INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Soft Shadow Maps: Efficient Sampling of Light Source Visibility

Lionel Atty — Nicolas Holzschuch — Marc Lapierre — Jean-Marc Hasenfratz — Charles Hansen — François X. Sillion

N° 5750

Novembre 2005

Thème COG

Rapport de recherche

N 0249-6399 ISRN INRIA/RR--5750--FR+ENG

Soft Shadow Maps:
Efficient Sampling of Light Source Visibility

Lionel Atty* , Nicolas Holzschuch * , Marc Lapierre† , Jean-Marc Hasenfratz*‡ , Charles Hansen*§ ,
François X. Sillion*

Thème COG — Systèmes cognitifs
Projet ARTIS

Rapport de recherche n° 5750 — Novembre 2005 —24 pages

Abstract: Shadows, particularly soft shadows, play an important role in the visual perception of a scene by providing visual cues about the shape and position of objects. Several recent algorithms produce soft shadows at interactive rates, but they do not scale well with the number of polygons in the scene or only compute the outer penumbra. In this paper, we present a new algorithm for computing interactive soft shadows on the GPU. Our new approach provides both inner- and outer-penumbra, and has a very small computational cost, giving interactive frame-rates for models with hundreds of thousands of polygons.

Our technique is based on a sampled image of the occluders, as in shadow map techniques. These shadow samples are used in a novel manner, computing their effect on a second projective shadow texture using fragment programs. In essence, the fraction of the light source area hidden by each sample is accumulated at each texel position of this *Soft Shadow Map*. We include an extensive study of the approximations caused by our algorithm, as well as its computational costs.

Key-words: Image Synthesis, Lighting Simulation, Real-Time, GPU programmation, Soft Shadows

ARTIS and MOVI are research teams in the GRAVIR laboratory, a joint research unit of CNRS, INRIA, INPG and UJF.

This research was conducted while Charles Hansen was in a sabbatical at the ARTIS research team.

* ARTIS/GRAVIR IMAG INRIA
† MOVI/GRAVIR IMAG INRIA
‡ Université Pierre Mendès-France
§ School of Computing, University of Utah

Cartes d’ombres douces : calcul efficace de la visibilité de la source lumineuse

Résumé : Les ombres, et en particulier les ombres douces, jouent un rôle important dans la perception d’une scène 3D. Elles fournissent des informations importantes sur la forme et la position des objets. De nombreuses recherches récentes ont montré comment produire des ombres douces en temps interactif, mais soit ces méthodes sont limitées soit dans les scènes qu’elles peuvent traiter, soit dans la qualité visuelle des résultats. Dans ce rapport, nous présentons un nouvel algorithme pour le calcul en temps-réel des ombres douces sur le GPU. Notre approche donne à la fois les pénombres intérieures et extérieures, consomme très peu de temps de calcul, ce qui lui permet des temps de rendu compatibles avec le temps réel pour des modèles de plusieurs centaines de milliers de polygones.

Notre technique est basée sur une carte de profondeur des obstacles, comme pour la technique « shadow map ». Les pixels de cette carte de profondeur sont utilisés d’une façon nouvelle. Nous calculons leur contribution à une texture d’ombre douce en utilisant des *fragment programs*. La proportion de la source lumineuse qui est masquée par chaque échantillon est calculée, puis accumulée dans cette texture, la *Soft Shadow Map*. Nous présentons également une analyse détaillée des approximations introduites par notre algorithme, ainsi que des ses coûts de calcul.

Mots-clés : Synthèse d’images, simulation de l’éclairage, temps réel, cartes graphiques, ombres douces

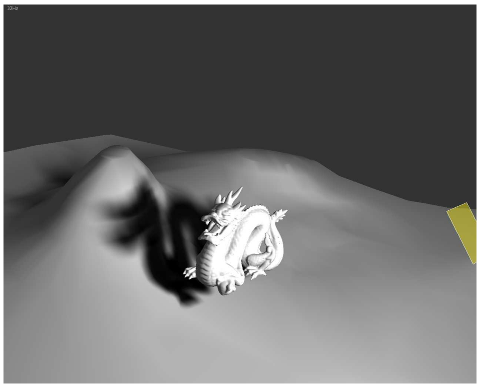


Figure 1: Applying our algorithm (200,000 polygons, occluder map 256×256, displayed at 32 fps).

1 Introduction

Shadows add important visual information to computer-generated images. The perception of spatial relationships between objects can be altered or enhanced simply by modifying the shadow shape, orientation, or position [WFG92, Wan92, KMK97]. Soft shadows, in particular, provide robust contact cues by the hardening of the shadow due to proximity resulting in a hard shadow upon contact. The advent of powerful graphics hardware on low-cost computers has led to the emergence of many interactive soft shadow algorithms (for a detailed study of these algorithms, please refer to [HLHS03]).

In this paper, we introduce a novel method based on shadow maps to interactively render soft shadows. Our method interactively computes a projective shadow texture, the *Soft Shadow Map*, that incorporates soft shadows based on light source visibility from receiver objects (see Fig. 1). This texture is then projected onto the scene to provide interactive soft shadows of dynamic objects and dynamic area light sources.

There are several advantages to our technique when compared to existing interactive soft-shadow algorithms: First, it is not necessary to compute silhouette edges. Second, the algorithm is not fill-bound, unlike methods based on shadow volumes. These properties provide better scaling for occluding geometry than other GPU based soft shadow techniques [WH03, CD03, AAM03]. Third, unlike some other shadow map based soft shadow techniques, our algorithm does not dramatically overestimate the umbra region [WH03, CD03]. Fourth, while other methods have relied on an interpolation from the umbra to the non-shadowed region to approximate the penumbra for soft shadows [WH03, CD03, BS02], our method computes the visibility of an area light source for receivers in the penumbra regions.

Our algorithm also has some limitations when compared to existing algorithms. First, our algorithm splits scene geometry into occluders and receivers and self shadowing is not accounted for. Also, since our algorithm uses shadow maps to approximate occluder geometry, it inherits the well known issues with aliasing from shadow map techniques. For large area light sources, the soft shadows tend to blur the artifacts but for smaller area light sources, such aliasing is apparent.

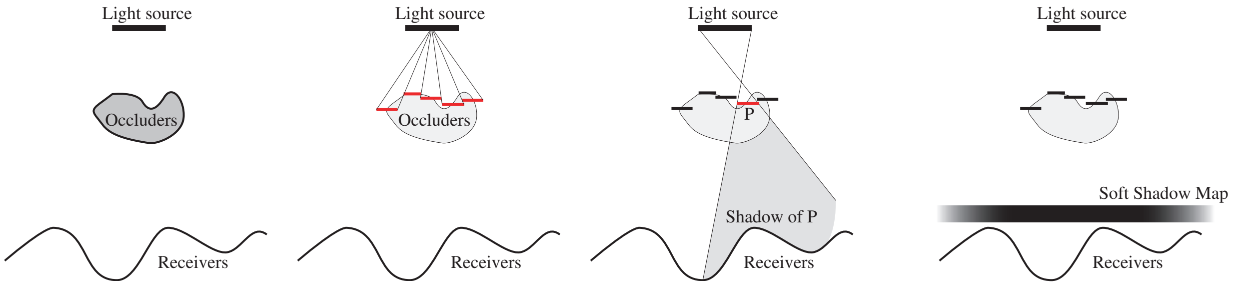
We acknowledge that these limitations are important, and they may prevent the use of our algorithm in some cases. However, there are many applications such as video games or immersive environments where the advantages of our algorithm (a very fast framerate, and a convincing soft shadow) outweigh its limitations. We also think that this new algorithm could be the start of promising new research.

In the following section, we review previous work on interactive computation of soft shadows. In Section 3, we present our algorithm. In the next two sections, we conduct an extensive analysis of our algorithm; first, in Section 4, we study the approximations in our soft shadows, then in Section 5 we study the rendering times of our algorithm. Both studies are done first from a theoretical point of view, then experimentally. Finally, in Section 6, we conclude and expose possible future directions for research.

2 Previous Work

Researchers have investigated shadow algorithms for computer-generated images for nearly three decades. The reader is referred to a recent state-of-the art report by Hasenfratz *et al.* [HLHS03], the overview by Woo *et al.* [WPF90] and the book by Akenine-Möller and Haines [AMH02].

The two most common methods for interactively producing shadows are shadow maps [Wil78] and shadow volumes [Cro77]. Both of these techniques have been extended for soft shadows. In the case of shadow volumes, Assarsson and Akenine-Möller [AAM03] used penumbra wedges in a technique based on shadow volumes to produce soft shadows.



(a) Scene view (b) Discretizing occluders (c) Soft shadows from one micro-patch (d) Summing the soft shadows

Figure 2: The main steps of our algorithm

Their method depends on locating silhouette edges to form the penumbra wedges. While providing good soft shadows without an overestimate of the umbra, the algorithm is fill-limited, particularly when zoomed in on a soft shadow region. Since it is necessary to compute the silhouette edges at every frame, the algorithm also suffers from scalability issues when rendering occluders with large numbers of polygons.

