Hardware-accelerated Rendering of Antialiased Shadows with Shadow Maps

Stefan Brabec and Hans-Peter Seidel

Computer Graphics Group
Max-Planck-Institut für Infomatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
Phone +49 681 9325-428 Fax +49 681 9325-499
E-mail: {brabec, hpseidel}@mpi-sb.mpg.de

Abstract

We present a hardware-accelerated method for rendering high quality, antialiased shadows using the shadow map approach. Instead of relying on dedicated hardware support for shadow map filtering, we propose a general rendering algorithm that can be used on most graphics workstations. The filtering method softens shadow boundaries by using a technique called percentage closer filtering which is commonly used in software renderers, e.g ray tracing. In this paper we describe how the software algorithm can be efficiently mapped to hardware. In order to achieve real-time or at least interactive frame rates we also propose a slightly modified shadow filtering method that saves valuable hardware resources while still achieving good image quality.

Keywords: Shadow Algorithms, Graphics Hardware, Frame Buffer Tricks, Image Processing, Rendering.

1 Introduction and Background

Recent developments in graphics hardware have led to systems which provide very high polygon throughput combined with sophisticated lighting and shading models. However, improvements in image quality are still focusing on local properties, e.g. using per fragment phong lighting. For realistic images, which should provide accurate impressions of the world in 3D, one must also consider the computation of global effects. One of the most important effects in this category are of course good-looking shadows. In the field of real-time and interactive rendering, there are only a few widely used algorithms. Casting shadows onto large receiver polygons is mostly done using projected geometry [1], whereas more complex receivers can only be handled using the shadow volume approach [2]. These two methods have in common that they still require CPU resources to compute geometry and may generate a large amount of additional polygons that have to be pushed down the graphics pipeline. The main benefit of these algorithms is that all computations are done in object space which usually results in highly accurate shadows.

Another class of algorithms is based on the shadow map or depth map algorithm [10]. These methods operate in image space which means that they rely on a sampling scheme to process the scene geometry. The shadow map algorithm can be used to compute shadows for arbitrary receiver and occluder geometry. Since sampling is the fundamental principal of all rasterization-based graphics architectures shadow mapping is the first choice for real-time rendering. SGI's Onyx Reality series of graphic workstations [5] were the first systems with direct shadow map support and recently ATI presented the first PC graphics card [6] with a similar feature based on the so called priority buffer.

However, sampling methods all suffer from undersampling artifacts when it comes to higher frequency parts and the same is true for the shadow map algorithm. As discussed by Reeves et al. [7], sampling problems occur during the generation phase as well as when performing the actual shadow test. The first can be solved by increasing the image resolution and by using stochastic sampling instead of sampling at regular grid points. In order to resolve sampling artifacts during the shadow test, Reeves proposes a filtering method called *percentage closer filtering* (PCF) which will be described later in more detail.

In this paper we describe several ways for mapping this filtering technique to hardware. First, we will briefly explain the shadow map algorithm and its hardware implementation. Second, we will take a closer look at Reeves' filtering technique and propose a hardware-only way to filter shadow maps. Third, we will present a modified PCF algorithm that can be used to render high-quality, antialiased shadows at interactive or even real-time frame rates. Finally we compare the different approaches in terms of image quality and rendering time.

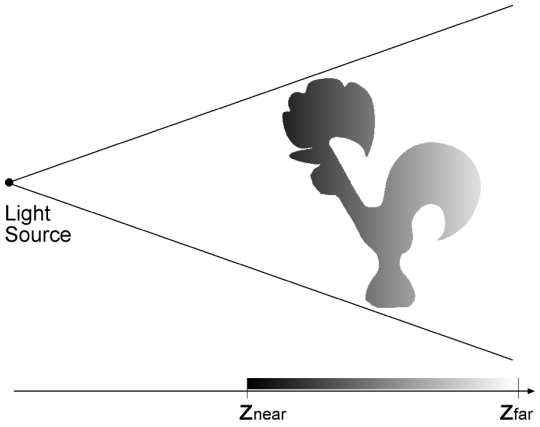


Figure 1. Mapping z values to color values using texture mapping.

2 Shadow Mapping

As presented by [10], the shadow map algorithm consists of two major steps. First, the scene is rendered as seen by the light source and the z values of the nearest pixels are stored away in the so called shadow map for later use. In the final step, the scene is rendered again, this time from the camera's point of view. During this pass, surface points are transformed to the light source coordinate system and compared to the corresponding z values stored in the shadow map which results in a binary value indicating whether the point is in shadow or lit.

