# Forward Shadow Mapping

Hansong Zhang

TR98-003
Department of Computer Science
University of North Carolina at Chapel Hill
zhangh@cs.unc.edu

**Abstract:**
Forward shadow mapping is a new approach to real-time shadow generation. The traditional shadow map algorithm maps the pixels in the eye's view *backward* into the depth buffers of light sources (i.e. shadow maps), which is similiar to and often implemented as an extension to texture mapping. Our algorithm reverses this process by using 3-D image warping techniques to transform shadow map pixels *forward* into the eye's view to directly indicate which pixels are lit; it does not interfere with normal texture mapping and easily supports anti-aliased shadow edges and projective textures. Access to shadow maps and projective textures is in pixel-sequential order. This algorithm has advantages when speed of texture mapping becomes the performance bottleneck, which is often the case in visual simulation and game applications.
**CR Categories and Subject Descriptors:** I.3.5 [**Computer Graphics**]: Picture/Image Generation; I.3.7 [**Computer Graphics**]: Three Dimensional Graphics and Realism - *color, shading, shadowing, and texture*
**Key Words and Phrases:** Shadow, lighting, image based rendering, 3D image warping, texture mapping

## 1 Introduction

Shadow generation has been a subject of active research in computer graphics. Surfaces in a scene not lit by the light source are in shadow. Shadows can significantly enhance the realism of synthesized images and often lead to better understanding of spatial relationships between objects.

The task of shadow determination is to tell whether portions of a scene visible from the eye are visible from the light sources. Areas seen by both the eye and a light are illuminated, while those visible to the eye but not the light lie in shadow with respect to that light. Multiple light sources can be handled individually in the same way. Obviously, shadow generation is visibility determination from multiple view points.

Depth buffering is a popular image-space approach to resolving visibility. A shadow algorithm can make use of the depth buffers of the light sources to de-

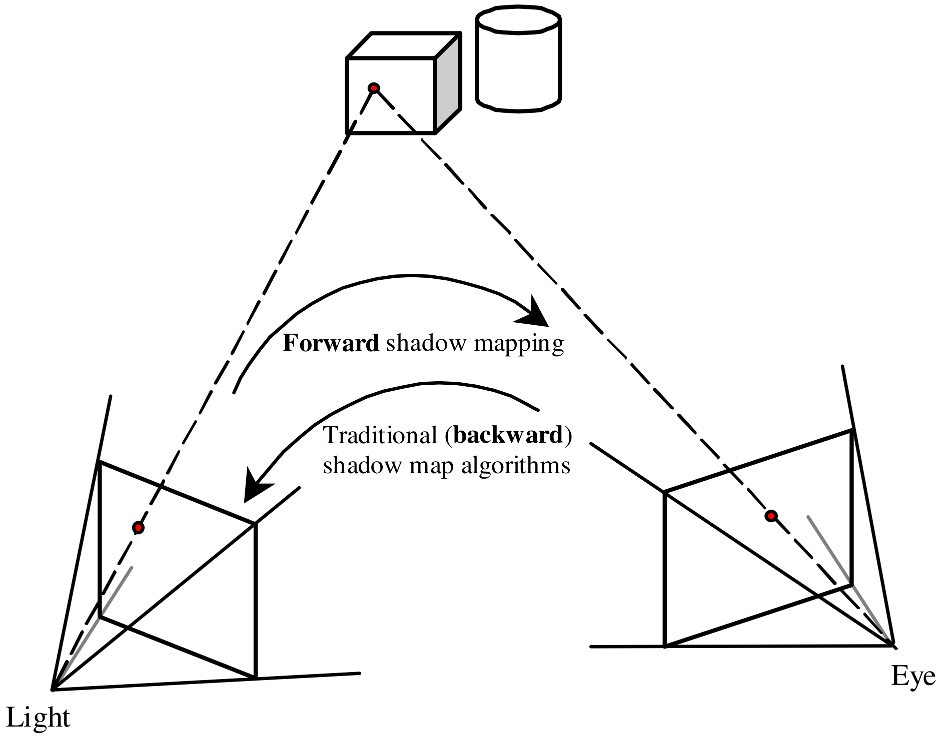


Figure 1: The basic concept of forward shadow mapping. Pixels in the light's depth buffer (the shadow map) are transformed, using 3-D image warping technqiues, into the eye's view to directly show which pixels are illuminated. It reverses the traditional shadow map algorithm which is a projection into light's view followed by random look-ups into the shadow map.

termine whether a 3-D point is visible to the lights; these depth buffers are also called *depth maps* or most commonly *shadow maps*.

