Eurographics Symposium on Rendering - Experimental Ideas & Implementations (2016)
E. Eisemann and E. Fiume (Editors)

4D-rasterization for Fast Soft Shadow Rendering

Paper1014

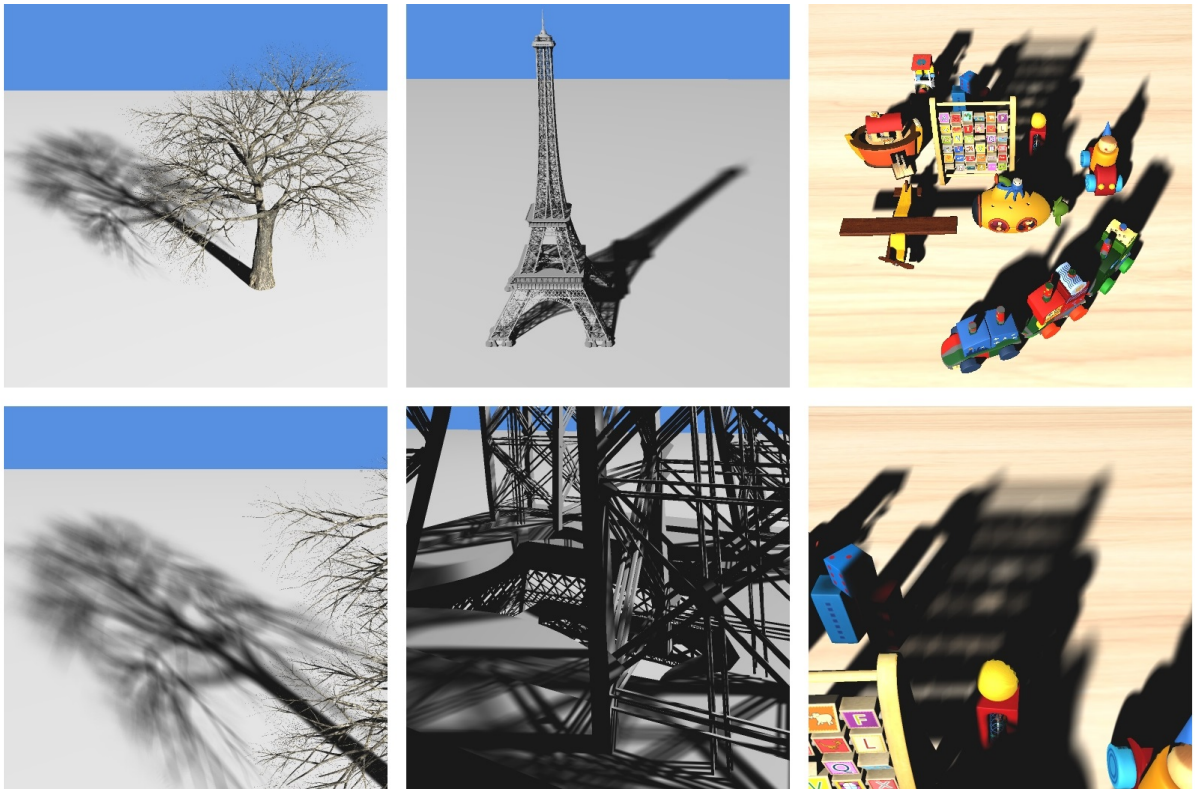


Figure 1: Soft shadows rendered with our method. Our images are identical to images rendered with ray tracing for the same area light source uniform sampling resolution. The average frame rate for our method vs. ray tracing is 5.2fps vs. 0.34fps for *Tree*, 10.0fps vs. 0.29fps for *Tower*, and 15.0fps vs. 0.60fps for *Toys*, which corresponds to 15×, 35×, and 25× speedups.

Abstract

This paper describes an algorithm for rendering soft shadows efficiently by generalizing conventional triangle projection and rasterization from 2D to 4D. The rectangular area light source is modeled with a point light source that translates with two degrees of freedom. This generalizes the projection of triangles and of output image samples, as seen from the light, to the locus of projections as the light translates. The generalized projections are rasterized to determine a conservative set of sample/triangle pairs, which are then examined to derive light occlusion masks for each sample. The algorithm is exact in the sense that each element of the occlusion mask of a sample is computed accurately by considering all potentially blocking triangles. The algorithm does not require any type of precomputation so it supports fully dynamic scenes. We have tested our algorithm on several scenes to render complex soft shadows accurately at interactive rates.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

Many light sources in scenes of interest to computer graphics applications have a non-zero area. Such lights cast soft shadows, and computing these shadows accurately greatly enhances the quality of the rendered images. However, rendering soft shadows is expensive as one has to estimate the fractional visibility of the area light source from each of millions of output image samples. A common approach is to discretize the light source with hundreds or even thousands of point light samples, which results in having to compute point-to-point visibility for billions of light rays every frame.

In this paper we present an algorithm for rendering complex soft shadows at interactive rates. We model the light rectangle as a point light source with two translational degrees of freedom. We estimate visibility to the light rectangle from output image samples by projecting and rasterizing the scene triangles and the output image samples as seen from the light. The two translational degrees of freedom of the point light source generalize the projection to the union of conventional projections as the point light source translates on the light rectangle. The generalized projections of triangles and samples are rasterized to define a conservative and tight set of output image sample/triangle pairs, which are examined to derive light occlusion masks for each output image sample. A light occlusion mask is a 2D bitmap with one bit per light sample, which is 1 if the light sample is occluded, and 0 otherwise. Whereas computing hard shadows cast by a fixed point light source can be done with conventional 2D rasterization, our 4D rasterization algorithm has to account for the two degrees of freedom added by the two translations of the point light source on the light rectangle.

