Real-Time Soft Shadows Using a Single Light Sample

Florian Kirsch
Hasso-Plattner-Institute for Software Engineering
at the University of Potsdam
Prof.-Dr.-Helmert-Straße 2-3
D-14482 Potsdam, Germany
kirsch@hpi.uni-potsdam.de

Jürgen Döllner
Hasso-Plattner-Institute for Software Engineering
at the University of Potsdam
Prof.-Dr.-Helmert-Straße 2-3
D-14482 Potsdam, Germany
doellner@hpi.uni-potsdam.de

ABSTRACT

We present a real-time rendering algorithm that generates soft shadows of dynamic scenes using a single light sample. As a depth-map algorithm it can handle arbitrary shadowed surfaces. The shadow-casting surfaces, however, should satisfy a few geometric properties to prevent artifacts. Our algorithm is based on a bivariate attenuation function, whose result modulates the intensity of a light causing shadows. The first argument specifies the distance of the occluding point to the shadowed point; the second argument measures how deep the shadowed point is inside the shadow. The attenuation function can be implemented using dependent texture accesses; the complete implementation of the algorithm can be accelerated by today's graphics hardware. We outline the implementation, and discuss details of artifact prevention and filtering.

Keywords

Soft Shadow Algorithms, Shadow Mapping, Graphics Hardware, Hardware-Accelerated Rendering

1. INTRODUCTION

Soft shadows play an important role for perceiving the arrangement of objects in a 3-dimensional scene. The location of a shadow mediates the relative positions of a light source, shadow blocker and receiver. The sharpness of the shadow's penumbra clarifies distance relationships between blocker and receiver. Soft shadows also provide a far more realistic impression of an image compared to hard shadows since in real-life there are no perfect point lights.

Currently, no algorithm is known that renders physically correct and dynamically updated soft shadows in arbitrary scenes and in real-time. This is in contrast to hard shadow algorithms. Dynamic soft shadow algorithms require simplifications to achieve interactive frame rates, including:

- Restrictions of the type of occluding and shadow-receiving shapes.
- Forbidden self-shadowing.
- Disregarding shape and size of an area light, and reducing the shadow computation on a single light-source position.
- Abandoning the goal of physical correctness, aiming only for convincing soft shadows.

The algorithm we present in this paper belongs to the group of depth-map algorithms. It is suited for real-time rendering, and, therefore, sacrifices physical correctness – it samples a single light position. Requirements of participating shapes are rather general if compared to other known real-time soft shadow algorithms:

- Any shape can be receiver of shadows without the need to treat every planar surface of the shape separately.
- Any shape can block light with the following restriction: In order to avoid artifacts it is advantageous if different blockers overlapping in light space do not differ too much in z-direction.
- Self-shadowing is possible; i.e., a shape can cast a shadow upon itself. However, these shadows will most likely have no penumbra.

We calculate the shadow as attenuation of the unshadowed light. The attenuation value at a point to be shaded is given by the result of a bivariate function:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Journal of WSCG, Vol.11, No.1., ISSN 1213-6972
WSCG'2003, February 3-7, 2003, Plzen, Czech Republic.
Copyright UNION Agency – Science Press

The first argument, the *shadow depth*, is the distance between the occluding point and the possibly shadowed point; the second, the *shadow width*, denotes the distance from a point to the nearest point that is geometrically illuminated. With today's graphics hardware, the attenuation function can be evaluated using a dependent texture access.

Our method is very fast because the only significant overhead compared to a hard shadow depth-map algorithm is the generation of the texture map containing shadow-width values. This is a simple operation that can be performed by the CPU or, through texture compositing, by graphics hardware. Our soft shadow algorithm appears to be one of the first of its kind that renders shadows entirely using today's graphics hardware. Therefore, even for dynamic shadows, frame rates that exceed 60 fps are achieved for 3D scenes of moderate complexity.

This paper proceeds in discussing related works. In Section 3, the algorithm is discussed from a theoretical point of view, and properties of the attenuation function are presented. Section 4 describes our implementation, Section 5 contains practical hints for quality improvement, and Section 6 gives a performance comparison. The paper concludes in Section 7.