This paper presents a new algorithm for shadow generation. Based on the *forward* 3-D warping of shadow maps into the eye's view, the algorithm reverses the mapping process of traditional shadow map algorithms (discussed in the next section) which often share resources with texture mapping. Our algorithm show its advantages when graphics performance is bound by texture mapping; it also easily supports anti-aliased shadow edges and integrates projective textures into the shadow generation process.

# 2 Related Work

Shadow map algorithms originated in [Wil78], which proposed an algorithm that renders the scene in two passes with depth-buffering. The first pass is from the light sources and generates the shadow maps. The scene is then rendered from the eye's view. 3-D points visible in the eye's view (corresponding to non-background pixels in the eye's image) are projected into the shadow maps; depth comparisons then show whether they are also visible to the lights. The image-space nature of this algorithm makes it simple to implement, applicable to any kind of primitives, easy to accelerate with hardware, and efficient enough to be used with complex scenes.

To solve the aliasing problem of the original shadow map algorithm, [HN85] proposed the use of per-pixel object identifiers and [RSC87] introduced percentage closer filtering. These improvements alleviate aliasing and produce soft shadow edges. [SKW$^+$92] recognizes that projective textures and shadow maps are both a simple extension to perspective-correct texture mapping; as a result, graphics systems such as the RealityEngine$^2$[Ake93] have sucessfully supported real-time shadows. The implementation of shadow maps on the PixelFlow [EMP$^+$97], a very different architecture, is also a variation of texture mapping. Extensive sharing of texture mapping hardware makes shadow maps easier and less expensive to implement. However, speed of texture mapping is often the limiting factor in modern graphics applications, and sharing of critical resources can lead to severe system imbalance.

Many other shadow algorithms have been developed. Compared to depth-buffer-based algorithms, they have limitations in generality, scalability and speed. Scanline shadow algorithms[App68, BK70] requires significant pre-processing and are thus not suitable for moving objects and lights. Polygon subdivision[AWG78, NN85], shadow volume algorithms [Cro77, BB84, TN85, Ber86, Max86, FGH$^+$85, Hei91], and the BSP-tree shadow volume method[CF89, CF92] are restricted to polygonal models and inefficient for scenes with a large number of polygons. Polygon subdivision algorithms algorithm is also difficult to implement robustly. For a survey of some of the algorithms see [WPF90]. Ray-tracing [Whi79, Kay79] and radiosity algorithms [GTGB84, CG85] produce high-quality shadows, but are still too expensive for real-time purposes.

Another area of research to which our work is related is image based rendering (IBR). The goal of IBR is to generate an image for a view (the *destination* image) directly from existing images for other views (*source* images), without an explicit geometric representation of the scene. The basic operation is the transformation of source image pixels to the destination image. [CW93] proposed a view interpolation method based on pre-computed per-pixel optical flow information. [MB95] points out that with per-pixel disparity values (which actually make pixels 3-D entities), source pixels can be re-projected directly into the destination image. This technique is also called 3-D image warping[MMB97], which is the term we use in this paper. An image with per-pixel disparity or Z value is often called a *depth image*.[1] [Zha97] discusses 3-D image warp using the OpenGL[WND85] graphics model.

Another way of generating a destination image from source images is to first reconstruct geometry from source images, and then render them for the destination view[DSV97, SDB97]. For scenes with large near-planar structures, this can be faster than per-pixel 3-D warping. On the other hand, it requires substantial computation in reconstruction, and thus is not applicable when source images change frequently.

[1]It is often unclear whether a "depth" image has disparity or Z values. In this paper, depth means Z. This is consistent with the notion of a depth buffer.

Figure 2: The process of forward shadow mapping

# 3 Forward Shadow Mapping

The forward shadow mapping algorithm applies 3-D image warping techniques to the problem of shadow generation. This section describes in detail the basic algorithm and an architecture-specific variation. The next section gives a comparison between forward shadow mapping and traditional shadow map algorithms.

