

Occlusion Textures for Plausible Soft Shadows

Elmar Eisemann and Xavier Décoret

ARTIS-GRAVIR/IMAG-INRIA[†]

Abstract

Authors' version: This paper presents a novel approach to obtain plausible soft shadows for complex dynamic scenes and a rectangular light source. We estimate the lights occlusion at each point of the scene using pre-filtered occlusion textures, which dynamically approximate the scene geometry. The algorithm is fast and its performance independent of the light's size. Being image based, it is mostly independent of the scene. No a priori information is needed, in particular there is no caster and receiver separation, making the method appealing and easy to use.

1. Introduction

Shadows are very important to estimate the spatial relationships of objects and to convey a sense of realism. For many years, computer graphics concentrated mostly on point lights which create hard shadows, surfaces being either lit or not. In the real world, most light sources are not punctual and create soft shadows made of umbra and penumbra regions. In the umbra no direct light arrives, whereas in the penumbra the light source is only partially occluded. The incoming light, or irradiance, at a small surface is given by a double integral over the surface and the light source, of a function involving energy, visibility and orientation. Several approaches [AAM02, AAM03, ADMAM03] have shown that the important information for convincing shadows lies in the visibility contribution. Orientation can be factored out of the integral. The problem thus simplifies to the calculation of the light's visible portion, which yet represents a challenging task. This paper presents a novel way to estimate this visibility function with the following properties:

- shadows are plausible, continuous, smooth, and account for real penumbræ (not just extended umbrae);
- the quality and speed of the method is independent of the size of the light source and the penumbræ;
- the algorithm is almost independent of scene complexity, does not require information about the scene and integrates well with various rendering paradigms (vertex shaders, point and image based rendering, . . .);
- there is no need to distinguish shadow casters from receivers.

Our approach is inspired by previous work [KM99, SS98] and exploits current graphics hardware to obtain real-time

performance for highly complex scenes. In general our method does not lead to physically correct soft shadows, as does no current real-time method. Nevertheless we show that our approximation behaves well in several test cases and compare the quality to reference solutions. Also we give a short error analysis that could be applied in practice to accelerate the algorithm.

The basic method described in this paper was originally published in [ED06b] at SIBGRAPI 2006. This extended version presents important implementation details, a solution for sunlight shadows, not previously described optimizations (Sec. 3.6), additional comparisons with previous work and a deeper analysis (Sec. 4.2 and 4.1).

2. Previous work

Lots of research focused on shadows and an exhaustive presentation is not possible here. We refer the reader to [WPF90, HLHS03] for surveys.

Point light sources do not create penumbræ and amounts to point-point-visibility. Image based techniques such as shadow mapping [Wil78] exploit this, yielding performance quasi independent of the actual scene complexity. This is interesting for complicated objects, but the discrete nature of images leads to aliasing. Percentage closer filtering [RSC87, BS01, Ura05] uses several shadow map samples to smooth the jaggy boundary of hard shadows. In [DL06], statistics are used to smooth the aliasing, but can introduce light leaks. Note that these artefact-smoothing methods are sometimes use to produce smooth shadows that should not be mistaken with soft shadows. Alternatively, shadow volumes [Cro77] give higher quality as they use the real objects geometry, but potentially suffer from computation overhead for costly for complex scenes (for vegetation, almost all edges are a silhouette, yielding shadow quads overdraw and expensive silhouette detection.

[†] ARTIS is a team of the GRAVIR/IMAG laboratory, a joint effort of CNRS, INRIA, INPG and UJF

In principle soft shadows could be created by evaluating several point lights. If sampling is too coarse, banding occurs. Using too many samples yields expensive creation and evaluation [Ura05], therefore, approximate methods have been proposed. Brabec and Seidel calculate maximum unoccluded radii on a depth map to approximate soft shadows [BS02]. In [KD03], appropriate width values are precalculated in a special map. Shadow maps have been used in [SAPP05] to calculate, in a preprocess, a special deep shadow map for a static light source, encoding occlusion for each point of a static scene. Dynamic objects can be inserted but cannot cast soft shadows. The somewhat opposite strategy was presented by Zhou et al. [ZHL*05]; shadow information is faithfully precalculated per object. These static elements can then be used dynamically. Due to huge storage necessities, the result is compressed on basis functions which are evaluated at run-time at each scene vertex, involving sorting. Agrawala et al. [ARHM00] warp several depth images to obtain layered attenuation maps which can be evaluated quickly at run-time. As the preprocess is quite involving, light source and scene have to be considered static. A purely image based approach has been presented by Arvo et al. [AHT04]. The camera view is filtered to detect hard shadow boundaries. Flood fill is used to create penumbras based on depth map information. This is not well supported by the current graphics hardware. The cost depends heavily on the size of the penumbra on the screen, which can be large depending on viewpoint and light source. As inner penumbra creation is an erosion, overlapping umbras lead to a complete disappearance and temporal incoherence. Although not physically correct, their goal was to create plausible shadows. We work in the same context but show in Section 4.1. Recently Atty et al. [AHL*06] presented an approach based on a single depth map. Separating occluder and receiver, depth map pixels replace the actual occluder and are projected on the receiving ground. The algorithm gives convincing results at high framerates but involves CPU usage which limits texture resolution. Related techniques were presented in [GBP06, BS06, ASK06], that replace the CPU interaction by a slightly more expensive fragment shader and don't need the occluder receiver separation anymore. The run-time complexity is linear with respect to the source's size and allows only to use rather small sources. Large sources also induce visible temporal incoherences (compare Section 4.1).