The fill-rate limitation is a well known limitation of shadow-volume based algorithms. Recent publications [CD04, LWGM04] have focused on limiting the fill-rate for shadow-volume algorithms, thus removing this limitation.

On shadow maps, Chan and Durand [CD03] and Wyman and Hansen [WH03] both employed a technique which uses the standard shadow map method for the umbra region and builds a map containing an approximate penumbra region that can be used at run-time to give the appearance, including hard shadows at contact, of soft shadows. While these methods provide interactive rendering, both only compute the outer-penumbra, the part of the penumbra that is outside the hard shadow. In effect, they are overestimating the umbra region, resulting in incorrectly looking soft shadows in the case of large area light sources. These methods also depend on computing the silhouette edges in object space for each frame; this requirement limits the scalability for occluders with large numbers of polygons.

Brabec and Seidel [BS02] and Kirsch and Doellner [KD03] use a shadow map to compute soft shadows, by searching at each pixel of the shadow map for the nearest boundary pixel, then interpolating between illumination and shadow as a function of the distance between this pixel and the boundary pixel and the distances between the light source, the occluder and the receiver. Their algorithm requires scanning the shadow map to look for boundary pixels, a potentially costly step; in practical implementations they limit the search radius, thus limiting the actual size of the penumbra region.

Soler and Sillion [SS98] compute a soft shadow map as the convolution of two images representing the source and blocker. Their technique is only accurate for planar and parallel objects, although it can be extended using an object hierarchy. Our technique can be seen as an extension of this approach, where the convolution is computed for each sample of an occlusion map, and the results are then combined.

Finally, McCool [McC00] presented an algorithm merging shadow volume and shadow map algorithms by detecting silhouette pixels in the shadow map and computing a shadow volume based on these pixels. Our algorithm is similar in that we are computing a shadow volume for each pixel in the shadow map. However, we never display this shadow volume, thus avoiding fill-rate issues.

3 Algorithm

3.1 Presentation of the algorithm

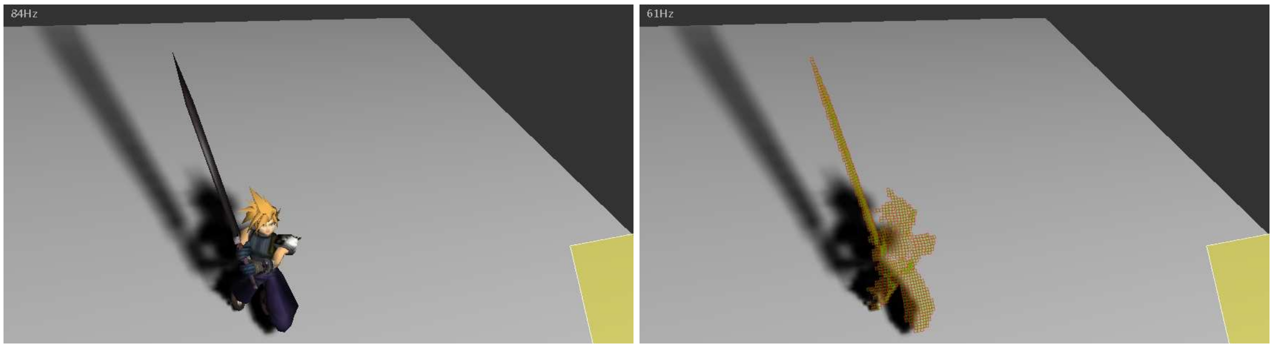
Our algorithm assumes a rectangular light source and starts by separating potential occluders (such as moving characters) from potential receivers (such as the background in a scene) (Fig. 2(a)). We will compute the *soft shadows* only from the occluders onto the receivers.

Our algorithm computes a *Soft Shadow Map*, (SSM), for each light source: a texture containing the texelwise percentage of occlusion from the light source. This soft shadow map is then projected onto the scene from the position of the light source, to give soft shadows (see Fig. 1).

Our algorithm is an extension of the shadow map algorithm: we start by computing depth buffers of the scene. Unlike the standard shadow map method, we will need two depth buffers: one for the occluders (the *occluder map*) and the other for the receivers.

```
Compute depth map of receivers
Compute depth map of occluders
for all pixels in occluder map
    Retrieve depth of occluder at this pixel
    Compute micro-patch associated with this pixel
    Compute extent of penumbra for this micro-patch
    for all pixels in penumbra extent for micro-patch
        Retrieve receiver depth at this pixel
        Compute percentage of occlusion for this pixel
        Add to current percentage of occlusion in the soft shadow map
    end
end
Project soft shadow map on the scene
```

Figure 3: Our algorithm



(a) The original model (b) The discretized occluder

Figure 4: Discretizing the occluders

The occluder map depth buffer is used to discretize the set of occluders (see Fig. 2(b)): each pixel in this occluder map is converted into a micro-patch that covers the same image area but is located in a plane parallel to the light source, at a distance corresponding to the pixel depth. Pixels that are close to the light source are converted into small rectangles and pixels that are far from the light source are converted into larger rectangles. At the end of this step, we have a discrete representation of the occluders.

The receiver map depth buffer will be used to provide the receiver depth, as our algorithm uses the distance between light source and receiver to compute the soft shadow values.

We compute the soft shadow of each of the micro-patches constituting the discrete representation of the occluders (see Fig. 2(c)), and sum them into the soft shadow map (SSM) (see Fig. 2(d)). This step would be potentially costly, but we achieve it in a reasonable amount of time with two key points: 1) the micro-patches are parallel to the light source, so computing their penumbra extent and their percentage of occlusion only requires a small number of operations, and 2) these operations are computed on the graphics card, exploiting the parallelism of the GPU engine. The percentage of occlusion from each micro-patch takes into account the relative distances between the occluders, the receiver and the light source. Our algorithm introduces several approximations on the actual soft shadow. These approximations will be discussed in Section 4.

The pseudo-code for our algorithm is given in Fig. 3. In the following subsections, we will review in detail the individual steps of the algorithm: discretizing the occluders (Section 3.2), computing the penumbra extent for each micro-patch (Section 3.3) and computing the percentage of occlusion for each pixel in the Soft Shadow Map (Section 3.4).

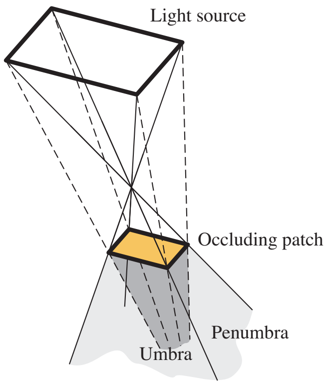
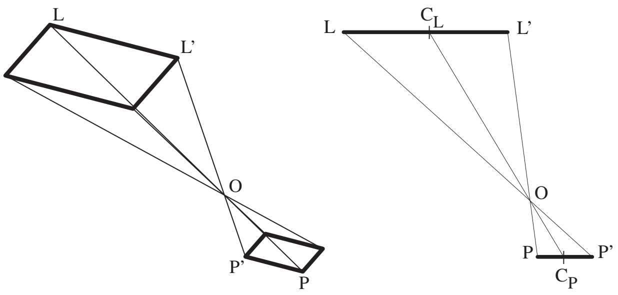


Figure 5: The penumbra extent of a micro-patch is a rectangular pyramid



(a) (b)

Figure 6: Finding the apex of the pyramid is reduced to a 2D problem

3.2 Discretizing the occluders

The first step in our algorithm is a discretization of the occluders. We compute a depth buffer of the occluders, as seen from the light source, then convert each pixel in this *occluder map* into the equivalent polygonal micro-patch that lies in a plane parallel to the light source, at the appropriate depth and occupies the same image plane extent (Fig. 4).

The occluder map is axis-aligned with the rectangular light source and has the same aspect ratio: all micro-patches created in this step are also axis-aligned with the light source and have the same aspect ratio.

3.3 Computing penumbra extents

Each micro-patch in the discretized occluder is potentially blocking some light between the light source and some portion of the receiver. To reduce the amount of computations, we compute the penumbra extent of the micro-patches, and we only compute occlusion values inside these extents.

Since the micro-patches are parallel, axis-aligned with the light source and have the same aspect ratio, the penumbra extent of each micro-patch is a rectangular pyramid (Fig. 5). Finding the penumbra extent of the light source is equivalent to finding the apex O of the pyramid (Fig. 6(a)). This reduces to a 2D problem, considering parallel edges (LL') and (PP') on both polygons (Fig. 6(b)). Since (LL') and (PP') are parallel lines, we have:

$$\frac{OL}{OP} = \frac{OL'}{OP'} = \frac{LL'}{PP'}$$

This ratio is the same if we consider the center of each line segment:

$$\frac{OC_L}{OC_P} = \frac{LL'}{PP'}$$

Since the micro-patch and the light source have the same aspect ratio, the ratio $r = \frac{LL'}{PP'}$ is the same for both sides of the micro-patch (thus, the penumbra extent of the micro-patch is indeed a pyramid).