## 3.1 Motivation

Implementations of shadow maps as an extension of texture mapping are very natural, given the fact that shadow and texture mapping are similiar operations and can share similiar hardware resources. However, this makes the texture hardware often the bottleneck of system performance. In many applications, speed of texturing has become the bottleneck even without shadow maps. The use of a shadow map, which is equivalent to applying a texture map to all geometry in the scene (or all visible geometry, for deferred shading architectures[DWS+88]) aggravates the situation. Shadow maps also compete with ordinary textures in the scene for texture memory. It is often desirable to use a *projective texture*[McR96] (an image projected by the light) to model a light's variation in intensities, in order to achieve effects like spotlights with gradual fall-off or lights shining through stained glass; this means more textures for the scene. Another problem is that applying more than one texture to the same polygon is a feature not commonly available in hardware and often has to be emulated using multi-pass rendering.

Still more shadow maps are required if we want to support shadow casting in wider angle than a single shadow map can handle. Real-world point lights or cylindrical lights can be modeled by multiple shadow maps covering the surface of a sphere or cylinder—or, in other words, modeled by multiple lights with limited fields of view. With so many shadow maps, the cost of multi-pass rendering soon gets out of control.

It is still true that we should maximize hardware sharing to minimize the amount of hardware (if any) specific to shadow generation. But for texture-rate bound applications, this sharing should not be of the texture mapping hardware, in order to maintain system balance.

The major benefit of the forward shadow mapping algorithm is the removal of the burden of shadow generation from the texture mapping resources. Our goal is to provide an algorithm that efficiently handles real-time, anti-aliased shadowing of textured scenes with wide-angle projective-texture-mapped lights.

## 3.2 Algorithm Overview

For brevity, we will use "a lit pixel" to mean a pixel that corresponds to a 3-D point in the scene that is lit and seen by the eye.

Shadow map pixels are samples of the portions of scene geometry visible to the light. We use 3-D image warping to transform shadow map pixels into the eye's view, so that we know where the illuminated portions fall in the eye's image and what their depths are in the eye space. Shadow map pixels are accessed in sequential order. A depth comparison between a warped pixel and the corresponding one in the eye's image decides whether the warped pixel is visible to the eye. If it is, the eye pixel is in the lit area of the eye's image. Note that this assumes the scene has already been rendered from the eye so that the eye's depth buffer is ready to be used.

If we have a projective texture we can access it in the same sequential order as shadow map pixels, and the warped pixels can then be assigned light intensities. Since memory access is all in pixel-sequential order, these effects are cheap to support.

The above is a high-level description of forward shadow mapping. Depending on what one does once a pixel is determined to be lit (or in shadow), many variations of the base algorithm are possible.

For deferred shading (DS) architectures, a light mask can be stored per-pixel to switch lights on/off, so warping can be scheduled before shading to set the masks. Actually, forward shadow mapping is DS-oriented since it requires that the eye's depth buffer be available before shadow computation (part of shading).

For architectures that shade before scan-conversion (SBSC), DS can be simulated to an extent with multi-pass rendering from the eye. This is inefficient for a large scene. Below we show a fast algorithm for SBSC architectures that has only one eye pass. It also produces anti-aliased shadow edges. 3-D warping (combined with optional projective textures) produces a modulation image, which is used to scale color values for each framebuffer pixel separately. This is an approximation to real shading, but in practice produces realistic-looking images. Modulation has been used for spotlight effects in [McR96].

It is also possible to draw modulation pixels directly into the framebuffer as soon as a shadow map pixel is warped. But, as will be discussed later, antialiasing can be easily done if we have the whole modulation image.

Steps of the algorithm are listed below. For brevity, the algorithm is given for one light source; extension to multiple lights is straightforward. Figure 2 illustrates how it works on a model by showing the intermediate and final images. Comments on the algorithm are given in the following subsections.

