrast situations.

These techniques can be implemented to operate only on the $O(n)$ candidate edge flags generated by the $O(n^2)$ initial θ_L thresholding pass.

3.4 Hardware Edge Detection

The graphics hardware can be used to implement the first difference thresholding edge detector, and can also be used to generate a hierarchical representation of the edge map to avoid the initial $O(n^2)$ data traversal cost.[1]

The idea is sketched in Figure 3. After the eye view is rendered, writing to the depth buffer is disabled. The contents of the depth buffer are shifted down in x and y by one sample and biased by $\pm\theta$, then copied back into the existing depth buffer, with depth testing enabled. When the samples are biased by $-\theta$ the depth test should be $>$, when they are biased by $+\theta$ the depth test should be $<$. The combined set of operations is

$$z[x + 1, y] + \theta < z[x, y],$$

$$z[x + 1, y] - \theta > z[x, y],$$

[1]This cost can be significant. In our implementation, run-length encoding the output of the edge detector so the reconstruction algorithm in Section 4 only had to scan $O(n)$ cells resulted in an order of magnitude improvement in the performance of the reconstruction algorithm.

$$z[x, y + 1] + \theta < z[x, y],$$

$$z[x, y + 1] - \theta > z[x, y];$$

this is equivalent to

$$z[x, y] - z[x + 1, y] > \theta,$$

$$z[x + 1, y] - z[x, y] > \theta,$$

$$z[x, y] - z[x, y + 1] > \theta,$$

$$z[x, y + 1] - z[x, y] > \theta,$$

which is in turn equivalent to

$$|\Delta_x z[x, y]| > \theta,$$

$$|\Delta_y z[x, y]| > \theta,$$

as desired.

The stencil and/or the color buffers can be used to accumulate the results of the depth comparisons, and the color buffer can be used to generate a hierarchical map of nonzero cells using hardware zooming and compositing. All of these operations can be performed entirely in hardware, using the existing OpenGL API (i.e., `glCopyPixels` and `glPixelZoom`). In fact, using certain proposed extensions, histograms can be computed in hardware to assist with threshold selection.

Unfortunately, we have found that with many current OpenGL implementations, depth buffer operations can be slow and are often not implemented correctly, i.e., the copied pixels are not passed through all fragment tests as specified in the standard, or the depth buffer is not read back correctly when the depth test is enabled. We hope future generations of graphics hardware will resolve these issues.

4. SURFACE RECONSTRUCTION

Once the edge flags are set, we can reconstruct the shadow polygons. The simplest approach generates only the parts of a fixed mesh that abut at least one marked edge; see Figure 4. Alternatively, we can generate specific clusters of polygons for each possible combination of edge flags, using table lookup on the 4-bit combined edge code for each cell, as in Figure 5.

When we need to reconstruct a quadrilateral, we need to choose how it will be split into triangles. A poor split results in rough-edged shadows. The usual approach of comparing the deviation of normals does not work well. Instead, the diagonal differences in depth across each cell should be computed and the quadrilateral split to cut *across* the diagonal difference of

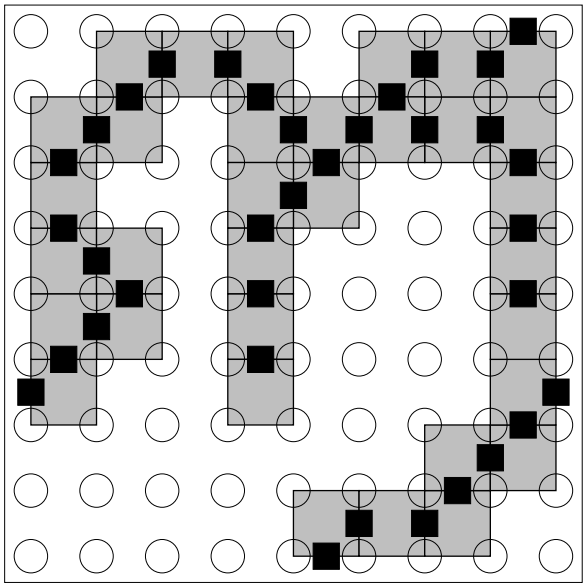

Fig. 4. A fixed quadrilateral mesh is edited to only contain elements that overlap a marked edge.

the greatest magnitude. This will produce shadows with 45° jaggies rather than 90° jaggies.

We can divide the triangle configurations into two categories based upon which way the quadrilateral is split. This information must be stored because it is necessary for capping, as described in Section 6.2.

5. MULTIPLE MAPS

Since a single shadow map is limited in its field of view, multiple shadow maps are needed to cast shadows omnidirectionally from a point light source. Each shadow map addresses a partition of space. The multiple shadow volume boundaries must be generated and fitted together carefully so cracks do not appear. Since the reconstructed surfaces are all part of a single star-shaped surface, they can all be rendered together in one pass of Step 2 of the algorithm in Section 2.

When rendering the shadow maps in the hybrid algorithm, the viewing frustum should be adjusted to render extra depth samples around the edges. This is visualized in Figure 6. This will permit the edge-detection and reconstruction process to respond correctly and consistently right up to the geometric edge of the shadow volume partition. The number of extra samples required will depend on the edge detector used.

