Eurographics Symposium on Rendering (2007)
Jan Kautz and Sumanta Pattanaik (Editors)

Soft Shadows by Ray Tracing Multilayer Transparent Shadow Maps

Feng Xie, Eric Tabellion and Andrew Pearce†

PDI/Dreamworks Animation Inc, USA

Abstract

We present a method for high quality soft shadows for area lights in cinematic lighting. The method is an extension of traditional shadow maps, so it has the advantage of image based shadow methods; the algorithm's complexity is independent of geometric complexity. We introduce multilayer transparent shadow maps, which can be used to produce high quality soft shadows for scenes with extremely complex geometry, fur, and volume objects. Instead of the traditional sampling and filtering of shadow maps, we compute the shadow factor by ray tracing the multilayer transparent shadow map. The result is soft shadows of quality similar to that achieved by stochastic ray tracing, but at a much lower cost.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism Shadowing, Raytracing;

1. Introduction

Shadows play an important role in lighting and rendering; as Da Vinci said, *'an object appears most in relief when it is between light and shadow'*, that place we call penumbra. When the light source has an extended area, the shadow transitions gently from darkness to softness, and gives strong visual cues to the boundaries of the object.

Physically correct soft shadows are very expensive to compute; for this reason, they are rarely used in production lighting. Instead, percentage closer filtering (PCF) based soft shadows (implemented through back projection of occluders using regular and deep shadow maps) are widely used in production despite their significant artifacts. As a result, lighters spend a lot of time manually tweaking lights to emulate the look of correct soft shadows.

In this paper, we introduce a general and efficient method for computing physically correct soft shadows using an enhanced shadow map structure, *multilayer transparent shadow map*. An MTSM stores multiple layers of depth and opacity or translucency values at each pixel, therefore, it is able to capture the position and visibility of most occluders for an area light source.

An MTSM may be sampled and filtered like a traditional depth map to create high quality shadows from small, point-like lights; more importantly, it can be ray traced to create high quality soft shadows for large area lights for all types of objects. By storing and probing all the depth layers at once, we exploit the spatial coherence among objects projecting to the same screen pixel, significantly reducing the cost (in time, memory, and disk storage) to ray trace shadow maps, thereby making physically correct shadows feasible for production usage.

2. Previous Work

There has been much work on shadow algorithms over the years. Woo's paper is still a good classic overview [WPF90] and Hasenfratz et al gave a more up to date review on recent work [HLHS03], with a focus on hardware accelerated shadows.

Traditionally, there have been two main approaches to soft shadow computation. One set of solutions is object space

† email: feng, et, apearce@pdi.com

based; several papers describe penumbra computation using wedges or blurred wedges. Most recently, Lane et al presented a soft shadow algorithm that uses a single ray and edge visibility computation to reconstruct the shadowed area [LAA*05]. The algorithm is significantly faster than distributed ray tracing in many situations, but the visibility algorithm is still bound by geometric complexity, and the advantage over classic ray tracing degrades significantly as the number of edges in the scene increases. Since most production setups have very complex geometry like dense foliage and fur, geometric or object space methods are still not practical.

Williams showed that for point light sources, a depth map generated from the light view may be used to compute the shadow factor of any point in the scene [Wil78]. Shadow maps may suffer from bias and aliasing, but given high enough resolution and proper sampling and filtering, they can deliver an antialiased shadow look independent of geometric complexity [RSC87].

Recently, there has been much work on hardware-assisted soft shadow computation. Most of that work focused on real time performance. Guennebaud's paper gave a good overview of exending PCF sampling to support area lights and its inherent limitations. Their paper focused on hardware acceleration of extended PCF and some techniques to reduce PCF based soft shadow artifacts [GBP06]. Bavoil et al explored the idea of using multilayer depth maps to improve the quality of PCF based soft shadows; they were able to achieve better contact shadows and reduce bias artifacts using multilayer shadow maps (MLSM) [BCS06]; however, the fundamental cause of the artifacts from using PCF based soft shadows remains unaddressed.

In 1998, Lischinski et al described ray tracing layered depth images (LDI) for computing secondary rays in image based rendering [LTG98]; however, because the source of their LDI is often range data captured from views other than those of the light sources, their method was prone to light leaks.

In 1999, Keating et al performed penumbra computation using quantized multilayer depth images [KM99]; they sampled the quantized MLSM using filtered deterministic ray marching (via correlation in the light samples among the shading surfaces). The combination reduced light leaks but also caused blockiness and banding in the shadow image.

Then in 2000, Agrawala et al [ARHM00] introduced ray tracing of multi-view shadow maps for creating soft shadows from area lights. Although this method delivers high quality soft shadows for regular opaque geometry, it doesn't support fur or transparent geometry; in addition, the cost of algorithm is linear in the number of views, so it is still quite expensive.

In the same year, Lokovic and Veach introduced deep shadow maps for fur and volume [LV00], but their method made no effort to support large area lights.

To overcome the quality limitations of the current soft shadow methods used in computer animation, we developed a new soft shadow technique by ray tracing multilayer transparent shadow maps. Our method produces soft shadows almost identical to stochastic ray tracing; it is significantly faster and more general than ray tracing multiview shadow maps.

