Smooth Penumbra Transitions with Shadow Maps

WILLEM H. DE BOER
Playlogic Game Factory B.V. [1]

We present a method for rendering aesthetically correct soft shadows that exhibit smooth inner and outer penumbrae and self-shadowing. Our approach is an extension of ordinary shadow mapping; in an additional step, occluder shadow silhouette edges, as seen from the center of an area light, are detected and extruded into so-called *Skirts*. This step is done entirely in image space, and as such does not require the construction of extra geometric primitives. During final rendering, penumbra regions are identified using the depth information of the *Skirts* and stochastic sampling of the shadow map. Our algorithm has been successfully implemented on programmable shader hardware, and interactive framerates are obtained. We observe that the main bottleneck of our implementation lies in the rasterisation stage; performance mainly depends on the size of the lightsource.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-dimensional graphics and realism

General Terms: Algorithms, human factors, performance

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

1. INTRODUCTION

Real-time soft shadow techniques have received a lot of attention by the graphics research community. Recent advances in graphics hardware have made it possible to extend existing hard shadow algorithms ([Williams 1978] and [Crow 1977]) to render soft shadows at interactive or real-time framerates. Most of these methods rely on object space techniques for extracting an occluder's shadow silhouette edges. These edges are essential to identifying penumbra regions, as we shall discuss later. Unfortunately, the performance of these object space approaches depends on the geometric complexity of the scene. In this paper, we are interested in modifying Williams' shadow map algorithm to incorporate aesthetically correct soft shadows. More specifically, we are looking for an algorithm that

(1) has a constant and controllable time complexity, independent of the complexity of the scene in terms of the number of primitives used (e.g. triangles, surface patches);
(2) does not exhibit any "banding" or other quantisation artifacts of soft shadow boundaries, and
(3) it must incorporate the "softening" effect of the shadow over distance.

[1]The author has since left the company, and now works for Microsoft Research in Cambridge, United Kingdom

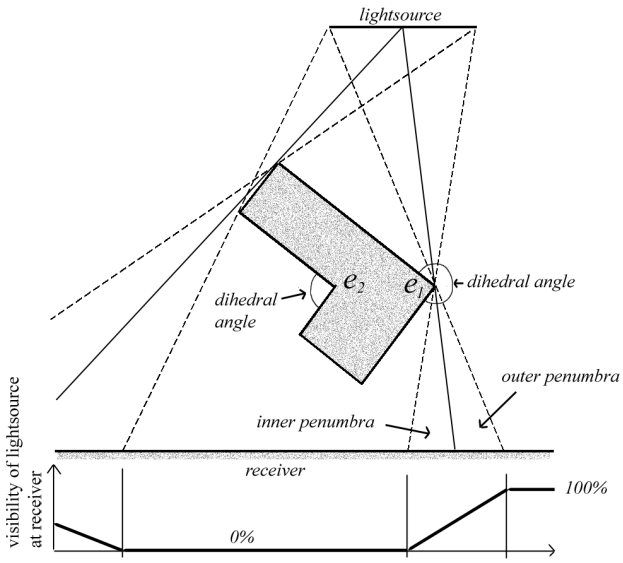


Fig. 1. A simple concave object casts a shadow on a plane.

As will become clear later, the last point is equivalent to being able to distinguish between so-called "inner" penumbrae and "outer" penumbrae.

