

# Shadow Mapping Based on Dual Depth Layers

D. Weiskopf and T. Ertl

Institute of Visualization and Interactive Systems  
University of Stuttgart

---

## Abstract

*Shadow maps are a widely used means for the generation of shadows although they exhibit aliasing artifacts and problems of numerical precision. In this paper we extend the concept of a single shadow map by introducing dual shadow maps, which are based on the two depth layers that are closest to the light source. Our shadow algorithm takes into account these two depth values and computes an adaptive depth bias to achieve a robust determination of shadowed regions. Dual depth mapping only modifies the construction of the shadow map and can therefore be combined with other extensions such as filtering, perspective shadow maps, or adaptive shadow maps. Our approach can be mapped to graphics hardware for interactive applications and can also be used in high-quality software renderers.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

---

## 1. Introduction

Shadows are important elements in creating a realistic image and in providing the user with visual cues about the spatial structure of the scene and the relative positions of objects. Studies have shown that shadows are crucial for spatial perception and that the use of shadows as depth cues enhances the accuracy of object positioning<sup>8</sup>. Classical algorithms for generating shadows date back to the early days of computer graphics. A representative for an object-space approach is the shadow volume algorithm<sup>3</sup>; image-space algorithms originate from shadow depth mapping<sup>16</sup>.

In this paper, we exclusively deal with this image-space approach because shadow mapping is an effective and widely used shadow determination technique and provides several important benefits. Not only does it allow for any class of geometric primitives without any additional precaution, it is also the only shadow approach that requires storage complexity independent of the number of objects in the scene<sup>18</sup>. Moreover, the algorithmic structure of shadow mapping lends itself to an efficient implementation on graphics hardware as depth maps readily fit into the concept of texturing. Direct support for shadow mapping dates back to sgi's InfiniteReality<sup>10</sup> series and is included in consumer market graphics cards like the nVIDIA GeForce 3. The current generation of GPUs (graphics processing units) support shadow

mapping by providing floating point precision both in programmable fragment operations and texture formats. Not only is shadow mapping used in interactive applications but also in high-quality offline rendering systems such as RenderMan or Maya.

Shadow mapping is a two-phase process. First, the shadow depth map is filled by rendering the scene from the perspective of the light source. Then, the scene is drawn as seen from the actual eye point. During rendering of each fragment, the region to be shaded is projected onto the respective cell in the shadow depth map and the fragment's depth is compared to the value stored in the depth map in order to determine shadowed regions.

Unfortunately, shadow mapping is subject to some unpleasant disadvantages. The quality of the shadow heavily depends on the resolution of the shadow map. Aliasing problems occur due the sampling during shadow testing, especially close to shadow edges. The reason for this is the fact that a fragment to be tested cannot be exactly mapped onto a texel in the depth map. A typical artifact is self-shadow aliasing, in which a polygon is considered to shadow itself because of the inaccurate sampling method.

In order to reduce aliasing artifacts and achieve a robust depth comparison, we extend the original shadow mapping

approach to *dual shadow mapping*, which takes into account not only the closest surface to the light source but the two closest surfaces. An adaptive bias is applied to the depth of the closest surface before the depth comparison with the current fragment is performed. The other parts of the original shadow mapping approach are not modified.

The rest of this paper is organized as follows. First, a review of previous work on shadow generation is given, where the focus is on methods to overcome aliasing artifacts. In Section 3, Woo's approach<sup>17</sup> to reducing self-shadow aliasing is described in more detail because it serves as a basis for dual shadow mapping, which is explained subsequently. An implementation of dual shadow mapping on graphics hardware is presented in Section 5; resulting images are discussed in Section 6. The paper ends with a brief conclusion.

## 2. Previous Work

There is a large body of literature dealing with shadow generation. We refer to the article by Woo et al.<sup>18</sup> for a survey of shadow algorithms in general, and to the book by Akenine-Möller and Eric Haines<sup>1</sup> for a presentation of real-time techniques.

We focus our review of previous work on issues of shadow mapping and, in particular, reducing aliasing artifacts. A widely used solution to self-shadow aliasing is the use of a constant bias (shift of depth values)<sup>11</sup>. The problem is that the value for an appropriate bias depends on the scene and is quite hard to specify. Wang and Molnar<sup>15</sup> report that even for some simple test scenes it is impossible to find an acceptable bias. Therefore, they propose a technique that reduces the need for a bias for the special case of a scene consisting of solid objects only. Their method works by rendering only back faces into the shadow map, relying on the fact that aliasing problems cannot occur in the neighborhood of back faces because the back faces of a closed surface are not illuminated anyway.