6. CLIPPING AND CAPPING

The scene is clipped when it is rendered both from the point of view of the light source and from the eye. This can lead to several serious practical problems.

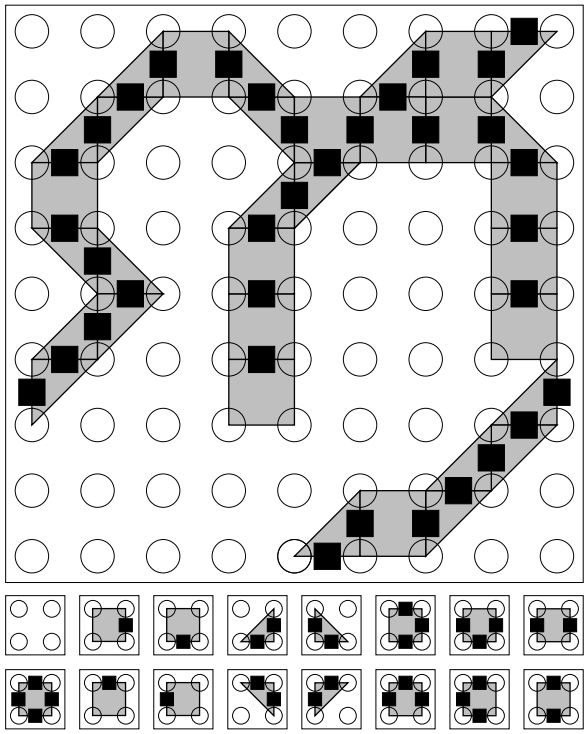

Fig. 5. Table lookup used to generate shadow mesh elements from the per-cell edge codes. The table shown at the bottom is designed to extend dangling edges by one pixel.

6.1 Light-View Clipping

In order to maximize depth precision in the shadow map, the near and far distances in the light view must be set as tightly as possible around the scene. If the near distance is too small, precision close to the far plane will suffer.

Objects closer to the light source than the near distance will be clipped during rendering, and so will either fail to cast a shadow or will cast a reduced shadow. If backface culling is used during rendering of the light view, it will appear as if holes have appeared in the occluder. Complete near-plane clipping is not critical, since it will appear that the light source has passed through the object.

Light-view far plane clipping is not a serious problem either if light-source attenuation is used. If a scene bounding volume is not available, the far plane can be set at a distance at which the light from the source can be neglected.

A bounding box around the viewing frustum in world space can be intersected with the scene bounding box to select a tight light-view projection. In this case if the eye view changes, the shadow map will have to be updated.

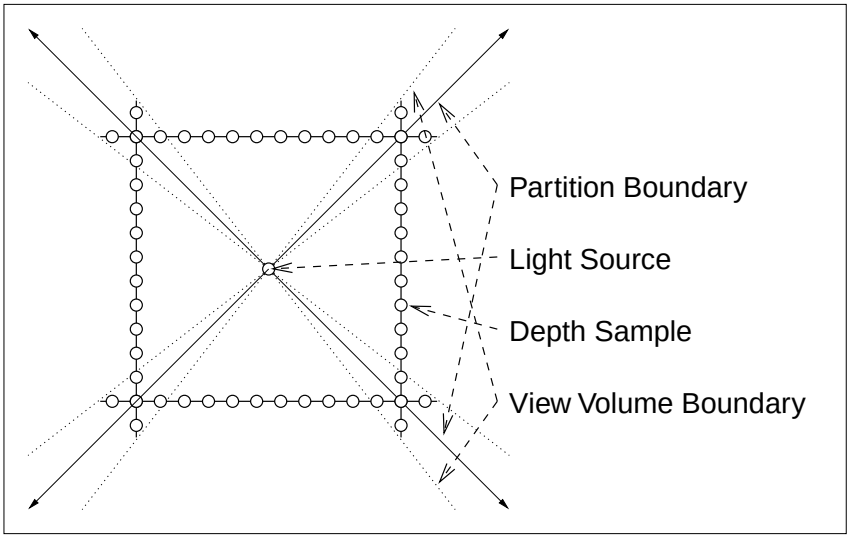

Fig. 6. Omnidirectional shadowing requires partitioned shadow maps that should overlap to maintain continuity.

6.2 Eye-View Clipping

The eye-view far plane clipping plane can be interpreted as a full-screen polygon, rendered in the background color at the maximum depth, which culls shadow volume polygons in the usual way with a depth test.

Unfortunately, if shadow polygons are clipped away by the near plane of the eye, parity information is lost. The standard "solution" of inverting the parity globally (or incrementing the shadow count globally) if the eye is in shadow [Bergeron 1985; Crow 1977] **does not work**, since the parity should only be inverted where the interior of the shadow volume is visible. This is demonstrated in Figure 7. The eye itself happens not to be in shadow in either of these examples, but the visible part of the near plane intersects the shadow volume boundary.