2. RELATED WORK

Hard Shadow Algorithms

To render hard shadows, two kinds of algorithms are widely used for real-time calculation of dynamic shadows. The first approach are shadow volume algorithms, first described by Crow [Cro77]. Here, shadow calculation takes place in object space by means of invisible semi-infinite shadow volumes. These are built by extruding occluding polygons into direction of the light. A pixel inside a shadow volume is known to be in shadow. The shadow volume algorithm was adapted to real-time graphics hardware by Heidmann [Hei91].

Depth-map shadow algorithms, first proposed by Williams [Wil78], are the second kind. Depth-maps are depth images of the scene rendered from the position of light in a pre-processing step. In the main rendering pass, for shadow testing, a point is transformed from camera space into light space. If the resulting z-value is greater than the corresponding value in the depth-map, then the point is considered to be in shadow. Projective shadows were the key to adapt depth-map shadows in real-time rendering systems [Seg92].

Williams' depth-map shadows can be simplified to a less general algorithm that is sometimes referred as projective shadows [Huu99]. During pre-processing, shadow blockers are rendered from the light source with black color; the background color is white. For shadow calculation, the resulting image – the *occlusion map* – is projected from the light source onto receiving shapes, and the resulting texture values directly modulate the shading intensities. This algorithm obviously does not permit self-shadowing. Despite, it is often used in practice because of its simplicity and its low hardware requirements. Our soft shadow algorithm is related to projective shadows, since it may use an occlusion map in order to generate the shadow-width map (see Section 4).

Soft Shadow Algorithms

It is possible to render soft shadows using hard shadow algorithms of either category (e.g., Brotman and Badler [Bro84]). To simulate an area light, several point light sources are distributed over its surface, calculating the shadow for each of these lights, and accumulating the resulting shading intensities. In practice, this idea is hardly usable due to the following reasons:

- It requires a significant number of point light samples to achieve smooth intensity changes in the penumbra: The number of different intensities exceeds the number of light source samples by only one.
- With today's real-time graphics hardware, an accumulation buffer is required for good visual quality: Many light samples reduce the intensity resolution available for each light sample. Using a common frame buffer providing 8 bits per color channel leads to banding artifacts. Accumulation buffers provide enough accuracy but are not accelerated by today's graphics hardware; they are emulated in software and, therefore, slow down rendering performance significantly.

Apart from the latter accumulation method, few shadow volume algorithms address soft shadow generation for interactive rendering. One exception is Chin and Feiner's Area SVBSP tree [Chi92] that, for static polygonal environments, stores penumbra and umbra regions in a pair of BSP trees.

Most other algorithms for soft shadows are based on variants of Williams' depth-map algorithm. We first cover algorithms to reduce aliasing effects by softening the shadow boundary. To do so, Reeves et al. proposed 'percentage closer filtering' [Ree87], which is nowadays integrated in rendering hardware that provides dedicated support for depth textures, but which can be also implemented deploying standard graphics hardware without relying on dedicated hardware support [Bra01]. When using projective shadows, blurring the occlusion map can create a similar effect. These methods sometimes are cited in the context of soft shadows. Although

generating soft shadow edges, they cannot simulate an important property of soft shadows though: The distance between occluding shape and shadowed shape is *not* taken into account, so the size of the penumbra is invariant.

Heckbert and Herf aim to calculate physically correct soft shadows by generating a radiance texture for each polygon [Hec97]. Such a texture contains all illumination information for the polygon, generated by sampling many light sources and accumulating the resulting intensities. In practice, the algorithm has the same limitations as Brotman and Badler's. It is possible to employ convolution operations to approximate the radiance textures, thus significantly reducing the number of light samples [Sol98]. This refinement, however, seems to be inappropriate for real-time rendering since for every radiance texture, e.g., every receiving polygon, an FFT must be calculated.

Haines describes soft shadows using plateaus [Hai01]. For each planar surface in shadow, a texture is created that contains the projection of the blocker. To simulate the penumbra, a cone is rendered for each silhouette edge of the blocker, and added to the texture.

Heckbert's at al. and Haines' algorithms are seriously restricted by the fact that for every planar surface in shadow, a separate texture is required. Our soft shadow algorithm has the advantage that it requires only a single texture for the whole scene, even if curved shapes are shadowed.