Another approach to reducing self-shadow aliasing is taken by Woo<sup>17</sup>, who averages the depths of the two closest surfaces (with respect to the light source) to determine the depth shadow map. This method works for closed and non-closed objects alike. Woo's technique is the basis for the shadow generation part of the original Talisman architecture<sup>14</sup>. More recently, Everitt et al.<sup>5</sup> have presented a GeForce 3-based implementation of Woo's technique based on the extraction of the two closest surfaces via depth peeling<sup>4</sup>.

Hourcade and Nicolas<sup>7</sup> propose the priority buffer as a different form of shadow mapping, which solves the biasing issue and lends itself to a straightforward implementation on graphics hardware. In the priority buffer, only object IDs are stored, not depths. However, if only a single ID is attached to each object, intended self-shadowing for concave geometries is not taken into account. On the other hand, artifacts at

triangle edges can occur if a separate ID is issued for each triangle.

Aliasing artifacts can also be reduced by adapting the resolution of the shadow map. Fernando et al.<sup>6</sup> replace the "flat" depth map by an adaptive, hierarchical representation that is continuously updated. This method requires a traversal and refinement of the hierarchical data structure and cannot be completely mapped to graphics hardware. Tadamura et al.<sup>13</sup> use multiple shadow maps with varying resolution to reduce aliasing for outdoor scenes. Their technique cannot be mapped to graphics hardware and therefore is not suitable for interactive applications. Another approach to alleviating problems of undersampling is based on filtering. Reeves<sup>11</sup> introduces percentage closer filtering as the appropriate way of filtering shadow maps. As opposed to normal texture filtering, the fragment's depth and the entry in the shadow map are first compared and afterwards the binary results are filtered to obtain the proportion of the region in the shadow. Deep shadow maps by Lokovic and Veach<sup>9</sup> extend the concept of filtering by storing approximate attenuation functions for each shadow map texel.

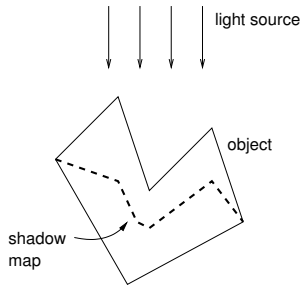
Finally, Stamminger and Drettakis<sup>12</sup> propose perspective shadow maps to reduce perspective aliasing. However, their technique aggravates the problem of self-shadowing because objects are scaled non-uniformly. Therefore, a robust depth comparison is especially useful in combination with perspective shadow maps.

## 3. Midpoint Shadow Maps

Woo<sup>17</sup> introduced a virtual, intermediate depth map to avoid many biasing and self-shadowing problems associated with shadow maps. In the first part of the shadow mapping algorithm, not only is the closest depth value with respect to the light source determined, but the two closest values are stored in separate buffers. Afterwards the two buffers are averaged into a single depth texture, which serves as the shadow map for the subsequent rendering of the shadowed scene.

For a closed surface of a solid object, this shadow map represents depth values of a virtual surface passing through the middle of the object. Therefore, we use the term *midpoint shadow mapping* for this approach. Figure 1 illustrates the midpoint shadow map for a polygonal test object.

On the one hand, this extension has the advantage that almost all potential precision problems are solved by shifting the bounds for depth comparisons away from the closest, unshadowed surface. On the other hand, self-shadowing and bias remain issues for a small number of cases. Figure 2 shows examples for the two major classes of problems. In Figure 2 (a), the fragment marked by the black dot might become erroneously unshadowed because the midpoint shadow map immediately to the right is behind the black dot. This *self-unshadowing* has two reasons. First, the finite sampling of the shadow map might cause a mapping



**Figure 1:** *Midpoint shadow map.* A directional light source is above. A virtual surface (dashed line) in between the two closest layers of the polygonal test object (solid line) serves as shadow map.

to an inappropriate depth value. Second, the distance of the second closest surface affects the position of the midpoint shadow map. Figure 2 (b) demonstrates that self-shadowing can appear in regions close to a silhouette line. If a silhouette point is further away from the light source than neighboring parts of the midpoint shadow map, the silhouette point might become self-shadowed.

#### 4. Dual Shadow Maps

To facilitate the subsequent discussions, we formalize midpoint shadow mapping. The depth of a point—or a fragment—that has to be shadow-tested is denoted  $z_{\text{frag}}$ ; the depths of the closest and second closest surfaces are named  $z_1$  and  $z_2$ , respectively. The midpoint shadow test can now be written as

$$z_{\text{frag}} < z_1 + z_{\text{bias}}(z_1, z_2) \quad , \quad (1)$$

with the bias function

$$z_{\text{bias}}(z_1, z_2) = z_{\text{bias, midpoint}}(z_1, z_2) = \frac{z_2 - z_1}{2} \quad .$$

Generalizing midpoint shadow maps, we adopt the conceptual point of view that shadow tests can be considered as a depth comparison with a virtual surface that is shifted away from the closest surface by a variable amount  $z_{\text{bias}}(z_1, z_2)$ . As the bias is a function of the depths of the two closest surfaces, we introduce the term *dual shadow mapping* for this more generic approach.