The near plane clip must be handled by capping the shadow volume boundary at the near plane of the view volume. Since the shadow volume boundary is not closed (due to the edge-detection elimination of redundant shadow polygons), does not have hidden surface removal applied to it, and may be hidden by parts of the existing scene depth values, we cannot generate caps in the usual manner, i.e., by detecting visible back-facing polygons, at least not on the same pass we use to set the shadow stencil flags. An additional pass could be used, but would require the rendering of *all* $O(n^2)$ shadow volume polygons and would overwrite the depth buffer.

Other hardware-accelerated approaches are possible, including using a projection shadow algorithm to cast appropriate shadows onto the near plane. However, we have found that when the shadow volume is regenerated from a shadow map, a simple and fast software capping algorithm is

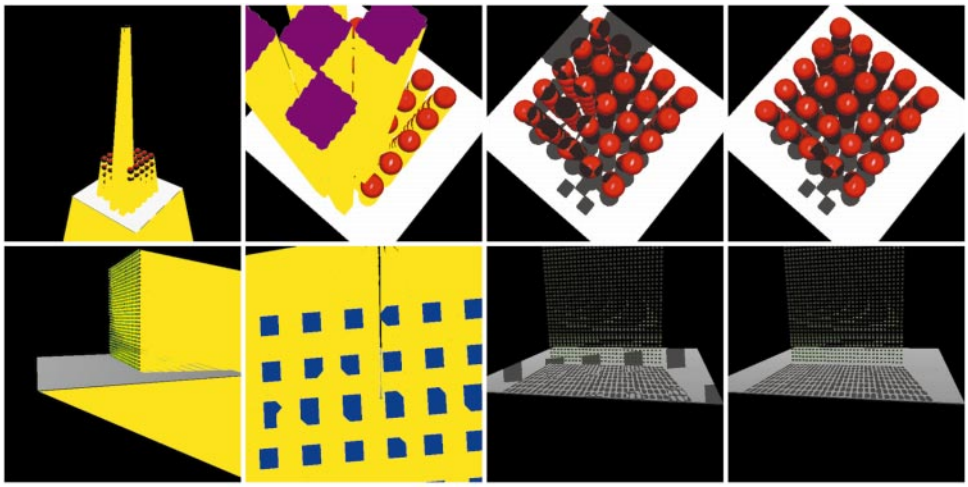

Fig. 7. Clipping the shadow volume boundary at the near plane of the eye can reverse the parity of the shadow test locally. This can happen with occluders behind the eye (top) or when looking towards the light source (bottom). From left to right: side views with the shadow volume drawn using visible polygons, views where the near plane facet of the viewing volume intersects the shadow volume, the same view but with the shadows rendered without capping (and with inversion artifacts), and finally the corrected images rendered with capping.

feasible, since we can take advantage of the coherent structure of the shadow volume polygons.

Our software solution involves the following steps:

(1) Transform the plane equation of the near clipping plane of the eye into light device coordinates.

(2) Transform the rectangular near plane facet of the view frustum into light device coordinates.

(3) Scan and convert the transformed near plane facet over the shadow map and test all covered shadow map samples against the transformed near plane equation.

(4) If all samples tested are on the eye side of the near plane, the parity test should be inverted globally, but no cap polygons need to be drawn.

(5) If all samples tested are opposite the eye side of the near plane, or if the near plane facet is not visible in the light view, no cap is needed.

(6) If there are samples on both sides of the near plane, cap polygons need to be generated.

Using either of the simple reconstruction processes presented in Section 4, cap polygon vertices can be reconstructed without referring to the actual silhouette polygons. There are $32 = 2 \times 2^4$ possible cases combining sample to near plane classification and the reconstruction split in each cell. We