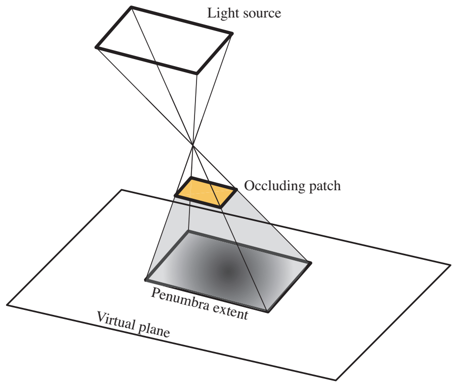


Figure 7: The intersection between the pyramid and the virtual plane is an axis-aligned rectangle

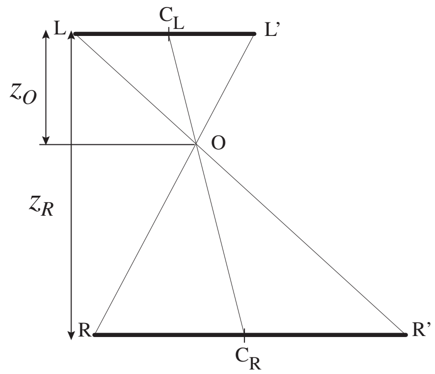


Figure 8: Computing the position and extent of the penumbra rectangle for each micro-patch.

We find the apex of the pyramid by applying a scaling to the center of the micro-patch (C_P), with respect to the center of the light source (C_L):

$$\overrightarrow{C_L O} = \frac{r}{1+r} \overrightarrow{C_L C_P}$$

We now use this pyramid to compute occlusion in the soft shadow map (see Fig. 7). We use a virtual plane, parallel to the light source, to represent this map (which will be projected onto the scene). The intersection of the penumbra pyramid with this virtual plane is an axis-aligned rectangle. We only have to compute the percentage of occlusion inside this rectangle.

Computing the position and size of the penumbra rectangle uses the same formulas as for computing the apex of the pyramid (see Fig. 8):

$$\begin{aligned} \overrightarrow{C_L C_R} &= \frac{z_R}{z_R - z_O} \overrightarrow{C_L O} \\ RR' &= LL' \frac{z_R - z_O}{z_O} \end{aligned}$$

3.4 Computing the soft shadow map

For all the pixels of the SSM lying inside this penumbra extent, we compute the percentage of the light source that is occluded by this micro-patch. This percentage of occlusion depends on the relative positions of the light source, the occluders and the receivers. To compute it, for each pixel on the receiver inside this extent, we project the occluding micro-facet back onto the light source [DF94] (Fig. 9). The result of this projection is an axis-aligned rectangle; we need to compute the intersection between this rectangle and the light source.

Computing this intersection is equivalent to computing the two intersections between the respective intervals on both axes. This part of the computation is done on the GPU, using a fragment program: the penumbra extent is converted into an axis-aligned quad, which we draw in a float buffer. For each pixel inside this quad, the fragment program computes the percentage of occlusion. These percentages are summed using the blending capability of the graphics card.

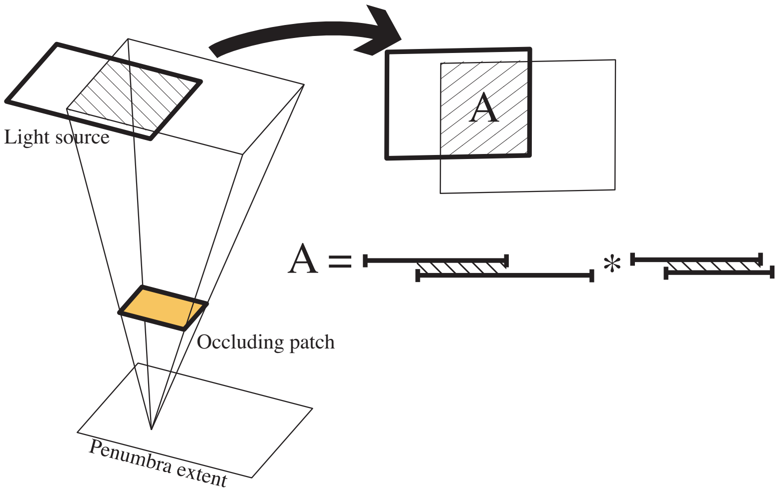
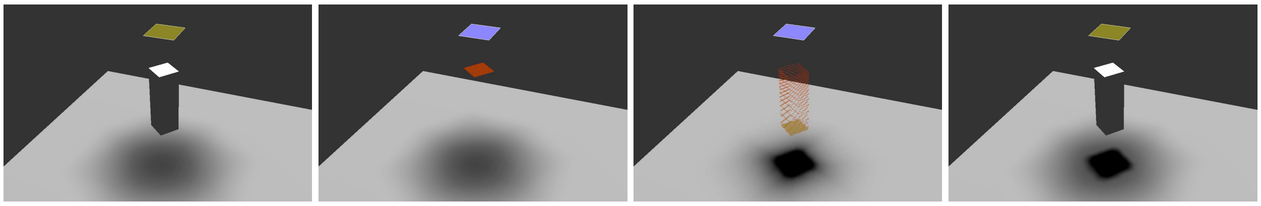


Figure 9: We reproject the occluding micro-patch onto the light source and compute the percentage of occlusion.



(a) Original algorithm

(b) Closest, front faces of the occluder discretized with their shadow

(c) Furthest, back faces of the occluder discretized with their shadow

(d) Combining the two soft shadow maps

Figure 10: The original algorithm fails for some geometry. The two-pass method gives the correct shadow.

To further optimize the computations, we use the `SAT` instructions in the fragment program assembly language: without loss of generality, we can convert the rectangle corresponding to the light source to $[0,1] \times [0,1]$. Each interval intersection becomes the intersection between one $[a,b]$ interval and $[0,1]$. Exploiting the `SAT` instruction and swizzling, computing the area of the intersection between the projection of the occluder $[a,b] \times [c,d]$ and the light source $[0,1] \times [0,1]$ only requires three instructions:

```
MOV_SAT rs,{a,b,c,d}
SUB rs, rs, rs.yxwz
MUL result.color, rs.x, rs.z
```

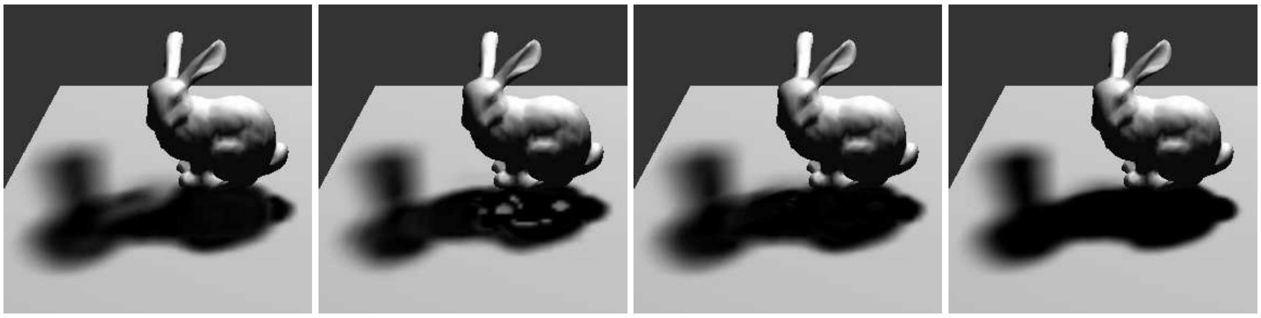
Computing the $[a,b] \times [c,d]$ intervals requires projecting the micro-patch onto the light source and scaling the projection. This uses 11 instructions: 9 basic operations (`ADD`, `MUL`, `SUB`), one reciprocal (`RCP`) and one texture lookup to get the depth of the receiver.

3.5 Two-sided soft-shadow maps

As with many other soft shadow computation algorithms [HLHS03], our algorithm exhibits artifacts because we are computing soft shadows using a single view of the occluder. Shadow effects linked to parts of the occluder that are not directly visible from the light source are not visible. In Fig. 10(a), our algorithm only computes the soft shadow for the front part of the occluder, because the back part of the occluder does not appear in the occluder map. This limitation is frequent in real-time soft-shadow algorithms [HLHS03].

For our algorithm, we have devised an extension that solves this limitation: we compute two occluder maps. In the first, we discretize the closest, front-facing faces of the occluders (see Fig. 10(b)). In the second, we discretize the furthest, back-facing faces of the occluders (see Fig. 10(c)).

We then compute a soft shadow map for each occluder map, and merge them, using the maximum of each occluder map. The resulting occlusion map has eliminated most artifacts (Fig. 10(d) and 11). Empirically, the cost of the two-pass algorithm is between 1.6 and 1.8 times the cost of the one-pass algorithm. Depending on the size of a model and the



(a) One pass (148 fps) (b) One pass with bottom patches (142 fps) (c) Two passes (84 fps) (d) Ground truth

Figure 11: Two-pass shadow computations enhance precision.

quality requirements of a given application, the second pass may be worth this extra cost. For example, for an animated model of less than $100,000$ polygons, the one-pass algorithm renders at approximately 60 fps. Adding the second pass drops the framerate to 35 fps – which is still interactive.