3. Background information

Given a point light source l, a point p is in shadow if the ray from p to l is blocked by some geometry in the scene. The shadow cast by a point light source has a sharp silhouette: every point in the scene is either completely in shadow or out. For a point light source l, a shadow map of appropriate resolution captures the visibility of geometry from the light source well and can be used to achieve a good quality shadow through percentage closer sampling and filtering [RSC87].

For an area light source, a point p is totally in shadow if every part of the light source is blocked, *umbra*; it's not shadowed if every point of l is visible; and it is partially in shadow if some portion of l is visible, *penumbra*. In fact, the shadow factor of p can be expressed in terms of the *area* of light blocked from p.

There are two reference methods that compute the shadowed area of an extended light source correctly. The first one generates many sample shadow maps of the area light (around 1000 light samples), computes the shadow factor of point p for each shadow map, then average the results. The second one is stochastic ray tracing, as described by [CC84].

Both methods are very expensive, resulting in many efforts to emulate the look of soft shadows using a single traditional depth map. Here is the classic method of emulating soft shadows using a single depth map that was first used in the production of the animated film *Antz*; we call it PCF based soft shadows in this paper for lack of a standard recognizable name and because it is an extension of traditional PCF shadows.

1. Let z_{min} be the smallest value of the depth map. Transform point p to the screen space of light l to get p's screen space location (s_x, s_y), then compute p's light space depth value $p.z$.
2. Use $p.z$ and z_{min} of the depth map to estimate the conservative filter size around (s_x, s_y).

$$filter\_size = resolution * light\_radius * (\frac{1}{z_{min}} - \frac{1}{p.z}) \quad (1)$$

3. Take stochastic samples within the filter region; a sample is a blocker if the z value is less than $p.z$; shadow fac-

tor is computed as the percentage of blockers among the samples .

The filter size computed from equation 1 can be excessively large when z_{min} is small. Most variations of the method involve multiple passes where a better estimate of the smallest blocker z is computed in the first pass to reduce the filter size for each shading point in the second pass. See [GBP06] for a detailed description and analysis of this algorithm.

The main drawbacks of this method are: loss of contact shadows for large area lights; darkened shadows due to wrong occlusion fusion; and light leaks due to storage of only the occluder closest to the center of the light source. These artifacts become more apparent with larger area lights, as shown by the column of images in figure 1e.

PCF based sampling of shadow maps may be easily extended to support multilayer shadow maps; figure 1d shows the effect of shadow computation by PCF sampling of MLSM; for small area lights, most of the light leaks are removed but the exaggeration of the umbra region remains due to wrong occluder fusion. However, the light leaks at the contact shadows become quite noticable as we increase the light radius.

4. Ray tracing multilayer shadow map

Stochastic ray tracing is a simple and general algorithm for computing high quality soft shadows as shown in figure 1a; the main drawback is the cost of computing the intersection of millions of rays with a complex scene. For example, in the image of Vanessa, figure 5, there are 3.05 million shading surfaces, and 256 samples for each light, which translates to over 760 million ray intersection tests against a scene composed of 105 million triangles. Even in a highly optimized ray tracer, this is prohibitively expensive for production lighting.

4.1. Ray tracing depth maps

There has been a lot of work in accelerating ray tracing against complex scenes using hierarchical traversal [RSH05]. In this paper, we focus on using a simplified representation of the scene to accelerate the computation of ray traced shadows.

Figure 1c shows a sequence of images of a simple test scene where the shadow computation is done by ray tracing a single depth map from the center of the light. It doesn't suffer from darkened shadowed artifacts in PCF based methods, but the light leaks are very noticeable, and they occur because the single depth map stores only the polygon closest to the light center through each pixel. Occluders for other parts of the light may be missing. The light leaks also increase with the size of the area light.

Agrawala et al solved the light leak problem by adding multiple reference views to represent the scene. Those reference views are usually taken from the corners of the area light. The single depth map is then replaced with a list of depth maps, and each sample ray is tested against each view until an intersection is found.

The main drawbacks of their algorithm are: heuristic based view placement; and the cost to setup and trace each view separately. To improve performance, they maintain a list of clipped rays and track blocker coherence . Aside from the data structure complexity, blocker coherence based acceleration degrades significantly for scenes with high edge complexity like fur and foliage.

To address these issues, we introduce ray tracing multilayer shadow maps.

4.2. Construction of multilayer shadow maps

Multilayer depth maps store more than one depth sample per pixel. To construct a multilayer shadow map, we store the first k layers of depth values in each pixel. The algorithm requires only a simple extension to the z buffer of a traditional scanline renderer. Instead of a single z value for each pixel, we maintain of list of k samples. As polygons are rasterized, if a z sample is closer than any of the existing k samples in a pixel, it is inserted into the list.

The extra cost incurred by constructing a k layer z buffer versus a single layer z buffer happens only when the incoming z sample needs to be inserted in the list of layers in the pixel. For a software scan line renderer, the cost of z buffer rasterization is dominated by tessellation and scan conversion; the actual cost of updating the pixel z is small in comparison. In the test scenes we used, construction cost of a 5-layer shadow map is only on average 5 percent more than that for a single-layer shadow map. In contrast, the cost of multi-view shadow maps is linear with respect to the number of views, so a 5-view shadow map will incur 5 times the cost of constructing a single shadow map.