1. Render the scene from the light's view; get the resulting depth buffer as the shadow map.
2. Render the scene normally from the eye's view.
3. Clear the modulation image with the dimming factor for areas not lit by any light.
4. Each shadow map pixel is warped to the eye's view, carrying the corresponding intensities from the projective texture; for each pixel in the eye's image covered by the warped pixel, compare the transformed depth with the depth in the eye's depth buffer. If they are approximately equal, then the the light intensities are accumulated in the modulation image.
5. Blur the modulation image for antialiased shadow edges.
6. Apply the modulation image. It modulates the normally rendered, eye's image for shadows and lighting effects defined by the projective texture.

## 3.3 Transforming the Shadow Map

We use existing 3-D warping techniques [MB95, Zha97] to warp the shadow map pixels to the eye's view. These techniques take advantage of the incremental nature of pixel-by-pixel computation, which results in much less operations than full (non-incremental) 3-D warping per pixel. The method described in [Zha97] is especially suited to our purpose since it deals directly with z-values from the depth buffer and uses the same coordinate systems as in conventional 3-D graphics.

Due to the fact that pixels are not real points, and that the eye and the light view generally have different sampling rates, the area a warped shadow map pixel covers in the eye's image has to be computed. Detailed discussion of this computation is beyond the scope of this paper, but it should be noted that the area is often estimated since it is too expensive to get the exact answer; it is often overestimated to avoid cracks between warped pixels. For interactive applications, pixels in the area are often treated as having the same color and depth.

We do not use mesh-based IBR methods (see section 2) because we want to support dynamic scenes and moving lights, which potentially changes shadow maps for each frame. The time required for processing depth values makes these methods impractical for us.

Note that the forward shadow mapping algorithm is independent of the 3-D warping methods it uses. Any warping algorithm can be plugged in without changing the shadow algorithm.

## 3.4 Updating the Modulation Image

Below we will use *pixel* to refer to both the 2-D location on an image and the 3-D point(s) in the scene it corresponds to. They are easy to distinguish in context.

For each pixel in the eye's image covered by the warped pixel, we have to decide whether the corresponding shadow map pixel is visible to the eye. This is done by comparing the warped depth to depths from the eye's depth buffer. If pixels are infinitely small and there is no numerical error, the warped depth should be equal to the depth at an eye pixel (meaning the two pixels are the projections of the same point in the scene) for the shadow map pixel to be visible—which in turn means the eye pixel is lit.

In practice this exact relationship will not happen, requiring error tolerance, $\epsilon$. We use an approximately-equal operator for depth comparison with an error bound. The error bound is an empirical value based on the resolution of the images and depth values. We could also use a *less-than* operator as in other shadow map algorithms, offsetting the warped depth by $\epsilon$ before the comparison. But the more strict *approximately-equal* operator also helps to trim away the outside part of the estimated coverage of a warped pixel that sits across the silhouette of a visible surface, if depth changes more than $\epsilon$ at the silhouette.

When an eye pixel is determined to be lit, the light intensities from the projective texture are added to the corresponding pixel in the modulation image. Warped pixels may overlap since their sizes are estimated, so we have to avoid adding more than once the intensities from the same light for an eye pixel. If there is only one light, the intensities can overwrite instead of being added to the modulation image. For multiple lights, the stencil buffer can be used to store light identifiers to avoid the problem.

### 3.5 Anti-aliasing

In order to produce smooth shadow edges, we simply blur the modulation image with some weighted-average filter. The bigger the filter kernel, the softer the edges appear, and the less precise the edge positions. It is interesting to note that the effect of blurring is analogous to the percentage closer sampling for antialiasing [RSC87] in the traditional shadow map algorithm, but enjoys better memory access behavior. Advanced filtering techniques and large filter kernels are easier to support in our algorithm.

## 4 Comparison

In pure-software implementations, forward shadow mapping (FSM) and traditional shadow map algorithms (or backward shadow mapping, BSM) differ mainly in strategies for resampling—in the eye space—the light-space samples of the scene geometry. Similar to texture mapping, BSM resamples by interpolating the nearby samples for a new sampling location[2]. FSM resembles splatting[Wes90], in which each light-space sample leaves a "footprint" (i.e. area covered by the warped pixel) on the eye's image. In this paper we have assumed simple footprints to reduce amount of computation.