Heidrich describes a soft shadow algorithm suited for linear light sources [Hei00]. For simple scenes, his algorithm requires only two light samples. For the light source a depth map and a visibility map is built. The visibility map contains the percentage of the whole light that reaches a point in the scene. This somehow resembles our notion of a shadow-width map. However, an edge detection process is needed for the generation of the visibility map, scaling badly with scene complexity. In our algorithm, generation time of the shadow-width map is independent of the scene geometry.

Single Sample Soft Shadow Algorithms

Parker et al. compute soft shadows using a single light sample [Par98]. Their algorithm is suitable to ray tracing: around an occluder, a bigger object is constructed whose opacity lowers towards the outer boundary. The contribution of a light is modulated by the opacity term, while also taking into account the relative positions of occluding and receiving shapes and the light source.

Recently Brabec et al. adopted Parker's algorithm to work on depth maps [Bra02]. During shadow calculation, after transforming a point into light space, they search the neighborhood of the transformed point in the depth map. If the point is lit but in the neighborhood a blocked pixel is found, the point is darkened a bit, taking into account the distance to the next blocked pixel. If otherwise the point is shadowed but in the neighborhood unblocked pixels are found, the point is not fully darkened but illuminated proportional to the distance to the next unblocked pixel.

Brabec's algorithm is suited for interactive frame rates. The shadow calculation is performed on the CPU, since the graphics hardware is not able to do the neighborhood search. The algorithm allows no self-shadowing. Otherwise, for its performance, the quality of shadows is good, despite the fact that they are not physically correct.

Our algorithm is strongly related to Brabec's algorithm: Instead of searching the neighborhood in the depth map, we look up the distance to the next unblocked pixel in a second texture, the shadow-width map. This allows for shadow computation using graphics hardware, lifting rendering performance to real-time speed. As a limitation, shadows can be rendered only in such areas where a pixel is geometrically in shadow. Hence, our shadows are smaller than their physically correct counterparts – typically not noticed by the human viewer.

3. ALGORITHM

Pre-processing

During a pre-processing step, our algorithm constructs two distinct images from the point of view of the light source, which are accessed later to calculate the intensity of a fragment:

- The first image is the depth-map as known in Williams' shadow algorithm.
- The second image we call *shadow-width map*. It is constructed as follows: Assume a plane A behind all shadow blockers that is perpendicular to the light's direction. Then, the *shadow width* (sw) of a point in the plane is given by its distance to the nearest point on the plane that is geometrically lit by the light – i.e., the shadow width is the distance to the next unblocked pixel.

$$\forall p \in A: \quad sw(p) = \min_{\substack{q \in A \\ \text{illuminated by light}}} \|q - p\|$$

A linear transformation can transform any point p in object space into light space, resulting in coordinates $p' = (x', y', z')$. We use these coordinates to lookup the corresponding value in the depth-map $z(x', y')$ resp. shadow-width map $w(x', y')$. Then, we define the *shadow depth* of p as

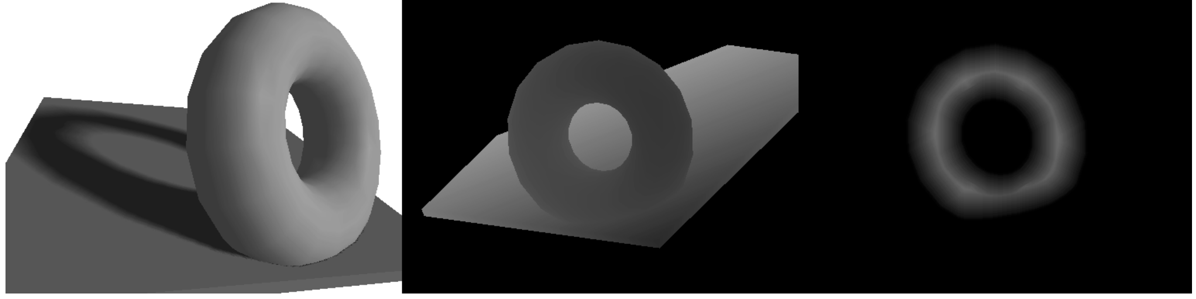

Figure 1. Left: torus with soft shadow. Center: depth-map. Right: shadow-width map.

$$sd(p) = z' - z(x', y')$$

and the *shadow width* of p as

$$sw(p) = w(x', y') .$$

The shadow depth can be interpreted as distance from p to the most distant point of a shape blocking the light of p. The shadow width can be explained as follows: If the light is a distant light, $sw(p)$ measures the distance to the nearest point that is geometrically illuminated. For point lights, $sw(p)$ approximates the steradian with respect to the light source between the nearest point fully illuminated and p (see Figure 2). In both cases, we use $sw(p)$ to measure how deep a point is in shadow.