The hardware implementation we use is based on the algorithm proposed by [4]. Instead of relying on dedicated shadow mapping capabilities, e.g. as done by [9], this method can be implemented on any graphics system which supports a rendering pipeline similar to the OpenGL graphics API [8].

The main idea of the algorithm is the mapping of z values to color values as depicted in Figure 1. Using automatic texture coordinate generation it is possible to use eye or world space coordinates (x, y, z, w) as texture coordinates (s, t, r, q). Combined with a one dimensional texture map defining a linear ramp between 0 and 1 a mapping from r to any of the red, green, blue, or alpha channels can be done. The resulting image is stored away in a 2D texture map for later use.

In the next step the scene is rendered once again, this time from the camera's point of view. Again a linear ramp 1D texture is used to map z values to color values, but this time an additional 4×4 texture matrix is specified which transforms points from camera to light source space.

In the third and final pass the scene is rendered from the camera's position with the precomputed shadow map applied as a projective texture and the blending unit setup to subtract the values from this pass from the values of the previous one. The result of this is a shadow mask in the specified color channel that contains values of 0 for lit and values > 0 for shadowed pixels[1].

This method can be implemented on any graphics system supporting OpenGL. However, some of the passes can be collapsed if multi texturing is supported. In the next sections we will focus on those machines that have support for the OpenGL Imaging Subset [11], a class of methods for real-time 2D image processing that are useful for many kinds of applications.

3 Hardware-based Percentage Closer Filtering

In this section we describe how Reeves percentage closer filtering technique can be used for hardware rendering by extending the shadow mapping technique explained in the previous section.

Basically, PCF works by reversing the order of filtering and comparing. Instead of first filtering the texture image over some specific region and using the resulting value for further processing, PCF performs the comparison step first. Figure 2 illustrates the scheme in the case of a 3×3 region

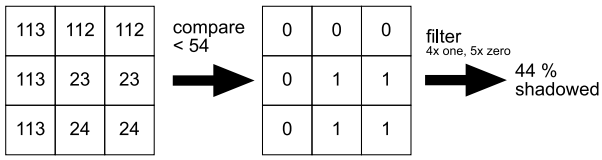


Figure 2. Percentage closer filtering.

in the shadow map. Nine z values are compared against a given surface z value which results in a 3×3 binary mask from which the percentage shadowing can be calculated by simple bit counting. The region that is sampled can be determined by projecting the pixel boundary rectangle onto the shadow map, an operation which is easy to implement in a software renderer but impossible for hardware rendering without a dedicated PCF filtering method.

However, it is possible to use PCF if we assume a constant filter region, e.g. a 2×2 footprint, which is the smallest, symmetric filter size. For this we have to compare each pixel's z against four z values stored in the shadow map. For a constant footprint we are able to generate a shadow map where each entry consists of n components and where n is the number of samples per footprint. Given a 2×2 footprint the four components can simply be stored using the red, green, blue, and alpha channel of the texture image. For the generation of the shadow map this means that we

[1] An additional frame buffer copy may be needed to bring values > 0 to 1 or any other value.

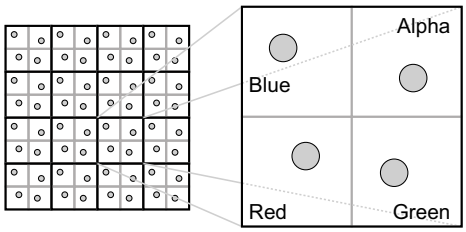


Figure 3. Stratified sampling and pixel packing.

have to render the scene four times where in each pass only one color channel is enabled for writing and the image plane is jittered as depicted in Figure 3. This stratified sampling scheme increases the effective resolution by a factor of two in each dimension, so instead of a 1024×1024 one component shadow map we have now generated a 1024×1024 shadow map with four depth values per texel.

Given such a packed shadow map it is relatively easy to adapt the shadow test proposed in Section 2 since we only have to extend the one component scheme to four components.

During the shadow test passes we first render the scene as seen by the camera but since we want to compare the surface point's depth against every component in the shadow map we replicate the z values over all four color channels. Next, the projection of the shadow map is done as before. This corresponds to Reeves' comparison step, resulting in a four component shadow mask.