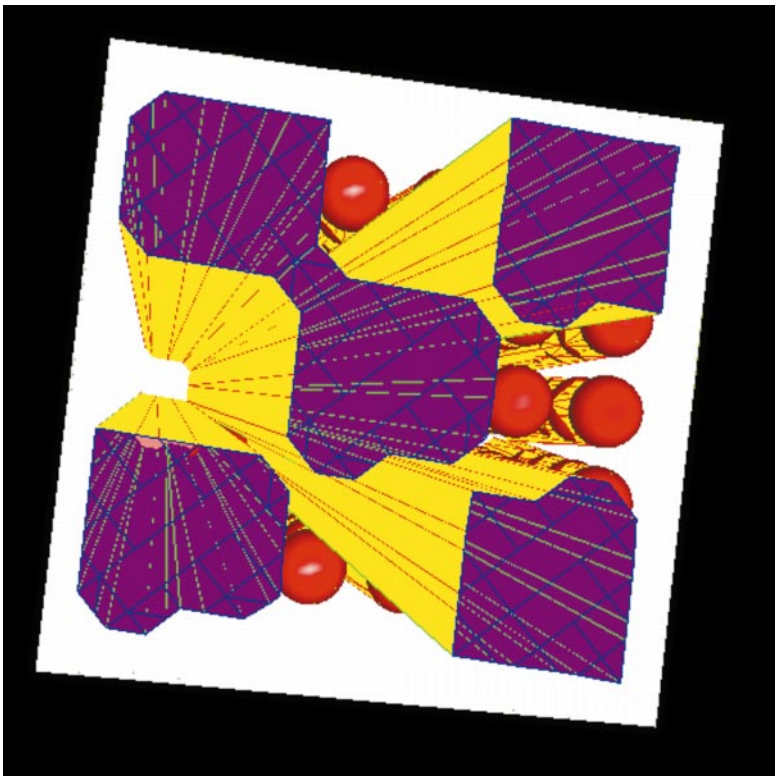

Fig. 8. Detail of a cap generated using a very low resolution shadow map. Rows of depth values with the same classification relative to the eye-view near plane are run-encoded and rendered as single rectangles. Cells with mixed classifications are clipped against the shadow volume reconstruction polygons to generate appropriate fragments to complete the cap.

can analyze each case and generate the appropriate cap polygons one cell at a time.

Once generated, the cap polygons are simply rendered with the rest of the shadow polygons to toggle the parity where needed.

A detail of a cap is shown in Figure 8. We do not currently attempt to trace and tesselate the contours of the cap to improve coherence, although this could be done. We do render runs with single polygons, after run-length encoding the results of the capping test. If contour tracing is used, holes in caps could simply be rendered as additional back-facing cap polygons; this would toggle the parity correctly and would also work with multiple shadow volumes (Section 8.2) using Crow's algorithm. As shown later, the computational and rendering cost of software capping is negligible.

Since the hardware is clipping the uncapped shadow volume boundary at the near plane of the eye, we do not have to do it in software. However, the implementation must avoid having the cap clipped away. We translate the device coordinate system itself in z to avoid this problem and to avoid having to shift the cap polygons away from their true positions.

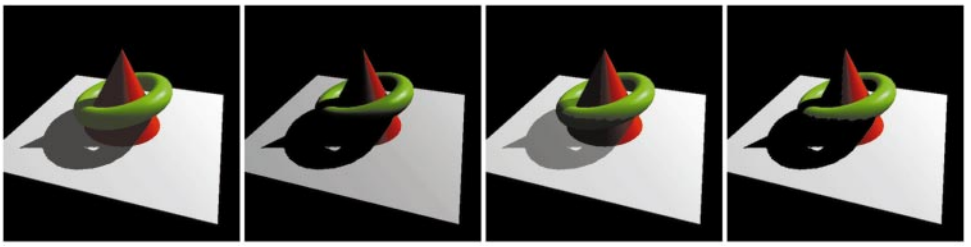

Fig. 9. Shadow-rendering modes, left to right: consistent ambient shadows; consistent (zero ambient illumination) black shadows; inconsistent composited shadows; and inconsistent (nonzero ambient illumination) black shadows.

To avoid numerical problems with the intersection of rays from the light source with the eye-view near plane facet, the intersection points (vertices of the cap polygons) are computed in homogeneous coordinates, without an explicit division. If the eye-view near plane facet is edge-on in the light source view, the cap vertices will then be mapped towards infinity in the eye view, as appropriate. The cap vertices need to be computed with reasonably high accuracy to avoid doubling up of edge pixels where the cap meets the clipped shadow volume. We have found this happens occasionally, but these salt-and-pepper artifacts are nearly invisible and are certainly preferable to gross inversion.

7. SHADOW-RENDERING MODES

Once the stencil bit has been set correctly, shadows can be rendered in a number of different modes:

Ambient shadows: If the scene renderer does not use the stencil buffer, the scene can be rerendered using only ambient illumination, masked to modify only pixels in shadow.

Black shadows: A single black polygon can be drawn over the entire scene to blacken all pixels in shadow.

Composited shadows: A semitransparent black polygon can be drawn over the entire scene to darken all pixels in shadow.

However, with use of the composited shadow-rendering mode, incorrect illumination, including highlights, will be present in the shadow. If the ambient illumination does not match the color drawn in shadow, artifacts can result along silhouette edges; look closely at Figure 9. If black shadows are used, the ambient illumination should also be zero.

The basic hybrid shadow algorithm can only render one light source at a time. Generally, multiple light sources should be rendered as follows:

(1) Render the eye view using only ambient illumination and load the image into the accumulation buffer.

(2) For each light source:
   (a) Render the eye view illuminated with the current light source and a zero ambient term.
   (b) Disable writing to the color and depth buffers and set the depth test to "<".
   (c) Clear the stencil buffer and render the current shadow volume, inverting the stencil for each fragment.
   (d) Enable drawing to the color buffer and turn off the depth test.
   (e) Draw a black polygon where the stencil is *set*.
   (f) Add the color buffer to the accumulation buffer.

(3) Recover the accumulation buffer without scaling it.

This algorithm works even if the base renderer uses the stencil buffer.

If the color precision is adequate, the stencil buffer is not used by the base renderer, and the base renderer uses only standard hidden surface removal, then compositing can be used instead of the accumulation buffer:

(1) Render the eye view using only ambient illumination.