4.3. Ray tracing multilayer shadow map

In [LTG98], Lischinski et al described ray tracing layered depth images for image based rendering; their algorithm may be easily modified to work with a multilayer depth map.

The intersection test of a shadow ray with a multilayer shadow map is done as follows:

First we transform the ray to the camera space of the shadow map; then we clip this camera space ray using the minimum z value of the shadow map, since we assume our light source is a flat area at the origin of the light space. (It is simple to extend this to support light sources that are not flat by using a conservative estimate of max z offsets of the area light in light space.) After this, we project the clipped camera space ray to the screen space of the shadow map, and

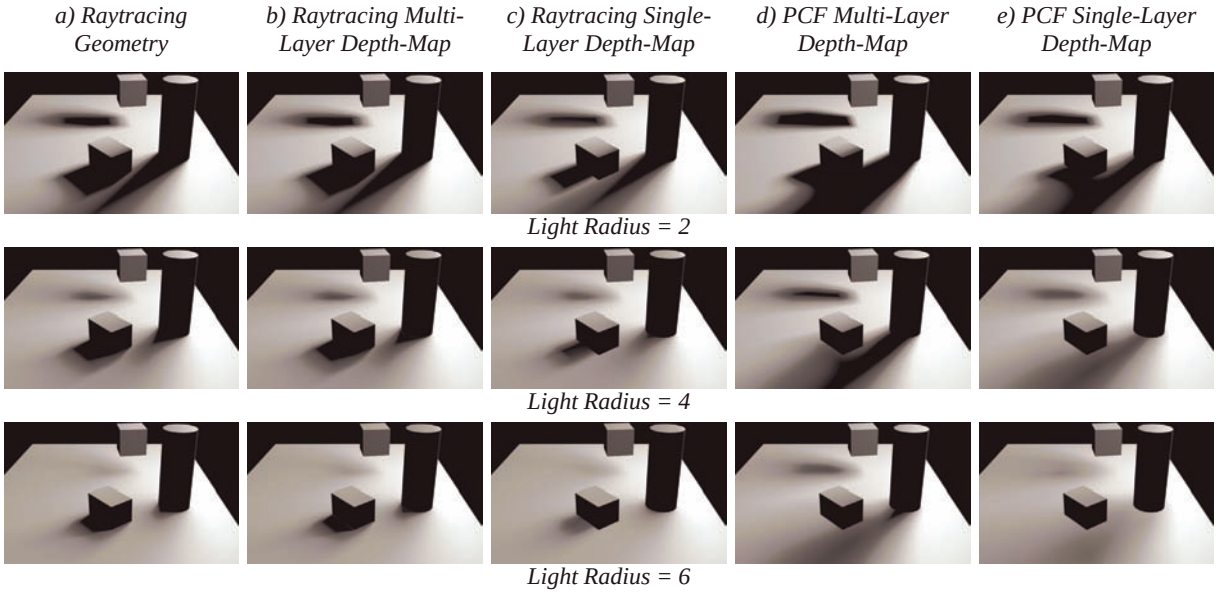


Figure 1: *Soft shadows for a simple test scene: PCF based sampling of MLSM still shows occluder fusion artifacts (d); while ray tracing single layer depth map suffers from light leaks (c). Ray tracing an MLSM (b) generates soft shadows almost identical to stochastic ray tracing (a).*

clip it using the viewports of the shadow map; now we have a screen space ray.

Given a screen space ray that starts at (s_x, s_y, s_z) and ends at (e_x, e_y, e_z); we perform the regular scan line conversion. As we walk through each pixel in the shadow map along the line, we compute z_{enter} and z_{exit} values of the ray across the pixel. Given $[z_{enter}, z_{exit}]$, we scan the depth samples in the pixel (as opposed to the single z value in a traditional depth map); if a depth sample $[z_i - z_{thresh}, z_i + z_{thresh}]$ overlaps $[z_{enter}, z_{exit}]$, then an intersection is found; else the walk terminates at the end of screen space ray. The *z threshold* bias depends on the depth map resolution, and the angle between the surface normal and light direction. See the quality section for more discussion on this parameter.

At the pixel level, the cost of finding a hit in a multilayer shadow map is $o(log(k))$ of the single layer shadow map, where k is the number of layers in the shadow map. The complexity of ray tracing a shadow map is the same as the complexity of line scan conversion, which is linear to the line's screen length. For each shading micropolygon p, the maximum screen length of all the sample rays for a light of radius r is:

$$screen\_length = r * resolution * (\frac{1}{z_{min}} - \frac{1}{p.z}) \qquad (2)$$

For shadow maps of size of at least 1kx1k, this value can be on the order of hundreds. Software scan conversion of a long line is expensive, but may be accelerated using hierarchical traversal. Given a multilayer depth map, we build a quadtree, where each node contains *min_z* and *max_z* of the 4 child nodes in the layer below. The hierarchical intersection test between a screen space ray and a quadtree is done as follows:

```
int trace(node, ray) {
  update ray.z_enter, ray.z_exit;
  if (no overlap with ray) return 0;
  if (leaf(node))
        return intersect(node, ray)
  else foreach nonempty child of node {
       if (trace(child, ray))
            return 1;
       return 0;
  }
}
```

Using a hierarchical z buffer, we reduce the cost of ray tracing shadow maps from linear to logarithmic in shadow map resolution and light radius.