Our algorithm is exact in the sense that it estimates visibility correctly between each output image sample and each light sample. Our algorithm does not require any precomputation and therefore it supports fully dynamic scenes, with moving light sources that can change size, and with moving geometry that can deform. We have tested our algorithm on multiple scenes with complex soft shadows. The soft shadows in Figure 1 are rendered by sampling the light rectangle uniformly at 32×32 resolution. Our algorithm produces images that are identical to those rendered by ray tracing for the same uniform light sampling resolution, but at 15 to 35 times higher frame rates. We compare our algorithm to NVIDIA's Optix ray tracer with bounding volume hierarchy (BVH) acceleration. The *Toys* scene is dynamic so ray tracing has to recompute the BVH for every frame. The *Tree* and *Tower* scenes are static, so the ray tracing performance reported in Figure 1 does not include the time to compute the BVH.

2. Related work

There has been extensive research on soft shadow rendering. In addition to this brief overview, we also refer the reader to in depth surveys of prior techniques [ESA11, HLHS03].

Shadow *simulation* methods start from hard shadows that are fitted with penumbra regions [Mol02, BS02, AHT04]. For example soft shadows can be simulated by blurring hard shadow edges [Fer05, MKHS10].

Shadow *approximation* methods simplify the blocking geometry to accelerate visibility querying. For example, back projection methods approximate blocking geometry with shadow mapping [GBP06]. Shadow mapping methods were improved by smooth contour detection and radial area integration [GBP07], with micro-quads and micro-triangles [SS08], with multiple shadow maps that reduce the artifacts of sampling blocking geometry from a single viewpoint [YFGL09], with exponential shadow filtering [SFY13, SDMS14], or with stochastic sampling [LSMD15].

Accurate soft shadow methods actually compute visibility between output image samples and light samples correctly. Our method belongs to this category. Ray tracing [Whi79] provides natural support for soft shadows by tracing rays between output image and light samples, but performance is a concern. One method casts a sparse and variable set of light rays per pixel to obtain coarse visibility estimates that are then filtered adaptively [MWR12, YMRD15]. The method gains performance by reducing the number of rays but this comes at the cost of reduced quality. Our method ammortizes the cost of estimating visibility for individual rays by leveraging projection followed by rasterization, which also implies considering a fixed number of light samples for each output image sample and each potentially blocking triangle. Stochastic approaches can vary the number of rays as needed, at the cost of having to compute the adequate sampling rate, and of lower quality in some difficult cases.

Hierarchical occlusion volumes are used to estimate the soft shadows cast by triangles [LA05], with the shortcoming that the approach scales poorly with the light size. Soft shadow volumes work with silhouette edges as opposed to all triangle edges [LAA*05], which gains efficiency by ignoring land-locked triangles that do not affect soft shadows. Like our method, the soft shadow volumes methods maps potentially blocking triangles to output image samples. Performance is affected by the complexity of the method, and by fragmented meshes, where virtually all triangle edges are silhouette edges. Forest et al. [FBP08] accelerate the approach but the poor handling of fragmented meshes remains.

Lazy visibility evaluation [MAAG12] computes accurate soft shadows by grouping light rays in *Plucker* space, with the caveat of poor scalability with scene complexity as a BSP tree has to be built for each blocking triangle. A method for estimating visibility between two rectangular patches can be applied to soft shadows, but the method relies on assumptions about receiver geometry [ED07]. Another method estimates penumbra regions using a point light source and blocker silhouettes, with the challenge of stable and efficient silhouette detection [JHH*09].

Several methods compute, in a first step, a set of potentially blocking triangles for each output image sample by projecting triangle shadow volumes and output image samples onto a grid, and then, in a second step, estimate light visibility from the output image sample viewpoint either by ray tracing [BW09, NIDN13] or by rasterizing the set [AL04, SEA08, WZKV14, LMSG14]. The 4D rasterization used by our method can be seen as an efficient projection of the triangle shadow volume. In the second step, our method gains efficiency by computing the rows of the occlusion masks directly.

We take the approach of higher dimensional rasterization to estimate visibility from a light rectangle. The benefits of higher dimensional rasterization have also been noted in the context of motion

blur, where rapidly moving triangles are rasterized in 3D by adding the time dimension to conventional 2D rasterization [AMMH07].

3. 4D Rasterization

First we present the 4D rasterization algorithm, and then we describe optimizations for performance.

3.1. General 4D rasterization algorithm

Consider the problem of computing hard shadows cast by a single point light source L. The problem can be solved by projecting from L the scene triangles and the output image samples onto an image plane discretized with a uniform grid, which we call a framebuffer. If an output image sample s projects at a framebuffer pixel p that is also touched by the projection of a triangle T, one checks whether T blocks L from s. Whereas a naive algorithm would consider all sample/triangle pairs, the framebuffer is used to identify a much smaller but conservative set of such pairs. Note that the framebuffer is used here to pair output image samples with potentially blocking triangles, and it is not used as a shadow map that samples the scene geometry in order to provide approximate visibility queries. The visibility relationship between an output image sample and a triangle is computed accurately in a second phase.

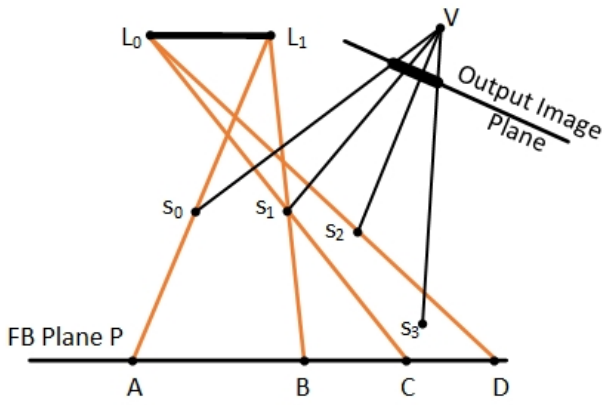


Figure 2: Construction of framebuffer plane P, parallel to light rectangle L_0L_1. The generalized projection of output image sample s_1 onto P is BC. The framebuffer extends on P from A to D to encompass the generalized projections of all output image samples.

