Plausible Image Based Soft Shadows Using Occlusion Textures

Elmar Eisemann   Xavier Décoret
ARTIS-GRAVIR/IMAG-INRIA*
Elmar.Eisemann@inrialpes.fr   Xavier.Decoret@inrialpes.fr

Abstract

This paper presents a novel image-based approach to render plausible soft shadows for complex dynamic scenes with rectangular light sources. The algorithm's performance is mostly independent of the scene complexity and the source's size. Occluders and receivers do not need to be separated and no knowledge about the scene representation is required, making the method easy to use. The main idea is to approximate the occlusion in the scene with pre-filtered occlusion textures. The visibility of the light source at a point in space is estimated by accumulating the occlusion caused by each texture, using a novel formula based on probabilities.

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 [3, 4, 5], 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:

- the resulting shadows are plausible, continuous and smooth, and account for real penumbrae (not just extended umbrae);

- the quality and speed of the method is independent of the size of the light source and the penumbrae;

- 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...);

- in particular, there is no need to distinguish shadow casters from receivers.

Our approach is inspired by previous work [20, 26] and exploits current graphics hardware to obtain real-time performance for highly complex scenes.

2 Previous work

Lots of research focused on shadows and an exhaustive presentation is not possible here. We refer the reader to [31, 18] for surveys.

Point light sources do not create penumbrae and represent a direct equivalence to point-point-visibility. This is exploited by image based techniques such as shadow mapping [29]. Using images results in independence to the actual scene complexity which is interesting for complicated objects. On the other hand, the discrete nature of images leads to aliasing. Percentage closer filtering [25, 8, 28], uses several shadow map samples to smooth the jaggy boundary of hard shadows. In [14], statistics are used to smooth the aliasing, but can introduce light leaks.

Shadow volumes [10] give higher quality, but they are potentially costly for complex scenes, like trees, where almost all edges is a silhouette. Shadow quads overdraw and silhouette detection become too expensive.

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 [28], therefore, approximate methods have been proposed. Brabec and Seidel calculate maximum unoccluded radii on a depth map to approximate soft shadows [7]. In [21], appropriate width values are precalculated in a special map. Shadow maps have been used in [27] 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. [34]; 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.[1] 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. [2]. The camera view is filtered to detect hard shadow boundaries. Flood fill is used to create penumbrae 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. Recently Atty et al. [6] 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. Guennebaud et al. [17] presented a related technique that eliminates most constraints but has linear run-time complexity with respect to the source's size.

Shadow volumes extensions for soft shadows have been presented in [3, 4, 5]. These geometry-based wedges give high quality shadows but rely on silhouette determination from the center of the source. Sampling artifacts are avoided and light sources can be textured [4], 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 [9] encounter such problems too. Here supplementary primitives are attached to silhouettes to describe the blocking influence of the edge. It is closely related to [24] and results in alias free, not soft, shadows, just like [32]. In [20], the scene is replaced by an MDI (Multiple depth image) ob-

tained 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.

3 Our approach

As explained in introduction, 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:

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

where v_P is the visibility function defined by:

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

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 describe 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 it supporting plane $\Pi_{\mathcal{O}}$ and its characteristic function in that plane:

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

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,
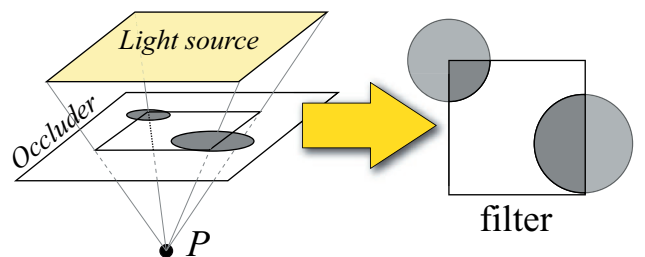

Figure 1. Shadow intensity as a filter.

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

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

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 a 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 tried three approaches: Mipmapping, NBuffers and Summed Area Tables.