5. Multilayer transparent shadow map

Ray tracing multilayer depth maps works well for complex scenes with trees, foliage and surfaces; however, a depth map that stores only z values does not lend itself to objects like fur and volume where light is both absorbed and filtered within the object. If for each pixel in the depth map, we shoot a ray from the center of the light to the center of the pixel (we call this light center ray to facilitate discussions below), and

store the depth (position) and visibility change (opacity) of all the surfaces or volume samples the ray intersects along the way; then we can use these samples to compute the visibility change of any light ray traveling through them.

When the scene is composed of millions of hairs or transparent particles, storing all the samples each light center ray intersects is infeasible. Lokovic and Veach introduced deep shadow maps, a compact representation of the visibility as a function of depth along each light ray [LV00]; we build an MTSM by extending the deep shadow construction algorithm.

5.1. Construction

For each light center ray, deep shadow stores a compressed form of the visibility function. Each deep pixel stores a list of control vertices composed of depth and visibility, (z_i, v_i). The visibility of any depth along the light center ray may be computed using linear interpolation of the control vertices; however, to compute the visibility of another light ray traversing through the samples, we need a way to estimate the visibility change and location of the original samples.

We know the accumulated visibility change and depth range of all the samples between two control vertices, so to reconstruct the average depth location and average visibility change of the original samples, all we need is their total number. We can then compute the average visibility change of each sample and the depth interval between them.

The structure of an MTSM pixel is as follows: each pixel has a list of layers, and each layer contains $(z_i, v_i, count_i)$, depth, accumulated visibility, and the count of the samples in between layer $i-1$ and layer i.

In addition, an MTSM pixel may need to store the depth and opacity of a subset of samples that are encountered after the visibility of the ray originated from the light center through the pixel reaches zero, since those samples may contribute to the visibility reduction of a ray starting from a different point on the light source. In the extreme case, an MTSM is a multilayer depth map, where each sample has full opacity; the need to store extra layers is clear here because they might be blockers for other points on the light sources. On the other hand, if the MTSM represents some homogeneous material like a hair ball or cloud, there is little need to store layers after the visibility is reduced to 0; the ability to store layers after reaching full opacity enables an MTSM to represent a more general set of shadow casters.

With two modifications, we can extend the deep shadow construction algorithm to build an MTSM. First, we track and store the count of original samples beween the control vertices. Second, once the accumulated visibility of the compressed samples reaches zero, we reset the accumulated visibility to 1 and compress the subsequent samples the same way. We reset the accumulated visibility up to a user specified number of times.

Since MTSM is an enhancement of deep shadow, it may be sampled and filtered in the same way to compute high quality shadows for fur and volume objects for small, point-like light sources. We focus next on how to use MTSM to compute soft shadows for area lights.

5.2. Ray tracing an MTSM

To compute soft shadows for semi-transparent objects due to a large area light source, we modify the multilayer depth map ray tracing algorithm to support a multilayer transparent shadow map.

```
ray.vis = 1;
void  trace(node, ray) {
  update ray.z_enter, ray.z_exit;
  if (no overlap) return;
  if (leaf(node)) {
    intersect(node, ray);
  } else foreach non empty child of node {
    trace(child, ray);
    if (ray.vis < threshold) return;
  }
}
```

The main difference with respect to ray tracing an MLSM is that instead of a binary ray depth sample intersection test, we compute the accumulated visibility along each ray. At the pixel level, the ray intersection test is modified to support compressed layers as follows. First, perform an overlap test of the ray's $[z_{enter}, z_{exit}]$ with depth range of all the layers in the pixel, then find the two adjacent layers the ray is traveling through inside the pixel. If ray's $[z_{enter}, z_{exit}]$ overlaps either layer's z value, we report an intersection. Otherwise, divide the z range $(z_{i-1}, z_i]$ into $count_i$ buckets, and report a hit if the ray intersects with any bucket boundary, since each boundary represents the depth location of a sample between the two layers.

If the ray intersects layer i or any estimated sample between layer $i-1$ and layer i, reduce *ray.vis* by dv_i where

$$dv_i = (v_i - v_{i-1})/count_i; \tag{3}$$

Unlike ray tracing of multilayer opaque shadow maps, the ray scan doesn't stop after finding the first intersection; instead, each intersection reduces ray visibility, and the ray scan stops when ray visibility falls below a threshold. The shadow factor of each ray is $1 - ray.vis$. The shadow factor of the light is the average shadow factor of all the sample rays.

6. Quality

In most cases, shadows generated by ray tracing multilayer shadow maps are almost identical to shadows generated using stochastic ray tracing, see figure 2; however, this method is subject to some inherent errors common to image based rendering. The next sections explore these errors and the techniques which can be used to minimize them.