Shadow-volume extensions for soft shadows have been presented in [AAM02, AAM03, ADMAM03]. These geometry-based wedges give high quality shadows but rely on a silhouette determination from the center of the source. Sampling artifacts are avoided and light sources can be textured [AAM03], but the approaches are unsuitable for highly complex models. The combination of the contributions of different occluders is done by additive accumulation. Hybrid approaches like [CD03] encounter such problems too. Here supplementary primitives are attached to silhouettes to describe the blocking influence of the edge. It is closely re-

lated to [PSS98] and results in alias free, not soft shadows, just like [WH03]. In [KM99], the scene is replaced by an MDI (Multiple depth image) obtained on the CPU, and ray-tracing is performed against this alternative representation. There are several similarities to our method and we will have a more precise discussion in Section 4. Precise shadows and accurate occluder fusion [ED07, SS07] have only been recently introduced, but such approaches are more costly than the here-presented solution.

3. Our approach

We approximate the shadow intensity at a point P for a rectangular light source \mathcal{S} in the presence of an occluder \mathcal{O} by the integral:

$$I(P) := \int_{\mathcal{S}} v_P(S) dS \quad (1)$$

where v_P is the visibility function defined by:

$$v_P : S \in \mathcal{S} \rightarrow 1 \text{ if } [P, S] \cap \mathcal{O} = \emptyset \text{ else } 0 \quad (2)$$

In other words, we count the number of rays from P to \mathcal{S} that are not blocked by \mathcal{O} . The shadow intensity function is three dimensional and generally very complex. In subsequent sections, we present a GPU friendly approximation.

3.1. Single planar occluder

First we consider a single planar occluder parallel to the light source. It is fully described by its supporting plane $\Pi_{\mathcal{O}}$ and its characteristic function in that plane:

$$\delta : \Pi_{\mathcal{O}} \mapsto \{0, 1\}, Q \rightarrow 1 \text{ if } Q \in \mathcal{O} \text{ else } 0 \quad (3)$$

Consider the frustum defined by a point P and the light, and the region where it intersects the occluder plane. There is a bijection between that region and light rays passing through P . Therefore, the shadow intensity at P is the integral of $1 - \delta$ over this region, normalized by the regions's size (Fig. 1). Because the light is rectangular and parallel to the occluder,

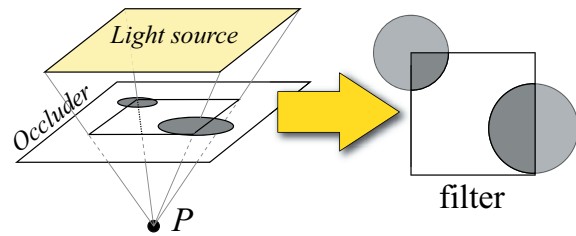


Figure 1: Shadow intensity as a filter.

this region is a rectangle whose size depends solely on the distances of P to the light and occluder planes:

$$s(P) := \frac{d(P, \Pi_{\mathcal{O}})}{d(P, \Pi_{\mathcal{S}})} \times \text{size}(\mathcal{S}) \quad (4)$$

The integral can be computed by filtering $1 - \delta$ with a box filter of size $s(P)$. Our approach is to encode $1 - \delta$ as an *occlusion texture* and to process it, as described in next section, such that a point P can be shaded by simply computing $s(P)$ using eq. (4) and performing a lookup of the appropriately filtered result.

3.2. Fast box filtering

To filter an occlusion texture with a rectangular kernel, we investigated three approaches: Mipmapping, NBuffers and Summed Area Tables.

Mipmapping was introduced to reduce aliasing of minified textures [Wil83]. It linearly interpolates between dyadically downsampled versions of the texture. It is widely supported by GPUs and was a natural candidate to filter occluder textures. In particular, rectangular kernels are supported with anisotropic filtering. Thanks to linear interpolation mipmapping gives smoothly varying shadows (Fig.2, top). However, it suffers from blocky artefacts that become particularly noticeable when the scene is animated. This results from the dyadic downsampling that may combine adjacent texels only at very high levels in the mipmap pyramid. Therefore, a slight shift of the occlusion texture can lead to large variations of the filtered function.

To alleviate this problem, Décoret introduced the NBuffers [D05] to allow prefiltering with continuously placed kernels. Originally this technique was used with a max filter and applied for culling of geometry. We use them to compute the mean value of neighboring pixels. Each level l holds, for each texel, the *normalized* response of a box filter with a kernel size of $2^l \times 2^l$. Via linear interpolation, intermediate kernel sizes can be approximated. It still does not compute the exact filtered function, but it significantly reduces the blocky artifacts (Fig.2 bottom) in particular for dynamic scenes. The construction of NBuffers is extremely fast. For a 256×256 texture, 8 levels need to be created, each of resolution 256×256 . Approximately 500M pixels are processed, each requiring exactly 4 texture lookups. Modern graphic cards can perform more than 50 times this amount of work at 30 fps.

Interestingly, the filtering by a rectangular kernel can be exactly computed using Summed Area Tables (SAT) [Cro84]. Unfortunately, although an efficient GPU implementation has been recently proposed [HSC*05], this approach still suffers from several limitations. First, 32 bit textures are required or severe precision artifacts will appear[†]. Currently, linear interpolation for such textures is not natively supported (this is why texels are noticeable on the close-up of Fig.2). Moreover, creation and transfer of such textures increase the bandwidth, slowing down the process.

[†] Shifting the values, as suggested by the authors, is not useful in our case, due to the bitmask nature of occluder textures

Speed is also impeded by the 4 texture lookups required to get the filter's response, when NBuffers require only one. Finally, contrary to NBuffers, the normalisation by the kernel size cannot be embedded and thus requires extra computations. However, as expectable, the resulting quality is higher, but the improvement does not compensate the performance penalty.