Mipmapping was introduced to reduce aliasing of minified textures[30]. It linearly interpolates between dyadically downsampled versions of the texture. It is 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, using mipmapping gives smoothly varying shadows (Fi.2, top). However, it suffers from blocky artefacts that become particularly noticeable when the scene is animated. This is because the dyadic downsampling 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 [12] to allow prefiltering with continuously placed kernels. 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) [11]. Unfortunately, although an efficient GPU implementation has been recently proposed [19], this approach still suffers from several limitations. First, 32 bit textures are required or severe precision artifacts will appear[1]. 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. 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 have implemented the three approaches. Figure 2 shows a qualitative and quantitative comparison. 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.
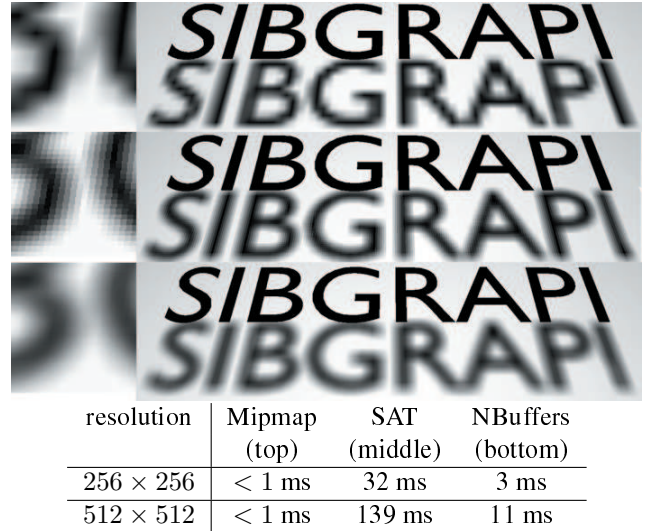
resolution	Mipmap (top)	SAT (middle)	NBuffers (bottom)
256×256	< 1 ms	32 ms	3 ms
512×512	< 1 ms	139 ms	11 ms

Figure 2. Comparison of filtering methods.

3.3 Multiple planar occluders

Let's now consider two or more 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 [26], 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 [26], 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[3, 4, 5]. During their floodfill, Arvo et al. [2] 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

---

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

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. 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 averaged. We propose a novel way to combine the contributions based on probabilities.

Our key observation is that the probability that a ray from P to $\mathcal{S}$ 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:

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

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, it does combine all occluders instead of selecting only one. Figure 3 shows a comparison.

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). Since we only need binary information in the occluder textures, approaches like slicemaps [13, 16] can be used to obtain many slices and would readily be usable with our approach. However, the more slices we have, the more texture lookups we must do to compute the combined shadow (note that the cost of pre-filtering is neglectable). In practice, we observed that 4 to 16 slices is a good tradeoff between speed and accuracy. Thus lightweight methods such as [23] can be used.

We perform a single rendering pass and a simple fragment shader to construct one to four RGBA textures (using MRTs). 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. It can actually be used with any modelling primitive and rendering method that produces a depth, such as point-based rendering, impostors, ray-tracing on GPU. One benefit of packing occluder textures in color channels is that mipmapping,

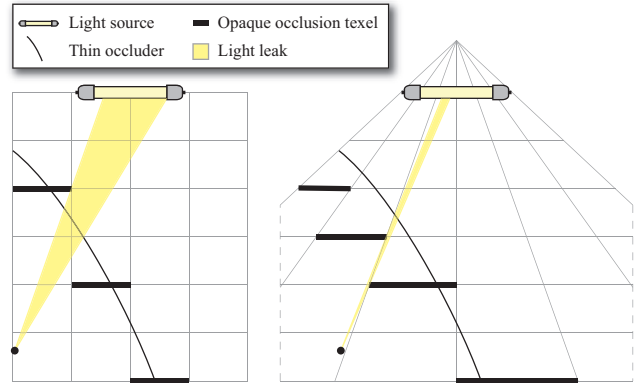NBuffers or SAT can be then computed for four slices in one step.

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. Using an orthogonal projection has two disadvantages. 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 lot of 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 distorsion. Second, during the shadow computations, this will involve kernels that are large and that can jut out from the occlusion textures (Fig.5). This would straighten the artifacts of the pre-filtering methods (interpolation or 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 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.

Light leaks.