4 Error Analysis and comparison

In this section, we analyze our algorithm, its accuracy and how it compares with the exact soft-shadows. We first study potential sources of error from a theoretical point of view, in Section 4.1, then we conduct an experimental analysis, comparing the soft shadows produced with exact soft shadows, in Section 4.2.

4.1 Theoretical analysis

Our algorithm is replacing the occluder with a discretized version. This discretization ensures interactive framerates, but it can also be a source of inaccuracies. From a given point on the receiver, we are separately estimating occlusion from several micro-patches, and adding these occlusion values together. We have identified three potential sources of error in our algorithm:

- We are only computing the shadow of the discretized occluder, not the shadow of the actual occluder. This source of error will be analyzed in Section 4.1.1.
- The reprojections of the micro-patches on the light source may overlap or be disjoined. This cause of error will be analyzed in Section 4.1.2.
- We are adding many small values (the occlusion from each micro-patch) to form a large value (the occlusion from the entire occluder). If the micro-patches are too small, we run into numerical accuracy issues, especially with floating-point numbers expressed on 16 bits. This cause of error will be analyzed in Section 4.1.3.

4.1.1 Discretization error

Our algorithm computes the shadow of the discretized occluder, not the shadow of the actual occluder. The discretized occluder corresponds to the part of the occluder that is visible from the camera used to compute the depth buffers, usually the center of the light source. Although we reproject each micro-patch of the discretized occluder onto the area light source, we are missing the parts of the occluder that are not visible from the shadow map camera but are still visible from some points of the area light source. This is a limitation that is frequent in real-time soft shadow algorithms [HLHS03], especially algorithms relying on the silhouette of the occluder as computed from a single point [WH03, CD03, AAM03].

We also use a discrete representation based on the shadow map, not a continuous representation of the occluder. For each pixel of the shadow map, we are potentially overestimating or underestimating the actual occluder by at most half a pixel.

If the occluder has one or more edges aligned with the edges of the shadow map, these discretization errors are of the same sign over the edge, and add themselves; the worst case scenario is a square aligned with the axis of the shadow map.

For more practical occluders the discretization errors on neighboring micro-patches compensate: some of the micro-patches overestimate the occluder while others underestimate it.

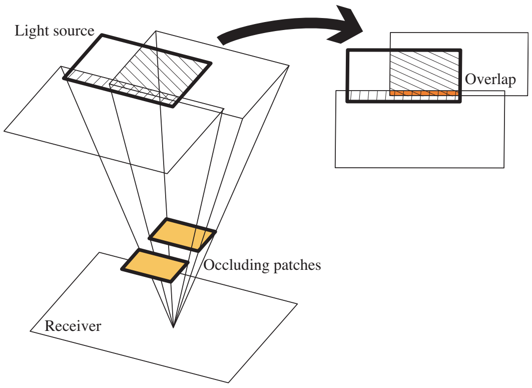


Figure 12: The reprojection of two neighboring micro-patches may overlap.

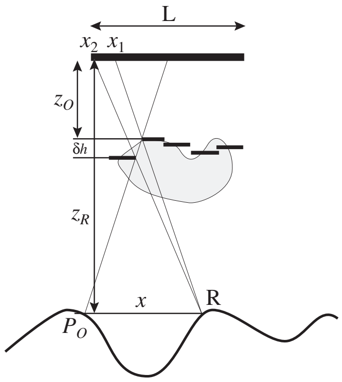


Figure 13: Computing the extent of overlap or gap between two neighboring micro-patches.

4.1.2 Overlapping reprojections

At any given point on the receiver, the parts of the light source that are occluded by two neighboring micro-patches should be joined exactly for our algorithm to compute the exact percentage of occlusion on the light source. This is typically not the case, and these parts may overlap or there may be a gap between them (Fig. 12). The amount of overlap (or gap) between the occluded parts of the light source depends on the relative positions of the light source, the occluding micro-patches and the receiver

If we consider the 2D equivalent of this problem (Fig. 13), with two patches separated by δh and at a distance z_O from the light source, with the receiver being at a distance z_R from the light source, there is a point P_0 on the receiver where there is no overlap between the occluded parts. As we move away from this point, the overlap increases. For a point at a distance x from P_0, the boundaries of the occluding micro-patches project at abcissa x_1 and x_2; as the occluding micro-patches and the light source lie in parallel planes, we have:

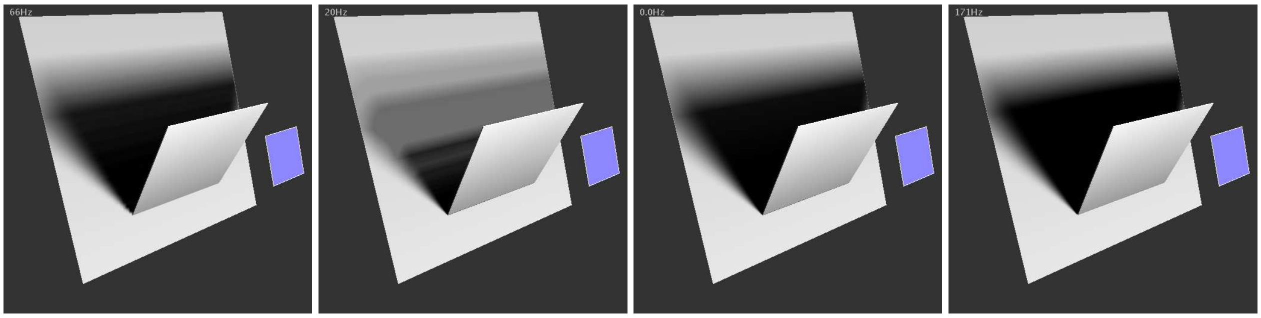
$$\begin{aligned} \frac{x_1}{x} &= \frac{z_O + \delta h}{z_R - z_O - \delta h} \\ \frac{x_2}{x} &= \frac{z_O}{z_R - z_O} \end{aligned}$$

The amount of overlap is therefore:

$$\begin{aligned} x_2 - x_1 &= x\left(\frac{z_O}{z_R - z_O} - \frac{z_O + \delta h}{z_R - z_O - \delta h}\right) \\ &= -x\frac{z_R \delta h}{(z_R - z_O)(z_R - z_O - \delta h)} \end{aligned} \tag{1}$$

x itself is limited, since the occlusion area must fall inside the light source:

$$|x| < \frac{L}{2}\frac{z_R - z_O}{z_O} \tag{2}$$



(a) 128^2 pixels, FP16 blending (66 Hz) (b) 512^2 pixels, FP16 blending (20 Hz) (c) 512^2 pixels, FP32 blending (CPU) (d) Ground truth (CPU)

Figure 14: Blending with FP16 numbers: if the resolution of the shadow map is too high, numerical issues appear, resulting in wrong shadows. Using higher accuracy for blending removes this issue (here, FP32 blending was done on the CPU).

The amount of overlap is therefore limited by:

$$|x_2 - x_1| < \frac{L}{2} \frac{z_R \delta h}{z_O(z_R - z_O - \delta h)} \tag{3}$$

Equation 3 represents the error our algorithm makes for each pair of micro-patches. The overall error of our algorithm is the sum of the modulus of all these errors, for all the micro-patches projecting on the light source at a given point. This is a conservative estimate, as usually some patches overlap while others present gaps; the actual sum of the occlusion values from all the micro-patches is closer to the real value than what our estimation tells (see Section 4.2).

The theoretical error caused by our algorithm depends on several factors:

Size of the light source: The maximum amount of overlap (Eq. 3) depends directly on the size of the light source. The larger the light source, the larger the error. Our practical experiments confirm this.

Distance between micro-patches: The maximum amount of overlap (Eq. 3) also depends linearly on δh, the distance in z between neighboring micro-patches. Since δh depends on the discretization of the occluder, the error introduced by our algorithm is related to the resolution of the bitmap: the smaller the resolution of the bitmap, the larger the error. Our practical experiments confirm this, but there is a maximum resolution after which the error does not decrease.

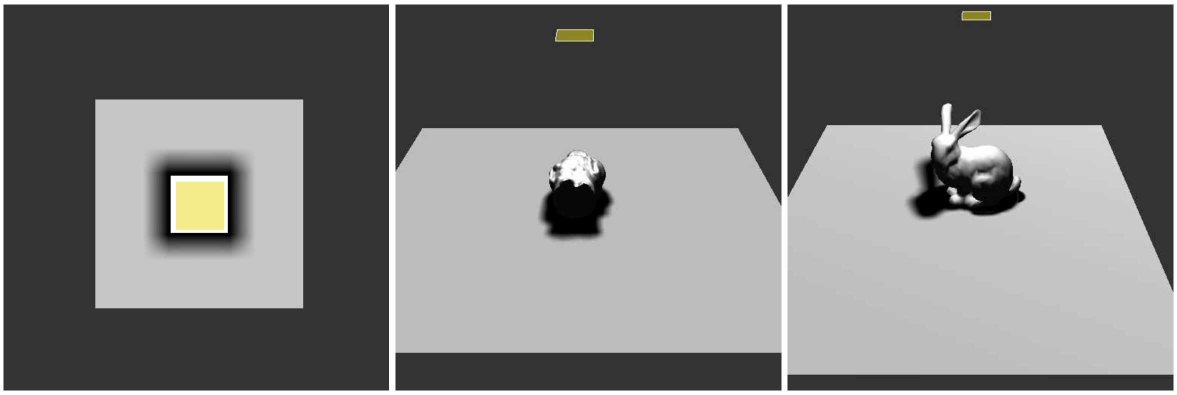
Distance to the light source/the receiver: If the occluder touches either the light source or the receiver, the amount of overlap (Eq. 3) goes toward infinity. When the occluder is touching the receiver, the area where the overlap occurs (as defined by equation 2) goes towards 0, thus the error does not appear. When the occluder is touching the receiver, the actual effect depends on the shape of the occluder. In some cases, overlaps and gaps can compensate, resulting in an acceptable shadow.