We implemented the three approaches (Fig. 2). In our opinion, NBuffers are currently the best tradeoff between performance and quality. On future hardware though, SAT may prove fast enough to become the preferred solution. In particular, it uses a single texture which is more cache friendly than the multiple textures required by NBuffers. To be able to perform a fair comparison we used the same hardware used in [HSC*05]. Our implementation leads to approximately the same timings (it differs by 1.5ms except for the case of a resolution of 512×512 where our implementation is 30% faster).



resolution	Mipmap (top)	SAT (middle)	NBuffers (bottom)
256×256	< 1 ms	5.1 ms	< 1 ms
512×512	< 1 ms	21.9 ms	1.7 ms
1024×1024	< 1 ms	96.8 ms	7.3 ms

Figure 2: Comparison of filtering methods.

3.3. Multiple planar occluders

We now consider several planar occluders. The shadows caused by each occluder independently can be computed as before. However, combining these shadows is a notably difficult problem. As pointed out in [SS98], the correct solution lies between the sum and the maximum of the occluders' contributions. Intuitively, two occluders can cover disjoint or overlapping parts of the light source.

Previous approaches more or less address this problem. In [SS98], the mean value between these two cases is suggested as an *ad hoc* solution. Assarson et al. use wedges to add or subtract light, and are thus inherently bound to combine them additively [AAM02, AAM03, ADMAM03]. To use a different combination method (still not exact) a costly

clearing step becomes necessary after each silhouette loop (see [ADMAM03]). Detecting silhouette loops also implies a lot more work on the CPU. During their floodfill, Arvo et al. [AHT04] need to keep track of the texel in a shadow map responsible for occlusion. When combining the occlusion for two such texels, it has to make a choice and therefore selects the one with maximum occlusion.

Additive approaches quickly saturate (it produces occlusion values greater than 100% that must be clamped) and overestimate the umbra: shadows look too dark and create unrealistic shadow gradients. Taking the maximum value gives visually more appealing results, but tends to create too bright shadows, in particular if the occluders are rather unstructured, like the foliage of a tree. Also gradient reversals happen easily in this situation. Taking the average does not make that much sense either, because the maximum only takes a single occluder into account, whereas the sum involves all occluders. Thus the ranges of these two values are too different to be meaningfully combined. We propose a novel way to combine the contributions.

Our key observation is that the probability that a ray from P to S is not blocked by the considered planar occluder is exactly $1 - V(P)$, where V is the shadow intensity function given by eq.(1). If we consider several occluders with a uniform distribution of occlusion, the probability that a ray is not blocked by the union of the occluders is the product of the probabilities. Thus we propose to combine the shadow intensities of several occluders using:

$$I_{1,\dots,n}(P) := \prod_{k=1}^n (1 - I_k(P)) \quad (5)$$

This formula has the same advantages as the sum. If an occluder does not block any ray ($I_k(P) = 0$), it does not influence the result. If it blocks all rays ($I_k(P) = 1$), the resulting intensity is zero. Compared to the maximum (Fig. 3), it does combine all occluders instead of selecting only one.

3.4. General scene

To treat general scenes, we approximate the occlusion they can cause with several occlusion textures. We cut the scene in slices parallel to the light source, and project everything inside a slice on its bottom plane (the one furthest from the light source). This positional information is binary and thus approaches like [DCB*04, ED06a] could be used to recover many layers at the cost of a single rendering step.

However, the more slices we have, the more texture lookups we need to compute the combined shadow (note that the cost of pre-filtering is mostly neglectable (see table 2). Currently we use 4 to 16 slices which seems to be a good tradeoff between speed and accuracy. To calculate the occluder texture representation, each slice has to be represented in one color channel. Multiple render targets (MRT) give the possibility to write into 4 buffers at the same

time, thus we can directly associate the slices to the correct color channel. This is related to lightweight methods such as [ND05].

The 4 to 16 color channels each represent one occluder texture. This pass is very fast and does not interfere with any CPU or GPU based vertex animation. Furthermore it is compatible with any kind of representation that produces a depth, such as point-based rendering, impostors, ray-tracing on GPU. Packing in color channels allows to compute mipmapping, NBuffers or SAT for four slices in parallel.

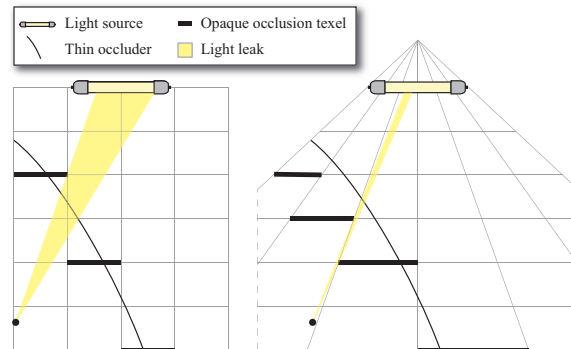


Figure 4: Orthogonal projection (left) causes more light leaks than perspective one (right).

The camera used during this rendering pass is very important as it controls how the scene is sliced. It is disadvantageous to use an orthogonal projection. First, a very large texture resolution is required for the camera's frustum to encompass the scene. Second, the projection onto occlusion textures breaks continuous surfaces into patches along lines not following the frustum center. The consequence is that light can shine through, where it is actually blocked by the real surface, causing *light leaks* (Fig.4). With perspective projection, the probability to have light leaks is much lower. Figure (4) shows the difference. On this figure, you can see that the center of projection (COP) is not placed on the light source. The reason is twofold. First, it would require a large field of view to encompass the scene, increasing texture distortion. Second, during the shadow computations, this would involve kernels that are large and that can jut out from the occlusion textures (Fig.5). This would rise interpolation as well as precision issues. Our choice is to place the COP slightly behind the light source, at a distance d , and to fit the frustum to the light source, choosing d so that the frustum contains the scene. We want to emphasize that using a projection from a particular COP only affects only the way we "x-ray" the scene to approximate occlusion, not the areas where shadows are computed. It does not relate to the approximation of silhouette edges from the center of the light source as in other methods, nor to the recovery of a depth map, which would only contain the first surfaces of a scene.