(2) For each light source:
   (a) Disable writing to the color and depth buffers and set the depth test to "<".
   (b) Clear the stencil buffer and render the current shadow volume, toggling the stencil.
   (c) Set the depth test to "=" and enable compositing to the color buffer. Set the source and destination compositing factors to one and one.
   (d) Render the eye view illuminated with the current light source and a zero ambient term only where the stencil is *zero*.

The "=" depth test ensures that only the frontmost surface's illumination will be composited. Use of this depth test assumes that changing the lighting model does not change the depth of any pixel in the scene.

In either case, for N light sources $N + 1$ eye-view renderings of the scene are needed, in addition to at least N light-view shadow map renderings.

8. EXTENSIONS

The hybrid algorithm can be extended in a number of ways by reconstructing better shadow volumes and/or by combining shadow volumes with Crow's shadow volume union algorithm.

8.1 Vectorization

In theory, the geometric representation of the shadow volume means that a more sophisticated reconstruction strategy could produce shadows that are an *improvement* over the shadows produced by the shadow map algorithm; currently our algorithm merely produces *equivalent* shadows.

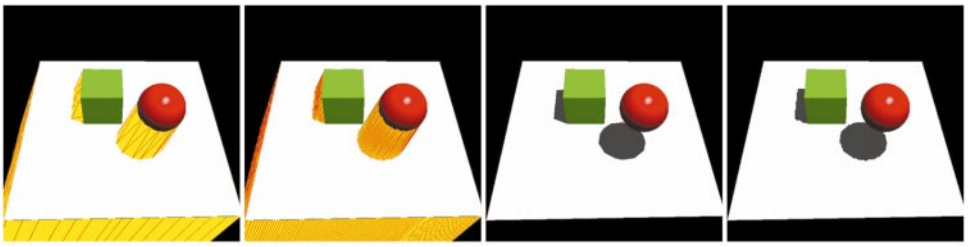

Fig. 10. A vectorized shadow volume, and the resulting shadows, are shown on the left for a low-resolution shadow map. On the right are the nonvectorized shadow volume and shadows generated from the same shadow map.

For instance, subpixel edge information can often be recovered by segmenting and vectorizing edge contours [Bloomenthal 1983]. We have experimented with this; some preliminary results are shown in Figure 10. Vectorization improves both quality and performance, since it permits replacing the numerous small polygons that are generated by the simpler reconstruction techniques with a smaller number of larger polygons, which can be more efficiently rasterized and produce more coherent shadows. The main difficulties are determining good shadow volume facets that are consistent with the depth samples in the shadow map, dealing consistently with nonvectorizable "blobs" and edge conditions, and implementing efficient capping of the resulting shadow volume.

8.2 Multiple Shadow Volumes

Shadow volumes could be reconstructed from shadow maps on a per-object basis, which would permit interactive manipulation of parts of the scene without having to recompute global shadow maps. This could also be done using, for instance, spatial decomposition or attachment hierarchies.

If object-space precision shadow volumes are available for some objects but not others, render only the objects that cannot generate object-precision shadow volumes into the light view. Then combine the shadow volume(s) reconstructed from the light view depth buffer with the object-precision shadow volumes using Crow's algorithm.

Since per-object shadow volumes can overlap and nest, counting would be needed in the stencil buffer, or the shadow volumes would have to be rendered one at a time (blackening parts of the scene after each). If the maximum amount of nesting is known for each per-object shadow volume, the number of bits needed for the total nesting count can be computed and the shadows "batched" to avoid overflow.

Certain algorithms may require the ability to incrementally extend shadow volumes. For instance, in hardware-assisted global illumination [Diefenbach and Balder 1997; Keller 1997], light may be reflected off a mirror by creating a virtual light source. In this case, the shadow map of the original light source is also reflected, but the reflected light may be blocked by additional occluders.

There are two ways to extend the reflected shadow map to account for these extra occluders. At the level of depth maps, the extra occluders can be rendered into a new depth map and merged with the reflected shadow map and the shadow volume polygons regenerated from the merged shadow map. Alternatively, a secondary shadow volume can be generated from a light view containing only the occluders. This new shadow volume will overlap the reflected shadow volume, and so the scene must be rendered using Crow's algorithm.

8.3 Shadow Edge Antialiasing

Shadow edges could be antialiased without any need to recapture the shadow map by jittering (with a small shear transformation) the back-projection of the already reconstructed shadow volume polygons. The magnitude of the jitter would be on the same order as the bias and the xy resolution error. An n-rooks sampling pattern for the jitter should be used to get the maximum number of grey levels for displacements of the edges.

By using multiple stencil bits, the scene would not have to be rerendered for each pass; only the shadow polygons themselves would have to be rerendered. After n shadow volume rendering passes into n separate bits of stencil, n renderings of a full-screen semitransparent polygon could be used to replace the original color in the frame buffer with an appropriate amount of black.

9. RESULTS AND TIMING

All values presented here were estimated using an average of the timings for 1000 random rotations of the test scenes in Figure 11, whose triangle counts are given in Table II. The ANSI C/OpenGL/GLUT test implementation was run on two machines: an SGI Octane with a 175MHz MIPS R10000 processor and the SI graphics engine (without texture mapping support), and an SGI O2 with a 225MHz MIPS R10000.