4.1.3 Floating-point blending accuracy

Our algorithm adds together many small scale occlusion values — the occlusion from each micro-patch — to compute a large scale occlusion value — the occlusion from the complete occluder. This addition is done with the blending ability of the GPU, using blending of floating-point buffers. At the time of writing, blending is only available in hardware for 16-bits floating-point buffers. As a result, we sometimes encounter problems of numerical accuracy.

Figure 14 shows an example of these problems. Unconventionally, *increasing* the resolution of the shadow map makes these problems *more* likely to appear (for a complete study of floating-point blending accuracy, see appendix A). The best workaround is therefore to use relatively low resolution for the occluder map, such as 128×128 or 256×256. While this may seem a low resolution compared to other shadow map algorithms, our shadow map is focused on the moving occluder (such as a character), not on the entire scene, so 128×128 pixels is usually enough resolution.

We see this is only as a temporary issue that will disappear as soon as hardware FP32 blending becomes available on graphics cards.



(a) Square plane (b) Buddha (c) Bunny

Figure 15: The test scenes we have used

4.2 Comparison with ground truth

We ran several tests to experimentally compare the shadows produced by our algorithm with the actual shadows. The reference values were computed using occlusion queries, giving an accurate estimation of the real occlusion of the light source. In this section, we review the practical differences we observed.

4.2.1 Experimentation method

For each image, we computed an error metric as thus: for each pixel in the soft shadow map, we compute the actual occlusion value (using occlusion queries), and the difference with the occlusion value computed using our algorithm. We summed the modulus of the differences, then divided the result by the total number of pixels lying either in the shadow or in the penumbra, averaging the error over the actual soft shadow. We used the number of pixels that are either in shadow or in penumbra and not the total number of pixels in the occluder depth map because the soft shadow can occupy only a small part of the depth map. Dividing by the total number of pixels in the depth map would have underestimated the error.

We have used 3 different scenes (a square plane parallel to the light source, a Buddha model and a Bunny model). These scenes exhibit several interesting features. The Budha and Bunny are complex models, with folds and creases. The Bunny also has important self-occlusion, and in our scene it is in contact with the ground, providing information on the behaviour of our algorithm in that case. The square plane is an illustration of the special case of occluders aligned with the axes of the occluder depth map.

We have tested both the one-pass and the two-pass versions of our algorithm. We selected four separate parameters: the size of the light source, the resolution of the shadow map and moving the occluder, either vertically from the receiver to the light source or laterally with respect to the light source. For each parameter, we plot the variation of the error introduced by our algorithm as a function of the parameter and analyze the results.

4.2.2 Visual comparison with ground truth

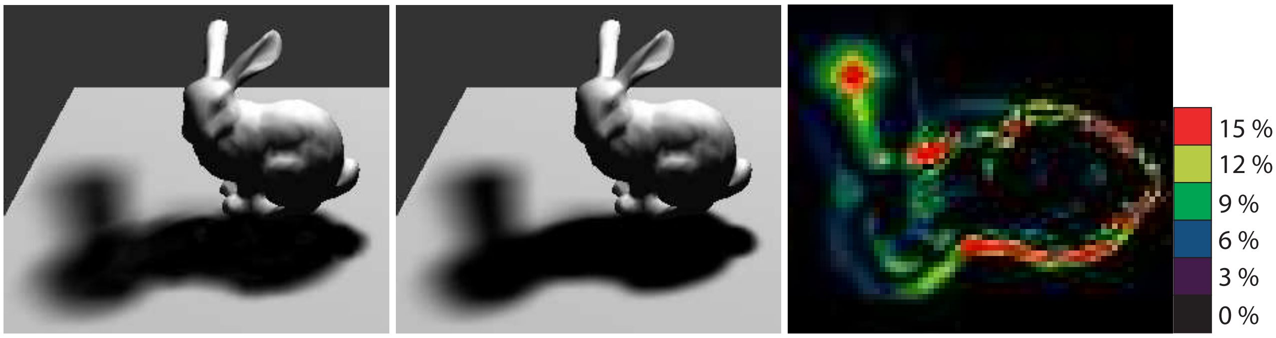
Fig. 16 shows a side by side comparison of our algorithm with ground truth. Even though there are slight differences with ground truth, our algorithm exhibits the proper behaviour for soft shadows: sharp shadows at places where the object is close to the ground, a large penumbra zone where the object is further away from the receiver. Our algorithm visibly computes both the inner and the outer penumbra of the object.

Looking at the picture of the differences (Fig. 16(d) and 16(g)) between the shadow values computed by our algorithm and the ground truth values, it appears that the differences lie mostly on the silhouette: since our algorithm only computes the soft shadow of the discretized object, as seen from the center of the light source. The actual shape of the soft shadow depends on subtle effects happening at the boundary of the silhouette.

4.2.3 Size of the buffer

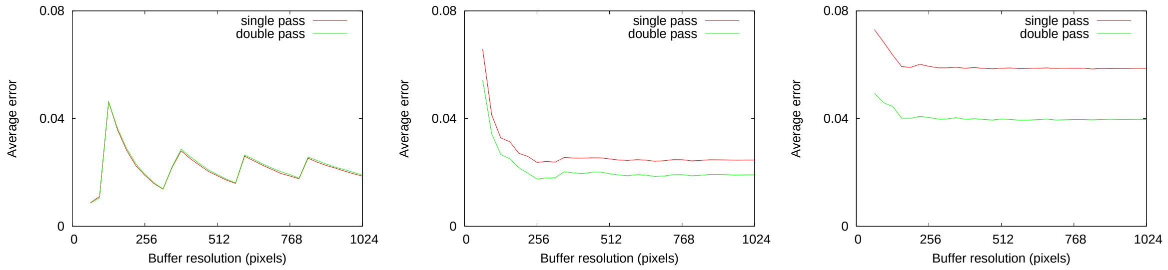
Figure 17 shows the average difference between the occlusion values computed with our algorithm and the actual occlusion values for our three test scenes, when changing the resolution of the shadow map. In these figures, the abcissa is the number of pixels for one side of the shadow map, so 128 corresponds to a 128×128 shadow map. For this test, we used

(a) Scene view (b) Our algorithm (c) Ground Truth (d) Difference between the occlusion values



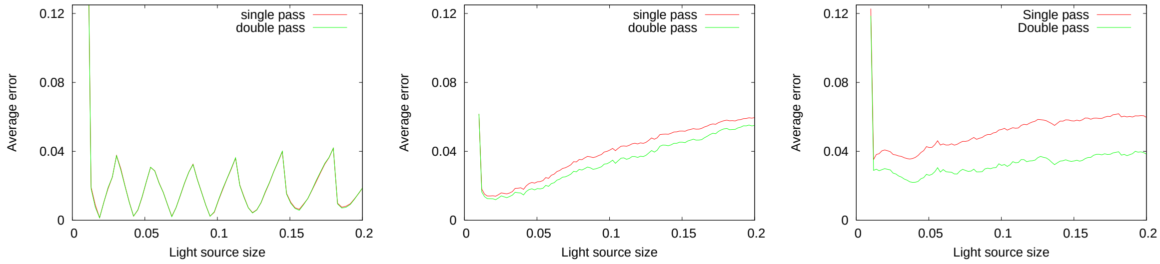
(e) Our algorithm (f) Ground Truth (g) Difference between the occlusion values

Figure 16: Visual comparison of our algorithm with ground truth.



(a) Square plane (b) Buddha (c) Bunny

Figure 17: Variation of the error with respect to the resolution of the shadow map



(a) Square plane (b) Buddha (c) Bunny

Figure 18: Variation of the error with respect to the size of the light source

non-power of two textures, in order to have enough sampling data. We can make several observations by looking at the data:

Two-pass version: the two-pass version of the algorithm consistently outperforms the single-pass version, always giving more accurate results. The only exception is of course the square plane: since it has no thickness, the single-pass and two-pass version give the same results.

Shadow map Resolution: as expected from the theoretical study (see Section 4.1.2), the error decreases as the resolution of the shadow map increases. What is interesting is that this effect reaches a limit quite rapidly. Roughly, increasing the shadow map resolution above 200 pixels does not bring an improvement in quality. Since the computation costs are related to the size of the shadow map, shadow map sizes of 200×200 pixels are close to optimal.