Note that the shadow width of a point is zero if the light source is visible from that point. In this case, also the shadow depth of the point is zero since no object casting a shadow is nearer the light than the point itself. This means that $sd(p) = 0$ automatically implies $sw(p) = 0$ and vice versa; and if $sd(p) \neq 0$, $sw(p)$ must be $\neq 0$ either.

Shadow Calculation

To calculate the illumination intensity for a given point p, shadow depth and shadow width are used as arguments of the *attenuation function* $I(sd(p), sw(p))$, whereby

$$I : R_+ \times R_+ \to [0,1] .$$

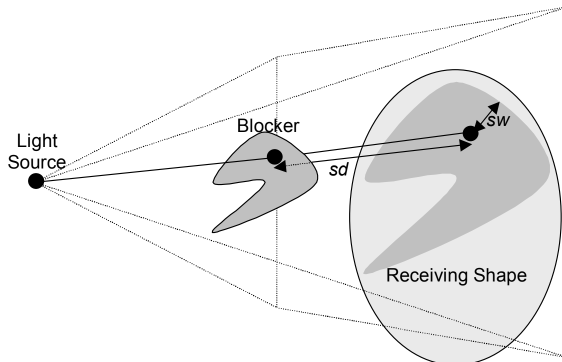

Figure 2. Geometric interpretation of shadow width and shadow depth.

$I(sd(p), sw(p))$ calculates the attenuation of the light source caused by a possible shadow at p. The co-domain of I is $[0,1]$, where zero denotes that the light is fully attenuated, i.e., a point is in the umbra of the shadow, and one denotes that light is not attenuated, i.e., a point is fully lit.

To provide soft shadows, I has the following properties:

1. $I(sd, sw) = 1$ if $sd = 0$ or $sw = 0$, to ensure that points are fully lit if they are visible from the light source.
2. $I(sd, sw) = I(sd', sw')$ if $\frac{sw}{sd} = \frac{sw'}{sd'}$. This property ensures that when scaling both shadow depth and shadow width by the same amount, the light attenuation remains unchanged.
3. $I(sd, sw) \leq I(sd, sw')$ if $sw > sw'$. If also $I(sd, sw') > 0$, we furthermore demand $I(sd, sw) < I(sd, sw')$. This property ensures that when approaching a shadow boundary, the light gets strictly monotonic brighter. In the umbra, the light obviously cannot get any darker than zero, so if $I(sd, sw') = 0$, also $I(sd, sw) = 0$ must hold.
4. For smooth attenuation, i.e., smooth intensity changes, $I(sd, sw)$ is continuous, which is not physically correct but feasible from a human perceptional point of view.

The following property directly follows from 2. & 3.:

5. $I(sd, sw) \leq I(sd', sw)$ if $sd < sd'$, respectively $I(sd, sw) < I(sd', sw)$ if additionally $I(sd', sw) > 0$. This ensures an important visual property of soft shadows: Where a shadowed polygon approaches the light blocker, the shadow boundary gets sharper.

Obviously, the choice of I is not unique, but depends of the attenuation changes at the boundary of the shadow. In general, $I(sd, sw)$ can be any function of the form

$$I(sd, sw) = \left\langle \begin{array}{ll} 1 & \text{if } sd = 0 \\ f\left(\frac{sw}{sd}\right) & \text{otherwise} \end{array} \right\rangle, \text{ clamped to } [0,1],$$

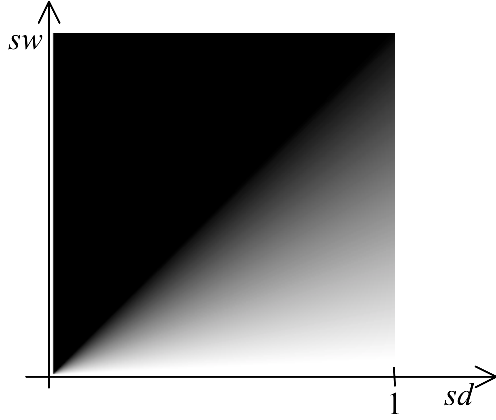

Figure 3. ***I(sd, sw)*** **with** $c_{scale} = 1$ **and** $c_{bias} = 0.05$**.**

where $f(x)$ is a continuous and strictly monotonic decreasing function that converges to one for $x \to 0$.