Figure 2: *Shrek soft shadows: notice that the visual artifacts of soft shadows using PCF sampling and ray tracing of single layer shadow map grow with light size, while ray tracing MLSM generates shadows identical to stochastic ray tracing*

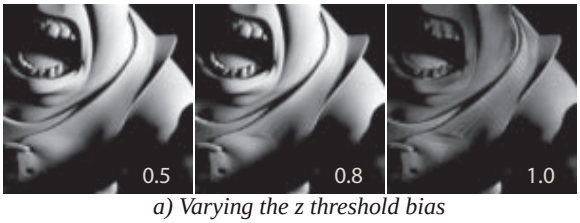
6.1. Sampling Error

As the screen space ray crosses a pixel, we report a positive hit if there exist a layer i where $[z_i - z_{thresh}, z_i + z_{thresh}]$ overlaps the $[z_{enter}, z_{exit}]$ of the screen space ray.

Light leaks can happen using this method; Keating et al used quantization to reduce light leaks [KM99], but this caused significant change to the shadow casting geometry, and noticeable error in the shadow generated. In contrast, Agrawala et al used the *'floor and wall'* strategy to connect adjacent pixels whose z values are within a constant *'gap bias'* [ARHM00].

We have experimented with both the *'floor and wall'* strategy and the constant *'z threshold'* bias in [LTG98], and found that the two strategies yield very similar results in most test scenes. In the end, a constant *'z threshold'* bias is easier for artists to control. Intuitively, bigger *'z threshold'* bias means the larger umbra region; while smaller *'z threshold'* bias means lighter shadows, and some light leaks caused by shadow casters at an oblique angle with respect to the light direction. See figure 3a and figure 3b for the effect of *'z threshold'* bias.

More importantly, we found that it's essential to specify *'z threshold'* bias in camera space because screen space z is nonlinear. Since the ray walk is in screen space, we have to perform a perspective divide to compute the light space z_{enter} and z_{exit} as the ray crosses a pixel. Fortunately, the perspective divide isn't done for every screen pixel along the ray because the hierarchical z buffer eliminates most false regions. Since the hierarchical z buffer is constructed using the screen space z values of the shadow map, the perspective divide doesn't happen until the ray traversal reaches a leaf node of the quadtree.



a) Varying the z threshold bias

b) Light leaks when z threshold bias=0.05

c) Light leak ray-tracing 2 layers

Figure 3: *Possible artifacts: z threshold bias and miss occluder*

6.2. Missing occluder from a shadow map

As we rely on the shadow map to represent the shadow casting geometry in the scene, light leaks may occur when a shadow casting object is missing from the shadow map. This can happen for 2 reasons:

1. The object is clipped by the view frustum of the light. If some part of the shadow casting scene actually intersects or lies outside the viewing frustum of the light center, that part would be missing from the shadow map and therefore unable to contribute to shadow computation from the other points on the light. The view frustum clipping problem is simple to fix if we know a priori the largest extent of the area light. If we push the light center back by this amount,

$$d = light\_radius / tan(.5 * angle(field\_of\_view)) \quad (4)$$

   then the view frustrum of the new light center will contain all the objects in view from any point on the light source using the same field of view.
2. The object is occluded by too many other objects. Given a light with a large radius and a scene with high depth complexity, a multilayer shadow map of finite layer count of 5 may not be able to capture all the objects that could be casting shadows onto other objects visible from camera view. There is no easy solution to this problem. If obvious light leaks are happening due to a missing occluder, users can usually remedy the situation by increasing the number of layers in the shadow map. Generally, due to the location coherence of the points on a light source, the depth complexity of the scene visible from different points on the light source is low. (Recall that an object has to be visible from at least one point on the light source to be able to cast shadows.) We've found in practice that 5 layers suffice for most production setups. See figure 3c for light leaks due to missing occluders.

7. Results

All the tests were done on a 2.2 Ghz AMD opteron with 4 gigabytes of memory. We have implemented our algorithm as part of the lighting computation in our deferred shading system where only visible micropolygons are shaded. The test scenes we used had a variety of challenges and demonstrate the scalability and robustness of our soft shadow algorithm. All the images and performance number reported used 256 light samples per shading micropolygon.

To simplify table labels and discussion, we use RT for ray tracing; MTSM for multilayer transparent shadow map; MLSM for multilayer shadow map; MVSM for multiview shadow map; SLSM for single layer shadow map; and PCF for PCF based shadows using regular or deep shadow maps (used for fur ball and Puss test).

For the first three test cases (Shrek, palm tree and fur ball), images were rendered at 600x300 resolution. Each test had

one spot light, and the resolution and maximum layer count of the multilayer shadow maps were set at 1kx1k and 5 respectively.

Quality wise, ray traced MTSM shadows and stochastic ray traced shadows are almost identical. Besides the rendered images, we present in table 1, the pixel difference between the images generated using PCF based shadows and ray traced MTSM shadows with respect to stochastic ray traced shadows. In all 3 tests, the error for ray traced MTSM shadows is about 5% to 10% of the error of PCF based shadows. In addition, Figure 4 shows the fast growth rate of the error of PCF based soft shadows with respect to light radius; in contrast, the error curve of ray traced MTSM shadows is almost flat with respect to light radius.