In this framework, a widely used constant bias<sup>11</sup> can be written as  $z_{\text{bias, const}}(z_1, z_2) = z_{\text{offset}}$ , with the constant offset  $z_{\text{offset}}$ . Here, the bias function is independent of the depth structure of the scene. The problem of this approach is to find an appropriate value for the offset in order to fulfill two contradicting requirements: if the value is too small, self-shadowing will occur; if the value is too large, objects that are close to their occluder will be erroneously illuminated (Figure 3). Typically, the bias is set according to the size

of the scene objects and can often be adjusted by the user during runtime.

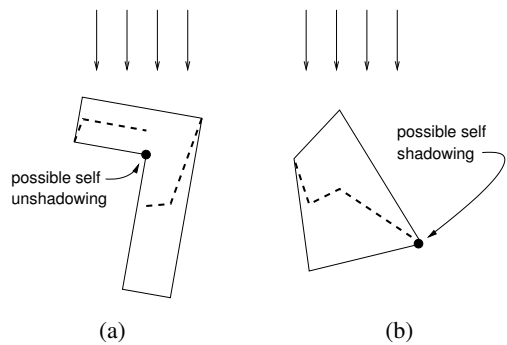
We propose to choose the bias function

$$z_{\text{bias}}(z_1, z_2) = \min\left(\frac{z_2 - z_1}{2}, z_{\text{offset}}\right) \quad , \quad (2)$$

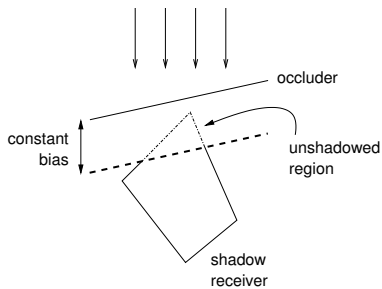
in order to combine the positive aspects of midpoint shadow maps and a constant offset. On the one hand, self-unshadowing due to the effects of a far-away second closest surface (Figure 2 (a)) is avoided by restricting the maximum bias to the constant  $z_{\text{offset}}$ . On the other hand, the bias will be determined by the midpoint approach if occluder and shadow receiver are close to each other. Therefore,  $z_{\text{offset}}$  can be set to a large value without losing the shadow on close-by objects.

Midpoint shadow maps are prone to self-shadowing artifacts at silhouette lines (Figure 2 (b)), and so is dual shadow mapping with our choice of bias function. Since front faces and back faces are very close to each other in the vicinity of a silhouette, the same mean depth value is used for the dual shadow map as for the midpoint shadow map.

For a solid object that is represented by a closed surface self-shadowing at silhouettes can be avoided by applying back face culling. In this way, the second closest surface is no longer the back face of the object but some front-facing surface further away; the bias function yields a larger shift and enables the illuminated silhouette region to pass the shadow test. Back face culling is compatible with dual shadow maps, whereas it introduces severe artifacts into the original midpoint approach. Figure 4 demonstrates how back face culling causes another class of self-unshadowing aliasing for midpoint shadow mapping. Typically, a region around a silhouette line of a shadow receiver is affected by self-unshadowing because an object in the background or—if there is no background object—the far clipping plane can shift the midpoint shadow map beyond the depth of the silhouette region (Figure 4 (a)). In Figure 4 (b), a variant of this self-unshadowing aliasing is illustrated. If the surface of a



**Figure 2:** *Self-unshadowing (a) and self-shadowing (b) problems for midpoint shadow maps.*



**Figure 3:** Shadow mapping with a large constant bias. The shadow receiver is closer to the occluder than the bias and thus is erroneously illuminated.

shadow receiver is almost parallel to the light direction, the midpoint shadow map will change rapidly, i.e., its slope is large. Due to sampling problems, parts of the receiver may lie in front of the corresponding sampled depth. This self-unshadowing aliasing is unpleasant and—for the silhouette artifacts—occurs quite often. Therefore, back face culling is not a suitable option for midpoint shadow mapping.

In contrast, our dual shadow mapping approach overcomes these aliasing problems and thus allows us to employ back face culling in order to avoid the self-shadowing artifacts at silhouette lines (Figure 2 (b)). Moreover, back face culling (for solid objects) and no culling (for example, for infinitesimally thin surfaces) can be used in the same scene without invalidating dual shadow mapping. Besides the distinction between closed, one-sided surfaces and open, two-sided surfaces, no other information about the scene is required. In particular, neither connectivity information nor assignment to object IDs is needed; a “triangle soup” can be used as input.