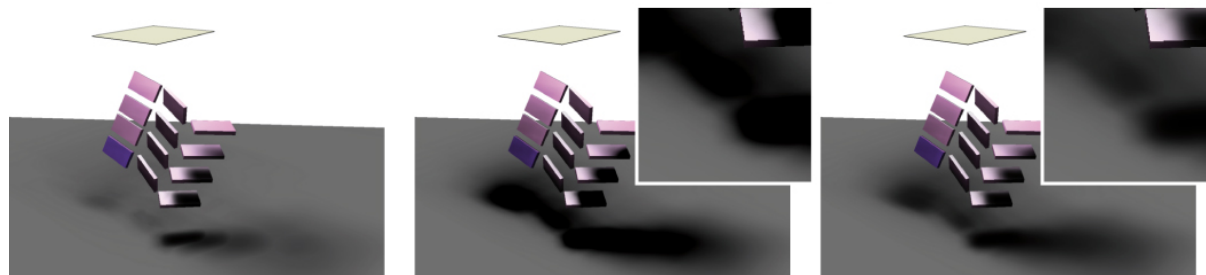


Figure 3: Comparison of maximum (left), sum (middle) and our combining approach (right).

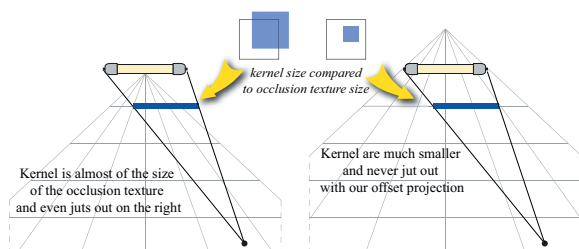


Figure 5: COP with offset: Filtering is simpler

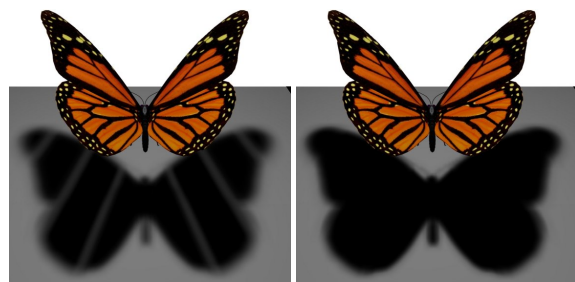


Figure 6: Fixing light leaks for thin occluders.

Light leaks.

As we have seen, our choice of perspective projection already limits light leaks, but some may still occur. It will become particularly visible in the case of thin geometry, such as a butterfly wings (Fig.6). For such geometry, we project each occlusion texture on its successor farther away from the source in order to “close” the discontinuities. Note that the projection can be performed *virtually*. During the slice creation, when converting distance into a color, it is sufficient to fill the succeeding channel too. This introduces no extra costs. Figure 6 shows how it drastically reduces the leaks. This method does probably not handle all situations but in practice, we did not encounter any leaks.

Projecting occlusion textures in this way affects the resulting shadows but only slightly. Umbrae are a little overestimated and the shadow gradient slightly differs. Conse-

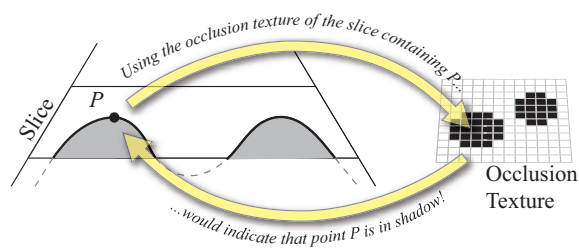


Figure 7: Auto-shadowing inside a slice.

quently, one can decide to enable this correction uniquely for thin objects, as for others, light leaks will be unlikely.

Self shadowing.

Our method does not distinguish shadow casters and receivers. Every point in the scene is shadowed, using only the occluder textures between it and the light source. The occlusion texture corresponding to the slice containing the point is not used, because any point in the slice would be shadowed by its own projection in the occlusion texture (Fig. 7). Simply ignoring the containing slice would cause discontinuities where the geometry crosses the clipping planes. Instead, we linearly fade out the contribution of a slice depending on the distance of the shaded point to the slice’s lower clipping plane.

Using slices for self occlusion is a coarse approximation, but it often works well in practice for the following reasons. For a slice far away from a point, the occlusion texture actually provides a good approximation of the occlusion caused by what is in the slice. For a slice nearby a point, it is theoretically more problematic but is often concealed by diffuse illumination. Indeed, for a watertight object, the front-facing faces block light from the back-facing ones. When they fall in the same slice, this effect is missed. However, the diffuse illumination for back-facing faces is zero and dominates the incorrect shadowing. They appear dark as they should (Fig.8). For non-watertight chaotic objects like trees, the diffuse illumination contains high frequencies which hide potentially incorrect shadowing. We insist that this concerns

only nearby slices. For distant slices, self-shadowing behaves correctly.

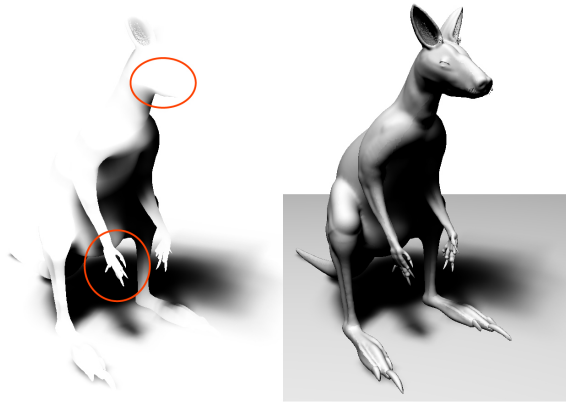


Figure 8: Our shadowing (left) may miss close self-occlusions. Fortunately diffuse illumination often compensates (right).

