GEARS: A General and Efficient Algorithm for Rendering Shadows

Lili Wang[†1], Shiheng Zhou[1], Wei Ke[2], and Voicu Popescu[3]

[1]China State Key Laboratory of Virtual Reality Technology and Systems, Beihang University, Beijing China
[2] Macao Polytechnic Institute, Macao China
[3]Computer Science, Purdue University, West Lafayette, Indiana, USA

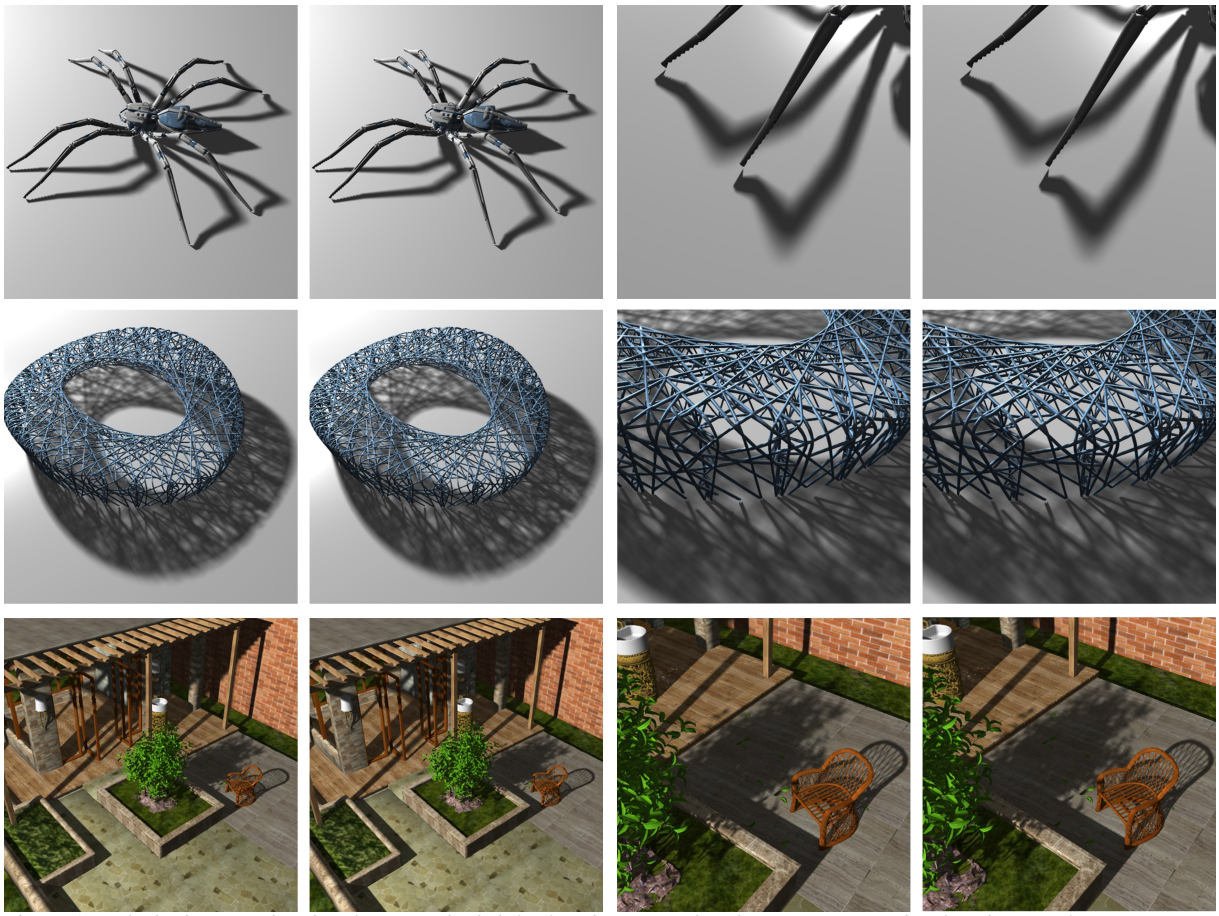


Figure 1: *Soft shadows rendered with our method (left of each pair) and with ray tracing (right of each pair), for comparison. The average frame rate for our method vs. ray tracing is 26.5fps vs. 1.07fps for the Bird Nest scene, 75.6fps vs. 5.33fps for the Spider scene, and 21.4fps vs. 1.98fps for the Garden scene. The shadow rendering times are 49.8ms vs. 1176.4ms, 14.3ms vs. 209.2ms, and 45.2ms vs. 561.8ms for the three scenes, respectively. The output image resolution is 512x512.*