We generalize this approach to rendering soft shadows. The rectangular area light source is modeled with a point light source that translates with two degrees of freedom. This translation generalizes the projections of triangles and of output image samples onto the framebuffer. The generalized projection is the locus of projections as the point light source translates. With this projection generalization, the framebuffer provides, as before, a conservative set of sample/triangle pairs that have to be examined for light visibility, which now could be fractional.

We render soft shadows according to Algorithm 1. The algorithm proceeds in five main steps. In ***STEP 1***, the scene is rendered from the output view deferring shadow computation (line 2). This defines

Algorithm 1 4D-rasterization for fast soft shadow rendering

Input: 3D scene *SCE* modeled with triangles, light rectangle L, output view V, light discretization resolution $m \times n$, 4D rasterization framebuffer resolution $w \times h$.
Output: *SCE* rendered from V with an $m \times n$ occlusion mask of L for each output sample s.

```
1:  STEP 1: render output image without shadows
2:  Render SCE from V to preliminary image I
3:  STEP 2: define frame buffer for 4D rasterization
4:  Define plane P for 4D rasterization frame buffer FB
5:  for each pixel s in I do
6:      FB.aabb = FB.aabb ∪ Proj4(s.xyz, L, P)
7:  STEP 3: assign output image samples to FB pixels
8:  for each output image sample s in I do
9:      s.aabb = Proj4(s.xyz, L, FB)
10:     for each FB pixel p in s.aabb do
11:         p.sampleSet = p.sampleSet ∪ {s}
12: STEP 4: assign triangles to FB pixels
13: for each triangle t in S do
14:     t.aabb = Proj4(t.v0, L, FB) ∪ Proj4(t.v1, L, FB) ∪
15:                                   ∪ Proj4(t.v2, L, FB)
16:     for each FB pixel p in t.aabb do
17:         p.triangleSet = p.triangleSet ∪ {t}
18: STEP 5: computation of light occlusion masks
19: for each FB pixel p do
20:     for each triangle t in p.triangleSet do
21:         for each output sample s in p.sampleSet do
22:             for i = 1 to m do
23:                 for j = 1 to n do
24:                     s.mask_ij |= Occlusion(L_ij, t, s.xyz)
```

the output image samples for which the light occlusion masks have to be computed.

STEP 2 defines the framebuffer FB used for 4D rasterization. The plane P of FB is defined first as a plane parallel to the light plane and sufficiently far away for all scene geometry to be located in between the light plane and the framebuffer plane (Line 4). The actual location of plane P is determined using the scene diameter. The rectangular uniform grid of FB is aligned with the light rectangle. The extent of the grid of FB is defined as the 2D axis aligned bounding box $FB.aabb$ that encompasses the generalized projections of the output image samples onto P (Lines 5-6).

The generalized projection *Proj4* of a 3D point A onto a plane P as given by a light rectangle $L_0L_1L_2L_3$ is the rectangle defined as the four intersections between the rays AL_0, AL_1, AL_2, and AL_3 and plane P. The $FB.aabb$ is discretized to a $w \times h$ uniform grid. Figure 2 illustrates in 2D the construction of FB. In this 2D illustration, the generalized projection of output image sample s_1 is segment BC, and $FB.aabb$ extends from A to D.

STEPS 3 and 4 assign output image samples and scene triangles to the pixels of FB, which defines an implicit sample to triangle assignment. A sample is assigned to all FB pixels covered by the generalized projection of the sample (Lines 8-11). A triangle is assigned to all FB pixels covered by the 2D axis aligned bounding

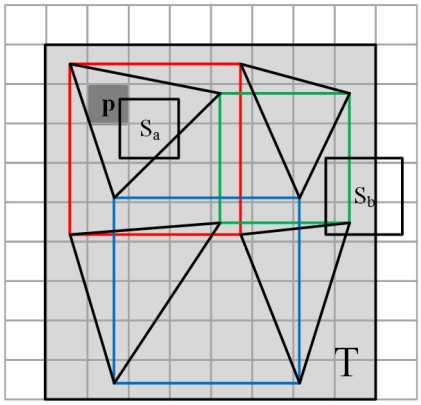


Figure 3: Generalized projection of triangles and of output image samples. The generalized projection of a triangle t is the grey shaded area T. T is computed as the 2D AABB of the generalized projections of the three vertices of t, shown with the red, green, and blue rectangles. The generalized projections of two output image samples s_a and s_b are the rectangles S_a and S_b. Both S_a and T cover framebuffer pixel p, which creates the (s_a,t) pair, so t is considered when computing the light occlusion mask of s_a. S_b is not completely in T so t does not need to be considered for the occlusion mask of S_b.

box (AABB) of the generalized projections of the three triangle vertices (Lines 13-17). Figure 3 shows the assignment of samples and triangles to the pixels of FB. The generalized projections of the three vertices of a triangle T are shown with the red, green, and blue rectangles. T is assigned to all FB pixels covered by the 2D AABB of these three rectangles. Rectangle S_a is the generalized projection of an output image sample s_a; s_a is assigned to all pixels touched by S_a. Pixel p is assigned both T and s_a, so the pair (s_a, T) is considered at STEP 5 for light occlusion computation.

STEP 5 computes the occlusion masks of the output image samples by processing each pixel p of the 4D rasterization frame buffer. All triangles stored at p (Line 20) are considered against all output image samples stored at p (Line 21). For each sample/triangle pair (s,t) the algorithm checks for each of the $m \times n$ light samples L_{ij} whether L_{ij} is occluded from s by t (Lines 22-24). An occlusion mask value of a sample s can be safely updated in parallel at multiple FB pixels p since we use a logical *or* operation.

This general 4D rasterization algorithm for rendering soft shadows is accelerated by avoiding considering the same output image sample/triangle pair multiple times (Section 3.2), and by evaluating at STEP 5 the occlusion of an entire row of light samples at once (Section 3.3).