3.5. Putting everything together

Figure 9 summarizes the algorithm. The scene is sliced and projected onto occlusion textures, which involves one rendering of the scene from the light’s point of view (Sec. 3.4). These occlusion textures are pre-filtered with different kernel sizes (Sec. 3.2). A second render pass is performed from the observer’s point of view. For each point P of the scene, all slices between it and the light source are visited. The shadow caused by each slice is determined by performing a texture lookup with a filter size corresponding to the projection of light on the slice as seen from P . The shadow contributions are combined using the formula (5) which performs better than the maximum or the additive approach (Sec. 3.3). The contribution of the slice closest to P is weighted according to the distance of P to the slice, in order to obtain smooth inter-slice variation of the shadow intensity. The result is then combined with per-pixel Phong shading and textures.

3.6. Implementation details

Conceptually, the algorithm is simple (see alg. 1). To shade a fragment, we retrieve the corresponding world point W . Then, for each occlusion texture, we find the kernel position p and size s implied by W and the light, deduce the surrounding NBuffer levels l and $l + 1$, lookup filtered occlusion in these levels, linearly interpolate the values, and accumulate the occlusion. The problem is that this approach is not implementable for two reasons. First, shaders currently do not allow dynamic access of an array of textures. Line 6 and 7 thus cannot be translated to shader instructions. The second problem is that currently, shaders can only access a fixed

Algorithm 1 ideal computation of visibility V for fragment f and light L

```

1:  $V = 0$ 
2:  $W = \text{world\_coordinate}(f)$ 
3: for  $i$  in occlusion_textures do
4:    $p, s = \text{kernel\_pos\_and\_level\_for}(W, L, i)$ 
5:    $l = \lfloor s \rfloor$ 
6:    $v_{lo} = \text{lookup}(p, \text{nbuffers}[l])$ 
7:    $v_{hi} = \text{lookup}(p, \text{nbuffers}[l+1])$ 
8:    $v = (1 - (s - l))v_{lo} + (s - l)v_{hi}$ 
9:    $V = \text{accumulate}(v, V)$ 
10: end for

```

number of textures, typically 16. If we use 4 occlusion textures of 256×256 (thus encoding 16 slices in the RGBA channels), we need $8 = \log_2(256)$ levels of NBuffers. Even on a simple configuration like this, $4 * 8 = 24$ different textures are used. We get around these two problems by using texture packing, and sequencing the algorithm in a way that texture arrays are accessed statically.

Let’s start with packing. Our 16 slices are encoded in 4 occlusion textures. Instead of generating 8 NBuffers for each, we generate 8 textures and pack in texture i the NBuffer level i for each occlusion texture (resulting in 8 textures of $4 \times 256 \times 256$). We now use only 8 textures instead of 24. With a resolution of 2048×2048 , we would only need three more NBuffer levels, thus still fitting the 16 textures limitation (and in particular leaving 5 textures usable for conventional shading).

Let’s now see how we re-order the algorithm. The key observation is that each slice i should be filtered with a kernel of size s_i , and that s_i is strictly increasing from the slice closest to the shaded point to the slice closest to the light. Thus, we can loop over the NBuffer levels l in order of increasing kernel size s_l , and increment a current slice index i . This index is initialized with the slice closest to the point and is increased every time $s_i < s_l$. Because of packing, this index is a shift of the horizontal texture coordinate used to access the current NBuffer level. This second version (see alg.2) works because the loop at line 5 is a static one, completely determined by the resolution of the occlusion textures.

Line 8 requires a comment. Because we encode four slices in the RGBA channels of each occlusion texture, increasing the current slice index is a bit more tricky than using $i = i + 1$. Luckily, this can be done efficiently using the swizzle operator of shading languages, and clever tricks. We use a `float4` for slice index and implement line 8 and 9 with `i=i.yzwx` and `delta += i.w*packing_offset`. Then, in line 12 and 13, when we do the lookup, we get back four slices as one RGBA color, and we extract the relevant one by doing a dot product with `i`. Note that this approach is purely arithmetic and no branching is used (also conceptually

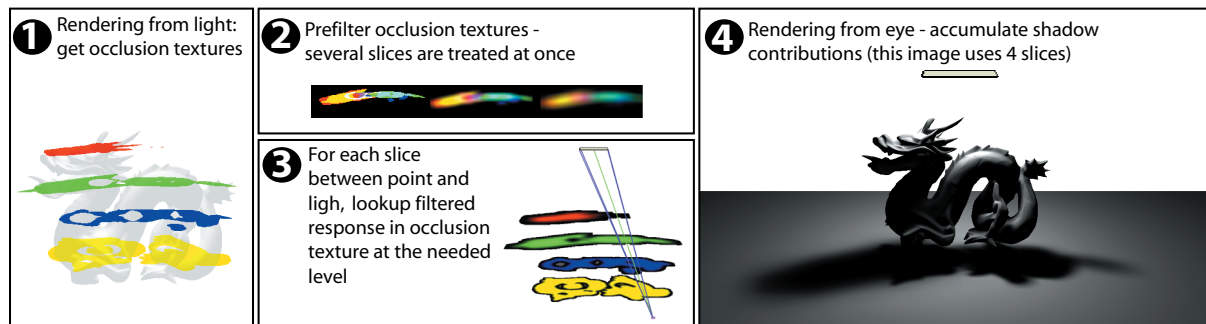


Figure 9: Summary of our algorithm.