The fact that the error does not decrease continuously as we increase the resolution of the occluder map is a little surprising at first, but can be explained. It is linked to the silhouette effect. As we have seen in Fig. 16, the error introduced by our algorithm comes from the boundary of the silhouette of the occluder, from parts of the occluder that are not visible from the center of the light source, but visible from other parts of the light source. Increasing the resolution of the shadow map does not solve this problem.

The optimal size for the shadow map is related to the size of the light source. As the light source gets larger, we can use smaller bitmaps.

Discretization error: the error curve for the square plane presents many important spikes. Looking at the results, it appears that these spikes correspond to discretization error (see Section 4.1.1). Since the square occluder is aligned with the axis of the shadow map, it magnifies discretization error.

4.2.4 Size of the light source

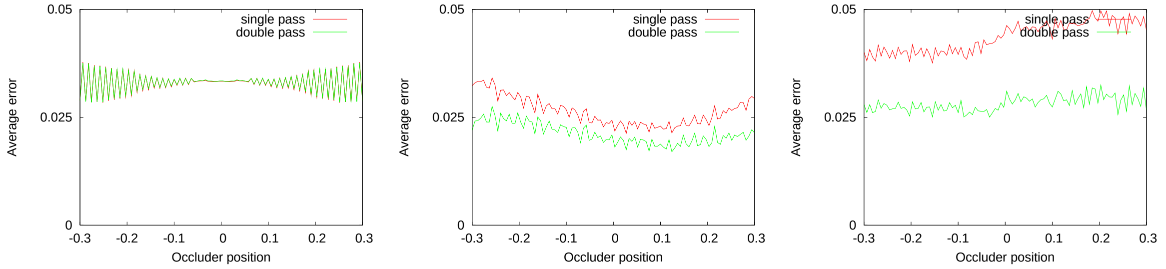
Figure 18 shows the average difference between the occlusion values computed with our algorithm and the actual occlusion values when we change the size of the light source for our three test scenes. The parameter values range from a point light source (parameter=0.01) to a very large light source, approximately as large as the occluder (parameter=0.2). We used a bitmap of 128×128 pixels for all these tests. We can make several observations by looking at the data:

Point light sources: the beginning of the curves (parameter=0.01) corresponds to a point light source. In that case, the error is quite large. This corresponds to an error of 1, over the entire shadow boundary; as we are computing the shadow of the discretized occluder, we miss the actual shadow boundary, sometimes by as much as half a pixel. The result is a large error, but it occurs only at the shadow boundary.

Light source size: except for the special case of point light sources, the error increases with the size of the light source. This is consistent with our theoretical analysis (see Section 4.1.2).

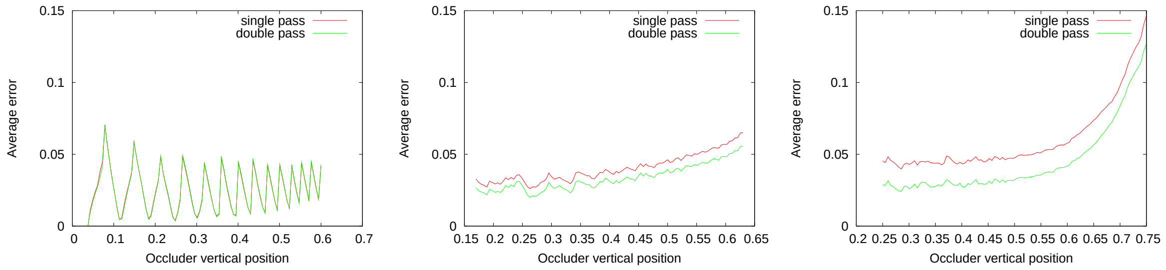
4.2.5 Occluder moving laterally

Figure 19 shows the average difference between the occlusion values computed with our algorithm and the actual occlusion values, when we move the occluder from left to right under the light source. The parameter corresponds to the position with respect to the center of the light, with 0 meaning that the center of the object is aligned with the center of the light. We used a bitmap of 128×128 for all these tests.



(a) Square plane (b) Buddha (c) Bunny

Figure 19: Variation of the error with respect to the lateral position of the occluder



(a) Square plane (b) Buddha (c) Bunny

Figure 20: Variation of the error with respect to the vertical position of the occluder

The error is at its minimum when the occluder is roughly under the light source, and increases as the occluder moves laterally. The Buddha and Bunny models are not symmetric, so their curves are slightly asymmetric, and the minimum does not correspond exactly to 0.

4.2.6 Occluder moving vertically

Figure 20 shows the average difference between the occlusion values computed with our algorithm and the actual occlusion values, when we move the occluder vertically. The smallest value of the parameter corresponds to an occluder touching the receiver, and the largest value corresponds to an occluder touching the light source. We used a bitmap of 128×128 for all these tests.

As predicted by the theory, the error increases as the occluder approaches the light source (see Section 4.1.2). For the Bunny, the error becomes quite large when the upper ear touches the light source.

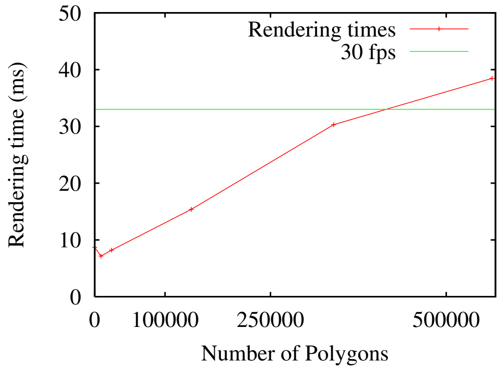
5 Complexity

The main advantages of our algorithm are its rendering speed and its scalability. With a typical setup (a modern PC, an occluder map of 128×128 pixels, a scene between $50,000$ polygons and $300,000$ polygons), we get framerates between 30 and 150 fps. In this section, we study the numerical complexity of our algorithm and its rendering speed. We first conduct a theoretical analysis of the complexity of our algorithm, in Section 5.1, then an experimental analysis, where we test the variation of the rendering speed with respect to several parameters: the size of the shadow map, the number of polygons and the size of the light source (Section 5.2). Finally, in Section 5.3, we compare the complexity of our algorithm with a state-of-the-art algorithm, Soft Shadow Volume [AAM03].

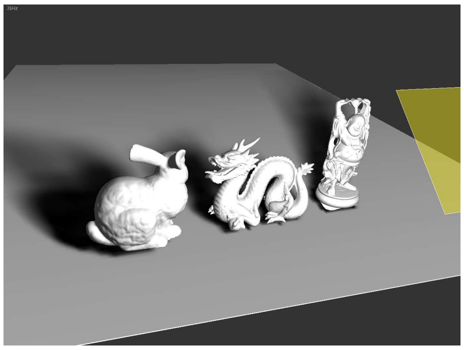
5.1 Theoretical complexity

Our algorithm starts by rendering a shadow map and downloading it into main memory. This preliminary step has a complexity linear with respect to the number of polygons in the scene, and linear with the size of the shadow map, measured in the total number of pixels.

Then, for each pixel of the shadow map corresponding to the occluder, we compute its extent in the occlusion map, and for each pixel of this extent we execute a small fragment program of 11 instructions, including one texture lookup.



(a) Rendering times (in ms)



(b) Our largest test scene (565,203 polygons)

Figure 21: Influence of polygon count

The overall complexity of this second step of the algorithm is the number of pixels covered by the occluder, multiplied by the number of pixels covered by the extent for each of them, multiplied by the cost of the fragment program. This second step is executed on the GPU, and benefits from the high-performance and the parallelism of the graphics card.

The worst case situation would be a case where each micro-patch in the shadow map covers a large number of pixels in the soft shadow map. But this situation corresponds to an object with a large penumbra zone, and if we have a large penumbra zone, we can use a lower resolution for the shadow maps. So we can compensate the cost for the algorithm by running it with bitmaps of lower resolution.

Using our algorithm with a large resolution shadow map in a situation of large penumbra results in relatively high computing costs, but a low resolution shadow map would give the same visual results, for a smaller computation time.

5.2 Experimental complexity

All measurements in this section were conducted on a 2.4 GHz Pentium4 PC with a GeForce 6800 Ultra graphics card. All framerates and rendering times correspond to *observed* framerates, that is the framerate for a user manipulating our system. We are therefore measuring the time it takes to display the scene *and* to compute soft shadows, not just the time it takes to compute soft shadows.