Volume xx (200y), Number z, pp. 1–13

Abstract

We present a soft shadow rendering algorithm that is general, efficient, and accurate. The algorithm supports fully dynamic scenes, with moving and deforming blockers and receivers, and with changing area light source parameters. For each output image pixel, the algorithm computes a tight but conservative approximation of the set of triangles that block the light source as seen from the pixel sample. The set of potentially blocking triangles allows estimating visibility between light points and pixel samples accurately and efficiently. As the light source size decreases to a point, our algorithm converges to rendering pixel accurate hard shadows.

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

1. Introduction

Rendering accurate soft shadows at interactive rates remains an open research problem. The core challenge is to estimate what fraction of an area light source is visible from each of the surface samples captured by the output image pixels. Consider a 3-D scene modeled with N triangles, a rectangular light source sampled with kxk light samples, and an output image of resolution wxh. The challenge is to compute whether there is a direct line of sight from each of the kxk light samples to each of the wxh pixel samples.

One approach is to trace kxk rays from each of the wxh pixels. Considering all wxh x kxk x N possible ray-triangle pairs is prohibitively expensive and an acceleration scheme is needed to avoid considering ray-triangle pairs that do not yield an intersection. Another approach is to render the scene kxk times to obtain kxk conventional shadow maps. Each shadow map needs to be rendered at a resolution comparable to w x h, and, even so, shadow map undersampling artifacts can occur. The cost of a brute force implementation of the approach is prohibitive: the entire scene has to be rendered k x k times at w x h resolution. For quality soft shadows typical k values are 16 and above, which implies rendering the scene hundreds of times. A third approach is to render the scene from each of the w x h pixel samples at k x k resolution, which computes directly and accurately the visibility masks needed for each pixel sample. The cost of a brute force implementation of the approach is, again, prohibitive: the entire scene has to be rendered w x h times, albeit at the lower k x k resolution.

In this paper we propose a soft shadow rendering method that accelerates this third approach. Whereas a brute force method renders all scene triangles in order to estimate the light visibility mask of a given pixel sample P, our method only renders triangles that are likely to block the light as seen from P. The set of potentially blocking triangles is determined by projecting pixel samples and triangle shadow volumes onto a regular grid. All pixel samples that project at a given grid cell are assigned all triangles whose shadow volume projections intersect the grid cell. Our method is (1) accurate, (2) efficient, and (3) general. We also refer the reader to the accompanying video.

(1) For each pixel sample P our method is guaranteed to render all triangles that block the light as seen from P; therefore our method is accurate, in the sense that it accurately assesses visibility from each pixel sample to each light sample. Fig. 1 shows that soft shadows rendered with our method are identical to soft shadows rendered with a ray tracer (i.e. NVIDIA's OptiX) for the same light sampling resolution (i.e. 16x16).

(2) Our method is efficient because the approximations employed are not only conservative, but they are also fast to compute and tight. First, the set of blocking triangles is not computed per pixel sample but rather per group of pixel samples, leveraging pixel to pixel coherence. Second, the regular grid is designed such that the projection of a triangle's shadow volume be approximated well with a simple axis aligned bounding box (AABB), which precludes unnecessary triangle to pixel sample assignments without resorting to expensive convex hull computations. Third, graphics hardware advances have brought programmability flexibility that enables an efficient implementation of our method. In particular, the atomic operations and the memory model provided by parallel programming environments such as CUDA enable storing and processing efficiently a variable number of pixel samples and a variable number of triangle ID's at each cell of a regular grid. Our method renders complex soft shadows at interactive rates. In all our experiments, our method is substantially faster than OptiX. The ray tracing performance reported in Fig. 1 does not account for the time needed to construct the kd-tree; the performance gap is even wider for dynamic scenes.