Algorithm 2 practical computation of visibility V for fragment f and light L

```

1:  $V = 0$ 
2:  $W = \text{world\_coordinate}(f)$ 
3:  $i = \text{index\_slice\_closest\_to}(W)$ 
4:  $p, s = \text{kernel\_pos\_and\_level\_for}(W, L, i)$ 
5:  $\delta = i * \text{packing\_offset}$ 
6: for  $l$  in  $\text{nbuffers\_levels}$  do
7:   while  $s < \text{kernel\_size\_for\_level}(l)$  do
8:      $i = i + 1$ 
9:      $\delta += \text{packing\_offset}$ 
10:     $p, s = \text{kernel\_pos\_and\_level\_for}(W, L, i)$ 
11:   end while
12:    $v_{lo} = \text{lookup}(p + \delta, \text{nbuffers}[l])$ 
13:    $v_{hi} = \text{lookup}(p + \delta, \text{nbuffers}[l + 1])$ 
14:    $v = (1 - (s - l))v_{lo} + (s - l)v_{hi}$ 
15:    $V = \text{accumulate}(v, V)$ 
16: end for

```

ally, it amounts to tests, and can also be implemented using `if/then` constructs).

A couple of other optimization can be done, but are not presented here for conciseness. In particular, packing offset and kernels position/size actually only depend on the distance from the shaded point to the COP. This simplifies and factors several computations. Deferred shading is also used to avoid doing computations on hidden fragments. We believe the reader can figure out these details by himself, and only focused on describing the nifty part of the algorithm. A detailed implementation will be available on the paper website.

3.7. Future Hardware Extensions

In DirectX 10 texture stacks will be introduced, that would make the presented work-around unnecessary and will improve our performance even further.

One possibility nowadays is to arrange NBuffer levels in

a 3D texture making the linear interpolation automatically available without extra cost. Currently, even though the function to render into 3D texture slices is documented it is not yet implemented in the drivers. Having this option the texture packing would also become unnecessary, making the algorithm even more efficient. Also the amount of texture units is increased making it possible to keep the textures separate without performing packing. Currently in our implementation, 16 slices impose a maximal resolution of 2096^2 . This is very acceptable, but texture stacks will even lift this restriction completely.

As mentioned before more than 16 slices could be generated from the scene using techniques as in [DCB*04, ED06a]. Redistributing the slices into color channels can be done independently of the scene geometry. This would also allow to do a first scene analysis to better fit slices to the objects. Currently we simply use a uniform distribution. Another interesting aspect is the fact that 32 bit textures will become available. Thus in one color channel several occlusion layers could be packed. This would make the lookups more cache friendly and accelerate the algorithm even further. New hardware will be based on scalar rather than vector processors, which is perfect for our purposes, because the color channels are treated separately in our algorithm and color is not use in its standard meaning.

4. Results and discussion

Our work is similar in spirit to that of Keating and Max [KM99] but the field of application is completely different. Their approach does not aim at real time and targets ray-tracing. Even without averaging several rays, it is still presented in a form that would not allow real-time performance. It uses small kernels mostly to avoid noise and applies it similarly to percentage closer filtering [RSC87]. Instead, we use convolution for acceleration purposes. We presented several solutions to approximate filtering efficiently, rather than performing it. Our method thus treats large light sources without penalty. Occlusion textures are efficiently created on the GPU and we avoid any CPU interaction. We

combine contributions differently, based on probability, and obtain convincing results without evaluating several sample rays. Of course, ray-tracing produces more realistic images.

We implemented our method using Cg 1.5 shading language and OpenGL. To make it possible to compare our algorithm to others we used the same test system as in [AHL*06] (a 2,4 Ghz Pentium 4 with a GeForce 6800 Ultra). Both the render pass to slice the scene and the computations of NBuffers are very fast, thus the rendering cost of our method is dominated by the final render pass and is almost the same in all our tests. Most of the images we show are levels of gray. This is to emphasize the shadows. Our method works seamlessly with textures and would even benefit from their presence, since texture maps would mask minor shadowing artifacts. Similarly, most of our examples show cast shadows on a flat ground to ease the perception. An arbitrary ground is possible. In practice it could be advantageous to know that some elements of the scene are a floor and it could be simply excluded from the slicing and afterwards take all slices into account. In practice we did not rely on a special treatment and we want to emphasize again that there is no caster/receiver distinction in our methods. Grazing angles on almost planar geometry can lead to problems

4.1. Quality Comparison

Our shadows are plausible and smooth even for extreme low resolution of occlusion textures. Separately, each single occlusion texture is piecewise linear and has a blocky appearance for a low resolution. Combining non-aligned textures leads to an artificial increase of resolution. This can be interpreted in terms of frequencies [DHS*05]. Slices can be seen as a decomposition of the shadow on basis functions. Each slice is looked-up with distinct filter size and thus represents a separate frequency range. A combination is a wealth of information, that is not equivalent to a single texture. Nevertheless, too low resolutions would still introduce artifacts during animation. The smallest entity is a pixel, thus the blocking contribution of very fine objects can be overestimated or missed. This is a problem that we share with all image based methods.

We believe that the introduction of occlusion textures to shadows is very important. Depth map based approaches can suffer from visible temporal incoherences for large sources in even simple situations. Imagine a small occluder close to the light. It might not even create an umbra region, but all objects, that are hidden in the depth map will not cast any shadow at all. Therefore whenever an object passes over another one there is a flickering in the penumbra region. Atty et al. [AHL*06] relied on two shadow maps to overcome this problem. Guennebaud et al. [GBP06] do not have this option, because they would need to separate occluder and receiver. Thus they are restricted to small sources not only by performance but also quality issues. For small light sources, due to a smaller overhead the technique is preferable, al-

though in this case percentage closer filtering approaches work well. Our method has less problems with occlusion because we rely on a slicing. Nevertheless, slicing might miss nearby occlusions, thus a slight flickering may still appear for vertical movements and grazing angles as shown in the accompanying video.

Interestingly although our approach is based on a very coarse assumption, the results are very convincing. We tried several very different test scenarios and compared our technique to a reference solution based on sampling.