Dual shadow mapping only modifies the construction of the shadow map. Therefore, it can be combined with most other improvements of shadow mapping to further reduce aliasing and enhance image quality. Perspective shadow mapping<sup>12</sup> can be applied to take into account the effects of the perspective transformation into the eye space during the sampling of the shadow map. Aliasing problems can be reduced by adaptive shadow mapping<sup>6</sup> or filtering<sup>11</sup>. If more information about the structure of the scene is available, the priority buffer storing object IDs could further increase the quality of shadow mapping.

## 5. Implementation on Graphics Hardware

Dual shadow mapping lends itself to an implementation on graphics hardware and on CPU alike. Therefore, both interactive applications and high-quality offline renderers benefit from this approach. In this section, only an implementation on graphics hardware is described since it should be straight-

forward to include the modifications into a CPU-based rendering system.

The following discussion is based on the functionality of state-of-the-art GPUs (graphics processing units). In particular, we utilize programmable vertex and fragment processing with floating point precision, floating point textures, and render-to-texture capabilities. Our example implementation is based on DirectX 9 and was tested on an ATI Radeon 9700. As we build upon the functionality laid out in the specifications of DirectX 9, our implementation will run on any future graphics hardware that is conform to DirectX 9.

The depth comparison for dual shadow map from Eq. (1) can be re-written as

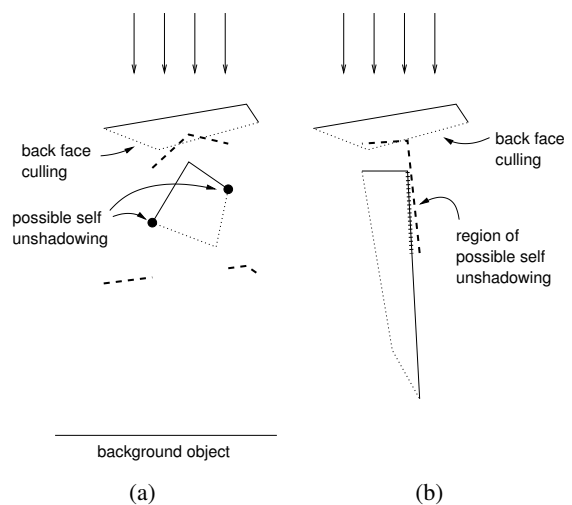
$$z_{\text{frag}} < z_{\text{dual}}(z_1, z_2) \quad , \quad (3)$$

with the depth texture

$$z_{\text{dual}}(z_1, z_2) = z_1 + z_{\text{bias}}(z_1, z_2) \quad . \quad (4)$$

Therefore, the complete shadow mapping algorithm consists of two main phases. The depth texture  $z_{\text{dual}}(z_1, z_2)$  is generated in the first phase. The second phase is identical to the original shadow mapping algorithm: the scene is rendered from the eye point by taking into account the depth comparison (3).

The main problem in phase one is to extract the two closest surfaces as seen from the light source. This can be accomplished by a two-pass rendering. In the first pass, the scene is rendered into the depth buffer, with depth testing being enabled. Texture coordinates are specified in a way to represent the depth with respect to the light source, i.e., the position of a vertex,  $(x, y, z, w)$  is re-used as its texture coordinates,  $(s, t, q, r)$ . A vertex program (vertex shader in the lan-



**Figure 4:** Self-unshadowing for midpoint shadow mapping with back face culling.

guage of DirectX) needs only one additional line of code to output the vertex coordinates as texture coordinates. Via interpolated texture coordinates, the fragment program (pixel shader in the notation of DirectX) has access to the depth of the current fragment and writes it directly into a texture with one floating point channel, i.e., such a texture is used as render target. Unfortunately, the fragment operation unit has no read access to the fragment's depth; therefore, the depth has to be transmitted from the transform and lighting stage to the fragment program via texture coordinates. (Therefore, slope-scale based depth bias, which is supported in DirectX 9 for  $z$  buffer rendering, cannot be used for shadow mapping.)

In the second pass, the scene is rendered into the depth buffer for a second time, after having cleared the depth buffer. Once again, a one-channel floating point texture of the same size is used as render target. Exactly the same scene as in the first pass—including the aforementioned texture coordinates—is rendered. In addition, another tuple of texture coordinates is specified to allow a one-to-one mapping between rendered fragments and the texels from the texture generated in the first pass. The fragment program compares the fragment's depth  $z_2$  to the depth  $z_1$  stored in the texture; if both depth values are equal (up to a very small tolerance level  $\epsilon$  due to inaccuracies in the floating point representation), the fragment will be discarded. In this way, the foremost surface is not rendered in this second pass. Finally, a third rendering pass combines the two textures for  $z_1$  and  $z_2$ , evaluates the function  $z_{\text{dual}}(z_1, z_2)$  by a fragment program, and writes the outcome into a render target.