5.2.1 Number of polygons

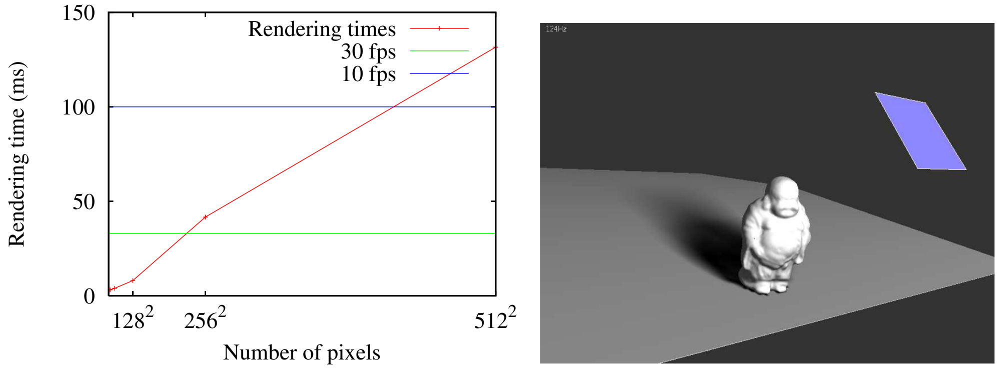
We studied the influence of the polygon count. Fig. 21(a) shows the observed rendering time (in ms) as a function of the polygon count, with a constant occluder map size of 128×128 pixels. The first thing we note is the speed of our algorithm: even on a large scene of $340,000$ polygons, we achieve real-time framerates (more than 30 frames per second). Second, we observe that the rendering time varies linearly with respect to the number of polygons. That was to be expected, as we must render the scene twice (once for the occluder map and once for the actual display), and the time it takes for the graphics card to display a scene varies linearly with respect to the number of polygons. For smaller scenes (less than 10,000 polygons, rendering time below 10 ms), some factors other than the polygon count play a more important role.

Our algorithm exhibits good scaling, and can handle significantly large scenes without incurring a high performance cost. The maximum size of the scene depends on the requirements of the user.

5.2.2 Size of occluder map

Fig. 22(a) shows the observed rendering times (in ms) of our algorithm, on a scene with 24,000 polygons (Fig. 22(b)), when the size of the occluder map changes. We plotted the rendering time as a function of the number of pixels in the occluder map (that is, the *square* of the size of the occluder map) to illustrate the observed linear variation of rendering time with respect to the total number of pixels.

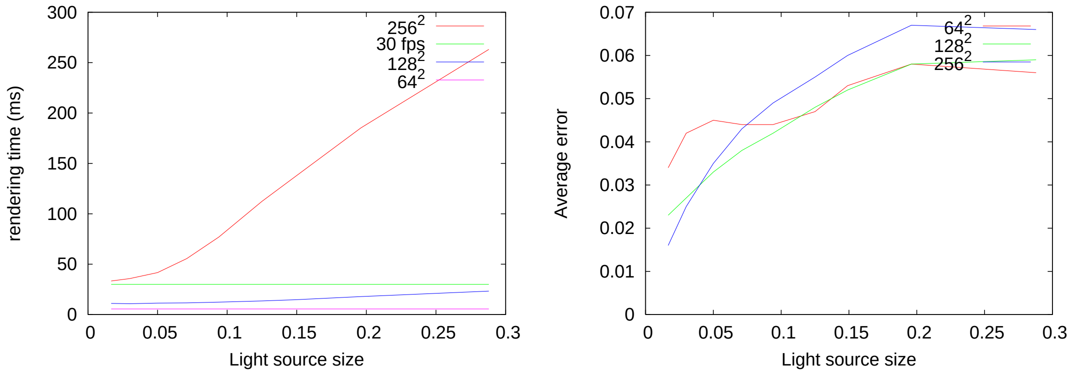
An occluder map of 512^2 pixels gives a rendering time of 150 ms – or 7 fps, too slow for interactive rendering. An occluder map of 128^2 or 256^2 pixels gives a rendering time of 10 to 50 ms, or 20 to 100 fps, fast enough for real-time rendering. For a large penumbra region, an occluder map of 128^2 pixels qualitatively gives a reasonable approximation, as in Fig. 22(b). For a small penumbra region, our algorithm behaves like the classical shadow mapping algorithm and artifacts can appear with a small occluder map of 128^2 pixels; in that case, it is better to use 256^2 pixels.



(a) Rendering times (in ms)

(b) Test scene (24,000 polygons)

Figure 22: Influence of the size the occluder map



(a) Rendering time

(b) Average error

Figure 23: Changing the size of the light source (floating bunny scene)

5.2.3 Light source size

Another important parameter is the size of the light source, compared to the size of the scene itself. A large light source results in a large penumbra region for each micro-patch, resulting in more pixels of the soft shadow map covered, and a larger computational cost. Fig. 23(a) shows the observed framerate as a function of the size of the light source. We did the tests with several bitmap resolutions (256^2, 128^2, 64^2). Fig. 23(b) shows the error as a function of the size of the light source, for the same bitmap resolutions.

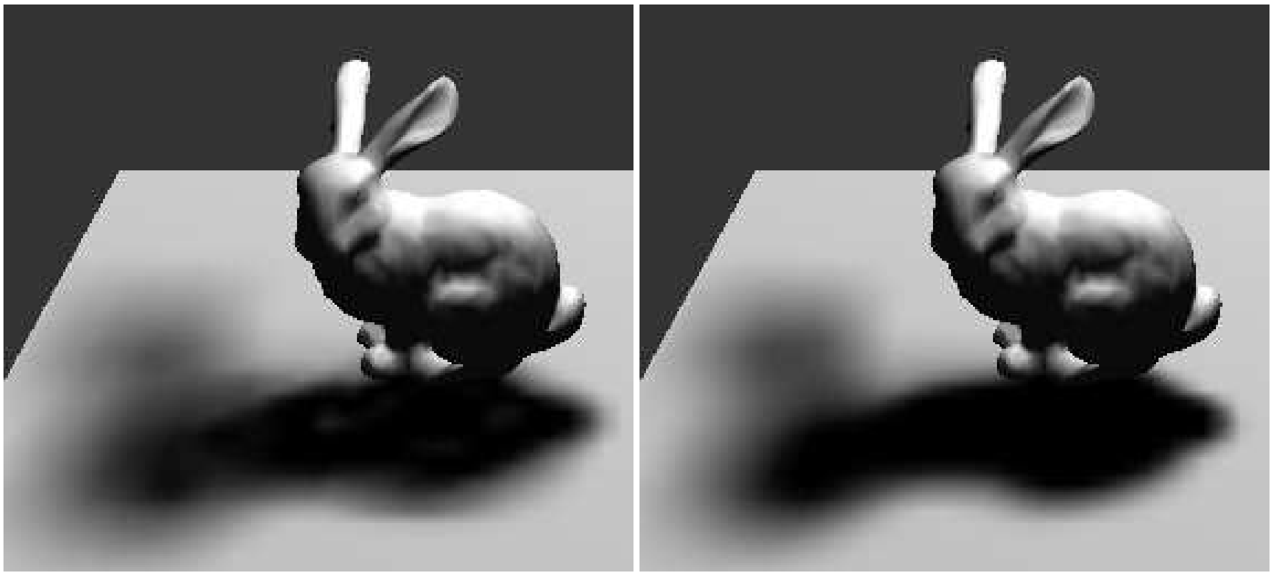
As you can see from Fig. 23(a), the rendering time increases with the size of the light source. What is interesting is the error introduced by our algorithm (see Fig. 23(b)). The error logically increases with the size of the light source, and for small light sources, larger bitmaps result in more accurate images. But for large light sources, a smaller bitmap will give a soft shadow of similar quality. A visual comparison of the soft shadows with a small bitmap and ground truth shows the small bitmap gives a very acceptable soft shadow (see Fig. 24).

This effect was observed by previous researchers: as the light source becomes larger, the features in the soft shadow become blurrier, hence they can be modeled accurately with a smaller bitmap.

5.3 Comparison with Soft-Shadow Volumes

Finally, we performed a comparison with a state-of-the art algorithm for computing soft shadows, the Soft-Shadow Volumes by Assarsson and Akenine-Möller [AAM03].

Fig. 25 shows the same scene, with soft shadows, computed by both algorithms. We ran the tests with a varying number of jeeps, to test how both algorithms scale with respect to the number of polygons. Fig. 25(c) shows the rendering times as a function of the number of polygons for both algorithms. These figures were computed using a window of 512×512 pixels for both algorithms.



(a) Bitmap of 64^2 (184 fps) (b) Ground truth

Figure 24: Large light sources with small bitmaps

(a) Soft Shadow Maps (b) Soft Shadow Volumes (c) Rendering times

Figure 25: Comparison with Soft-Shadow Volumes

Our algorithm scales better with respect to the number of polygons. On the other hand, soft shadow volumes provide a better looking shadow (see Fig. 25(b)), closer to the actual truth.

6 Conclusion and Future Directions

In this paper, we have presented a new algorithm for computing soft shadows in real-time on dynamic scenes. Our algorithm is based on the shadow mapping algorithm, and is entirely image-based. As such, it benefits from the advantages of image-based algorithms, especially speed.

The largest advantage of our algorithm is its high framerate, hence there remains plenty of computational power available for performing other tasks, such as interacting with the user or performing non-graphics processing such as physics computations within game engines. Possibly the largest limitation of our algorithm is the fact that it does not compute self-occlusion and it requires a separation between occluders and receivers. We know that this limitation is very important, and we plan to remove it in future work, possibly by using layered depth images.