The objects shown above the ground planes in Figure 11 are reflected and duplicated below the ground plane for the purposes of this test, so all views have approximately the same visual complexity. Table II gives the total number of triangles in each scene, and the average number of polygons in the corresponding shadow volumes. Only a single-view shadow map was used.

Depth precision on both machines was 24 bits. Two spatial resolutions (256×256 and 512×512) of both the image and the shadow map were tested, to determine scaling of the algorithm with resolution.

Edges were flagged with a modified Canny edge detector. The template-based reconstruction scheme shown in Figure 5 was used, with the max-diagonal cut splitting rule for quadrilaterals.

The test program is self-contained and may be downloaded from

```
http://www.cgl.uwaterloo.ca/Projects/rendering/Papers/
```

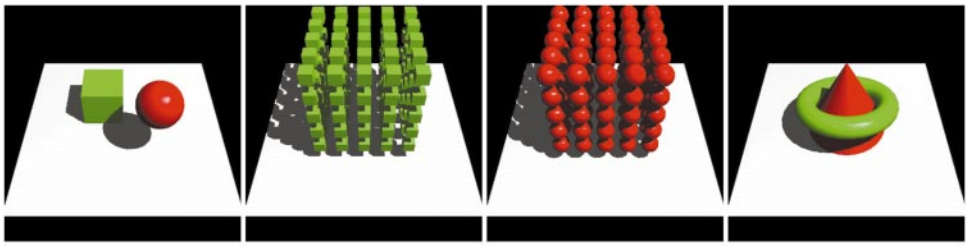

Fig. 11. Test scenes A, B, C, and D, left to right.

9.1 Timings

Per-phase and overall timings are given in Table I. For the per-phase timings given, ambient illumination was zero and black shadows were rendered using a single black stenciled rectangle. Overall rates for ambient shadows (which require an additional rendering of the scene lit only with ambient illumination) and unshadowed scenes are also given for comparison. Note that the primary shadow-related computational costs are edge detection and shadow volume rendering.

The edge-detection cost is primarily data access; increasing the complexity of the edge detector does not change this timing significantly. Our edge-detector implementation is entirely in software. The first pass of the edge detector could theoretically be implemented in hardware via OpenGL, but we did not have access to a machine for which our standard-compliant OpenGL implementation both worked and was faster than a straightforward software implementation.

Note as well the long time for depth buffer read-back on the O2. This is mysterious, as the O2 has a unified memory architecture; even with a format conversion the readback should be no more costly than the edge detection, but it is by a significant factor. The timings could probably be improved significantly on the O2 by tuning the driver.

The shadow volume-rendering cost is primarily vertex-transformation limited. Our preliminary results indicate that vectorization will decrease the time to render the shadow volume by an order of magnitude, while only slightly increasing reconstruction time. Capping, however, would be significantly more complex.

The shadowing cost is, as expected, relatively independent of scene complexity, but is approximately linearly dependent on the resolution of the shadow map. Doubling the shadow map resolution (quadrupling the number of depth samples) less than doubles the number of shadow volume polygons. Note as well that the primary rendering time for scene C was much longer than any of the other scenes, but the total time spent reconstructing and rendering the shadow volume was approximately equivalent to scene B, and in fact slightly less. Therefore, this technique should be especially effective for complex scenes containing curved surfaces. For simple scenes, unfortunately, the resolution-dependent overhead swamps the base rendering time of the scene itself.

Table I. Comparative Timings for Various Phases of the Algorithm (unless otherwise noted, units are milliseconds; rates do not include swap synchronization latency)

Resolution	256 × 256							
Machine	Octane SI				O2			
Scene	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
Light view render	4.0	5.5	100.1	8.3	6.2	9.7	178.3	14.7
Depth read	1.4	1.4	1.5	1.4	57.0	57.2	57.2	57.1
Edge detect	17.3	18.9	19.2	17.5	15.4	16.0	16.4	15.3
Reconstruct	2.8	10.5	10.1	3.0	10.0	46.6	44.0	10.4
Eye view render	7.2	12.7	246.6	16.0	9.1	15.9	275.8	24.1
Volume render	34.4	96.3	78.0	35.5	10.3	35.0	33.0	11.1
Volume cap	0.38	0.42	0.55	0.46	0.33	0.53	0.82	0.49
Shadow render	0.8	0.8	0.8	0.8	1.2	1.1	1.1	1.1
Total time	68.3	146.5	456.9	83.0	109.5	182.0	606.6	134.3
Unshadowed (Hz)	138.9	78.7	4.1	62.5	110.0	62.9	3.6	41.5
Shadowed, Black (Hz	14.6	6.8	2.2	12.1	9.1	5.5	1.6	7.4
Shadowed, Ambient (Hz)	12.9	6.1	1.2	9.5	8.4	5.0	1.0	6.1
Resolution	512 × 512							
Machine	Octane SI				O2			
Scene	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
Light view render	6.1	7.9	101.7	10.0	9.2	12.3	184.8	18.1
Depth read	4.5	4.5	4.3	4.5	209.5	209.2	209.9	209.6
Edge detect	73.2	76.5	76.8	73.7	71.8	74.4	75.2	72.1
Reconstruct	5.5	22.8	20.9	6.2	18.4	108.7	98.4	21.3
Eye view render	10.9	16.2	248.3	18.9	11.2	17.4	280.2	26.8
Volume render	66.6	199.1	155.0	69.0	28.4	101.0	82.5	30.8
Volume cap	0.66	0.79	1.22	1.00	0.65	1.29	1.65	1.08
Shadow render	3.3	3.3	2.9	2.9	4.1	4.1	4.2	4.1
Total time	170.8	331.1	611.1	186.2	353.3	528.4	936.9	383.9
Unshadowed (Hz)	91.7	61.7	4.0	52.9	89.3	57.5	3.6	37.3
Shadowed, Black (Hz	5.9	3.0	1.6	5.4	2.8	1.9	1.1	2.6
Shadowed, Ambient (Hz)	5.6	2.9	0.98	4.8	2.8	1.9	0.78	2.5