The resulting texture is used as shadow map in phase two. Here, the scene is rendered from the eye point, with illumination computations and shadow testing being enabled. Texture coordinates are computed in a vertex program to represent the position of the vertex with respect to the light source. A fragment program compares the fragment's depth in the coordinate system of the light source with the depth stored in the shadow map according to Eq. (3). Depending on the result of the comparison, the fragment is drawn as shadowed or unshadowed pixel.

The new functionality introduced by dual shadow mapping is completely restricted to phase one. The rendering of the scene from the eye point in phase two is not modified—dual shadow maps are transparent to the implementation of this rendering stage. Therefore, filtering techniques, which concern only phase two, can be readily combined with dual shadow maps. We have implemented two variants of percentage closer filtering. First, a bilinear interpolation of the binary results for the four closest depth map texels is realized. Since bilinear percentage closer filtering is not directly supported in the specifications of DirectX 9, the weights for bilinear interpolation are explicitly computed in the fragment program by extracting the fractional coordinates within the respective texture cell. Second, a filtering by a  $4 \times 4$  jitter

matrix is implemented; the code is based on an ATI shadow demo<sup>2</sup>.

## 6. Results

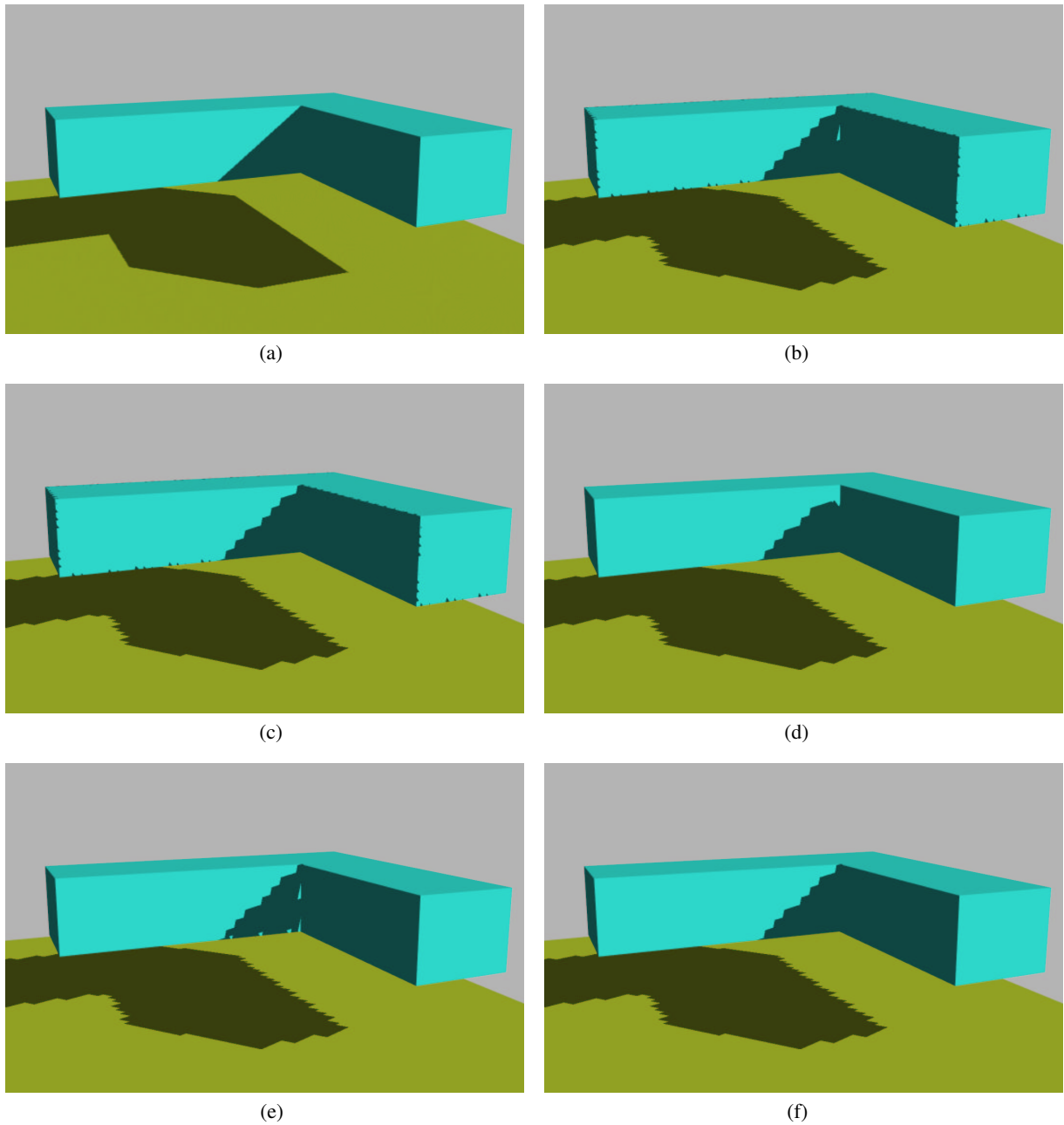
In this section, some example images are included to demonstrate the features of dual depth mapping in comparison to midpoint shadow maps and the approach with constant bias. All images were generated on a PC with ATI Radeon 9700 graphics by using our implementation based on DirectX 9.