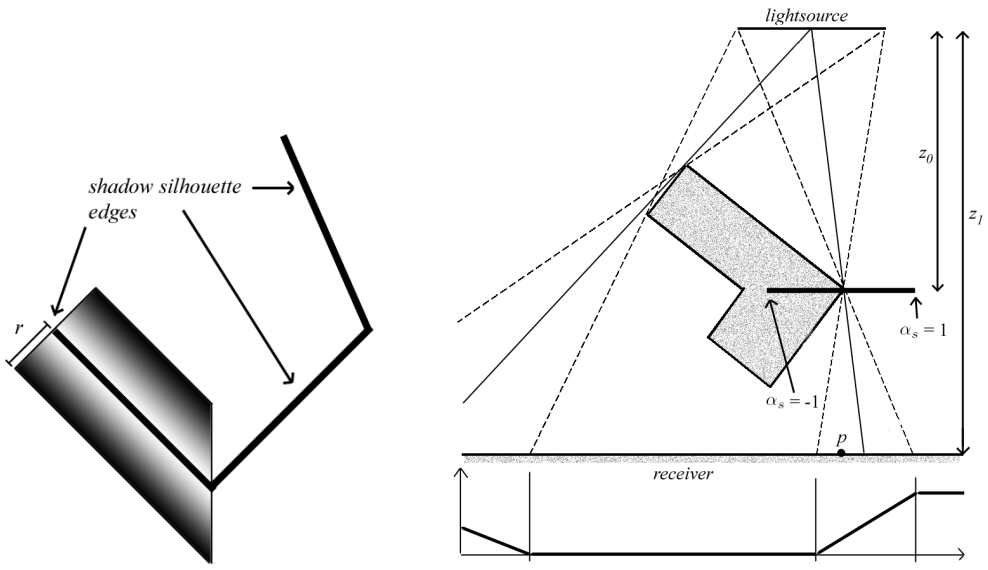
Many soft shadow algorithms construct explicit representations of penumbrae, by extending shadow silhouette edges outwards in light space [Chan and Durand 2003] [Wyman and Hansen 2003] [Haines 2001] using geometric primitives. We will also build explicit penumbra representations, but in contrast to other approaches we will (i) construct these in image space, and (ii) build them in a way that allows us to render both inner penumbrae as well as outer penumbrae.

2. SMOOTH PENUMBRA TRANSITIONS

2.1 Preliminaries

Consider Figure 1, which is a cross section of a very simple scene consisting of one occluder and a receiver. We see that edge e_1 on the right casts a hard shadow boundary onto the receiver. This edge is a silhouette edge as seen from the centre of the area lightsource; it is shared by two faces, only one of which is facing the light. Edge e_2 is also a silhouette edge, however, it does not cast a shadow boundary. The main observation is that shadow boundaries are cast only by those silhouette edges, such as e_1, for which the *dihedral angle* is bigger than π radians. We will refer to such edges as *shadow silhouette edges*. Note that with this construction, the edge below and to the left of e_2 is a silhouette edge, and its dihedral angle implies that it should also cast a shadow. However, because it lies in shadow itself it will not be classified as such.

For area lightsources, shadow boundaries are not hard, but rather they are "spread out" to a penumbra region. (From hereonafter, when we refer to an area



(a) (b)

Fig. 2. (a) Three silhouette edges and a Skirt belonging to one (top view). (b) Calculating a point's visibility value (side view). The thick black line represents a sideview of a Skirt.

lightsource, we will mean a disk-shaped lightsource with radius, r.) We will distinguish between two types of penumbra regions: the outer penumbra, which separates the completely lit region from the hard shadow boundary; and the inner penumbra which separates the umbra from the hard shadow boundary.

We also note that the inner penumbra is situated where an umbra would have been, had the scene been lit by a point lightsource. As the inner penumbra increases in size (with an increase in distance between receiver and occluder) the umbra decreases in size accordingly. This gives rise to the "softening" effect of the cast shadow the further a receiver is removed from its occluder.

2.2 Introducing shadow skirts

As has been shown by other researchers, it is possible to construct special "primitives" that help us to detect penumbra regions; smoothies and the cones and sheets of penumbra maps [Chan and Durand 2003] [Wyman and Hansen 2003] are examples of primitives that have been used in the past. In contrast to these approaches, we construct our primitives using image processing techniques. Nevertheless, they still have a very intuitive *geometric* interpretation: Figure 2 a illustrates our penumbra or *Skirt*, for a single shadow silhouette edge; each shadow silhouette edge is associated with a unique *Skirt*. We note that a *Skirt* has the following three properties:

(1) Its width is fixed to twice the radius of the lightsource;

0.5	1.0	0.5
1.0	0.0	1.0
0.5	1.0	0.5

Fig. 3. The 3×3 construction filter. Note how the centre entry is 0; we explicitly add the corresponding texel value to the filter sum, which guarantees us that shadow silhouette edges always have the value 1.