3.2. Avoiding redundant output sample/triangle pairs

We accelerate the general 4D rasterization algorithm described above based on the fundamental observation that an output image sample/scene triangle pair needs to be considered only if the generalized projection of the sample is *completely* covered by the generalized projection of the triangle. If the generalized projection of a sample is not completely covered by the generalized projection of a triangle, the triangle does not occlude the light rectangle as seen from the sample. Using Figure 3 again, the generalized projection S_b of a sample s_b is not fully covered by the generalized projection of t (gray shaded area T), therefore the (s_b, t) pair does not have to be considered since t does not occlude the light as seen from s_b. We use this *full coverage requirement* property (see Appendix for proof) to optimize Algorithm 1.

First, we avoid unnecessary sample/triangle pairs by modifying STEP 3 as shown in Algorithm 2. Based on the *full coverage requirement* property, we only need to worry about sample/triangle pairs for which the triangle generalized projection completely contains the generalized projection of the sample. Therefore it is sufficient to assign a sample to only one of the pixels covered by the generalized projection of the the sample. The generalized projection of the output image sample is replaced with a conventional projection from just one corner of the light, i.e. L_0. This way the sample is assigned to exactly one FB pixel p, and a sample/triangle pair is considered at most once since a triangle covers p at most once. The optimization removes the redundancy of considering the same sample/triangle pair multiple times.

Algorithm 2 STEP 3 optimization

```
1: STEP 3: assign output image samples to FB pixels
2: for each output image sample s in I do
3:     p = Proj(s.xyz, L0, FB)
4:     p.sampleSet = p.sampleSet ∪ {s}
```

We use the same *full coverage requirement* property for a second optimization of Algorithm 1. STEP 5 is modified as shown in Algorithm 3. Lines 5-6 add a trivial rejection of the sample/triangle pair (s,t) if the generalized projection of s is not fully contained in the generalized projection of t. Whereas the first optimization avoids considering the same sample/triangle pair at multiple pixels, this second optimization reduces the number of pairs considered at the same pixel. A pair is considered only if the triangle could occlude the light as seen from the sample. Lines 7-17 show that the occlusion mask is now computed one entire row at a time, as described in Section 3.3.

3.3. Row-based light occlusion computation

Referring again to the optimized STEP 5 of Algorithm 1 shown in Algorithm 3, lines 7-9 compute the three planes given by the three triangle edges and the output image sample, and line 10 computes the triangle plane. Lines 11-16 update the occlusion mask for an entire row at the time.

If the light row segment $L_{i1}L_{in}$ does not intersect plane P_0, then the entire light row segment is either completely occluded or completely unoccluded, based on the sidedness of L_{i1} with respect to P_0 (Line 13). Here it is assumed that the light is sampled with a resolution of 16×16, hence the four hexadecimal digits used to represent the partial row occlusion mask M_0. If $L_{i1}L_{in}$ does intersect P_0 (Line 14), then the light row segment is partitioned in two subsegments $L_{i1}L_{ik}$ and $L_{ik}L_{in}$, with one occluded and one not (Lines 15-16). Figure 4 illustrates the case when $L_{ik}L_{in}$ is occluded and $L_{i1}L_{ik}$ is

Algorithm 3 STEP 5 optimization

```
1:  STEP 5: computation of light occlusion masks
2:  for each FB pixel p do
3:      for each triangle t in p.triangleSet do
4:          for each output sample s in p.sampleSet do
5:              if Proj4(s.xyz, L, P) ⊄ t.aabb then
6:                  continue
7:              P0 = Plane(s.xyz, t.v0, t.v1)
8:              P1 = Plane(s.xyz, t.v1, t.v2)
9:              P2 = Plane(s.xyz, t.v2, t.v0)
10:             P3 = Plane(t.v0, t.v1, t.v2)
11:             for i = 1 to m do
12:                 if ((Li1Lin ∩ P0) = Lik) = ∅ then
13:                     M0 = P0(Li1)>0 ? 0xFFFF : 0x0000
14:                 else
15:                     k = Li1Lik / Li1Lin; temp = (1<<k) -1
16:                     M0 = P0(Li1)>0 ? temp : ¬ temp
17:                 s.maski |= M0 & M1 & M2 & M3
```

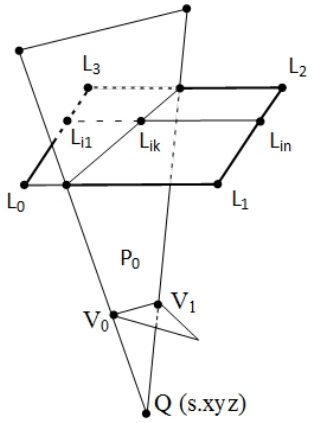


Figure 4: Direct (non-iterative) computation of row i of the light occlusion mask for output image sample Q. The intersection L_{ik} between the row segment $L_{i1}L_{in}$ and the triangle edge plane QV_0V_1 defines the column k where the row bits switch from *not occluded* to *occluded*.

not, which corresponds to the second value in the conditional assignment in Line 16. The partial row occlusion masks M_1, M_2, M_3 corresponding to the planes P_1, P_2, P_3 are computed the same way (not shown in Algorithm 3 for conciseness). The partial masks are combined and the row occlusion mask of s is updated (Line 17).

4. Results and discussion

In this section, we discuss the quality of the shadows rendered by our method, we report frame rate measurements, and we discuss limitations. We tested our algorithm with several scenes: *Tree* (367ktris), *Tower* (514ktris), *Toys* (263ktris), *Dragon* (81ktris), *Coaster* (1,096ktris), and *Church* (240ktris) (see Figs. 1 and 5). All performance measurements reported in this paper were recorded on measured on a workstation with a 3.5GHz Intel(R) Core(TM) i7-4770 CPU, with 8GB of RAM, and with an NVIDIA GeForce GTX 780 graphics card. We compare our method to NVIDIA's Optix ray tracer [Nvi16], for the same uniform sampling of the light rectangle. The average frame rates were computed over the paths shown in the accompanying video.