In hardware implementations, BSM borrows the random look-up capacity built into texturing mapping hardware and adds depth comparisons as a shadow map-

[2]Note that it is the results of depth comparison, not the depth values themselves, that are interpolated [RSC87].

specific extension.[3] FSM, on the other hand, shares hardware with an ordinary framebuffer (the color and Z buffers), which provides random memory access needed by the warped pixels for depth comparisons and accumulation of light intensities. No special-purpose extension is needed.

FSM is independent of texture mapping so that the latter can be fully dedicated to scene contents. It also makes it possible for shadow generation to run in parallel with texturing. It accesses shadow maps *and* projective textures in pixel-sequential order.

# 5 Implementation

Our algorithm has been implemented on Silicon Graphics workstations using OpenGL. Ideally, the algorithm should be implemented inside graphics hardware in order for it to work at its full potential, and to conduct direct performance comparisons with hardware implementations of traditional shadow map algorithms. Without support for such low-level programming, our current implementation nontheless serves as a proof of concept and also runs at interactive frame rates.

3-D warping is done on the host, and runs in parallel if multiple CPUs are available. The depth buffers from the lights and the eye are read into the main memory to support warping. The modulation image, once constructed, is drawn onto the framebuffer with the proper blending function which treats incoming colors as modulation factors for already existing colors in the framebuffer.

Figures 3 and 4 show the effects of shadow map resolutions and filtering of the modulation image. The filter uses a $3 \times 3$ averaging kernel. As expected, filtering helps to enhance image quality, especially when the shadow map is low resolution. On a 4 processor SGI Onyx2 InfiniteReality, frame rate is 7 frames per second with filtering and 9 without.

Figure 5 is a frame taken from an interactive demonstration of our algorithm. A robot walks in a soft-edge spotlight; with the robot moving, the shadow map has to be updated every frame. It runs at $640 \times 480$ VGA resolution. On the same machine as mentioned above, the frame rate is on average 6 frames per second. Table 5 shows the allocation of frame time, averaged over 100 frames.

# 6 Conclusion and Future Work

In this paper we have introduced a new shadow algorithm based on forward 3-D warping of shadow maps. This algorithm provides new design alternatives for both software and hardware shadow determination. It does not interfere with ordinary texture mapping. It accesses shadow maps and projective textures in sequential order and easily supports anti-aliased shadow edges. The algorithm has been implemented and demonstrates shadowing at interactive rates.

Future plans include implementation of the algorithm inside graphics hardware, faster implementation of 3-D warping algorithms, and better methods for computing the pixel coverage of warped pixels.

[3] On highly programmable architectures like PixelFlow, shadowing only involve a special software shading function, with no change to hardware.

| Operations | Time (ms) |
|---|---|
| Render from light | 26.2 |
| Render from light | 26.2 |
| Read light depth | 11.7 |
| Render from eye | 27.2 |
| Read eye depth | 12.3 |
| 3-D warping | 67.1 |
| Modulation | 8.8 |
| Misc. overhead | 14.0 |
| Total frame | 167.3 |

Table 1: Frame time allocation for the robot demo

# References


[Ake93] Kurt Akeley. RealityEngine graphics. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 109–116, August 1993.

[App68] A. Appel. Some techniques for shading machine renderings of solids. In *IFIP*, volume 32, pages 37–45, 1968.

[AWG78] P. Atherton, K. Weiler, and D. Greenberg. Polygon shadow generation. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, volume 12, pages 275–281, August 1978.

[BB84] L. S. Brotman and N. I. Badler. Generating soft shadows with a depth buffer algorithm. *IEEE Computer Graphics and Applications*, 4(10):71–81, October 1984.

[Ber86] P. Bergeron. A general version of crow'S shadow volumes. *IEEE Computer Graphics and Applications*, 6(9):17–28, 1986.

[BK70] W. J. Bouknight and K. C. Kelly. An algorithm for producing halftone computer graphics presentations with shadows and movable light sources. In *Proc. AFIPS JSCC*, volume 36, pages 1–10, 1970.

[CF89] Norman Chin and Steven Feiner. Near real-time shadow generation using BSP trees. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 99–106, July 1989.

[CF92] Norman Chin and Steven Feiner. Fast object-precision shadow generation for areal light sources using BSP trees. In *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25, pages 21–30, March 1992.