(2) it has a grey value, α, which is 1 (i.e. maximum intensity) at its associated shadow silhouette edge, and which fades to 0 (i.e. minimum intensity) towards both its lateral edges;
(3) it contains depth information.

Figure 2 b shows how to calculate the shadow value of a point in the penumbra region. Such a point, $\mathbf{p}$, is "occluded" by a *Skirt*, and must therefore lie in a penumbra region. We calculate $\mathbf{p}$'s visibility value $V(\mathbf{p}) \in [0, 1]$ as follows:

$$V(\mathbf{p}) = \frac{1}{2}\left(\frac{\alpha_s}{(1 - \frac{z_0}{z_1})} + 1\right) \tag{1}$$

where $\alpha_s \in [-1, 1]$ is the transformed grey value, α, of the *Skirt* corresponding to $\mathbf{p}$:

$$\alpha_s = \begin{cases} (\alpha - 1) & \text{if } \mathbf{p} \text{ is in shadow,} \\ (1 - \alpha) & \text{otherwise.} \end{cases} \tag{2}$$

Note that Eq. (1) incorporates the ratio of distances between receiver, occluder, and lightsource. This is done for the same reason as with smoothies [Chan and Durand 2003], and allows us to calculate the correct penumbra size for any point on the receiver. It is important to note that with this construction, in image space, penumbra regions can never be wider than the width of a *Skirt*.

3. ALGORITHM

3.1 Constructing the *Skirt*s

Before we can construct *Skirts*, we need to find all shadow silhouette edges in the scene. A very useful property of a shadow silhouette edge is that – for non-intersecting geometry – it is a line of discontinuity in the shadow map [McCool 2000]. Therefore, by applying an edge-detection filter to the shadow map we can identify all shadow silhouette edges. We store these edges with a grey value of 1 in an otherwise black *Skirt* buffer. We then construct the *Skirts* by iteratively applying a special 3×3 filter to this buffer. Figure 3 illustrates the entries of the filter kernel. With each iteration, we widen every *Skirt* in the buffer by 1 shadow map texel in all directions. We perform as many iterations as the radius of the

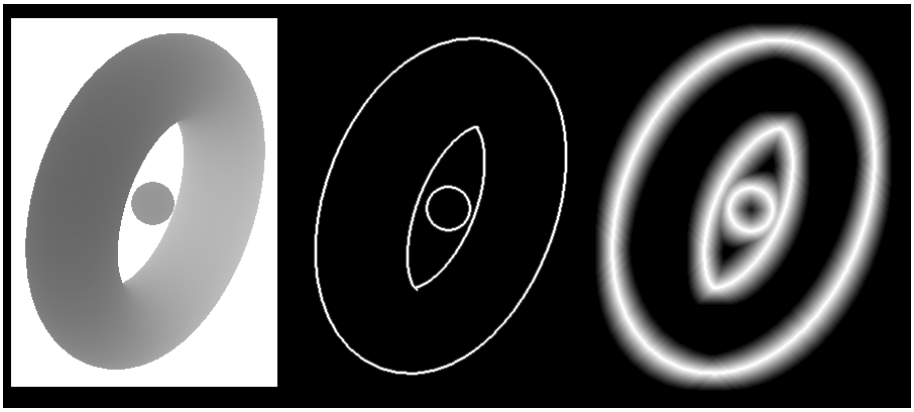


Fig. 4. An example of a shadow map (left), its shadow silhouette edges (middle), and the associated final Skirts buffer (right).

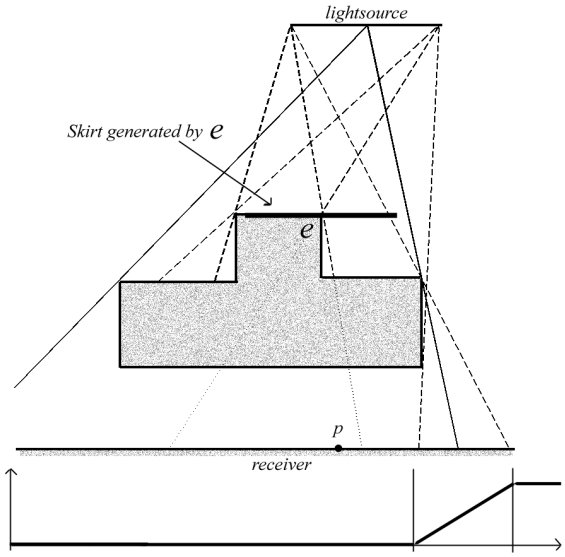


Fig. 5. A case where point **p** is incorrectly classified as lying in an inner penumbra.