Figure 5 shows a simple “L”-shaped object above a greenish surface. The scene is illuminated by directional light from right above. Figure 5 (a) is a high-quality rendering and serves as a benchmark. In all other images, a heavily undersampled shadow texture is used to emphasize possible aliasing artifacts. Midpoint shadow mapping without back face culling, as illustrated in (b), reveals both self-unshadowing (at the vertical crease) and self-shadowing artifacts (at the silhouettes as seen from the light source). In image (c), dual shadow mapping without back face culling avoids self-unshadowing, but shows the same self-shadowing artifacts at the silhouettes. If the constant depth bias in the original shadow mapping approach is chosen too large, surfaces lying in the shadow can become erroneously illuminated like the region left to the vertical crease in image (d). By enabling back face culling, self-shadowing artifacts can be removed for midpoint shadow mapping (image (e)); however, additional self-unshadowing in the neighborhood of silhouettes is introduced. Finally, the combination of dual shadow mapping and back face culling for solid objects gives quite convincing results even for heavily undersampled shadow maps, as it can be seen in (f).

In Figures 6 (d)–(f), bilinear shadow filtering is demonstrated for the original shadow mapping technique, midpoint shadow mapping, and dual shadow mapping, respectively. Corresponding unfiltered images are shown in Figures 6 (a)–(c). Filtering helps to reduce the visibility of artifacts and generates more naturally looking soft shadow edges. Nevertheless, self-unshadowing artifacts in the midpoint approach remain clearly noticeable (on the lower left part of the double torus in (b) and (e)).

Figure 7 shows a more realistic image taken from a factory scene. Percentage closer filtering with a  $4 \times 4$  jitter matrix is applied, and a shadow map of acceptable resolution is used. However, midpoint shadow mapping in image (c) still shows some self-unshadowing artifacts, e.g., at the left edge of the wall in the background and around the transmission roles in the upper right. This artifacts are avoided by dual shadow mapping in image (d).

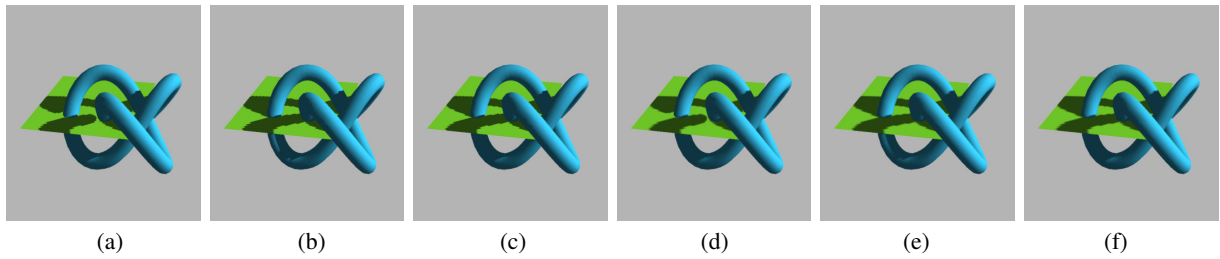
Performance measurements for a Windows XP PC with ATI Radeon 9700 GPU and AMD Athlon XP 2200+ CPU are given in Table 1. The test scene from Figure 7 with 13,284 triangles and 13,177 vertices was rendered on a  $1100^2$  viewport. Performance numbers for a rather small