As we have seen, our choice of perspective projection already limits the 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
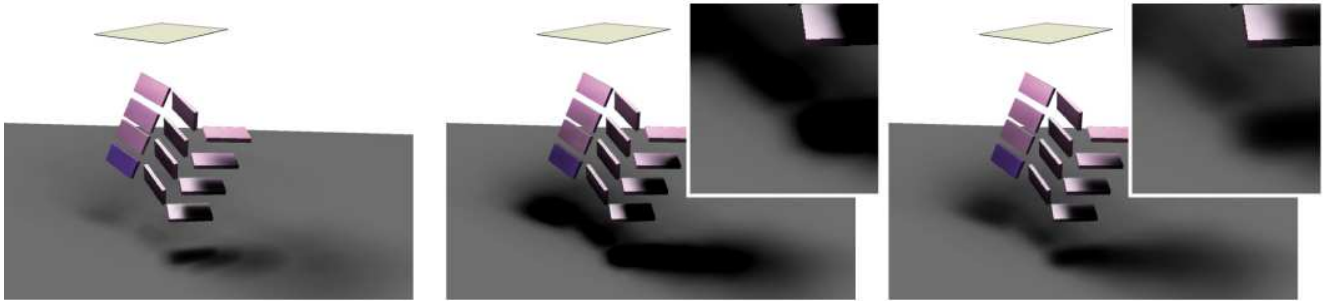
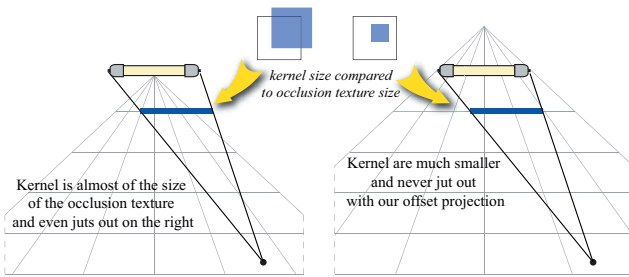**Figure 3. Comparison of maximum (left), sum (middle) and our combining approach (right).**
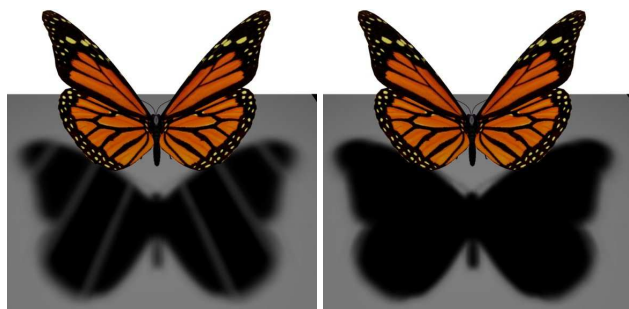

Figure 5. COP with offset: Filtering is simpler

Figure 7. Auto-shadowing inside a slice.

Figure 6. Fixing light leaks for thin occluders.

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 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-

project each occlusion texture on its successor farther away from the source in order to "close" the discontinuities. Realize, that this 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 during our tests.

Projecting occlusion textures in this way affects the resulting shadows but only slightly. Umbrae are a little over-estimated and the shadow gradient slightly differs. Consequently, one can decide to enable this correction uniquely for thin objects, as for others light leaks will be unlikely.
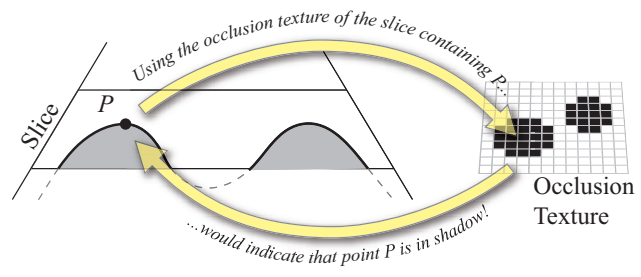
shadowing behaves correctly and in particular, we do not need to distinguish casters from receivers.
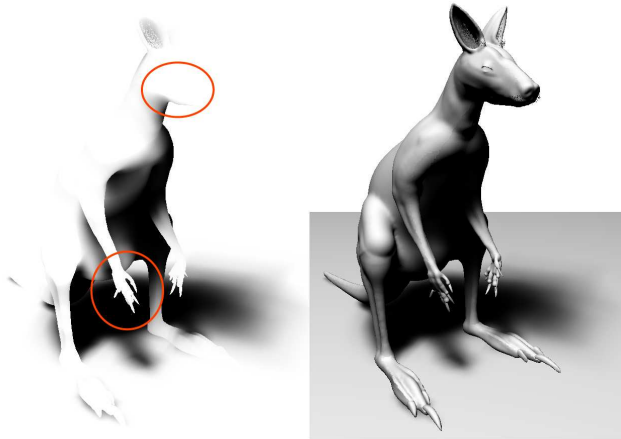

Figure 8. Our shadowing (left) may miss close self-occlusions. Fortunately diffuse illumination often compensates (right).