lightsource, r. Figure 4 shows examples of the different maps used in the *Skirt* creation process. The following procedure outlines the filter algorithm for a single texel, t.

```
ConstructSkirt( texel t, lightradius r )
b <- current Skirt buffer value at t
F_min <- shadow map depth at t
S <- 0
for each texel t' that surrounds t do
```

```
        b' <- skirt buffer value at t'
        if b' > 0 then S <- S + Filter[t']
        F_min = minimum(F_min, shadow map depth at t')
    endfor
    return (b + (S/(r*6))) and F_min
    end
```

The procedure returns two values. Both these values are stored in the *Skirt* buffer. The first value is the new α value of the *Skirt* at texel t (see Section 2). The second value is the minimum shadow map depth of all 8 texels in the immediate neighbourhood of t, and t *itself*. This value is the z_0 used in Eq. (1).

3.2 Rendering the final scene

During the final rendering pass, we modify the existing shadow map algorithm to return a "smooth" visibility value $V(\mathbf{p})$ for each rendered pixel, $\mathbf{p}$. First we obtain a binary visibility value $V_s(\mathbf{p})$ using the shadow map. Our next step depends on the outcome of this value: If $V_s(\mathbf{p}) = 1$, we calculate the final visibility value using Eq. (1), and we are done; if $V_s(\mathbf{p}) = 0$ we perform an extra test that takes care of a special case which occurs in certain lighting configurations with concave objects, as illustrated in Figure 5. Here $V(\mathbf{p}) = 0$, and simultaneously $\mathbf{p}$ is occluded by a *Skirt*. $\mathbf{p}$ is therefore classified as lying in an inner penumbra. This is obviously wrong, and $\mathbf{p}$ should be in umbra. To solve this, we construct a disk with radius r_p and with centre at $\mathbf{p}$ [Valient and de Boer 2004] [Brabec et al. 2002]. If this disk is completely shadowed, $\mathbf{p}$ is considered to be in an umbra, and $V(\mathbf{p}) = 0$. Otherwise, we calculate the final visibility value using Eq. (1). The entire strategy is embodied in the following procedure.

```
VisibilityTest( point P, lightradius r )
F <- shadow map value at P
if F >= P.z then
    if P occluded by Skirt
        return V calculated using Eq. 1
    else
        return 1
endif
if P occluded by Skirt
    D <- disk with centre at P and radius rp = r * ((z1-z0)/z0)
    if D is not in shadow then
        return V calculated using Eq. 1
endif
return 0
end
```

3.3 Second-depths

As was shown by Brabec et al., the disk approach may fail for points on receivers that are of non-constant z in light space. There are several ways of alleviating this problem [Brabec et al. 2002] [Wang and Molnar 1994] [Valient and de Boer 2004], but it is out of the scope of the paper to subject these to a thorough scrutiny. None

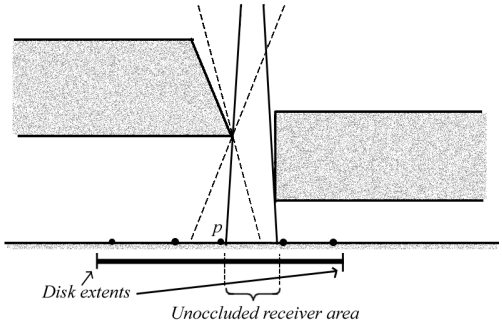


Fig. 6. A point, **p**, for which the samples in the disk all miss the unoccluded region, resulting in **p** being classified as lying in umbra.

of the existing solutions are entirely fail-safe; however, we note that the cases in which our algorithm suffers from this issue is only a subset of the cases for which these solutions were originally designed. Bearing this in mind, we will test the disk against second-depth occluders [Wang and Molnar 1994]. Unfortunately, this will not remove the problem for receivers that are too thin [Wang and Molnar 1994]. The original authors propose using so-called "virtual" samples, but they point out a case in which this additional approach may still fail.