(3) Our method is general, as it works with any scene modeled with triangles, without requiring a partitioning of the scene into blockers and receivers, and without restrictions on scene geometry such as planarity of receivers or strong connectivity of blocker meshes. Moreover the method supports fully dynamic scenes, including moving and deforming blockers and receivers, as well as light sources with changing size, location, and orientation. As the light source area decreases, our method naturally convergences to the ir-

† Chairman Eurographics Publications Board

regular z-buffer method for rendering hard shadows [JLBM05, JMB04, ZM10], producing pixel-accurate results, and avoiding the classic shadow map resolution issues.

2. Related Work

Several excellent surveys provide a comprehensive and in depth review of existing soft shadow rendering methods [ESA11, HLHS03]. In this brief overview, we distinguish between image space and geometry space methods. Image space methods, such as shadow map methods [GBP06, YDF*10, SGYF11], project the 3-D scene onto planes to compute z-buffers used to determine visibility. Geometry space methods estimate visibility in 3-D space to determine umbra and penumbra regions, e.g. the penumbra wedge and shadow volume based methods [Mol02, JHH*09, FBGP09]. Our method uses triangle shadow volumes, thus it is a geometry space method.

Based on result accuracy, soft shadow methods can be divided into three categories: shadow simulation methods, shadow approximation methods, and accurate shadow methods. Shadow simulation methods usually start from hard shadows which are fitted with penumbra regions [Mol02, BS02, AHT04]. For example soft shadows can be simulated by blurring hard shadow edges [Fer05, MKHS10]. Simulation methods are fast and are thus suitable for applications where performance is at a premium, but the shadows rendered can be substantially wrong.

Shadow approximation methods approximate the blocking geometry to facilitate visibility querying. For example, back projection methods approximate blocking geometry with shadow mapping. Guennebaud et al. [GBP06] estimate the visibility between the light source and the pixel sample by using shadow map samples that are back projected onto the light image plane. Back projection was later improved for more accurate and faster soft shadows by smooth contour detection and radial area integration [GBP07]. Schwarz and Stamminger [SS08] approximate model the shadow map samples with micro-quads and micro-triangles, which improves quality. All these approaches rely on a single shadow map to model blocker geometry, which is not conservative. Heuristic strategies are employed to fill in gaps in the shadows and to generate more accurate contours of blockers. Yang et al. [YFGL09] use multiple shadow maps to reduce the artifacts of sampling blocking geometry from a single viewpoint. A concern is rendering efficiency, which is alleviated by grouping coherent pixels in tiles and by computing a single visibility factor per tile. Shadow approximation methods produce shadows that are based on actual visibility computation and achieve interactive frame rates at the cost of approximating the blocking geometry. We do not approximate blocking geometry, but, like Schwarz and Stamminger [SS07], we do use bitmasks to estimate partial light source visibility.

Accurate soft shadow methods rely on the accurate computation of visibility between pixel samples and light samples. Our method belongs to the category of accurate soft shadow methods. Ray tracing [Whi79] naturally supports accurate soft shadows by tracing rays between pixel samples and light points, but peformance is a concern. Laine and Aila [LA05] describe a method based on hierarchical occlusion volumes for triangles to accelerate the estimation of whether a triangle blocks the ray between a pixel sample and the light source. Increasing the size of the area light source decreases the efficiency of this algorithm. The same researchers propose another acceleration scheme, dubbed soft shadow volumes, which is based on finding and working with silhouette edges as opposed to triangles [LAA*05]. The approach takes advantage of the observation that soft shadow computation does not need to worry about triangles whose projection is landlocked by the projection of neighboring connected triangles when seen from the light. The shadow wedges cast by silhouette edges are assigned to output image pixel samples using a multi-resolution hemicube, which is similar to the 2-D grid we employ to assign blocking triangles to pixel samples. Both methods leverage the general idea of computing point in volume inclusion tests in 2-D by projection and rasterization. However, performance is limited by the overall complexity of the method that implies finding silhouette edges, tracing one ray per pixel, and reconstructing and resampling the visibility function. Another limitation is poor performance for fragmented meshes, when virtually all triangle edges are silhouette edges. Forest et al. [FBP08] accelerate the soft shadow volumes approach to interactive rates using depth complexity sampling. The method eliminates the need to trace a ray per pixel, it streams silhouette edges as opposed to constructing a data structure for storing them, and it provides a good quality/performance tradeoff.