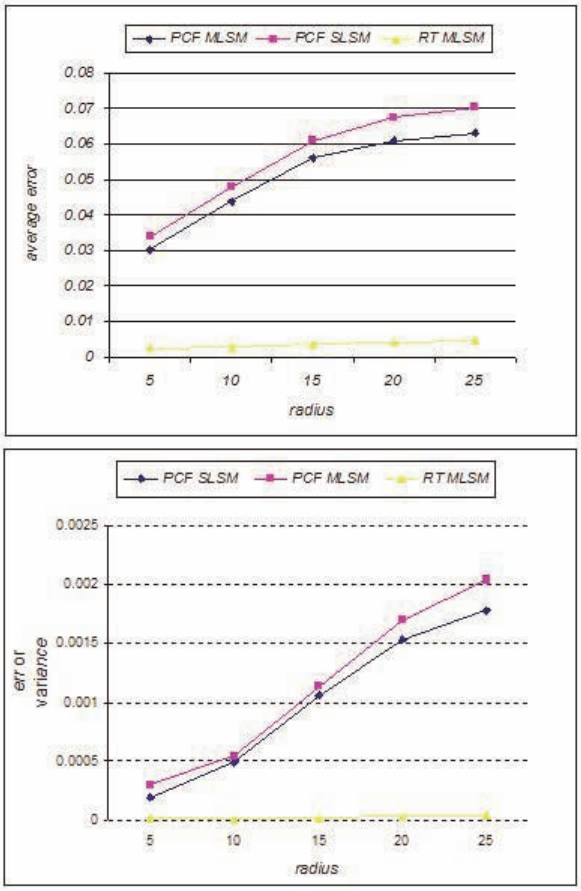


Figure 4: *Error growth of PCF based shadows and ray traced MLSM shadows for Shrek*

Table 2 shows the performance comparison for the different soft shadow methods (the numbers only include shadow computation time during shading) and Table 3 shows the geometric and shading complexity of each test as well as the resolution and construction cost of the shadow maps used.

For regular geometry (Shrek and palm tree), ray tracing multiview shadow maps produces high quality soft shadows. However it is about 5 times slower than ray tracing multilayer shadow maps; because we had to use 5 views (a center view plus 4 corner views) to remove all the light leaks for the large area lights we tested. For scenes with fur, ray tracing multiview shadow maps suffers from significant aliasing due to the high frequency of edges in regular shadow maps; see figure 13.

The fur ball test demonstrates the performance and quality of ray traced MTSM shadows. The shadows look very similar to ray traced shadows, for both the self shadows on the fur and the soft penumbra on the floor. In contrast, PCF based deep shadow method generates soft self shadows on the fur, but produces an incorrect penumbra on the floor.

Compared to stochastic ray traced shadows using scene geometry, the performance benefit of ray traced MLSM shadows increases with geometric complexity. The cost benefit factor ranges from 5 to 20. The cost of ray tracing MLSM between the 3 tests are fairly consistent, which validates that the cost of ray tracing shadow maps is bound by image complexity. In contrast, the cost to ray trace the fur ball is about 10 times the cost to ray trace Shrek due to increased geometric complexity.

Both Vanessa and Puss tests are rendered at 2kx1k resolution, using 2kx2k MTSM with maximum 5 layers. Puss is lit with one key light, while Vanessa is lit with 4 lights (the common set up in production character lighting); since the shadow cost of the 4 lights are consistent, we only report the average. We only compare the quality and performance of our method with PCF based soft shadows for these two tests because the complexity of the shadowing geometry in these tests exceed the memory limitation of our ray tracer. However, based on the shading surface count and shadowing triangle count of the tests, and the cost to ray trace the fur ball, we can conservatively estimate that the cost for ray traced shadows would be at least 20 hours for Vanessa vs. 9 minutes using ray traced MTSM.

Even though we doubled the shadow map resolution in the Vanessa and Puss tests, the per shadow query cost is similar to the Shrek test because both MTSMs have high blocker density and because the algorithm complexity is logarithmic in shadow map resolution.

		Shrek	Palm	Fur
Mean error	PCF	.063	.017	.016
	RT MTSM	.0044	.003	.005
STD error	PCF	.042	.014	.05
	RT MTSM	.005	.003	.002

Table 1: *Average and standard deviation of pixel difference w.r.t to ray traced shadows*

	RT MSM	RT Geometry	RT MVSM	PCF or Deep*
Shrek	57	207	273	18
Palm tree	61	446	311	20
Fur ball	98	2662	NA	33*
Puss	240	NA	1480	83 *
Vanessa	556	NA	NA	156

Table 2: *Per frame shadow computation time in seconds, excluding shadow map construction time*

	shading sample #	shadow triangle #	MTSM res	MTSM cost
Shrek	215k	316k	1kx1k	2
Palm tree	263k	310k	1kx1k	5
Fur ball	373k	4562k	1kx1k	15
Puss	1754k	13M	2kx2k	20
Vanessa	3050k	109M	2kx2k	30

Table 3: *Geometric and shading complexity, shadow map resolution and construction time*

8. Summary

In this paper, we presented stochastic ray tracing of multilayer shadow maps, an efficient algorithm for generating high quality soft shadows.

The algorithm is simple to implement and places no constraint on the light shape or size. The complexity of the algorithm is logarithmic in shadow map resolution and independent of geometric complexity. It is easily parallellizable; once the shadow map is loaded into shared memory, all the shadow ray tests may be executed in parallel.