3.4 Summary

We can now summarise our algorithm in the following five steps:

(1) Render the shadow map, as seen from the centre of the area lightsource [Williams 1978] [Brabec et al. 2002];

(2) generate the shadow silhouette edges by applying an edge detection filter to the shadow map, and storing the result in a *Skirt* buffer;

(3) construct the *Skirt*s by iteratively calling `ConstructSkirt` for each texel in the *Skirt* buffer;

(4) render the shadow map again, this time using second-depth occluders;

(5) render the final scene using the modified shadow map test, `VisibilityTest`; and the second-depth shadow map.

Note that step (1) and (4) can be combined into one step.

4. IMPLEMENTATION

We have implemented our algorithm in DirectX9 on an ATI Radeon 9800SE graphics card, which supports `vs_2_0` and `ps_2_0` shaders. Steps (1) through to (5) of our algorithm are performed using vertex and pixel shaders; the host processor coordinates between these steps. Unfortunately, this shader version does not support dynamic and static branching. Therefore, `if`-statements are unrolled, and the resulting shaders are executed in an entirely linear fashion; in effect, the disk test is performed for every pixel on the screen. We store linear light space depths in

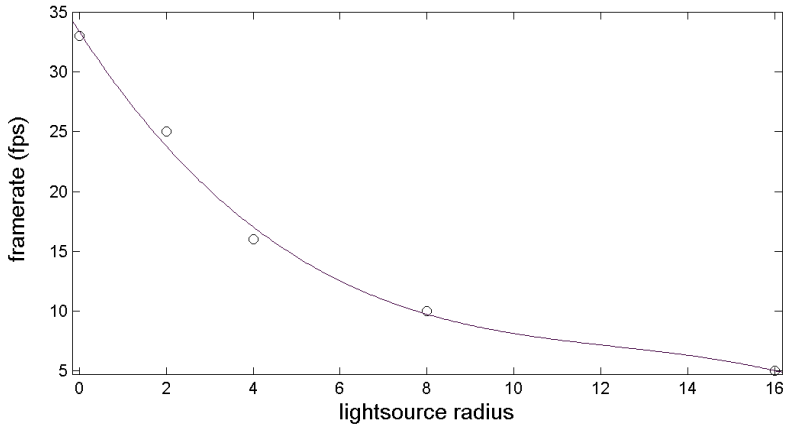


Fig. 7. Performance figure for the "torus and sphere" scene.

a shadow map [Brabec et al. 2002] with 16-bit depth precision. A simple thresholded magnitude edge detection filter kernel, similar to McCool's [McCool 2000] is used, and the resulting edges are stored in the red channel of a floating point `D3DFMT_G16R16` render target. We then construct the *Skirt*s by applying the filter to the buffer, and alternating between this buffer and an extra `D3DFMT_G16R16` buffer as source and destination (i.e. a ping-pong buffer). The final buffer then has in its red channel the grey values of the *Skirt*s, and its green channel contains the *Skirt*s' depth information.

We have implemented the disk used for inner penumbra detection as 12 samples taken within this disk; the locations of these are determined according to a Poisson disk distribution [Cook 1986] [Valient and de Boer 2004]. We observe that this small number of samples will result in under-sampling artifacts in certain situations in which *Skirts* overlap, as shown in Figure 6. This issue is resolved by testing every possible shadow map sample that falls within the disk, similar to Brabec et al.'s method. However, current hardware puts a limit on the number of texture reads that can be performed in a single pixel shader, which forces us to use a fixed number of samples.

We alleviate the problem by multiplying the disk radius, r_p, by a light radius factor, $f \in (0, 1]$. The denominator in Eq. 1 is multiplied by the same factor. Unfortunately, the smaller the value of f, the smaller the width of the resulting inner penumbra region.