4.1. Quality

Our method correctly estimates visibility between light source samples and output image pixel samples. Consequently the soft shadows rendered are identical to those rendered by ray tracing, when using the same uniform sampling of the light rectangle. The only approximation made by our method that influences quality is the resolution at which the light rectangle is sampled, i.e. the resolution of the occlusion masks. Whereas 8×8 is clearly insufficient, 16×16 results in occasional shadow discretization artifacts, 32×32 produces consistently good results, and there is virtually no improvement for an even higher light sampling resolution (Figure 6).

4.2. Performance

We implemented our algorithm using a combination of shaders (STEP 1) and CUDA code (STEPS 2-5), see Algorithm 1, and the optimizations in Algorithms 2 and 3. It is true that the generalized projections of output image samples and of triangles could be rasterized with the hardware rasterizer. However, each pixel needs to accumulate a variable number of samples and triangles, which is more easily handled using CUDA. We report the dependency of the frame rate on the various parameters of our algorithm. Unless otherwise specified, the light sampling resolution is 32×32, the resolution of the framebuffer FB used to assign triangles to samples is 128×128, and the output resolution is 512×512.

Table 1 gives the average frame rate for our test scenes for various output image resolutions, as well as the speedups versus ray tracing. Our method provides interactive rates for the 512×512 resolution, the frame rate decreases linearly with the number of output image pixels, and our method is substantially faster than ray tracing in all cases. STEPS 1, 2, and 3 have negligible time cost. Table 3 gives the time costs of STEP 4, which assigns triangles to FB pixels, and of STEP 5, which computes the occlusion masks for each sample. STEP 5 has the dominant cost.

Table 1: Frame rates [fps] and speedup versus ray tracing for various output resolutions.

	512^2	$1,024^2$	$1,280^2$
Tree	5.2 (18×)	1.4 (15×)	0.8 (11×)
Tower	10 (35×)	3.4 (21×)	1.8 (20×)
Toys	15 (25×)	3.8 (18×)	2.2 (18×)
Dragon	20 (15×)	5.5 (9×)	3.5 (9×)
Coaster	6.7 (14×)	1.9 (11×)	1.2 (11×)
Church	13 (10×)	3.2 (6×)	2.2 (6×)

Table 2 shows the benefits of three optimizations of our algorithm. The optimization of STEP 3 shown in Algorithm 2 which avoids storing a sample at multiple FB pixels avoids significant sample redundancy, as seen in column 2. Column 3 gives the significant frame rate speedups provided by the early rejection of sample/triangle pairs in the optimization of STEP 5 (Lines 5-6 in Algo-

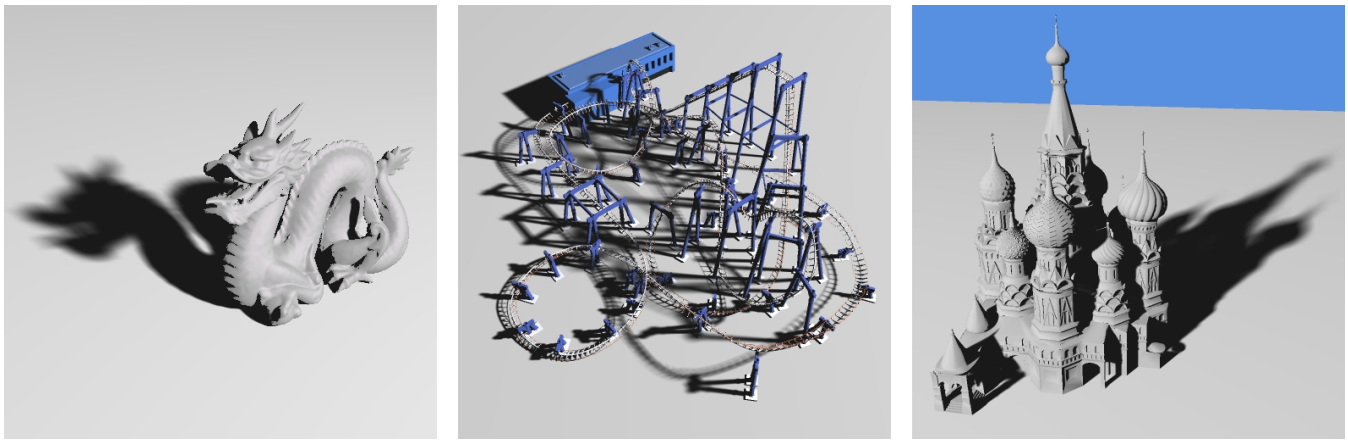
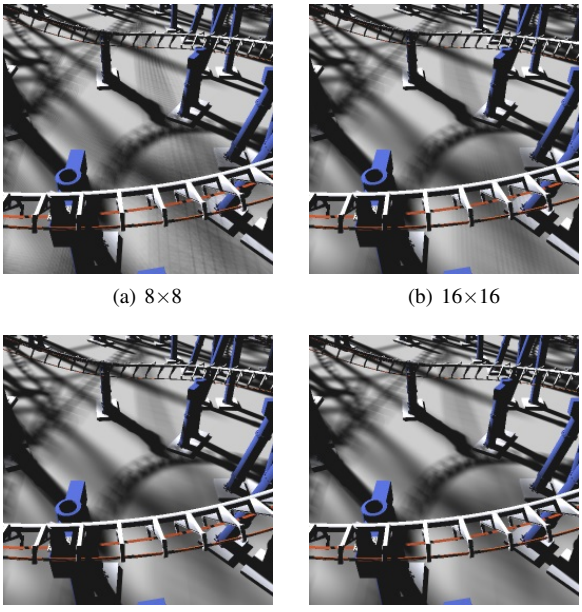


Figure 5: Additional scenes used to test our method.



(a) 8×8 (b) 16×16 (c) 32×32 (d) 64×64

Figure 6: Quality as a function of light sampling resolution.

rithm 3). Column 4 gives the significant speedups of processing one entire occlusion bitmask row at the time (Lines 11-15 in Algorithm 3).