In our implementation, we use the attenuation function:

$$I(sd, sw) = \left\langle \begin{matrix} 1 & \text{if } sd = 0 \\ 1 + c_{bias} - c_{scale} * \frac{sw}{sd} & \text{otherwise} \end{matrix} \right\rangle$$, clamped to $[0,1]$ (see also Figure 3).

c_{bias} and c_{scale} are non-negative constants. c_{scale} adjusts the softness of the shadow. Let sd be fixed; if c_{scale} is large $I(sd, sw)$ becomes zero even for small sw. In this case, yet close to the geometrical shadow boundary the light is fully darkened. Consequently, a large value of c_{scale} shrinks the size of the penumbra, whereas a small value enhances its size.

A valid value for c_{bias} is zero. In certain cases, we found a value slightly greater than zero advantageous to reduce intensity artifacts at the outer border of the soft shadow (see Section 5).

4. IMPLEMENTATION

To implement the presented algorithm in a way that it can benefit from acceleration by graphics hardware, the following steps must be taken for each fragment f resulting from rasterizing primitives:

1. The shadow depth value sd must be available for f.
2. The shadow width value sw must be available for f.
3. The result of the attenuation function I with sd and sw as arguments must be calculated. It is multiplied with the regular fragment's color resulting in the final color, which is copied into the frame-buffer.

Attenuation Function

Step 3 is the easiest to solve given today's graphics hardware that supports dependent texture accesses. Suppose the former texture units have been set in a way that sd and sw are determined correctly. Then, $I(sd, sw)$ can be looked up in a lookup-texture containing intensity values.

Generating Shadow Depths

Step 1 is more difficult but can be solved by the approach of Segal et al. [Seg92] and Heidrich [Hei99]. Here, we summarize the approach of Heidrich: The scene is rendered from the light source, applying a texture that encodes the distance to the light source in the alpha channel of the frame-buffer (using texture coordinate generation). The resulting alpha channel is copied into a texture, and represents the depth-map. To calculate sd, this texture is projected from the light source onto the scene in object space. A second texture is applied to the scene, which uses the texture that was formerly used to create the depth-map. Thus, sd is the difference between the fetched texture values of the first and the second texture.

Heidrich calculates the difference of the alpha texture values using texture environment functions. To use the result in a dependent texture lookup, we must calculate it at an earlier stage using texture shaders. This is possible with per-fragment dot-product facilities of current hardware: Instead of encoding the depth values in the alpha channel, we use the green channel. A value of one is stored in the red channel. This texture is projected onto the scene using the first texture unit. In the second texture stage, we calculate the dot-product of the result of the first texture lookup with (*s*, -1, 0), where *s* is generated to hold the distance of a fragment to the light source z'. This results to

$$\begin{aligned} & red * s + green * t + blue * r \\ & = 1 * s - 1 * green \\ & = z' - z(x', y') = sd \end{aligned}$$

Generating Shadow Widths

For Step 2, we calculate a texture that contains shadow width values; it is projected onto the scene to make sw available for all fragments. For this, we generate an *inverted occlusion map* at the same time when generating the depth-map. The occlusion map is rendered from the light source, choosing intensity of one for shadow blockers, and zero for other objects. The blue channel is used to store the occlusion map.

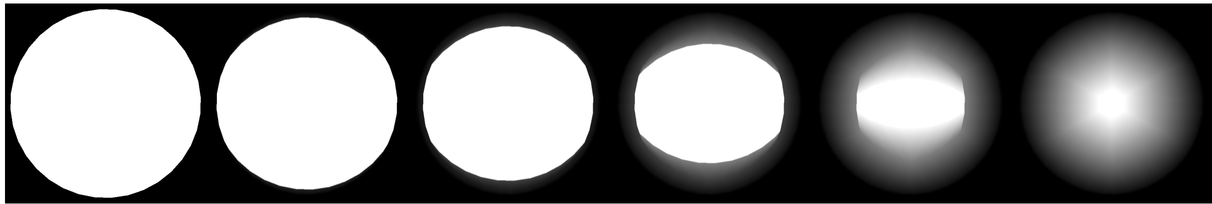

Figure 4. Generation of a shadow-width map from a sphere using texture compositing. Neighbors in 6 directions are sampled. From left to right: inverted occlusion map; result of algorithm after 4, 5, 6, 7, and 8 iterations. The starting difference value is 1/256.