Table II. Complexity of Test Scenes and Average Reconstructed Shadow Volumes in Triangles (a shadow volume complexity ratio of 2 indicates linear growth)

Scene:	**A**	**B**	**C**	**D**
Scene Complexity	3,756	3,132	450,132	7,332
Average Shadow Volume Complexity (256 × 256)	2,322	8,707	8,399	2,402
Average Shadow Volume Complexity (512 × 512)	3,544	16,456	15,093	4,021
Ratios of Shadow Volume Complexities	1.53	1.89	1.80	1.67

10. CONCLUSIONS

We have presented a hardware-assisted shadow algorithm that regenerates a polygonal shadow volume from depth buffer information.

This algorithm requires a one-bit stencil buffer, but does not require texture mapping or access to a polygonal representation of the scene. In fact, the algorithm imposes no constraints on how the hardware is used to

generate the image, other than the requirement that correct depth values be generated.

ACKNOWLEDGMENTS

The help, advice, and tolerance of members of the Computer Graphics Lab at the University of Waterloo, especially Glenn Evans, Kirk Haller, Ion Vasilian, and Peter Harwood, is appreciated.

REFERENCES

ATHERTON, P., WEILER, K., AND GREENBERG, D. 1978. Polygon shadow generation. In *Proceedings of the Conference on SIGGRAPH*, ACM Press, New York, NY, 275–281.

BEHRENS, U. AND RATERING, R. 1998. Adding shadows to a texture-based volume renderer. In *Proceedings of the 1998 IEEE Symposium on Volume Visualization* (VVS '98, Research Triangle Park, NC, Oct. 19–20, 1998), A. Kaufman, R. Yagel, and W. E. Lorensen, Eds. ACM Press, New York, NY, 39–46.

BERGERON, P. 1985. Shadow volumes for non-planar polygons: Extended abstract. In *Proceedings of the Conference on Graphics Interface* (May 1985), 417–418.

BERGERON, P. 1986. A general version of Crow's shadow volumes. *IEEE Comput. Graph. Appl. 6*, 9 (Sept.), 17–28.

BLINN, J. 1988. Me and my (fake) shadow. *IEEE Comput. Graph. Appl. 8*, 1 (Jan. 1988), 82–86.

BLOOMENTHAL, J. 1983. Edge inference with applications to antialiasing. In *Proceedings of the Conference on SIGGRAPH* (July 1983), ACM Press, New York, NY, 157–162.

BOUKNIGHT, W. AND KELLY, K. 1970. An algorithm for producing half-tone computer graphics presentations with shadows and movable light sources. In *Proceedings of on AFIPS Spring Joint Computer Conference*, AFIPS Press, Arlington, VA, 1–10.

CANNY, J. 1986. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 6 (Nov. 1986), 679–698.

CHEN, S. E. AND WILLIAMS, L. 1993. View interpolation for image synthesis. In *Proceedings of the ACM Conference on Computer Graphics* (SIGGRAPH '93, Anaheim, CA, Aug. 1–6, 1993), M. C. Whitton, Ed. ACM Press, New York, NY, 279–288.

CHIN, N. AND FEINER, S. 1989. Near real-time shadow generation using BSP trees. *SIGGRAPH Comput. Graph. 23*, 3 (July 1989), 99–106.

CHRYSANTHOU, Y. AND SLATER, M. 1995. Shadow volume BSP trees for computation of shadows in dynamic scenes. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics* (Monterey, CA, Apr. 9–12, 1995), M. Zyda, Ed. ACM Press, New York, NY, 45–50.

CROW, F. 1977. Shadow algorithms for computer graphics. In *Proceedings of the Conference on SIGGRAPH* (July 1977), ACM Press, New York, NY, 242–248.