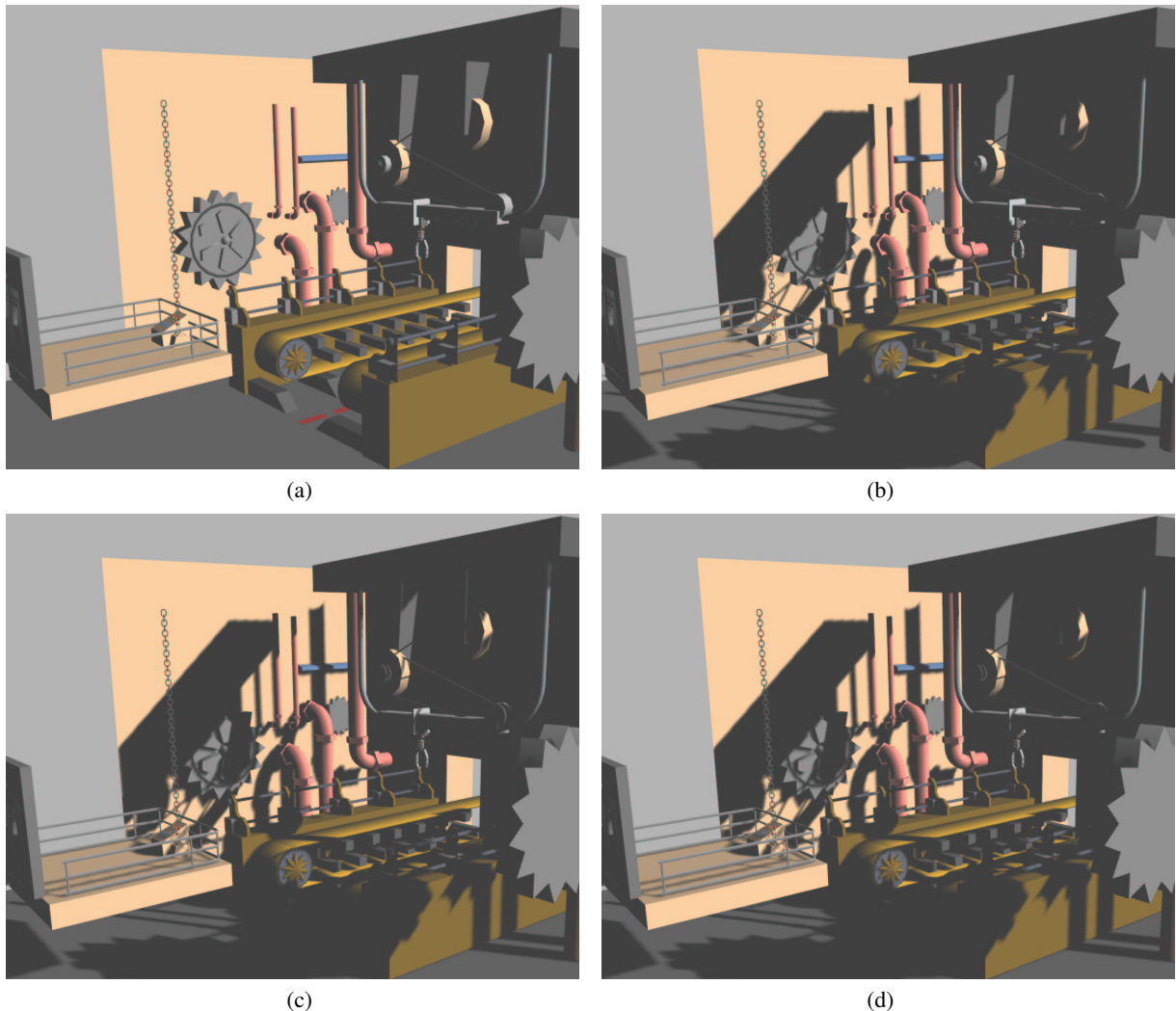
**Figure 5:** Comparison of shadow mapping approaches. Directional light comes from right above. Image (a) shows a high-quality rendering as a benchmark; all other images are based on a heavily undersampled shadow texture to emphasize artifacts. Image (b) illustrates midpoint shadow mapping without back face culling, (c) shows dual shadow mapping without back face culling, (d) presents the original shadow mapping with a large constant bias, (e) and (f) show midpoint shadow mapping with back face culling and dual shadow mapping with back face culling, respectively.

depth texture of  $256^2$  and a large depth texture of  $2048^2$  are included to compare the efficiency of the first phase of the shadow mapping algorithm. Midpoint and dual shadow mapping run at the same speed because their fragment programs

differ only minimally. For large depth textures, these two approaches show roughly half of the performance of the original shadow mapping technique due to the additional rendering costs in phase one. For smaller depth textures and / or



**Figure 6:** Bilinear shadow filtering. Images (a)–(c) are generated by the original shadow map technique, midpoint shadow mapping, and dual shadow mapping, respectively. Here, no filtering is applied. Bilinear filtering is combined with the same three techniques in (d)–(f).