Table 4 gives the average frame rate for our test scenes for various light sampling resolutions. The time cost of our algorithm is linear with the number of rows in the occlusion bit masks. As the number of light samples increases four times, the frame rate only decreases two times. This is an important advantage over algorithms that compute the bits in the masks one at the time, for which the frame rate is reduced fourfold when the light sampling resolution doubles. Table 5 gives the average frame rate for our test scenes for various lengths of the light rectangle diagonal. The diagonals of our scenes are 114, 54, 46, 58, 28 and 22, respectively. The

Table 2: Benefits of algorithm optimizations.

	Sample redundancy	**Early rejection speedup**	**Row-based speedup**
Tree	100	3.3×	13×
Tower	88	4.4×	12×
Toys	40	3.8×	12×
Dragon	206	4.8×	10×
Coaster	50	2.5×	13×
Church	183	12×	11×

Table 3: Time costs [ms] for the main steps of our algorithm.

	Tree	*Tower*	*Toys*	*Dragon*	*Coaster*	*Church*
*STEP*4	27	16	5	5	15	18
*STEP*5	160	72	60	36	121	52

center of the light is 69, 50, 43, 32, 37 and 47 away from the center of the scene. For these test scenes the frame rates are approximately inversely proportional to the light rectangle diagonal, and not to the rectangle area.

Table 4: Frame rates [fps] for various light resolutions.

	8×8	**16×16**	**32×32**	**64×64**
Tree	12	8.2	5.2	2.2
Tower	20	15	10	5.4
Toys	30	24	15	6.3
Dragon	45	30	20	9.1
Coaster	15	11	6.7	3.2
Church	24	18	13	6.7

Table 6 gives the average frame rate for our test scenes for various resolutions of the framebuffer FB used to assign triangles to samples. The best performance is obtained for 128×128. When FB

Table 5: Frame rates [fps] for various lengths of the light rectangle diagonal.

	1	**2**	**3**	**4**	**5**
Tree	19	9.4	5.2	3.2	2.3
Tower	29	17	10	6.8	5.0
Toys	26	20	15	12	9.5
Dragon	43	30	20	15	10
Coaster	20	11	6.7	4.3	3.1
Church	36	21	13	8.6	5.2

resolution is too low, the generalized projection of triangles is approximated coarsely which leads to considering a large number of sample/triangle pairs unnecessarily. When FB is too fine, STEP 4 is expensive, as each triangle is assigned to a large number of FB pixels, and the number of samples per FB pixel is imbalanced.

Table 6: Frame rates [fps] for various resolutions of the framebuffer used to pair triangles with samples.

	256×256	**128×128**	**64×64**	**32×32**
Tree	4.5	5.2	4.6	3.1
Tower	8.4	10	9.9	7.8
Toys	15	15	9.2	6.1
Dragon	18	20	17	12
Coaster	6.9	6.7	6.4	5.2
Church	9.4	13	12	8.7

An essential aspect of our algorithm is the efficiency of the assignment of triangles to output image samples, so we analyze it in detail. A perfect assignment would only pair a triangle with an output image sample if the triangle is relevant, i.e. if the triangle blocks at least one light sample. Our algorithm is conservative, in the sense that a sample is assigned all, but not only, blocking triangles. The algorithm stores a set of triangles and a set of output image samples at each FB pixel, and then considers, for each FB pixel, all sample/triangle pairs given by the two sets (Lines 2-4 in Algorithm 3). Many of these pairs are trivially rejected as irrelevant (Lines 5-6). Not all remaining pairs are relevant, since the generalized projection of the triangle is approximated with a 2D AABB of the generalized projections of its three vertices. Table 7 shows the efficiency of the assignment achieved by our algorithm with two numbers, for each scene, which report the percentage of relevant sample/triangle pairs before and after early rejection. The trivial rejection improves the assignment efficiency considerably, which is between 28% and 65%.

Table 7: Efficiency of triangle to sample assignment, as a percentage of relevant sample/triangle pairs from the total number of pairs considered by our algorithm, before and after early rejection.

	Tree	*Tower*	*Toys*	*Dragon*	*Coaster*	*Church*
Before	15%	14%	23%	26%	8%	10%
After	44%	45%	53%	65%	28%	37%

Whereas Table 1 provides a frame rate comparison between our method and ray tracing for equal quality, we have also performed a quality comparison for equal frame rate. As shown in Table 8, to achieve the same performance, ray tracing has to reduce the light sampling resolution considerably, which results in objectionable artifacts (Figure 7).

Table 8: Equal performance quality comparison of our algorithm to ray tracing.

	Frame rate [fps]	**4D rasterization bit mask res.**	**Ray tracing # of light rays**
Tree	5.2	32×32 = 1,024	72
Tower	10.0	32×32 = 1,024	36
Toys	15.0	32×32 = 1,024	42
Dragon	20.0	32×32 = 1,024	72
Coaster	6.7	32×32 = 1,024	81
Church	21.4	32×32 = 1,024	100

4.3. Limitations

Like for all exact soft shadow rendering algorithms, the cost of our algorithm is linear with the number of pixels in the output image, so it increases fourfold as the resolution increases twofold in each direction. Fast frame rates are obtained only for a relatively low output image resolution of 512×512. However, our algorithm is efficient, in the sense that for about 50% of the sample/triangle pairs (s,t) that are processed t actually occludes the light as seen from s. In other words, assuming the same processing of a sample/triangle pair, the frame rates achieved are within a factor of two of the frame rates of an ideal exact algorithm that only considers relevant sample/triangle pairs.