In a post-processing step, the occlusion map can be converted into a shadow-width map. The blue intensity of a texel having a neighbor texel much darker is replaced by the intensity of that neighbor texel plus a small difference value. This is done for all texels in the map, and the whole process is repeated considering different directions for the neighbors. Since the difference value added to a neighbor's intensity always remains the same, the final blue intensity of a texel approximately measures the distance to the nearest texel that was not occluded by a shadow casting object.

When the occlusion map is copied into system memory, this algorithm is easily implemented regarding neighbor texels in the left, right, top, and bottom direction. To avoid the costly frame-buffer copy operation, a variant of the algorithm may be also implemented that uses the graphics hardware. The inverted occlusion map is used as texture. It is bound to all available texture units, and the textures are applied to a quad that covers the whole viewport. One texture is applied unchanged. The other textures are moved to the different neighbor directions, and the small difference value is added to their texels. The intensity of a rendered fragment results from the minimum of all fetched texture values. The process is repeated multiple times, each time employing a new texture holding the actual frame-buffer, and with doubled difference and neighbor distance. The algorithm stops when the difference value is greater than one (see Figure 4).

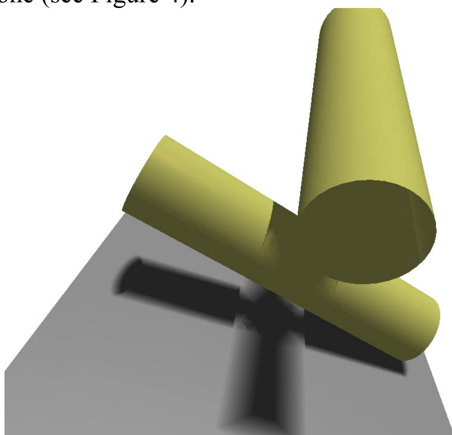

Figure 5. Artifacts due to overlapping blockers.

5. DISCUSSION

Possible Artifacts

Our algorithm handles all shadowed surfaces well. For objects that cast a shadow, problems arise if several light-blocking objects overlap in light space, or several parts of a single concave light-blocking object overlap. In both cases, only a single depth can be stored in the depth-map at a position. This results in shadow-depth discontinuities and to artifacts. As visual effect shadows abruptly get softer along shadow edges that cross other shadow edges (see Figure 5). The effect is disturbing in particular when it suddenly appears and disappears, for instance, in the case of moving objects.

The shadow-width map can also have artifacts due to united areas of projections of different blockers (or parts of blockers) in the inverted occlusion map. The results are shadow-width values that are too high. The visual effect is mostly notable for moving blockers, when shadow widths suddenly increase and the shadow gets darker.

Self-Shadowing

Self-shadowing shapes, e.g., shapes that are both blocker and receiver, are best treated as blocker when generating the inverted occlusion map. For a possibly shadowed point on a light blocker, the shadow width will be too high because the region around the point will enhance the blocker's silhouette in the inverted

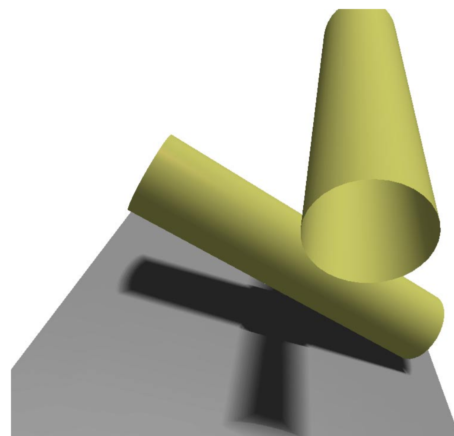

Figure 6. Without self-shadowing, reduced artifacts.