In order to compute one scalar value per pixel which should represent the percentage shadowing, we have to count the non zero values in the shadow mask and divide this sum by the number of samples taken per pixel. All this can be performed within a single frame buffer to frame buffer copy. Assuming a color depth of 8 bits per compo-

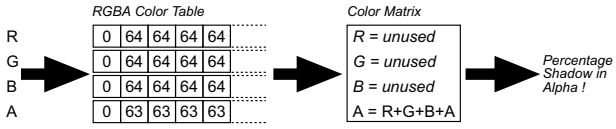


Figure 4. Computing the percentage of shadowing.

nent, we setup the OpenGL imaging pipeline as depicted in Figure 4. At first, incoming values are transformed using a RGBA color table. This clamps values to either 0 (completely lit) or 64, which represents 25% shadow in total[2]. Second, a simple 4×4 color matrix is used to sum up the contributions of all color channels and to pass out the result as the new alpha value. After this, five different levels of shadowing per pixel are stored in the alpha channel of the frame buffer: 0% shadowed (lit), 25%, 50%, and 75% for partially shadowed pixels and 100% for pixels that are completely shadowed.

This algorithm works very well for footprints of size 2×2 since all components can be processed simultaneously using the four color channels. If it comes to larger filter sizes, e.g. 3×3 or 4×4, the algorithm needs to be split up into parts of a maximum of four components per texel. The resulting contributions can then be summed up using the accumulation buffer [3].

Although theoretically possible, filter sizes greater than 2×2 are no longer practical for interactive or real-time applications. Considering a filter region of 4×4, the generation of the shadow map would require 16 rendering passes and four RGBA texture maps to store the results. During the shadow test, another eight passes are necessary to perform the subtraction plus up to four frame buffer copies and accumulation buffer operations.

4 Fast PCF for Real-Time Applications

In Section 3 a hardware-based algorithm for percentage closer filtering using Reeves' original method was proposed. However, for real-time applications this method requires way too much hardware resources, especially when it comes to larger filter sizes. In this section we present a slightly modified version of Reeves' PCF that overcomes these limitations.

Considering the generation of the shadow map, a footprint of $n\times n$ reduces the effective resolution by a factor of n in each dimension. For hardware graphics this means that we either have to use a very large shadow map or need to render the scene several times to different color channels. Since the number of rendering passes and the image resolution are critical for real time frame rates, *real* PCF is not well suited for very complex and dynamic scenes or machines with limited hardware resources.

A faster way of performing percentage closer filtering can be achieved if we try to retain the effective resolution of the shadow map and use a filtering scheme that softens shadow boundaries by just looking at adjacent texels to compute the shadow mask.

To do this, we render the shadow map as explained in Section 2 but instead of using only one color channel we store the encoded depth values in all four channels. Next, we want to generate a packed shadow map where each texel consists of four adjacent pixels as shown in Figure 5. Collecting neighboring pixels in that manner is not a trivial task since most rasterization hardware does not provide efficient

[2]Using a value of 63 in one of the color channels ensures that the value for completely shadowed pixels sums up to 255.

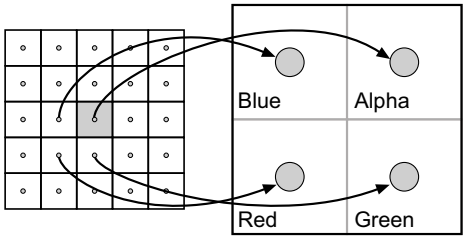


Figure 5. Fast PCF filtering and pixel packing.

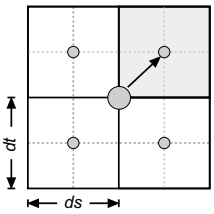


Figure 6. Texture coordinate offset.

methods for changing the position of pixels[3]. One exception to this is the OpenGL Imaging Subset, which does provide a method for performing convolutions on image data in either one or two dimensions. In the case of a 3×3 RGBA convolution the new color of a pixel P' is computed as

$$P'_{ij} = \sum_{m=0}^{2}\sum_{n=0}^{2} C_{mn}P_{m+1,n+j} \quad , \tag{1}$$

where C is the 3×3 RGBA convolution filter and P a specific pixel in the input image. This weighted sum can be adapted to perform the pixel packing as depicted in Figure 5 if we setup the filter kernel to collect only one color component for each color channel. Using

$$C = \begin{pmatrix} [0,0,0,0] & [0,0,0,0] & [0,0,0,0] \\ [0,0,\mathbf{1},0] & [0,0,0,\mathbf{1}] & [0,0,0,0] \\ [\mathbf{1},0,0,0] & [0,\mathbf{1},0,0] & [0,0,0,0] \end{pmatrix}$$

the resulting pixel value P' consists of the red channel taken from the lower left, the green channel from the lower mid pixel and so on. Although one column and one row of the filter kernel is not used at all, we prefer filter sizes with odd widths and heights, e.g. 3×3 or 5×5, since graphics hardware is normally optimized for this kind of filter sizes[4].