[CG85] Michael F. Cohen and Donald P. Greenberg. The Hemi-Cube: A radiosity solution for complex environments. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 31–40, August 1985.

[Cro77] Franklin C. Crow. Shadow algorithms for computer graphics. In *Computer Graphics (SIGGRAPH '77 Proceedings)*, volume 11, pages 242–248, July 1977.

[CW93] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 279–288, August 1993.

[DSV97] Lucia Darsa, Bruno C. Silva, and Amitabh Varshney. Navigating static environments using image-space simplification and morphing. In *1997 Symposium on Interactive 3D Graphics*, pages 7–16, April 1997.

[DWS+88] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: A VLSI system for high performance graphics. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 21–30, August 1988.

[EMP+97] John Eyles, Steven Molnar, John Poulton, Trey Greer, Anselmo Lastra, Nick England, and Lee Westover. PixelFlow: The realization. In *1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 57–68. ACM SIGGRAPH / Eurographics, ACM Press, August 1997.

[FGH+85] Henry Fuchs, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Frederick P. Brooks, Jr., John G. Eyles, and John Poulton. Fast spheres, shadows, textures, transparencies, and image enhancements in Pixel-Planes. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 111–120, July 1985.

[GHC97] Steven Gortler, Liwei He, and Michael F. Cohen. Rendering layered depth images. Technical Report MSR-TR-97-09, Microsoft Research, 1997.

[GTGB84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modelling the interaction of light between diffuse surfaces. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 212–22, July 1984.

[Hei91] Tim Heidmann. Real shadows, real time. *Iris Universe*, 18:28–31, 1991.

[HH97] Paul S. Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, School of Computer Sciene, Carnegie Mellon University, 1997.

[HN85] J. C. Hourcade and A. Nicolas. Algorithms for antialiased cast shadows. *Computers and Graphics*, 9(3):259–265, 1985.

[Kay79] Douglas S. Kay. Transparency, refraction, and ray tracing for computer synthesized images. Master's thesis, Cornell U., January 1979.

[Max86] Nelson L. Max. Atmospheric illumination and shadows. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 117–24, August 1986.

[MB95] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 39–46. ACM SIGGRAPH, August 1995.

[McR96] Organizer: Tom McReynolds. Programming with opengl: Advanced rendering. *SIGGRAPH'96 Course Notes*, pages 27–28, August 1996.

[MMB97] William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3d warping. In *1997 Symposium on Interactive 3D Graphics*, pages 7–16, April 1997.

[NN85] Tomoyuki Nishita and Eihachiro Nakamae. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 23–30, July 1985.

[RSC87] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 283–291, July 1987.

[SDB97] François Sillion, George Drettakis, and Benoit Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. In *Computer Graphics Forum (Proc. of Eurographics '97)*, volume 16, pages 207–218, Budapest, Hungary, September 1997.

[SKW+92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul E. Haeberli. Fast shadows and lighting effects using texture mapping. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 249–252, July 1992.

[TN85] E. Nakamae T. Nishita, I. Okamura. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. *ACM Transaction on Graphics*, 20(2):123–146, April 1985.

[Wes90] Lee Westover. Footprint evaluation for volume rendering. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 367–376, August 1990.

[Whi79] T. Whitted. An improved illumination model for shaded display. In *Computer Graphics (Special SIGGRAPH '79 Issue)*, volume 13, pages 1–14, August 1979.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, volume 12, pages 270–274, August 1978.

[WND85] Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide, 2nd Edition*. Addison-Wesley, 1985.

[WPF90] Andrew Woo, Pierre Poulin, and Alain Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, November 1990.

Figure 3: 512×512 final images using a 512×512 shadow map. The projective texture is the same as in Figure 2. The modulation image is blurred for (a) and not blurred for (b).

[Zha97] Hansong Zhang. A traditionalist view of 3-d image warping. Technical Report 97-043, Department of Computer Sciene, UNC-Chapel Hill, September 1997.

Figure 4: $512 \times 512$ final images using a $256 \times 256$ shadow map. The projective texture is the same as in Figure 2. The modulation image is blurred for (a) and not blurred for (b).

Figure 5: A frame from an interactive demo of forward shadow mapping (a walking robot in a spotlight)