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, our algorithm is simple. The fragment shader must get the world coordinates of the current fragment, and loop over the occlusion textures. For each, it computes the appropriate kernel size and then deduces the two levels of the NBuffer to interpolate for the filter response. However, this conversion is a complex formula. Moreover, it is not possible to randomly index an array of textures in a shader. Fortunately, it is possible to rewrite this

conceptual algorithm in a practical one. We maintain a current slice index, starting with the one closest to the point. We then loop over the NBuffer levels. Inside the loop, if the kernel size required for the current slice is between that of the current and next levels, we accumulate the shadow contribution and update the current slice index. This works because the kernel sizes are increasing along the slices. Various tricks are used, such as using swizzling masks to increase the current slice index, knowing that slices are packed in RGBA channels (`s = float4(1,0,0,0); s = s.wxyz;`). Moreover, since we have 4 to 16 occlusion textures with several NBuffer levels for each, the limit of 16 texture units is reached and texture packing of the NBuffers must be done[2]. A detailed description was beyond the scope of this paper. Please refer to its webpage for details (via the authors' webpages on http://artis.imag.fr/).

4 Results and discussion

We implemented our method using Cg 1.4 shading language and OpenGL, running on a Pentium 4 with a GeForce 7800. Figure 10 shows some results. 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. For 800×600 images, we run between 20Hz and 30Hz. 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, and we want to emphasize again that there is no caster/receiver distinction in our methods.

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. Combining non-aligned textures leads to an artificial increase of resolution. This can be interpreted in terms of frequencies [15]. 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 range of frequencies. 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. Also, as the smallest entity is a pixel, the blocking contribution of very fine objects can be overestimated or missed. Like the antennas in figure 6 which block too much light. This is a problem that we share with all image based methods.

---

[2]Note that this does not cause problems with filtering because our kernels are always entirely contained in the occluder texture (see Sec.3.4)
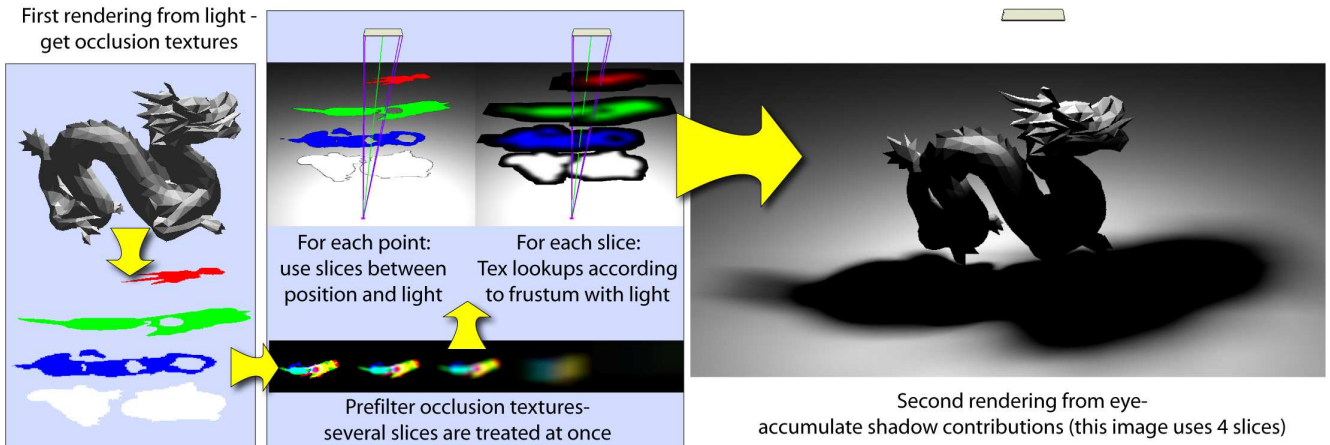
Figure 9. Summary of our algorithm.