After this convolution, which is applied when we copy the frame buffer contents to the RGBA shadow map, we have packed four depth samples into a single texel. This differs from the pixel packing scheme presented in Section 3 since the resolution remains constant (using only one rendering pass for the shadow map).

Performing the shadow test and computing the percentage shadowing term is very similar to the hardware-based PCF algorithm in Section 3, except that the texture coordinates need to be slightly offset by $(\frac{ds}{2}, \frac{dt}{2})$ (see Figure 6) to account for the new center pixel (since the convolution filter packs the lower left part as the new center pixel, as illustrated in Figure 5).

Having only one rendering pass for the shadow map generation, it becomes affordable to use larger filter sizes. In the case of a 4×4 footprint we can split up the computation into four shadow mapping phases and use the accumulation buffer to sum up the results. For each pass we use a 3×3 convolution that samples either the upper left, upper right, lower left, or lower right 2×2 region as explained before. With this multipass method, $4 * 5 = 20$ shadowing levels can be generated.

5 Results

We have implemented the described methods on Silicon Graphics Octane VPro/8 and O2 workstations using OpenGL as an underlying graphics SDK. Since the execution time of the algorithms depends on high polygon throughput (rendering the scene several times from different points of view) and high fill rates (frame buffer and texture map copies) this kind of machines, which are optimized for both classes of applications, are ideal platforms. Furthermore, a hardware-accelerated OpenGL imaging pipeline, needed for the fast PCF algorithm presented in Section 4 is only supported on mid- and high-level graphics workstations.

A comparison of the different variants of shadow filtering techniques is depicted in Figure 7. In order to make differences more noticeable, a small part (red rectangle) of each image is magnified (Figure 7e — 7h). The scene consists of about 7000 polygons and was rendered using an image resolution of 800×600 pixels with normal OpenGL lighting and one light source enabled.

Starting from left to right, the first image shows the result of the shadow mapping approach presented in Section 2. Using a shadow map resolution of 512×512 pixels, undersampling artifacts at shadow boundaries are quite noticeable (blocky edges). On an SGI Octane VPro/8, this scene can be rendered at about $25 - 30$ frames per second[5].

The second column shows the same scene but this time with percentage closer filtering applied as described in Section 3. With only three more grey levels, the shadows look much more realistic. The shadow map still has a resolution of 512×512, but since we used a 2×2 filter, which re-

[3] Apart from some global methods, e.g. for scaling image data.

[4] This is due to the fact that most image processing convolutions, e.g. Gaussian blur, are symmetric and pixel centered.

[5] All times presented here include the generation of the shadow map (as in fully dynamic scenes).

quires four rendering passes during the shadow map generation phase, we virtually increased the resolution by packing four depth values into a single texel. Frame rates using this method drop down to about 10 fps, which is due to the three additional rendering passes needed for shadow map generation.

With fast percentage closer filtering enabled (Section 4), we can achieve nearly real-time frame rates of about $15 - 20$ fps. Using a 2×2 footprint combined with the modified pixel packing method, shadow boundaries are well smoothed (Figure 7c and 7g) resulting in an image quality comparable to the normal PCF method.

The last column shows the result of fast percentage closer filtering using a 4×4 footprint. Having about 20 different levels of shadowing, blockiness is reduced to a minimum. As described in Section 4, filters of sizes larger than 2×2 need to be implemented using multipass rendering and an accumulation buffer to sum up the results. Due to this, a frame rate of only 5 fps can be achieved. If we restrict ourselves to stationary lighting, the shadow map generation becomes a precomputing step which makes PCF with filter sizes larger than 2×2 affordable.