Our algorithm processes an entire row of the occlusion bit mask at once, leveraging bit shift operations, which can support up to 64×64 bit mask resolutions, limitation due to the maximum length of integers on 64 bit architectures. One possible future work direction is to lookup the partial occlusion mask inferred by a triangle edge plane (e.g. P_0 in Figure 4) directly into an LUT, which also has the potential to accelerate STEP 5 by eliminating the for loop on the bit mask rows (Line 11 in Algorithm 3). Another limitation of our algorithm is a frame rate dependence on the prevalence of soft shadows in the output image, which can cause frame rate fluctuations during scene exploration. Providing a frame rate guarantee is more challenging in the case of algorithms like ours that evaluate visibility by projection followed by rasterization, compared to ray tracing. Projection followed by rasterization has the benefit of amortizing the cost of individual rays, but the individual rays of ray tracing can be budgeted with more flexibility to meet a desired performance.

5. Conclusions and Future work

We have presented an algorithm for rendering complex soft shadows accurately at interactive rates. Our algorithm probes visibility from the light source with a generalized projection and rasterization of blocking triangles. The algorithm has the desirable properties of generality and of simplicity, which are prerequisites for

benefiting from future advances of raw GPU compute power, and for widespread adoption in applications.

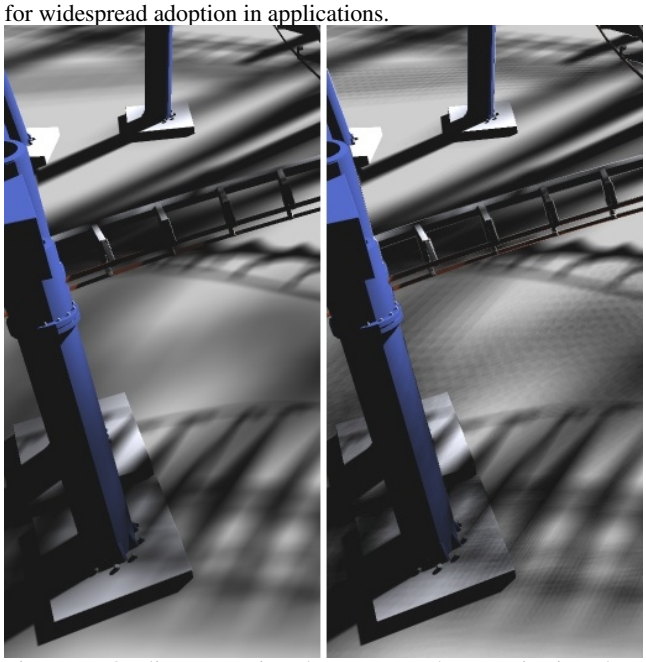


Figure 7: Quality comparison between our 4D rasterization algorithm (left) and ray tracing (right) for equal performance, which forces ray tracing to undersample the light.

We compared our method to ray tracing for the same uniform sampling of the rectangular light source, where we have a significant performance advantage. Of course, ray tracing can choose an irregular, and/or stochastic light sampling pattern. Compared to uniform light sampling, irregular sampling pattern brings higher quality for the same number of light samples. In future work, our method could be extended to work with a jittered set of light sampling locations obtained by perturbing an initially uniform set of sampling locations, which will alleviate shadow banding. Our approach of ammortizing the cost of estimating triangle occlusion of light samples by projection followed by rasterization does not easily support adapting the sampling pattern. This lack of flexibility brings a quality guarantee that stochastic/adaptive methods cannot easily meet, as an aggressive reduction of the number of light samples considered for a given output image sample can result in light leaks.

Our work addresses the problem of soft shadows by sampling the light rectangle with one thousand point light sources. However, the algorithm leverages the locality and the uniform structure of the light samples, and future work is needed to generalize to the interactive rendering of scenes lit with many area and point light sources.

6. Appendix

Full Coverage Requirement Property

Let s be an output image sample and let t be a scene triangle. Let S be the generalized projection of s with a light rectangle $L_0L_1L_2L_3$ onto a plane P, and let T be the AABB of the generalized projection of t. If $S \not\subset T$, then t does not occlude $L_0L_1L_2L_3$ from s.

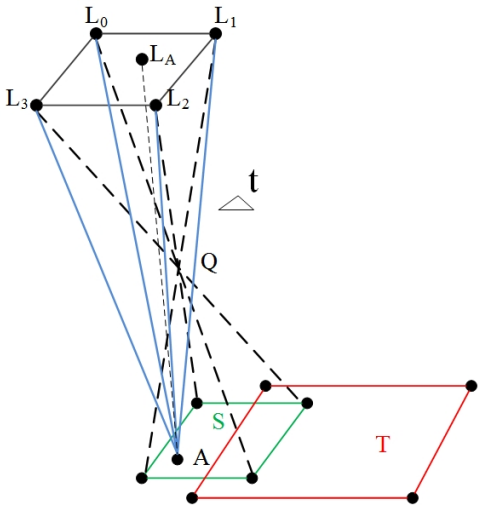


Figure 8: The generalized projection S of output image sample Q is not completely contained by the generalized projection T of triangle t, therefore t does not occlude any point of the rectangular light source $L_0L_1L_2L_3$ as seen from Q, and the pair (Q,t) does not need to be considered when rendering soft shadows.

Proof

Since $S \not\subset T$, $S-T \neq \emptyset$. (see Figure 8)
Let A be a point such that $A \in S-T$.
Since $A \notin T$, $AL \cap t = \emptyset$, $\forall$ point $L \in L_0L_1L_2L_3$.
Therefore $T \cap$ pyramid $AL_0L_1L_2L_3 = \emptyset$. (1)
Let $Q = s.xyz$, and let $L_A = AQ \cap L_0L_1L_2L_3$.
Since $A \in S$, $L_A \in L_0L_1L_2L_3$.
Therefore $Q \in$ pyramid $AL_0L_1L_2L_3$.
Therefore pyramid $QL_0L_1L_2L_3 \subset$ pyramid $AL_0L_1L_2L_3$. (2)
From (1) and (2) it follows that $t \cap$ pyramid $QL_0L_1L_2L_3 = \emptyset$.
This terminates the proof that t does not occlude $L_0L_1L_2L_3$ as seen from s.