Compared to ray tracing multiview depth maps, our method is 4-5 times faster. More importantly, our method can generate high quality soft shadows for semi-transparent geometry and fur, which is a limitation for multiview depth maps. In addition, multilayer shadow map is more scalable in dealing with missing geometry, a fundamental problem in using depth maps to represent the scene, because adding a new layer has little impact on the cost of ray tracing a multilayer shadow map.

Compared to PCF based soft shadows and deep shadows, our algorithm is about 3 times slower, but the quality of our soft shadows is more than 10 times better visually and quantitatively. The extra computation cost is justified when more accurate shadows lead to significant reduction in the time lighters spend tweaking shadows.

9. Conclusion and future work

Ray tracing multilayer transparent or opaque shadow map allows us to achieve the look of physically correct soft shadows at an order of magnitude faster than stochastic ray tracing. Until stochastic ray tracing becomes feasible, we believe ray tracing multilayer shadow map is an efficient and scalable solution for high quality soft shadows in cinematic lighting.

In the future, we would like to investigate into GPU accelerated ray tracing of MTSM, as well as acceleration techniques based on more effective sampling and bundling of shadow rays.

References

[AMA02] AKENINE-MOLLER T., ASSARSSON U.: Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Eurographics Rendering Workshop 2002 Proceedings* (2002), pp. 297–306.

[ARHM00] AGRAWALA M., RAMAMOORTHI R., HEIRICH A., MOLL L.: Efficient image-based methods for rendering soft shadows. In *Proceedings of SIGGRAPH 2000* (2000), Computer Graphics Proceedings, Annual Conference Series, ACM, pp. 375–384. 2, 7

[BCS06] BAVOIL L., CALLAHAN S. P., SILVA C. T.: Robust soft shadow mapping with depth peeling. SCI Institute Technical Report UUSCI-2006-028, August 2006. 2

[CC84] COOK R. L., CARPENTER L.: Distributed ray tracing. In *Proceedings of SIGGRAPH 1984* (1984), Computer Graphics Proceedings, Annual Conference Series, ACM, pp. 137–145. 2

[Cro77] CROW F.: Shadow algorithms for computer graphics. In *Proceedings of SIGGRAPH 1977* (1977), Computer Graphics Proceedings, Annual Conference Series, ACM, pp. 242–248.

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

[ED06] EISEMANN E., DÉCORET X.: Plausible image based soft shadows using occlusion textures. In *Proceedings of the Brazilian Symposium on Computer Graphics and Image Processing, 19 (SIBGRAPI)* (2006), Oliveira Neto Manuel Menezes deCarceroni R. L., (Ed.), Conference Series, IEEE, IEEE Computer Society.

[GBP06] GUENNEBAUD G., BARTHE L., PAULIN M.: Real-time soft shadow mapping by backprojection. In *Eurographics Symposium on Rendering* (2006), Eurographics, pp. 227–234. 2, 3

[GK93] GREENE N., KASS M.: Hierarchical z-buffer visibility. In *Proceedings of SIGGRAPH 93* (1993), Kajiya J., (Ed.), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 231–240.

[HLHS03] HASENFRATZ J.-M., LAPIERRE M., HOLZSCHUCH N., SILLION F. X.: A survey of real-time soft shadow algorithms. In *Proceedings of EUROGRAPHICS 2003* (2003). 1

[KM99] KEATING B., MAX N.: Shadow penumbra for complex objects by depth dependent filtering of multi-layer depth images. In *Eurographics Rendering Workshop 1999 Proceedings* (1999). 2, 7

[LAA*05] LANE S., AILA T., ASSARSSON U., LEHTINEN J., AKENINE-MOLLER T.: Soft shadow volumes for ray tracing. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (2005), Computer Graphics Proceedings, Annual Conference Series, ACM, pp. 1156–1165. 2

[LLA06] LEHTINEN J., LAINE S., AILA T.: An improved physically-based soft shadow volume algorithm. In *Proceedings of Eurographics 2006* (2006), Eurographics Association and Blackwell Publishing Ltd, pp. 303–312.

[LTG98] LISCHINSKI D., TAMPIERI F., GREENBERG D. P.: Image based rendering for non-diffuse synthetic scenes. In *Eurographics Rendering Workshop 1998 Proceedings* (1998). 2, 3, 7

[LV00] LOKOVIC T., VEACH E.: Deep shadow maps. In *Proceedings of SIGGRAPH 2000* (2000), Akeley K., (Ed.), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 385–392. 2, 5

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *Proceedings of SIGGRAPH 87* (1987), Stone M. C., (Ed.), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 283–291. 2

[RSH05] RESHETOV A., SOUPIKOV A., HURLEY J.: Multi-level ray tracing algorithm. In *Proceedings of SIGGRAPH 2005* (2005), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 1176–1185. 3

[SAPP05] ST-AMOUR J.-F., PAQUETTE E., POULIN P.: Soft shadows from extended light sources with penumbra deep shadow maps. In *Graphics Interface 2005* (May 2005), pp. 105–112.