5. RESULTS

Figure 7 shows the performance of our implementation for a simple scene consisting of a torus, a sphere, and a plane. Since the key steps of our algorithm operate entirely in image-space, the geometric complexity of the scene is of no importance. The size of the lightsource determines the width of the *Skirt*s; hence, this does affect our algorithm's performance. The units of the x-axis in the figure can accordingly be

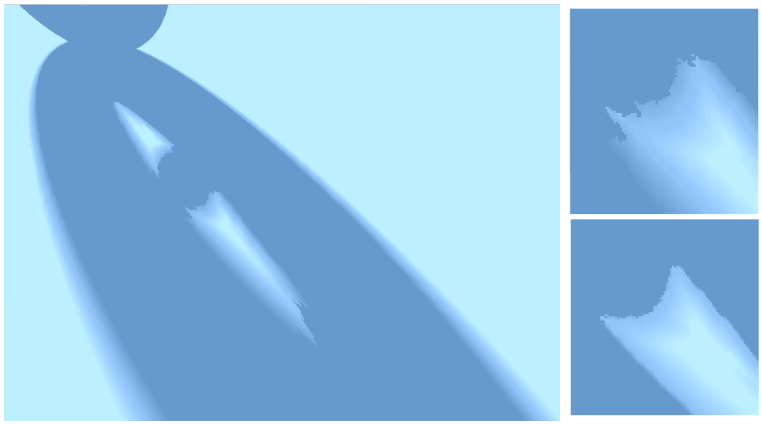


Fig. 8. An artifact at overlapping Skirts due to stochastic sampling (left). The top right figure shows an enlargement of the artifact, and the bottom right image shows a light radius factor $f = 0.2$.

interpreted as the number of "ping-pong" iterations (i.e. rendering passes) required in step (3).

The artifact due to the fixed number of disk samples used is illustrated in Figure 8. To completely solve this particular case, a light radius factor of 0.2 was used. Notice how the inner penumbra has drastically decreased in size; this accounts for the rather harsh transition between the inner penumbra and the outer penumbra, which degrades overall shadow quality. In practice, we have found that a light radius factor between 0.6-0.9 yields the best results with the least amount of degradation in overall shadow quality.

Finally, Figure 9 and Figure 10 show a comparison of our algorithm with the fractional-disk algorithm [Valient and de Boer 2004] and smoothies [Chan and Durand 2003].

6. DISCUSSION

Our algorithm bears some similarities with existing approaches. A similar image-space method has been proposed by Kirsch et al. [Kirsch and Doellner 2003]. Their implementation also relies on constructing an auxiliary buffer (in their case a so-called *shadow width* map) in several passes; however, every pass of our algorithm roughly accounts for two passes of Kirsch's to generate the same penumbra width, which makes our method more efficient. Also, Kirsch's implementation relies on a user-defined distinction between blockers and receivers, whereas our method does not need this distinction. Another crucial difference is that our method does not suffer from the artefacts described in Section 5 of Kirsch's paper. Our use of the fractional disk approach accounts for this. Lastly, their method does not support inner penumbrae.

The disk approach that is used for classifying inner penumbrae is equivalent to the disk used by Valient and de Boer [Valient and de Boer 2004]. Their algorithm also constructs auxiliary buffers which are similar to ours; however, these buffers