**Figure 7:** Shadow mapping with  $4 \times 4$  percentage closer filtering for a factory scene. Image (a) shows no shadow, (b) shadow mapping with a constant bias, (c) midpoint shadow mapping, and (d) dual shadow mapping.

**Table 1:** Rendering performance in fps.

shadow map size	constant bias		midpoint / dual	
	256 <sup>2</sup>	2048 <sup>2</sup>	256 <sup>2</sup>	2048 <sup>2</sup>
no shadow	167.5			
no filtering	106.2	69.1	85.3	30.6
bilinear filtering	44.8	36.6	40.6	21.9
4 × 4 filtering	27.2	23.9	25.5	16.6

more sophisticated filtering during phase two, the difference between the shadow mapping methods becomes much smaller.

## 7. Conclusion

We have presented dual shadow mapping, which computes an adaptive depth bias based on the two closest depth layers. The bias function takes the minimum of a constant offset and the half distance between these two depth layers in order to combine the advantages of a constant bias and midpoint shadow mapping. Dual shadow mapping handles a collection of triangles and does not require further information on the structure of the scene. Furthermore, back face culling for solid objects can be mixed with non-culled open surfaces in the same scene to increase the rendering performance and the robustness of the shadow tests. Finally, our approach lends itself to a hardware-accelerated implementation for interactive applications, but also is applicable to high-quality software renderers.

Although the issue of finding an appropriate constant bias has not been addressed directly, this problem is greatly alleviated by dual shadow mapping. In the limit of an infinite offset, dual shadow mapping converges to the original midpoint approach and, thus, still provides the acceptable results of this state-of-the-art technique. Therefore, the offset can be chosen by taking a value somewhere (or even well) above the value that would be used for implementations with only a constant offset. Stated differently, the final results are much less dependent on this value.

## References

1. T. Akenine-Möller and E. Haines. *Real-Time Rendering*. A.K. Peters, Natick, second edition, 2002. 2
2. ATI. Shadow map demo. <http://www.ati.com/developer/samples/dx9/ShadowMap.html>, 2003. 5
3. F. C. Crow. Shadow algorithms for computer graphics. *Computer Graphics (SIGGRAPH '77 Proceedings)*, 11(2):242–248, 1977. 1
4. C. Everitt. Interactive order-independent transparency. White paper, nVIDIA, 2002. 2
5. C. Everitt, A. Rege, and C. Cebenoyan. Hardware shadow mapping. White paper, nVIDIA, 2001. 2
6. R. Fernando, S. Fernandez, K. Bala, and D. P. Greenberg. Adaptive shadow maps. In *SIGGRAPH 2001 Conference Proceedings*, pages 387–390, 2001. 2, 4
7. J. C. Hourcade and A. Nicolas. Algorithms for antialiased cast shadows. *Computers and Graphics*, 9(3):259–265, 1985. 2
8. G. S. Hubona, P. N. Wheeler, G. W. Shirah, and M. Brandt. The relative contributions of stereo, lighting, and background scenes in promoting 3D depth visualization. *ACM Transactions on Computer-Human Interaction*, 6(3):214–242, Sept. 1999. 1
9. T. Lokovic and E. Veach. Deep shadow maps. In *SIGGRAPH 2000 Conference Proceedings*, pages 385–392, 2000. 2
10. J. S. Montrym, D. R. Baum, D. L. Dignam, and C. J. Migdal. InfiniteReality: A real-time graphics system. In *SIGGRAPH 97 Conference Proceedings*, pages 293–302, 1997. 1
11. W. T. Reeves, D. H. Salesin, and R. L. Cook. Rendering antialiased shadows with depth maps. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):283–291, 1987. 2, 3, 4
12. M. Stamminger and G. Drettakis. Perspective shadow maps. *ACM Transactions on Graphics*, 21(3):557–562, July 2002. 2, 4
13. K. Tadamura, X. Qin, G. Jiao, and E. Nakamae. Rendering optimal solar shadows with plural sunlight depth buffers. *The Visual Computer*, 17(2):76–90, 2001. 2
14. J. Torborg and J. Kajiya. Talisman: Commodity real-time 3D graphics for the PC. In *SIGGRAPH 96 Conference Proceedings*, pages 353–364, 1996. 2
15. Y. Wang and S. Molnar. Second-depth shadow mapping. Technical Report TR94-019, Department of Computer Science, University of North Carolina - Chapel Hill, Dec. 1994. 2
16. L. Williams. Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH '78 Proceedings)*, 12(3):270–274, 1978. 1
17. A. Woo. The shadow depth map revisited. In D. Kirk, editor, *Graphics Gems III*, pages 338–342. AP Professional, Boston, 1992. 2
18. A. Woo, P. Poulin, and A. Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, Nov. 1990. 1, 2