Eisemann and Decoret [ED07] developed a method for estimating visibility between two rectangular patches, which can be applied to soft shadows. They approximate the shadow volume of a blocking triangle with 4 rays per triangle vertex, one for each of the corners of a rectangular light. We use the same approximation and we also show that the approximation is conservative. Their method is not fully general: the method only allows casting shadows on a plane or a height field, and the method requires a separation between blocker and receiver, which precludes self-shadowing. Johnson et al. [JHH*09] use a point light source and edges of blockers to estimate penumbra regions, and then refine penumbra pixel intensities with extra visibility tests involving the actual area light source. The challenge of the method is a stable and efficient detection of silhouettes. Like the method we present, Benthin and Wald [BW09] construct frusta at pixel samples. However, they estimate the fractional light visibility from pixel samples by shooting rays whereas we determine and rasterize the set of potentially blocking triangles.

Sintorn et al. [SEA08] propose extending alias-free shadow maps (AFSM) [AL04] to soft shadows by accelerating

the computation of per pixel visibility masks. Like in our method, pixel samples are first projected onto the AFSM, which is a regular grid in front of the light. An AFSM location stores a variable number of pixel samples using a list. Then, the shadow volume of each blocking triangle is projected onto the AFSM. Finally, visibility bitmasks are updated for all pixel samples stored at an AFSM location covered by the shadow volume projection. One important difference between AFSM and our method is in the way the projection of the shadow volume of blocking triangles is approximated. AFSM approximates the projection by inscribing the light source into a bounding ellipsoid, by computing extremal points for the shadow volume projection, and by computing the 2-D convex hull of the extremal projection points. The convex hull is then rasterized to determine the AFSM locations covered. Our method designs the regular grid such that the shadow volume can be approximated well with a simple AABB of the projection of extremal points. This makes the expense of computing and of rasterizing the convex hull unnecessary. A second fundamental difference between the AFSM method and ours is that AFSM does not bound the number of rendering passes. A rendering pass can only update a constant number of visibility bitmasks. For example, for 8 render targets, 4 channels per pixel, and 32 bits per channel, a rendering pass can update only 4 16x16 visibility bitmasks. Additional rendering passes are needed until the AFSM location with the most pixel samples is fully treated. Our experiments show that, for a 512x512 output image resolution, even if an AFSM with a resolution of 512x512 is used, the maximum number of pixel samples in an AFSM location remains high (e.g. 31, which implies 8 rendering passes). Our method assigns triangles to pixel samples in a first pass and completes the soft shadow computation in a second pass, executed in parallel over all pixel samples.

3. GEARS Algorithm

Given a 3-D scene S modeled with triangles, an area light source modeled with a rectangle (L_0L_1 in Fig. 2), and a desired view with eye E and image plane I_0I_1, our algorithm renders soft shadows as follows:

1. Compute the output image without the soft shadows.
   a. Render S from E to obtain image I_0I_1.
2. Unproject each pixel p in I_0I_1 to pixel sample P.
3. Assign potentially blocking tris. to pixel samples P.
4. For each P, compute the frac. visibility v_p of L_0L_1.
   a. Construct camera PL_0L_1 with eye P and image plane L_0L_1 (orange frustum in Fig. 2).
   b. Render with PL_0L_1 all blocking triangles assigned to P on visibility bit mask M_P.
   c. Compute v_p as the percentage of unoccluded light samples in M_P. In Fig. 2, $v_p = LL_1 \,/\, L_1L_0$.
5. Add the contribution of light L_0L_1 to each pixel p of I_0I_1 using the computed fractional visibility v_p.