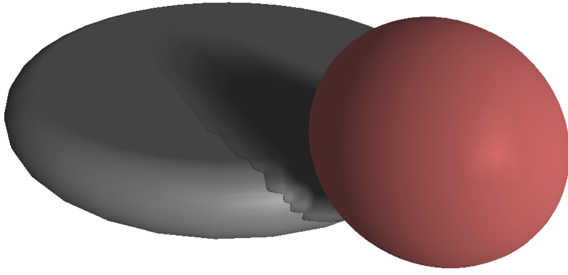

Figure 7. Depth-map resolution of 64×64. Where blocker and shadow receiver are close, artifacts are visible. Everywhere else the shadow is not affected.

Figure 8. The scene used for performance study.

Depth-map resolution	256×256	512×512	1024×1024	2048×2048
Software	61.6	34.3	12.6	3.2
Hardware 1/1 (hard shadows)	97.9	79.8	42.9	17.0
Hardware ¼	86.1 / 77.4	56.4 / 43.6	22.4 / 15.1	7.0 / 4.4
Hardware 1/16	77.5 / 64.4	43.6 / 30.0	15.2 / 9.2	4.4 / 2.5
Hardware 1/64	70.3 / 54.9	35.5 / 22.9	11.4 / 6.6	3.2 / 1.7
Hardware 1/256	64.0 / 47.5	29.3 / 17.8	8.8 / 4.9	2.4 / 1.3

Table 1. Performance in fps. Depth-map and shadow-width map are updated every frame, the shadow-with map with software or differing graphics hardware algorithms. The hardware algorithm samples three (left) or six (right) neighbor texels.

occlusion map. Therefore, the shadow at the point will be darker than expected. In practice, shadows onto self-shadowing shapes often have no penumbra at all.

If we avoid shapes that are both blocker and receiver, the visual effect of depth-map artifacts can be reduced. In this case, the back faces of shadow-casting shapes can be rendered into the depth-map using a 'z-greater' test. When calculating the shadow, the shadow artifacts in the depth-map cause a shadow with harder edge. As illustrated in Figure 6, this strategy leads to visually more convincing results.

Filtering

If the geometry of the occluding shape is simple and linear texture filtering is enabled, even shadow maps with an extremely low resolution can yield an acceptable quality (see Figure 7, which was created using a depth-map resolution of 64×64 texels). Linear filtering, however, has drawbacks since it produces intensity discontinuities at the outer boundary of the shadow. In Figure 7, this effect is visible where disk and sphere are close. The reason for discontinuities is that inside the shadow, the texture coordinates of the texel in the lookup-texture differ completely from the coordinates outside the shadow, which are (0, 0) – linear filtering between them makes no sense. As a future extension of the algorithm, programmable texture filter functions, e.g., as exposed in the GL_NV_fragment_program extension [Kil02], could solve this filtering problem.

6. PERFORMANCE

For our performance study, a Pentium IV with 1.300 Mhz and GeForce3 Ti200 with 64MB graphics hardware was used. For generating the shadow-width map, we distinguished between the software method and the graphics hardware method (Section 4).

The graphics hardware method, we recorded timing results for several starting difference values. This value decides how often the loop to create the shadow-width map is performed (and, on the other side, is responsible for the size of the penumbra). We also distinguished between sampling three and six neighbor texels to create the shadow-width map[1]. Our test scene is depicted in Figure 8. The resolution of the main window was 512×512 pixels. The results are given in Table 1; numbers are in frames per second.

For moderate depth-map resolutions (e.g., 256×256), the algorithm performs in real-time. For higher resolutions, performance is at least interactive.

The results are nearly independent of the canvas resolution. As expected, the resolutions of the depth-map and the shadow-width map determine the rendering speed since those textures are updated every frame. If they are not dynamically updated, the

[1] These numbers are best suited for GeForce3 hardware since GeForce3 can apply four textures at once, and to sample three neighbor texels, three textures and the main texture must be applied.

rendering speed is nearly indistinguishable from common depth-map algorithms generating hard shadows.

Currently, the performance of the software method is limited by the operation that copies frame-buffer contents to main memory. For testing purposes, we disabled the creation of the shadow-width map, leaving enabled the frame-buffer copy. The number of frames per second increased only by about 40% (depth-map resolution: 512×512).

Finally, the performance of the algorithm turns out to be independent of the scene complexity, in particular, if high resolutions for both kinds of shadow textures are used.