The first scene is shown in figure ?? . Even 4 slices give a good approximation for the trees branches. Realize that the overlapping of the branches would cause severe problems for depth map based methods. The fact that the shadows of the big tree are very accurate on the ground is not a coincidence. In fact if you take a planar occluder and you move it by a distance ϵ it has more influence if the occluder was close to the receiving point. Basically this influence is relative to the distance. Now in our case this movement is a projection, that is almost vertical. which is even better, because the error is also relative to the angle. In other words, the further away objects are, the more aggressively they could be simplified. This is directly exploitable in our algorithm as by recovering the result from one NBuffer texture we actually get the response for 4 slices found at the same distance. Also if the ground of the scene is known one could use more planes close to the ground, less close to the light. Nevertheless in the current implementation we did not make use of this. Because we want to capture self occlusion and thus suppose that at each height there is the same density of geometry (casting and receiving). In this particular scene it would have been a good idea to use less slices for the tree (in particular because the self occlusion is visible, but rather restricted) and the shadows on the ground are more important.

The second scene is a frame from an animation with a moving light. We did not even rely on bounding boxes and did not place all slices on the bunny (In particular you see the last slice being even unusable). We then stopped the animation at a random point. Again distant shadows are well approximated with a little number of slices. Whereas self shadowing of the ears is lost using 4 slices.

The next scene shows the dragon casting a shadow on an uneven ground. Still we remain close to the original solution. The ground is not treated any different than the dragon in this scene, there is no separation. As usual the highest error occurs at detailed geometry level.

The complicated box scene from figure 3 is interesting as it shows that our probabilistic combination is a very good trade-off between the two extremities.

Finally figure 3 shows the influence of texture resolution. For big lights several image based approaches argued that it is sufficient to take low resolution textures. This is not generally true. The tree details would be missed, making the

shadows flicker slightly when the texture resolution is very low. Even though the still looks appealing. The quality is high even for lower resolutions, when zooming in we can see the artifacts from sampling with almost 1000 samples. The light source is huge (the size of the whole scene) and the branches add an almost random noise, whereas our approach smoothes these artifacts out due to the filtering.

4.2. Timings

In this section we compare the speed of our algorithm with the current state-of-the-art solutions by Assarson et al. [ADMAM03] and Atty et al. [AHL*06].

We used the jeep scene from [AHL*06] with a varying number of jeeps. It is obvious that geometry based algorithms are quickly outperformed by image based techniques. Soft shadow volumes do not scale to a usable amount of geometry. Atty's algorithm behaves linear, just like ours, but the fact that lots of fragments have to be projected on the source, just like in Guennebaud's approach, the algorithm slows down rapidly with the number of occluding fragments. We do not have to reproject, because this calculation is already inherent in the filtering, our solution depends solely on a render step of the geometry not the scene configuration. We want to underline that we compared with our solution using 16 slices and a more than four times higher texture resolution, even though 4 slices and 128^2 would already have given a satisfactory result. Not only can we deal with big texture sizes, we also outperform the current state-of-the-art image based soft-shadow algorithm that even benefits from an occluder/receiver separation and cannot treat self shadowing.

Finally figure ?? depicts a more challenging scene, and our resulting frame rate. Each bunny has approximately 70.000 polygons, thus leading to a final complexity of almost 500.000 polygons and we still have a real-time performance.

5. Conclusion and Future Work

We presented a novel image-based soft shadow algorithm, that is fast and especially well-adapted to GPU. It does not rely on precalculation and integrates smoothly with animated scenes. The resulting shadows are plausible although not physically correct. In particular, inner and outer penumbras are handled. Although the method is image based, shadows are smooth, even at low texture resolution. Some artifacts can occur, due to the limited number of slices/resolution and approximated filtering; self-occlusion might fail locally and a slight flickering can occur for vertical movements as the weighting for the closest slice changes. However, the complex task of inter-object shading is seamlessly handled, through the introduction of a novel way of combining shadow contributions based on probabilities. No distinction between casters and receivers is required. The method is output sensitive depending only on the amount of

shaded points rather than on the nature or size of the shadow. To our best knowledge, it is the only approach that possesses all these properties. It outperforms current state-of-the-art algorithms and delivers very convincing shadows relatively close to the ground truth.

An important area of future investigation concerns the slice placement. Litmaps introduced in [D05], or CC Shadow volumes [LWGM04] could serve to determine the receivers in a scene, thus we can based on this information place less slices in the remaining part based on the mentioned error measure. It is possible to create per object representations, relating to the idea that the slices should evolve in a "continuous" way with objects and viewpoint.

Finally, hierarchical branching could be interesting, as one lookup gives us information about four slices. On our test hardware, it is currently not advantageous, showing that shader optimization becomes difficult.

Acknowledgements: We thank the reviewers for their comments and remarks. Special thanks go to S. Lefebvre for his suggestions and very early input. We also thank especially C. Soler and H. de Almeida Bezerra for several discussions. We want to thank U. Assarson, L. Atty, N. Holzschuch, M. Lapierre, J.-M. Hasenfratz, C. Hansen and F. Sillion for the comparison with our algorithm. Stanford, DeEspona for the models.