DIEFENBACH, P. J. AND BADLER, N. I. 1997. Multi-pass pipeline rendering: Realism for dynamic environments. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics* (SI3D '97, Providence, RI, Apr. 27–30, 1997), A. van Dam, Ed. ACM Press, New York, NY, 59–70.

DRETTAKIS, G. AND FIUME, E. 1994. A fast shadow algorithm for area light sources using backprojection. In *Proceedings of the ACM Conference on Computer Graphics* (SIGGRAPH '94, Orlando, FL, July 24–29, 1994), D. Schweitzer, A. Glassner, and M. Keeler, Eds. ACM Press, New York, NY, 223–230.

ELBER, G. AND COHEN, E. 1996. Adaptive isocurve-based rendering for freeform surfaces. *ACM Trans. Graph. 15*, 3, 249–263.

FUCHS, H., GOLDFEATHER, J., HULTQUIST, J. P., SPACH, S., AUSTIN, J. D., BROOKS JR., F. P., EYLES, J. G., AND POULTON, J. 1986. Fast spheres, shadows, textures, transparencies, and image enhancements in pixel-planes. In *Advances in Computer Graphics I*, G Enderle, M Grave, and F Lillehagen, Eds. Eurographic Seminars, Tutorials and Perspectives in Computer Graphics. Springer-Verlag, New York, NY, 169–187.

HAINES, E. AND GREENBERG, D. 1986. The light buffer: A shadow-testing accelerator. *IEEE Comput. Graph. Appl. 6*, 9 (Sept.), 6–16.

HEIDRICH, W. 1999. High-quality shading and lighting for hardware-accelerated rendering. Ph.D. Dissertation. University of Erlangen, Erlangen, Germany.

JANSEN, F. AND VAN DER ZALM, A. 1991. A shadow algorithm for CSG. *Comput. Graph. 15*, 2, 237–247.

KELLER, A. 1997. Instant radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (SIGGRAPH '97, Los Angeles, CA, Aug. 3–8), G. S. Owen, T. Whitted, and B. Mones-Hattal, Eds. ACM Press/Addison-Wesley Publ. Co., New York, NY, 49–56.

KILGARD, M. 1997. OpenGL-based real-time shadows. http://reality.sgi.com/mjk_asd/tips/rts/.

MARR, D. AND HILDRETH, E. 1980. Theory of edge detection. *Proc. Roy. Soc. London B207*, 187–217.

REEVES, W. T., SALESIN, D. H., AND COOK, R. L. 1987. Rendering antialiased shadows with depth maps. *SIGGRAPH Comput. Graph. 21*, 4 (July 1987), 283–291.

ROSSIGNAC, J. AND REQUICHA, A. 1986. Depth-buffering display techniques for constructive solid geometry. *IEEE Comput. Graph. Appl. 6*, 9 (Sept.), 29–39.

SEGAL, M., KOROBKIN, C., VAN WIDENFELT, R., FORAN, J., AND HAEBERLI, P. 1992. Fast shadows and lighting effects using texture mapping. *SIGGRAPH Comput. Graph. 26*, 2 (July), 249–252.

SOLER, C. AND SILLION, F. X. 1998. Fast calculation of soft shadow textures using convolution. In *Proceedings of the 25th Annual Conference on Computer Graphics* (SIGGRAPH '98, Orlando, FL, July 19–24, 1998), S. Cunningham, W. Bransford, and M. F. Cohen, Eds. ACM Press, New York, NY, 321–332.

STEWART, A. J. AND GHALI, S. 1994. Fast computation of shadow boundaries using spatial coherence and backprojections. In *Proceedings of the ACM Conference on Computer Graphics* (SIGGRAPH '94, Orlando, FL, July 24–29, 1994), D. Schweitzer, A. Glassner, and M. Keeler, Eds. ACM Press, New York, NY, 231–238.

WANGER, L. 1992. The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In *Proceedings of the 1992 ACM SIGGRAPH Symposium on Interactive 3D graphics* (Cambridge, MA, Mar. 29–Apr. 1), M. Levoy, E. E. Catmull, and D. Zeltzer, Eds. ACM Press, New York, NY, 39–42.

WEILER, K. AND ATHERTON, P. 1988. Hidden surface removal using polygon area sorting. In *Tutorial: Computer Graphics; Image Synthesis*, K. I. Joy, C. W. Grant, N. L. Max, and L. Hatfield, Eds, Computer Science Press, Inc., New York, NY, 209–217.

WESTERMANN, R., SOMMER, O., AND ERTL, T. 1999. Decoupling polygon rendering from geometry using rasterization hardware. In *Proceedings of the Tenth Eurographics Workshop on Rendering* (June 1999),

WIEGAND, T. 1996. Interactive rendering of CSG models. *Comput. Graph. Forum 15*, 4, 249–261.

WOO, A. 1993. Efficient shadow computations in ray tracing. *IEEE Comput. Graph. Appl. 13*, 5 (Sept.), 79–83.

WOO, A., POULIN, P., AND FOURNIER, A. 1990. A survey of shadow algorithms. *IEEE Comput. Graph. Appl. 10*, 6, 13–32.

Received: March 1998; revised: October 1999; accepted: January 2000