Our work is similar in spirit to that of Keating and Max [20] but the field of application is completely different. Their approach does not aim at real time and targets ray-tracing. Without averaging several rays, it is still presented in a form that would not allow real-time performance. It uses convolution mostly to avoid noise and applies it similarly to percentage closer filtering[25]. 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 created on the GPU whereas MDIs are created on the CPU. We combine contributions differently, based on probability, and obtain convincing results without evaluating several sample rays. Of course, sampling produces more realistic images.

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 penumbrae 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.

As pointed out by Hasenfratz et al. [18], an unsolved problem is to provide best soft shadows for a given time constraint. Our algorithm gets quite close to this goal. Its run-time is mostly predictable and depends on the number of shadowed pixels in the final output. Currently we investigate using forward shadow mapping [33] to calculate a solution as seen from the light and transfer it back to the final output. The slice structure makes this possible. This fixes in advance the number of fragments as well as the number of texture lookups and speed becomes completely controllable via resolution.

An important area of future investigation concerns the slice placement. Slicing could be restricted to the part of the scene that can cast shadows that are visible by the observer. Litmaps introduced in [12], or CC Shadow volumes [22] could serve that purpose. It is possible to create per object representations, following [34]. This relates to the idea that when the viewer or some elements of the scene move, the slices should evolve in a "continuous" way.

Finally, hierarchical branching could be interesting, as one lookup gives us information about four slices. On our test hardware (GeForce 6800TD-7800), it is currently cheaper to do a lot more work than branching. This shows, that in general shader optimization becomes difficult and we did not focus on this topic, as our approach is already fast. Optimizations are certainly possible, especially on assembler level or with respect to the card's architecture.
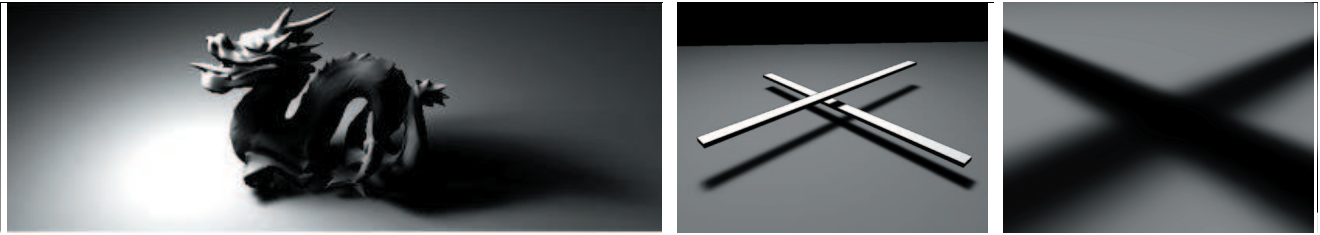
Figure 10. Examples of our method. Notice the self shadowing and the smoothness of shadows. Images are rendered at about 25Hz at a resolution of 800×600 **using 16 occlusion textures of** 512×512**.**

References

[1] M. Agrawala, R. Ramamoorthi, A. Heirich, and L. Moll. Efficient image-based methods for rendering soft shadows. In *Proc. of Siggraph'00*, 2000.

[2] J. Arvo, M. Hikorpi, and J. Tyystjärvi. Approximate soft shadows with an imag-space flood-fill-algorithm. In *Proc. of Eurographics'04*, 2004.

[3] U. Assarson and T. Akenine-Möller. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Proc. of Workshop on Rendering'02*, Springer Computer Science. Eurographics, Eurographics, 2002.

[4] U. Assarson and T. Akenine-Möller. A geometry-based soft shadow volume algorithm using graphics hardware. In *Proc. of Siggraph'03*, 2003.

[5] U. Assarson, M. Dougherty, M. Mounier, and T. Akenine-Möller. An optimized soft shadow volume algorithm with real-time performance. In *Proc. of Workshop on Graphics Hardware'03*, 2003.