References

- [AAM02] ASSARSON U., AKENINE-MÖLLER T.: Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Proc. of Workshop on Rendering'02* (2002), Springer Computer Science, Eurographics, Eurographics. 1, 2, 4
- [AAM03] ASSARSON U., AKENINE-MÖLLER T.: A geometry-based soft shadow volume algorithm using graphics hardware. In *Proc. of Siggraph'03* (2003). 1, 2, 4
- [ADMAM03] ASSARSON U., DOUGHERTY M., MOUNIER M., AKENINE-MÖLLER T.: An optimized soft shadow volume algorithm with real-time performance. In *Proc. of Workshop on Graphics Hardware'03* (2003). 1, 2, 4, 9
- [AHL*06] ATTY L., HOLZSCHUCH N., LAPIERRE M., HASENFRATZ J.-M., HANSEN C., SILLION F.: Soft shadow maps: Efficient sampling of light source visibility. *Computer Graphics Forum* (2006). 2, 8, 9
- [AHT04] ARVO J., HIKORPI M., TYYSTJÄRVI J.: Approximate soft shadows with an imag-space flood-fill algorithm. In *Proc. of Eurographics'04* (2004). 2, 4
- [ARHM00] AGRAWALA M., RAMAMOORTHY R., HEIRICH A., MOLL L.: Efficient image-based methods for rendering soft shadows. In *Proc. of Siggraph'00* (2000). 2

- [ASK06] ASZODI B., SZIRMAI-KALOS L.: Real-time soft shadows with shadow accumulation. In *Short Paper Eurographics* (2006). 2
- [BS01] BRABEC S., SEIDEL H. P.: Hardware-accelerated rendering of antialiased shadows with shadow maps. In *Proceedings of CGI'01* (2001). 1
- [BS02] BRABEC S., SEIDEL H.: Single sample soft shadows using depth maps. In *Proc. of Graphics Interface'02* (2002). 2
- [BS06] BAVOIL L., SILVA C. T.: Real-time soft shadows with cone culling. In *Technical Sketch at SIGGRAPH* (2006). 2
- [CD03] CHAN E., DURAND F.: Rendering fake soft shadows with smoothies. In *Proc. of Symposium on Rendering'03* (2003). 2
- [Cro77] CROW F.: Shadow algorithms for computer graphics.in computer graphics. In *Proc. of Siggraph'77* (1977). 1
- [Cro84] CROW F. C.: Summed-area tables for texture mapping. In *Proc. of Siggraph'84* (1984). 3
- [DÓ5] DÉCORET X.: N-buffers for efficient depth map query. In *Proc. of Eurographics'05* (2005). 3, 9
- [DCB*04] DONG Z., CHEN W., BAO H., ZHANG H., PENG Q.: Real-time voxelization for complex polygonal models. In *Proc. of Pacific Graphics'04* (2004). 4, 7
- [DHS*05] DURAND F., HOLZSCHUCH N., SOLER C., CHAN E., SILLION F.: A frequency analysis of light transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)* 24, 3 (aug 2005). 8
- [DL06] DONNELLY W., LAURITZEN A.: Variance shadow maps. In *Proc. of I3D'06* (2006). 1
- [ED06a] EISEMANN E., DÉCORET X.: Fast scene voxelization and applications. In *Proc. of I3D'06* (2006). 4, 7
- [ED06b] EISEMANN E., DÉCORET X.: Plausible image based soft shadows using occlusion textures. In *Proceedings of SIBGRAPI 2006* (Oct. 2006), pp. 155–162. 1
- [ED07] EISEMANN E., DÉCORET X.: Visibility sampling on GPU and applications. *Computer Graphics Forum (Proceedings of Eurographics 2007)* 26, 3 (Sept. 2007), 535–544. 2
- [GBP06] GUENNEBAUD G., BARTHE L., PAULIN M.: Real-time soft shadow mapping by backprojection. In *Eurographics Symposium on Rendering* (2006). 2, 8
- [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). 1
- [HSC*05] HENSLEY J., SCHEUERMANN T., COOMBE G., LASTRA A., SINGH M.: Fast summed-area table generation and its applications. In *Proc. of Eurographics'05* (2005). 3
- [KD03] KIRSCH F., DOELLNER J.: Real-time soft shadows using a single light sample. *Journal of WSCG* (2003). 2
- [KM99] KEATING B., MAX N.: Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. In *Proc. of Workshop on Rendering'99* (1999). 1, 2, 7
- [LWGM04] LLOYD B., WENDT J., GOVINDARAJU N. K., MANOCHA D.: Cc shadow volumes. In *Proc. of EG Symposium on Rendering'04* (2004), Springer Computer Science, Eurographics, Eurographics Association. 9
- [ND05] NGUYEN H., DONNELLY W.: *Hair Animation and Rendering in the Nalu Demo*. Addison Wesley, 2005. 4
- [PSS98] PARKER S., SHIRLEY P., SMITS B.: *Single sample soft shadows*. Tech. Rep. UUCS-98-019, University of Utah, 1998. 2
- [RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *Proc. of Siggraph'87* (1987). 1, 8
- [SAPP05] ST-AMOUR J.-F., PAQUETTE E., POULIN P.: Soft shadows from extended light sources with penumbra deep shadow maps. In *Proc. of Graphics Interface'05* (2005). 2
- [SS98] SOLER C., SILLION F.: Fast calculation of soft shadow textures using convolution. In *Proc. of Siggraph'98* (1998). 1, 3, 4
- [SS07] SCHWARZ M., STAMMINGER M.: Bitmask soft shadows. *Computer Graphics Forum (Proceedings of Eurographics 2007)* 26, 3 (Sept. 2007), 515–524. 2
- [Ura05] URALSKY Y.: Efficient soft-edged shadows using pixel shader branching. In *GPU Gems 2* (2005), Addison Wesley. 1, 2
- [WH03] WYMAN C., HANSEN C.: Penumbra maps: Approximate soft shadows in real-time. In *Proc. of Symposium on Rendering'03* (2003). 2
- [Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *Proc. of Siggraph'78* (1978). 1
- [Wil83] WILLIAMS L.: Pyramidal parametrics. In *Proc. of Siggraph'83* (1983). 3
- [WPF90] WOO A., POULIN P., FOURNIER A.: A survey of shadow algorithms. *IEEE Comput. Graph. Appl.* 10, 6 (1990). 1
- [ZHL*05] ZHOU K., HU Y., LIN S., GUO B., SHUM H.-Y.: Precomputed shadow fields for dynamic scenes. In *Proc. of Siggraph'05* (2005). 2