Figure 8 shows another example scene. On the left side the scene was rendered without filtering. Although the shadow map resolution was increased to 1024×1024 pixels, the shadow boundaries are still very blocky. Using fast PCF with a filter size of 2×2, shadow boundaries appear well smoothed (right). The rendering times are about 15 versus 10 frames per second.

6 Conclusion and Future Work

In this paper we showed how Reeves' percentage closer filtering can be applied for hardware-based shadow map rendering. With this approach, shadows of high quality can be rendered at interactive or real-time frame rates. As the algorithm makes intensive use of the OpenGL imaging extensions, a hardware-only implementation is currently only possible for certain graphics workstations. For consumer-class PC graphic cards additional memory transfers from frame buffer to host (and back) are necessary to emulate imaging operations in software. So hopefully we will see accelerated 2D imaging operations even on consumer-class graphics in the near future making the algorithm suitable for games and other interactive applications.

When discussing the algorithm we did not address sampling artifacts due to the limited depth resolution. When encoding depth values as color values we loose a lot of precision because of the (normal) 8 bits per color channel. A solution to this is already possible since some architectures, e.g. SGI's Octane VPro or InfiniteReality, support color depths of 12 bits per channel, a trend that will probably be picked up for future consumer-class graphics cards.

Up to now we have restricted ourselves to constant filter regions for computing the shadow map. However, better image quality could be achieved with a mipmap-like method for choosing a suitable filter size depending on the size of the projected pixel boundaries. But since this would require additional texture memory, which still is a very valuable resource, we consider the algorithm to be more practical and efficient in its current state.

References

[1] J. F. Blinn. Jim Blinn's corner: Me and my (fake) shadow. *IEEE Computer Graphics and Applications*, 8(1):82–86, Jan. 1988.

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

[3] P. E. Haeberli and K. Akeley. The accumulation buffer: Hardware support for high-quality rendering. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, pages 309–318, Aug. 1990.

[4] W. Heidrich. *High-quality Shading and Lighting for Hardware-accelerated Rendering*. PhD thesis, University of Erlangen, Computer Graphics Group, 1999.

[5] M. J. Kilgard. Realizing opengl: Two implementations of one architecture. *1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 45–56, August 1997.

[6] ATI Technologies Inc. Radeon charisma engine and pixel tapestry architecture. White Paper, 2000. Available from `http://www.ati.com`.

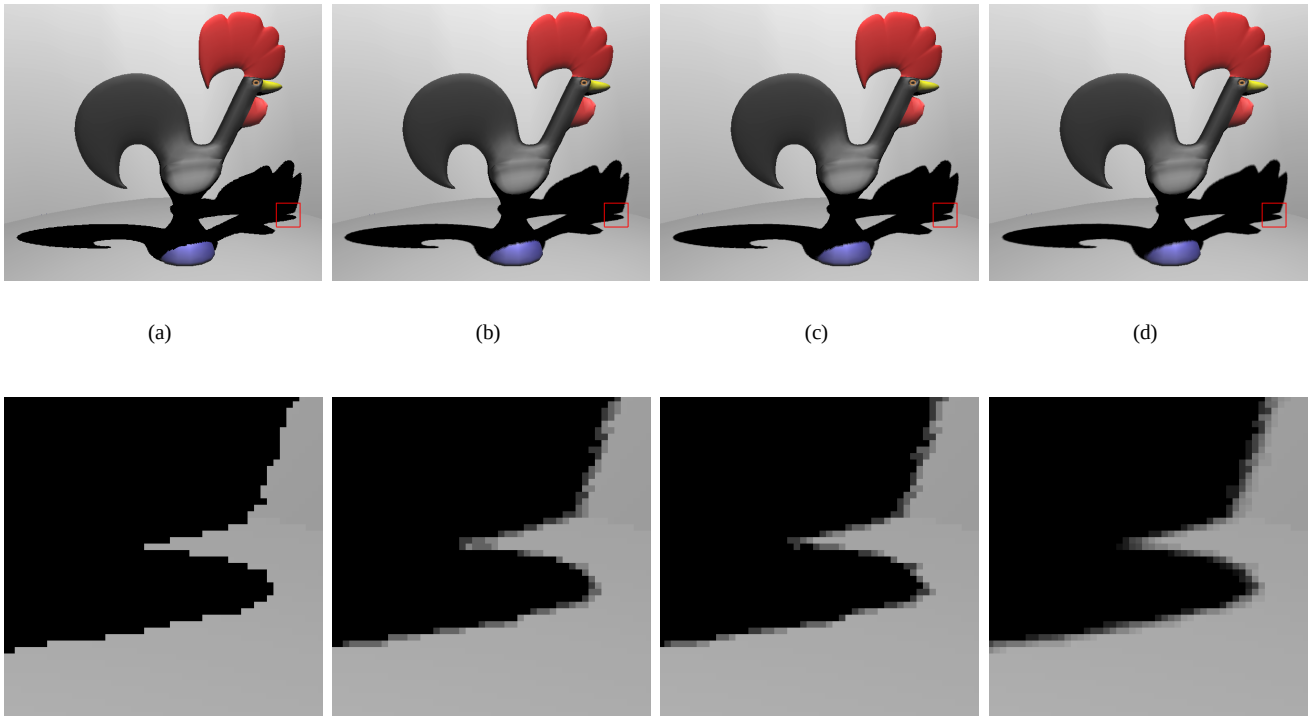
[7] W. T. Reeves, D. H. Salesin, and R. L. Cook. Rendering antialiased shadows with depth maps. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, pages 283–291, July 1987.