[6] L. Atty, N. Holzschuch, M. Lapierre, J.-M. Hasenfratz, C. Hansen, and F. Sillion. Soft shadow maps: Efficient sampling of light source visibility. *Computer Graphics Forum*, 2006.

[7] S. Brabec and H. Seidel. Single sample soft shadows using depth maps. In *Proc. of Graphics Interface'02*, 2002.

[8] S. Brabec and H. P. Seidel. Hardware-accelerated rendering of antialiased shadows with shadow maps. In *Proceedings of CGI'01*, 2001.

[9] E. Chan and F. Durand. Rendering fake soft shadows with smoothies. In *Proc. of Symposium on Rendering'03*, 2003.

[10] F. Crow. Shadow algorithms for computer graphics.in computer graphics. In *Proc. of Siggraph'77*, 1977.

[11] F. C. Crow. Summed-area tables for texture mapping. In *Proc. of Siggraph'84*, 1984.

[12] X. Décoret. N-buffers for efficient depth map query. In *Proc. of Eurographics'05*, 2005.

[13] Z. Dong, W. Chen, H. Bao, H. Zhang, and Q. Peng. Real-time voxelization for complex polygonal models. In *Proc. of Pacific Graphics'04*, 2004.

[14] W. Donnelly and A. Lauritzen. Variance shadow maps. In *Proc. of I3D'06*, 2006.

[15] F. Durand, N. Holzschuch, C. Soler, E. Chan, and F. Sillion. A frequency analysis of light transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, 24(3), aug 2005.

[16] E. Eisemann and X. Décoret. Fast scene voxelization and applications. In *Proc. of I3D'06*, 2006.

[17] G. Guennebaud, L. Barthe, and M. Paulin. Real-time soft shadow mapping by backprojection. In *Eurographics Symposium on Rendering*.

[18] J.-M. Hasenfratz, M. Lapierre, N. Holzschuch, and F. Sillion. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4), Dec. 2003.

[19] J. Hensley, T. Scheuermann, G. Coombe, A. Lastra, and M. Singh. Fast summed-area table generation and its applications. In *Proc. of Eurographics'05*, 2005.

[20] B. Keating and N. Max. Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. In *Proc. of Workshop on Rendering'99*, 1999.

[21] F. Kirsch and J. Doellner. Real-time soft shadows using a single light sample. *Journal of WSCG*, 2003.

[22] B. Lloyd, J. Wendt, N. K. Govindaraju, and D. Manocha. Cc shadow volumes. In *Proc. of EG Symposium on Rendering'04*, Springer Computer Science. Eurographics, Eurographics Association, 2004.

[23] H. Nguyen and W. Donnelly. *Hair Animation and Rendering in the Nalu Demo*. Addison Wesley, 2005.

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

[25] W. T. Reeves, D. H. Salesin, and R. L. Cook. Rendering antialiased shadows with depth maps. In *Proc. of Siggraph'87*, 1987.

[26] C. Soler and F. Sillion. Fast calculation of soft shadow textures using convolution. In *Proc. of Siggraph '98*, 1998.

[27] J.-F. St-Amour, E. Paquette, and P. Poulin. Soft shadows from extended light sources with penumbra deep shadow maps. In *Proc. of Graphics Interface'05*, 2005.

[28] Y. Uralsky. Efficient soft-edged shadows using pixel shader branching. In *GPU Gems 2*. Addison Wesley, 2005.

[29] L. Williams. Casting curved shadows on curved surfaces. In *Proc. of Siggraph'78*, 1978.

[30] L. Williams. Pyramidal parametrics. In *Proc. of Siggraph'83*, 1983.

[31] A. Woo, P. Poulin, and A. Fournier. A survey of shadow algorithms. *IEEE Comput. Graph. Appl.*, 10(6), 1990.

[32] C. Wyman and C. Hansen. Penumbra maps: Approximate soft shadows in real-time. In *Proc. of Symposium on Rendering'03*, 2003.

[33] H. Zhang. Forward shadow mapping. In *Proc. of Workshop on Rendering'98*, 1998.

[34] K. Zhou, Y. Hu, S. Lin, B. Guo, and H.-Y. Shum. Precomputed shadow fields for dynamic scenes. In *Proc. of Siggraph'05*, 2005.