An important aspect of our algorithm is that we can use low-resolution shadow maps in places with a large penumbra, even though we still need higher resolution shadow maps for places with small penumbra, for example close to the contact between the occluder and the receiver. An obvious improvement to our algorithm would be the ability to use hierarchical shadow maps, switching resolutions depending on the shadow being computed. This work could also be combined with perspective-corrected shadow maps [SD02, WSP04, MT04, CG04], in order to have higher resolution in places with sharp shadows close to the viewpoint.

In its current form, our algorithm still requires a transfer of the occluder map from the GPU to the main memory, and a loop, on the CPU, over all the pixels in the occluder map. We would like to design a GPU only implementation of our algorithm, using the future render-to-vertex capabilities.

7 Acknowledgments

The authors wish to thank Ulf Assarsson for doing the tests for the comparison with soft-shadow volumes.

References

[AAM03] ASSARSSON U., AKENINE-MÖLLER T.: A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003) 22*, 3 (July 2003), 511–520.

[AMH02] AKENINE-MÖLLER T., HAINES E.: *Real-Time Rendering*, 2nd ed. A K Peters, 2002.

[BS02] BRABEC S., SEIDEL H.-P.: Single sample soft shadows using depth maps. In *Graphics Interface* (2002).

[CD03] CHAN E., DURAND F.: Rendering fake soft shadows with smoothies. In *Rendering Techniques 2003 (Proceedings of the Eurographics Symposium on Rendering)* (June 2003), pp. 208–218.

[CD04] CHAN E., DURAND F.: An efficient hybrid shadow rendering algorithm. In *Rendering Techniques 2004 (Proceedings of the Eurographics Symposium on Rendering)* (2004), pp. 185–195.

[CG04] CHONG H., GORTLER S. J.: A lixel for every pixel. In *Rendering Techniques 2004 (Proceedings of the Eurographics Symposium on Rendering)* (2004).

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. *Computer Graphics (Proceedings of SIGGRAPH 77) 11*, 2 (July 1977), 242–248.

[DF94] DRETTAKIS G., FIUME E.: A fast shadow algorithm for area light sources using backprojection. In *Computer Graphics (SIGGRAPH 1994)* (1994), Annual Conference Series, ACM SIGGRAPH, pp. 223–230.

[HLHS03] HASENFRATZ J.-M., LAPIERRE M., HOLZSCHUCH N., SILLION F.: A survey of real-time soft shadows algorithms. *Computer Graphics Forum 22*, 4 (Dec. 2003), 753–774.

[KD03] KIRSCH F., DOELLNER J.: Real-time soft shadows using a single light sample. *Journal of WSCG (Winter School on Computer Graphics 2003) 11*, 1 (2003).

[KMK97] KERSTEN D., MAMASSIAN P., KNILL D. C.: Moving cast shadows and the perception of relative depth. *Perception 26*, 2 (1997), 171–192.

[LWGM04] LLOYD B., WENDT J., GOVINDARAJU N., MANOCHA D.: Cc shadow volumes. In *Rendering Techniques 2004 (Proceedings of the Eurographics Symposium on Rendering)* (2004), pp. 197–205.

[McC00] MCCOOL M. D.: Shadow volume reconstruction from depth maps. *ACM Transactions on Graphics 19*, 1 (2000), 1–26.

[MT04] MARTIN T., TAN T.-S.: Anti-aliasing and continuity with trapezoidal shadow maps. In *Rendering Techniques 2004 (Proceedings of the Eurographics Symposium on Rendering)* (2004).

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002)* (2002), 557–562.

[SS98] SOLER C., SILLION F. X.: Fast calculation of soft shadow textures using convolution. In *Computer Graphics (SIGGRAPH 1998)* (1998), Annual Conference Series, ACM SIGGRAPH, pp. 321–332.

[Wan92] WANGER L.: The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In *1992 Symposium on Interactive 3D Graphics* (Mar. 1992), vol. 25, pp. 39–42.

[WFG92] WANGER L., FERWERDA J. A., GREENBERG D. P.: Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications 12*, 3 (1992), 44–58.

[WH03] WYMAN C., HANSEN C.: Penumbra maps: Approximate soft shadows in real-time. In *Rendering Techniques 2003 (Proceedings of the Eurographics Symposium on Rendering)* (June 2003), pp. 202–207.

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *Computer Graphics (Proceedings of SIGGRAPH 78) 12*, 3 (Aug. 1978), 270–274.

[WPF90] WOO A., POULIN P., FOURNIER A.: A survey of shadow algorithms. *IEEE Computer Graphics & Applications 10*, 6 (Nov. 1990), 13–32.

[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Rendering Techniques 2004 (Proceedings of the Eurographics Symposium on Rendering)* (2004).

A Floating-point blending accuracy

In this section, we review the issues behind the hardware blending accuracy problems we have encountered and propose a temporary fix for these issues.

All the accuracy issues are linked to the fact that hardware blending is, at the time of writing, only available for 16-bits floating point numbers. NVidia graphics hardware stores these floating-point numbers using s10e5 format: one bit of sign, 10 bits of mantissa, 5 bits of exponent, with a bias of 15 for the exponent. The important point for addition is that the mantissa is stored on 10 bits. As a result, adding a large number X and a small number ε will give an inaccurate result if $\varepsilon < 2^{-10}X$:

$$X + \varepsilon = X \quad \text{if} \quad \varepsilon < 2^{-10}X \ \ (\text{inFP16})$$

For example, $2048 + 1 = 2048$ (in FP16 format) and $0.5 + \frac{1}{2049} = 0.5$ (also in FP16 format).

In some cases, the addition of the contribution from all micro-patches will be 1 (meaning complete occlusion of the light source). As a consequence, we can expect numerical accuracy issues if some micro-patches hide less than 2^{-10} of the light source. Because $32^2 = 2^{10}$, it means that the width of the reprojection of one micro-patch should be larger than $\frac{1}{32}$ of the width of the light source.

This translates easily into conditions for the position of the occluder:

$$\frac{1}{z_O} < \frac{1}{z_R} + \frac{64 \tan\alpha}{NL} \tag{4}$$

where L is the width of the light source, N is the resolution of the bitmap, α is the half-angle of the camera used to generate the shadow map, z_O is the distance between the light source and the occluder and z_R is the distance between the light source and the receiver.

Bitmap resolution: The most important thing is that increasing N makes this error *more* likely to appear. This explains why using a bitmap of 512×512 pixels we see a poor looking shadow, while the 128×128 bitmap gives the correct shadow (see Fig. 14).

Light source size: In equation 4, the size of the light source appears in a product with the resolution of the bitmap. If the light source is large, the bitmap must be low resolution in order to avoid FP16 blending errors. Fortunately, a large light source means a large penumbra for most occluders, so a low resolution bitmap might be enough for these penumbra effects.

Occluder position: As the occluder moves closer to the receiver, the likeliness of blending errors gets lower.

Camera half-angle: Similarly, increasing the camera half-angle improves the FP16 blending accuracy.

Basically, all these conditions amount to the same thing: using less pixels to describe the occluder in the shadow map. While this improves the FP16 blending accuracy, it obviously degrades the discretization of the occluder and also increases the overlapping between reprojections of neighboring pixels.

In our experiments (see Fig. 14) the blending accuracy problem appears very often when the resolution of the shadow map is larger than 512×512, sometimes with a shadow map resolution of 256×256 and very rarely with a shadow map resolution of 128×128.

The problem will disappear when hardware blending will become available on higher accuracy floating point numbers. FP32 have a mantissa of 23 bits, allowing the use of micro-patches that block less than 2^{-23} of the light source, meaning that the width of the back-projection of the micro-patch should be at least larger that 2^{-11} than the width of the light source (64 times smaller than the current threshold). Compared with the actual method, it would allow the use of shadow maps with a resolution above 4096×4096.

In the meantime, we use the following workaround: since the blending loses accuracy when we add a small number (the occlusion from a single micro-patch) to a potentially larger number (the sum of the occlusions from several micro-patches), we separate the blending in several steps, using an intermediate buffer; assuming we have M micro-patches in the discretization of the occluder, we split these M micro-patches into $\lfloor\sqrt{M}\rfloor$ batches, that will be added separately. The contribution of each batch is a larger scale number than the contribution of each micro-patch. As a consequence, they can be safely added together. The modified algorithm is:

- split the M micro-patches into $\lfloor\sqrt{M}\rfloor$ batches.
- for each batch:
  - sum the contribution of all the micro-patches, using FP16 blending

   - move the result to an intermediate buffer, adding it to the contributions of the previous batches.

- use the resulting sum for the soft shadow.

In this scheme, small values get added to small and medium values, and medium values get added to medium and large values. We never encounter the situation where a small value is added to a large one, thus we avoid accuracy issues.

Another workaround would be using a hierarchical shadow map, resulting in large patches in places with full occlusion, and smaller patches in the penumbra zone. We are working on this.

Depending on the scene and the relative position of the occluders, the workaround may or may not be necessary; for occluders that are not in contact with the receiver, high resolution is not necessary on the shadow map. In that circumstance, a shadow map resolution of 128×128 or smaller is sufficient in our experience, and blending accuracy issues are not visible.

Contents

List of Figures

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399