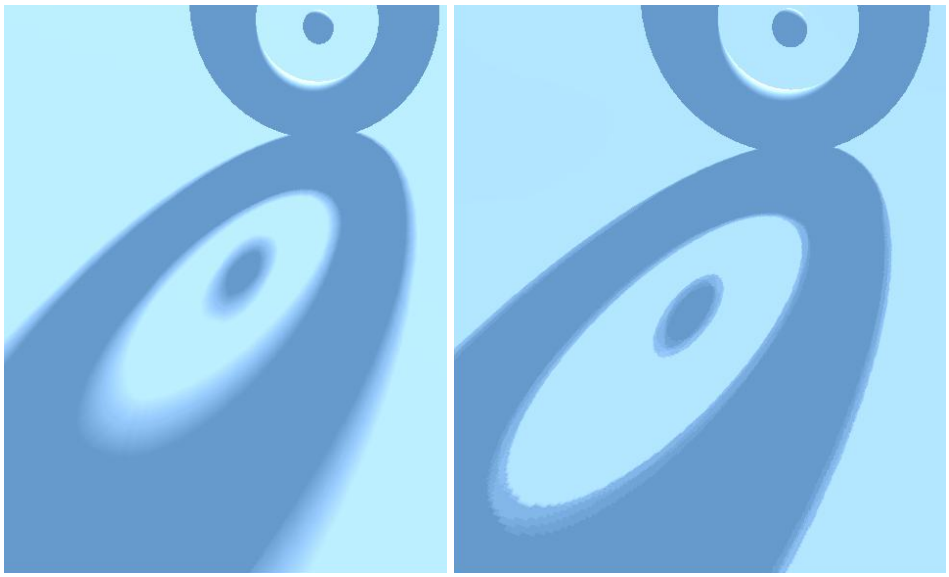


Fig. 9. Visual comparison of our algorithm with fractional-disk approach, shadow map size is 512×512 and the lightsource $r = 8$. (a) Our algorithm (b) fractional-disk

serve a different purpose. Unlike our method, their approach does not produce smooth penumbra transitions.

Brabec et al. [Brabec et al. 2002] also use a disk. In contrast to our method, this disk must be used for both inner as well as outer penumbrae. Therefore, our approach may benefit from scenes in which shadows occupy only a relatively small region.

We have experimented with improving the performance of step (3), by using a bigger *Skirt* construction filter, and separating it into two 1-dimensional filters [Mitchell et al. 2003], both as wide (or tall) as twice the lightsource's radius. We can then reduce step (3) to a 2-pass filter process. Unfortunately, a naive implementation of this proved not to give satisfactory results. A more sophisticated filter procedure is needed, but this remains future work.

7. ACKNOWLEDGEMENTS

We would like to thank Steve Hill for suggestions, and Ralph Egas and Gino van den Bergen for proofreading this paper.

REFERENCES

Brabec, S., , and Seidel, H.-P. 2002. Single sample soft shadows using depth maps. *Graphics Interface*, 219–228.

Brabec, S., Annen, T., and Seidel, H.-P. 2002. Practical shadow mapping. *Journal of Graphics Tools 7,* 4, 9–18.

Fig. 10. Visual comparison of our algorithm with smoothies. Shadow map size is 256×256 and the lightsource $r = 12$. (a) Our algorithm (b) smoothies. Notice how with our approach, the umbra shrinks and has disappeared almost completely at the head's shadow. Model courtesy of Playlogic Game Factory BV, copyright 2004.

Chan, E. and Durand, F. 2003. Rendering fake soft shadows with smoothies. *Proceedings of Eurographics Symposium*, 193–194.

Cook, R. 1986. Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5,* 1, 185–192.

Crow, F. 1977. Shadow algorithms for computer graphics. *Computer Graphics (Proceedings of SIGGRAPH '77)*, 242–248.

Haines, E. 2001. Soft planar shadows using plateaus. *Journal of Graphics Tools 6,* 1, 19–27.

Kirsch, F. and Doellner, J. 2003. Real-time soft shadows using a single light sample. *International Winter School of Computer Graphics, Journal of WSCG 11,* 2, 255–262.

McCool, M. 2000. Shadow volume reconstruction from depth maps. *ACM Transactions on Graphics 19,* 1, 1–26.

Mitchell, J., Ansari, M., and Hart, E. 2003. Advanced image processing with directx 9 pixel shaders. *ShaderX2 - Shader Programming Tips and Tricks with DirectX 9*, 439–464.

Valient, M. and de Boer, W. 2004. Fractional-disk soft shadows. *ShaderX3 - Advanced Programming with OpenGL and DirectX9*.

Wang, Y. and Molnar, S. 1994. Second-depth shadow maps. *UNC-CS Technical Report TR94-019*.

Williams, L. 1978. Casting curved shadows on curved surfaces. *Computer Graphics (Proceedings of SIGGRAPH '78) 12,* 3, 270–274.

Wyman, C. and Hansen, C. 2003. Penumbra maps: Approximate soft shadows in real-time. *Proc. of the 14th Symposium on Rendering*, 202–207.