[8] M. Segal and K. Akeley. *The OpenGL Graphics System: A Specification (Version 1.2)*, 1998.

[9] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli. Fast shadow and lighting effects using texture mapping. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, pages 249–252, July 1992.

[10] L. Williams. Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, pages 270–274, Aug. 1978.

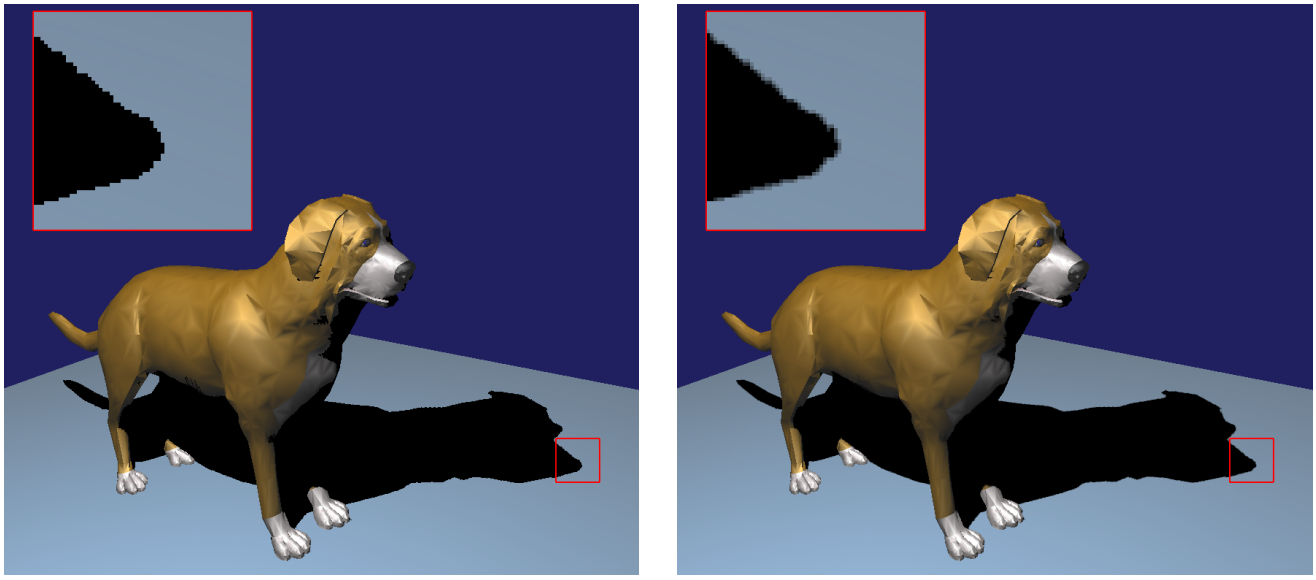
[11] M. Woo, J. Neider, T. Davis, and D. Shreiner. *OpenGL Programming Guide, Third Edition*. Addison-Wesley, 1999.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 7. Comparison of shadow filtering techniques. (a,e): without filtering. (b,f): normal PCF, 2×2 filter. (c,g): fast PCF, 2×2 filter. (d,h): fast PCF, 4×4 filter, multipass.



(a) (b)

Figure 8. Test scene. (a): without filtering. (b): fast PCF, 2×2 filter.