References

[AHT04] ARVO J., HIRVIKORPI M., TYYSTJRVI J.: Approximate soft shadows with an image-space flood-fill algorithm. *Computer Graphics Forum 23*, 3 (2004), 271–280. 2

[AL04] AILA T., LAINE S.: Alias-free shadow maps. *Proceedings of the Fifteenth Eurographics conference on Rendering Techniques* (2004), 161–166. 2

[AMMH07] AKENINE-MÖLLER T., MUNKBERG J., HASSELGREN J.: Stochastic rasterization using time-continuous triangles. *Proceedings of the 22Nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware* (2007), 7–16. 3

[BS02] BRABEC S., SEIDEL H.: Single sample soft shadows using depth maps. *Graphics Interface* (2002), 219–228. 2

[BW09] BENTHIN.C., WALD I.: Efficient ray traced soft shadows using multi-frusta tracing. *Advances in Computer Graphics Hardware* (2009), 135–144. 2

[ED07] EISEMANN E., DCORET X.: Visibility sampling on gpu and applications. *Computer Graphics Forum 26*, 3 (2007), 535–544. 2

[ESA11] EISEMANN E., SCHWARZ M., ASSARSSON U.: Real-time shadows. *CRC Press* (2011). 2

[FBP08] FOREST V., BARTHE L., PAULIN M.: Accurate shadows by depth complexity sampling. *Computer Graphics Forum 27*, 2 (2008), 663–674. 2

[Fer05] FERNANDO R.: Percentage-closer soft shadows. *ACM SIGGRAPH 2005 Sketches* (2005). 2

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

[GBP07] GUENNEBAUD G., BARTHE L., PAULIN M.: High-quality adaptive soft shadow mapping. *Computer Graphics Forum 26*, 3 (2007), 525–533. 2

[HLHS03] HASENFRATZ J., LAPIERRE M., HOLZSCHUCH N., SILLION F.: A survey of real-time soft shadows algorithms. *Computer Graphics Forum 22*, 4 (2003), 753–774. 2

[JHH*09] JOHNSON G., HUNT W., HUX A., MARK W., BURNS C., JUNKINS S.: Soft irregular shadow mapping: fast, high-quality, and robust soft shadows. *ACM Symposium on interactive 3D graphics* (2009), 57–66. 2

[LA05] LAINE S., AILA T.: Hierarchical penumbra casting. *Computer Graphics Forum 24*, 3 (2005), 313–332. 2

[LAA*05] LAINE S., AILA T., ASSARSSON U., LEHTINEN J., AKENINE-MOLLER T.: Soft shadow volumes for ray tracing. *ACM Transactions on Graphics 24*, 3 (2005), 1156–1165. 2

[LMSG14] LECOCQ P., MARVIE J.-E., SOURIMANT G., GAUTRON P.: Sub-pixel shadow mapping. In *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2014), ACM, pp. 103–110. 2

[LSMD15] LIKTOR G., SPASSOV S., MÜCKL G., DACHSBACHER C.: Stochastic soft shadow mapping. In *Computer Graphics Forum* (2015), vol. 34, Wiley Online Library, pp. 1–11. 2

[MAAG12] MORA F., AVENEAU L., APOSTU O., GHAZANFARPOUR D.: Lazy visibility evaluation for exact soft shadows. In *Computer Graphics Forum* (2012), vol. 31, Wiley Online Library, pp. 132–145. 2

[MKHS10] MOHAMMADBAGHER M., KAUTZ J., HOLZSCHUCH N., SOLER C.: Screen-space percentage-closer soft shadows. *ACM SIGGRAPH 2010 Posters* (2010), 1–1. 2

[Mol02] MOLLER T.A.AND ASSARSSON U.: Approximate soft shadows on arbitrary surfaces using penumbra wedges. *Eurographics Symposium on Rendering/Eurographics Workshop on Rendering Techniques* (2002), 297–305. 2

[MWR12] MEHTA S. U., WANG B., RAMAMOORTHI R.: Axis-aligned filtering for interactive sampled soft shadows. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 163. 2

[NIDN13] NABATA K., IWASAKI K., DOBASHI Y., NISHITA T.: Efficient divide-and-conquer ray tracing using ray sampling. In *Proceedings of the 5th High-Performance Graphics Conference* (2013), ACM, pp. 129–135. 2

[Nvi16] NVIDIA: Nvidia optix ray tracing engine. *http://developer.nvidia.com/optix* (2016). 5

[SDMS14] SELGRAD K., DACHSBACHER C., MEYER Q., STAMMINGER M.: Filtering multi-layer shadow maps for accurate soft shadows. *Computer Graphics Forum 34*, 1 (2014), 205–215. 2

[SEA08] SINTORN E., EISEMANN E., ASSARSSON U.: Sample based visibility for soft shadows using alias-free shadow maps. In *Computer Graphics Forum* (2008), vol. 27, Wiley Online Library, pp. 1285–1292. 2

[SFY13] SHEN L., FENG J., YANG B.: Exponential soft shadow mapping. *Computer Graphics Forum 32*, 4 (2013), 107–116. 2

[SS08] SCHWARZ M., STAMMINGER M.: Microquad soft shadow mapping revisiteds. *Eurographics 2008 Annex to the Conference Proceedings: Short Papers* (2008), 295–298. 2

[Whi79] WHITTED T.: An improved illumination model for shaded display. *Communications of The ACM 23*, 6 (1979), 343–349. 2

[WZKV14] WANG L., ZHOU S., KE W., V. P.: Gears: A general and efficient algorithm for rendering shadows. *Computer Graphics Forum 33*, 6 (2014), 264–275. 2

[YFGL09] YANG B., FENG J., GUENNEBAUD G., LIU X.: Packet-based hierarchal soft shadow mapping. *Computer Graphics Forum 28*, 4 (2009), 1121–1130. 2

[YMRD15] YAN L.-Q., MEHTA S. U., RAMAMOORTHI R., DURAND F.: Fast 4d sheared filtering for interactive rendering of distribution effects. *ACM Transactions on Graphics (TOG) 35*, 1 (2015), 7. 2