7. CONCLUSION

The presented real-time rendering algorithm generates soft shadows of dynamic scenes using a single light sample. As its main advantage only a single texture is required to render shadow of arbitrary complex scenes. In addition, the algorithm supports real-time rendering of shadows in dynamic scenes. For shadows in static scenes, its performance does not differ significantly from common depth-map algorithms generating only hard shadows. Furthermore, the algorithm is fully accelerated by today's graphics hardware. The latter point is important since currently, performance of graphics hardware advances faster than that of CPUs, and, therefore, we expect that our algorithm will scale well with upcoming graphics hardware.

As a direct extension of the depth-map shadow algorithm by Williams, applications using that algorithm can seamlessly integrate the presented soft shadow algorithm.

Of course, the produced soft shadows are not physically correct. But, they can satisfy the need for more realism of interactive applications. Artifacts are avoided if shadow blockers and self-shadowing shapes are treated carefully.

As future work, we will investigate user parameters that control soft shadow effects. It would be favorable if those parameters could be determined automatically.

The algorithm uses few and hardware-accelerated resources. We anticipate that it cooperates well with other rendering techniques such as bump-mapping, planar or environmental reflections, and programmable shading.

8. REFERENCES

[Bra01] Brabec, S., and Seidel, H. P. Hardware-accelerated rendering of antialiased shadows with shadow maps. Computer Graphics International (CGI 2001), 219-228, 2001.

[Bra02] Brabec, S., and Seidel, H. P. Single sample soft shadows using depth maps. Graphics Interface (GI 2002 Proceedings), 219-228, May 2002.

[Bro84] Brotman, L. S., and Badler, N. I. Generating soft shadows with a depth buffer algorithm. IEEE Computer Graphics and Applications, 4(10):71-81, October 1984.

[Chi92] Chin, N., and Feiner, S. Fast object-precision shadow generation for area light sources using BSP trees. Computer Graphics (Symposium on Interactive 3D Graphics), 25:21-30, 1992.

[Cro77] Crow, F. C. Shadow algorithms for computer graphics. Computer Graphics (SIGGRAPH '77 Proceedings), 11(2):242-248, July 1977.

[Hai01] Haines, E. Soft planar shadows using plateaus. Journal of Graphics Tools JGT, 6(1):19-27, 2001.

[Hec97] Heckbert, P. S., and Herf, M. Simulating soft shadows with graphics hardware. Technical Report TR CMU-CS-97-104, Carnegie Mellon University, January 1997.

[Hei91] Heidmann, T. Real shadows real time. Iris Universe, 18:28-31, November 1991.

[Hei99] Heidrich, W. High-quality shading and lighting for hardware-accelerated rendering. Ph.D. Thesis, Universität Erlangen, Februar 1999.

[Hei00] Heidrich, W., Brabec, W., and Seidel, H. P. Soft shadow maps for linear lights. Rendering Techniques 2000: Proc. Eurographics Workshop on Rendering, 11:269-280, June 2000.

[Huu99] Nguyen Hubert Huu. Casting shadows on volumes. Game Developer, 6(3):44-53, 1999.

[Kil02] Kilgard, M. J. (editor). NVidia OpenGL extension specifications for the CineFX Architecture (NV30). NVidia Corporation, August 2002.

[Par98] Parker, S., Shirley, P., and Smits, B. Single sample soft shadows. Technical Report UUCS-98-019, University of Utah, October 1998.

[Ree87] Reeves, W. T., Salesin, D. H., and Cook, R. L. Rendering antialiased shadows with depth maps. Computer Graphics (SIGGRAPH '87 Proceedings), 21(4):283-291, July 1987.

[Seg92] Segal, M., Korobkin, C., van Widenfelt, R., Foran, J., and Haeberli, P. E. Fast shadows and lighting effects using texture mapping. Computer Graphics (SIGGRAPH '92 Proceedings), 26(2):249-252, July 1992.

[Sol98] Soler, C., and Sillion, F. Fast calculation of soft shadow textures using convolution. Computer Graphics (SIGGRAPH '98 Proceedings), 321-332, July 1998.

[Wil78] Williams, L. Casting curved shadows on curved surfaces. Computer Graphics (SIGGRAPH '78 Proceedings), 12(3):270-274, August 1978.