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *Proceedings of SIGGRAPH 78* (1978), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 270–274. 2

[WPF90] WOO A., POULIN P., FOURNIER A.: A survey of shadow algorithms. *IEEE Computer Graphics and Applications 10*, 6 (11 1990), 13–32. 1

Figure 5: *Vanessa using ray traced MTSM: soft shadows on the neck and body, while retaining all the contact details of eye lashes and necklace*

Figure 6: *Vanessa using PCF based deep shadows: all the contact shadows are extremely blurred and some artifacts due to darkened penumbra on her face.*

Figure 7: *Palm tree using ray traced MTSM*

Figure 8: *Palm tree using ray traced shadows*

Figure 9: *Palm tree using PCF based shadows: notice the shift of the penumbra of the ground*

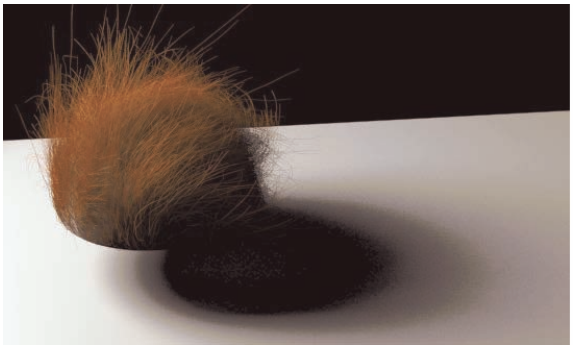


Figure 10: *Fur ball using ray traced MTSM*

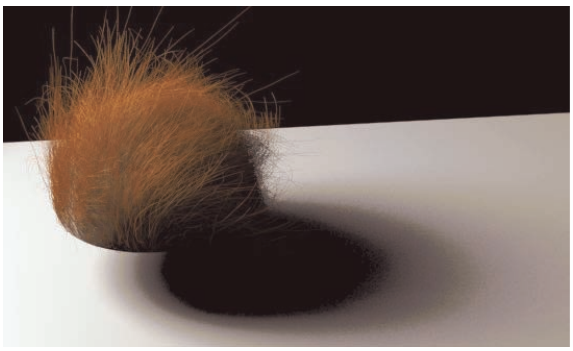


Figure 11: *Fur ball using ray traced shadows*

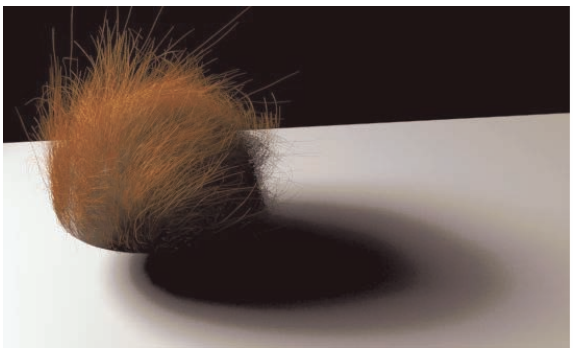


Figure 12: *Fur ball using deep shadows: widening and shifting of penumbra of the ground*

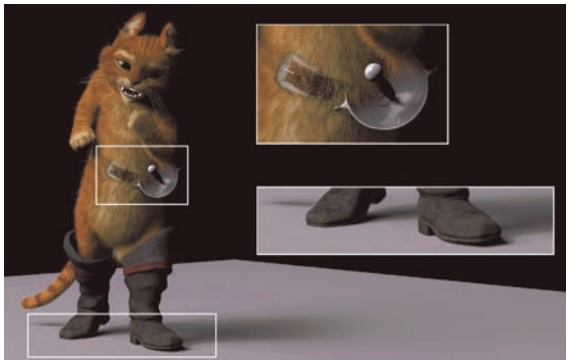


Figure 13: *Puss using ray traced MVSM: noisy artifacts on the fur*

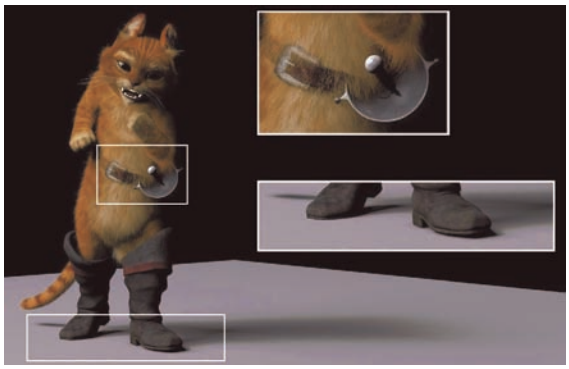


Figure 14: *Puss using ray traced MTSM: details on contact shadows and soft fur shadows*

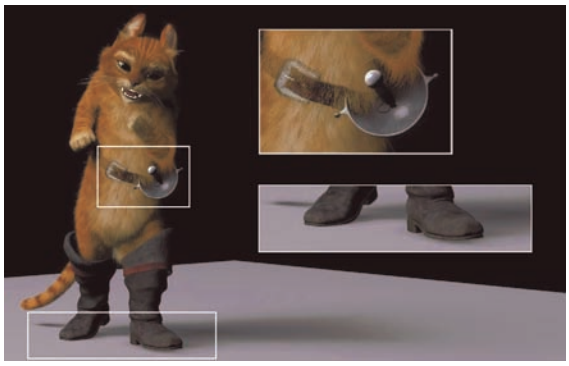


Figure 15: *Puss using by deep shadows: soft fur shadows but blurry or missing contact shadows for the boots, belt and sword handle*