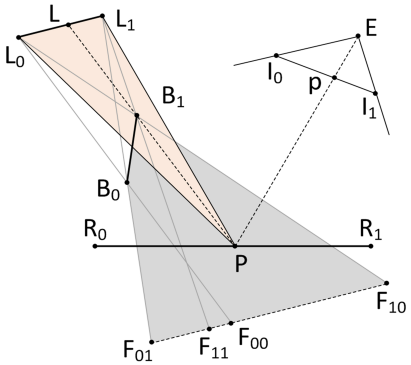


Figure 2: *Soft shadow computation overview for light L_0L_1, output image I_0I_1 with viewpoint E, blocker B_0B_1, and receiver R_0R_1.*

Step 1 is part of the regular rendering of the current frame. Step 2 is a simple pass over the output image pixels to compute a 3-D point per pixel by unprojection using the pixel's z-value. Step 3 computes which triangles should be considered for each output image pixel. Step 3 is an acceleration scheme that avoids having to consider all triangles for each output image pixel. The acceleration scheme is described in the next subsection. Step 4 computes a visibility bit mask for every pixel by rendering the potentially blocking triangles determined at Step 3. Step 5 takes a final pass over the output image pixels to finalize the deferred soft shadow computation.

3.1. Triangle to pixel assignment

Step 3 computes a superset of the triangles that block the light as seen from each pixel's 3-D sample point. Given a scene S, a rectangular area light source $L_0L_1L_2L_3$, and the pixel samples of the output image, step 3 computes a regular 2-D grid G that stores at each cell (u,v) the set of pixel samples P_{uv} that project at (u,v), and a set of potentially blocking triangles T_{uv} for P_{uv}. The grid is computed as follows:

3.1. Construct camera C with grid G as image plane.
3.2. Project pixel samples with C and assign them to the cells of G.
3.3. For each triangle T in S:
   a. Construct shadow volume V of T.
   b. Project V with C and assign T to all the cells of G that are touched by the projection of V.

(3.1) Camera C is used to decide whether a pixel sample is inside the shadow volume of a triangle, and it is constructed as follows.

In order to estimate visibility from pixel samples P to light samples L, the view frustum of camera C must contain all segments PL. We satisfy this condition by constructing the view frustum of camera C such that it contains the 3-D AABB of the four light vertices L_j and of all pixel samples P. The 3-D AABB is illustrated in 2-D as $A_0A_1A_2A_3$ in Fig. 3.

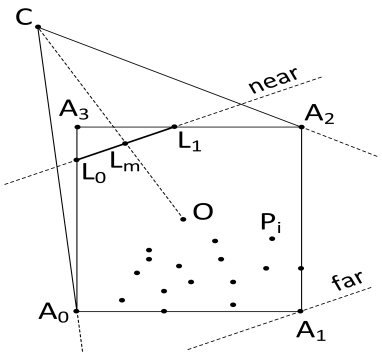


Figure 3: *2-D illustration of the camera used to define and populate the grid.*

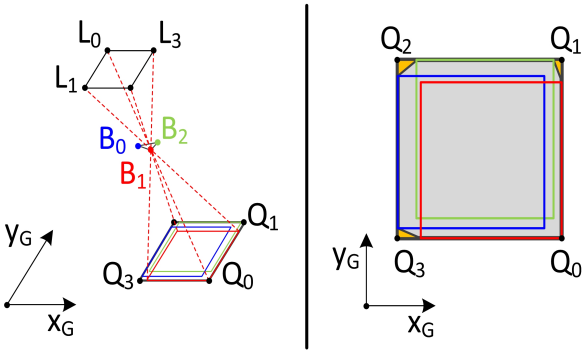


Figure 4: *Left: projection of shadow volume of triangle $B_0B_1B_2$ onto 2-D grid with axes x_G and y_G. Right: 2-D AABB $Q_0Q_1Q_2Q_3$ is a tight approximation of the shadow volume projection.*

A second requirement that the construction of camera C has to satisfy is that of minimizing the projection of the shadow volumes of the triangles. The smaller the projection, the fewer grid cells to which the triangle is assigned, and the more effective the acceleration scheme. Small shadow volume projections are obtained when camera C has a small field of view, as more orthographic projections avoid the magnification of nearby geometry produced by perspective projections. However, the rays of camera C have to converge to the light rays as the size of the light decreases. To the limit, when the light source becomes a point, the eye C has to be the light point. We place eye C on the line connecting the center O of $A_0A_1A_2A_3$ to the center of the light L_m, with distance CL_m proportional to the diagonal of the area light source. This way, as the light source decreases to a point, our algorithm converges to the irregular z-buffering algorithm for pixel-accurate hard shadows [JLBM05, JMB04, ZM10].

A third requirement that the construction of camera C has to satisfy is to provide a simple and accurate approximation of the projection of the shadow volume of each triangle. For this, we choose the image plane of camera C to be parallel to the light plane, with the image frame edges being parallel to the edges of the light rectangle. In other words, the grid axes are parallel to the light rectangle axes. This way, the 4 light corners L_0, L_1, L_2, and L_3, project the 3 vertices B_0, B_1, and B_2 of a blocking triangle to 3 axis aligned rectangles (Fig. 4, left). Moreover, as it is typically the case for the detailed scenes of interest in today's graphics applications, the majority of blocking triangles are small, which makes that, for most triangles, the projection of the shadow volume is approximated well by the 2-D AABB of the 3 projected rectangles. In Fig. 4, right, the 2-D AABB $Q_0Q_1Q_2Q_3$ is an excellent approximation of the actual projection of the shadow volume, shown in grey. $Q_0Q_1Q_2Q_3$ only overestimates the shadow volume projection by the 3 small corner triangles, shown in orange.

The far plane of camera C is given by the farthest corner of the 3-D AABB (i.e. A_1 in Fig. 3). The near plane is set to the light plane, as the light is single sided and only casts rays in one of the half spaces define by light plane.

(3.2) A pixel sample is assigned to the grid cell where the pixel sample projects with camera C. Fig. 5 uses the same notation as before. The image plane of camera C, i.e. the grid, is illustrated on the far plane. There are 4 grid cells G_0-G_3, G_0 is assigned 4 pixel samples, and G_3 none.

(3.3) A triangle is assigned to all grid cells that contain a pixel sample for which the triangle might block the light. First, the shadow volume of the triangle is computed by extruding the triangle along the 12 lines defined by the 4 light corners and the 3 triangle vertices (i.e. lines B_iL_j in Fig. 4). The vertices are extruded to the far plane of camera C. In Fig. 2, which is 2-D, there are 2x2 extrusion points F_{01}, F_{11}, F_{00}, and F_{10}. The shadow volume is defined by 15 points: 12 extrusion points and the 3 vertices of the triangle. Then, the 15 points of the shadow volume are projected with C onto G, and the triangle is assigned to all grid cells touched by the

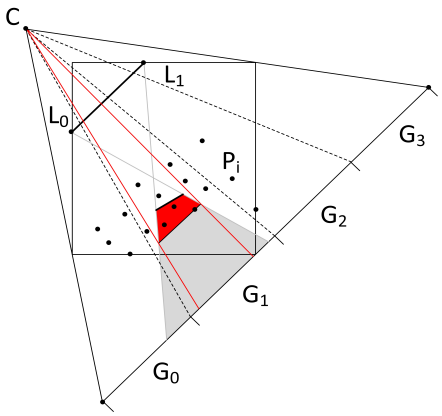
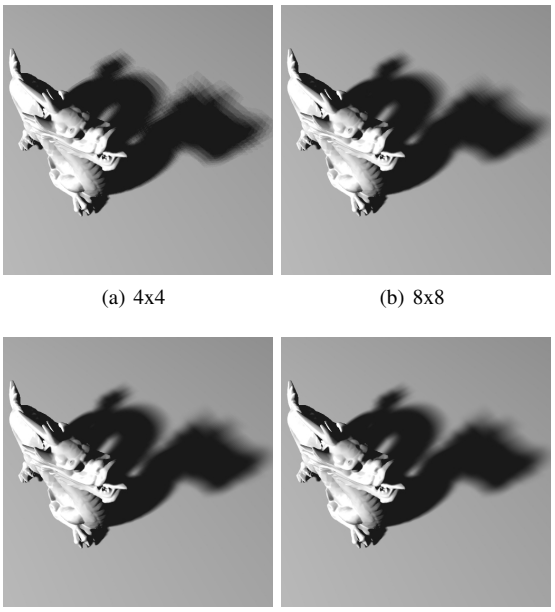


Figure 5: *Pixel sample and triangle assignment to grid.*

2-D AABB of the 15 projected points. In Fig. 5 the triangle whose shadow volume is shown shaded in grey is assigned to grid cells G_0 and G_1.

As described above, the shadow volume of a triangle is extruded to the far plane of camera C, which was constructed based on the 3-D AABB of *all* pixel samples. However, the shadow volume only needs to be extruded to the farthest pixel sample in the grid cells touched by the projection of the shadow volume. Extruding the shadow volume too far leads to unnecessary triangle to grid cell assignments. In Fig. 5, assigning the triangle to G_0 is unnecessary.

We address this problem with the following optimization. First we compute the farthest sample P_{far} in the set of grid cells touched by the projection of the shadow volume. This is done efficiently by iterating over the grid cells covered by the triangle and by computing the maximum of the grid cell maximum depths, which are pre-computed when the samples are projected to grid cells at step 3.2. Then we correct the shadow volume by only extruding the triangle up to the depth of P_{far}. Finally we project the corrected shadow volume to determine a smaller yet still conservative set of grid cells to which to assign the triangle. In Fig. 5 the corrected shadow volume is shown in red.The projection of this smaller shadow volume does not touch G_0 which avoids the unnecessary assignment.



(a) 4x4 (b) 8x8

(c) 16x16 (d) 32x32

Figure 6: Quality dependence on resolution of visibility masks.

4. Results and Discussion

In this section we discuss the quality of the shadows rendered by our method, we give a brief overview of the implementation, we report performance measurements, and we discuss limitations.

4.1. Quality

Our method is accurate in the sense that it correctly estimates visibility between light source samples and output image pixel samples. This results in soft shadows that are identical to those computed by ray tracing, when using the same number of light rays (see Fig. 1 and accompanying video). The only approximation made by our method that influences quality is the resolution of the visibility masks (Fig. 6). Whereas a resolution of 4x4 is insufficient, 8x8 produces good results, and there is virtually no improvement beyond 16x16.

4.2. Implementation overview

Referring back to the algorithm overview given in Section 3, step 1, i.e. rendering the scene preliminarily, without shadows, and step 2, i.e. i.e. unprojection to compute pixel samples, are implemented using the Cg shading language. The 3-D pixel samples are stored in a floating point texture. Steps 3, 4, and 5 are implemented in CUDA. Step 3 computes a set of potentially blocking triangles for each pixel sample according to the algorithm given in Section 3.1.

Step 3.1, which constructs camera C with grid G as its image plane, and step 3.2, which assigns a grid cell to each pixel sample, are executed together, with a single pass over the pixel samples, executed in parallel over the pixel samples, with one CUDA thread per pixel sample. The pass computes the 3-D AABB of the 3-D pixel samples with an atomic write which is executed only when the AABB needs to be extended to contain a point that the current AABB does not contain. Each pixel sample is projected to the image plane, and the 2-D AABB of the projections of the pixel samples is computed with an atomic write. The 3-D AABB is used to define the frustum of camera C and the 2-D AABB is used to define the grid G. The resolution of G is an input parameter. The projection of the pixel sample provides the mapping from pixel samples to grid cells.

Step 3.3 assigns potentially blocking triangles to grid cells to complete the mapping from pixel samples to potentially blocking triangles. Step 3.3 is executed in parallel over all triangles, with one CUDA thread per triangle, in two passes. In the first pass, the shadow volume of the triangle is projected to grid G with camera C to compute the 2-D AABB of its projection, in grid cell coordinates, and the number of cells $cellsN$ touched by the triangle (i.e. the area of the 2-D AABB). The second pass creates an array of pairs $(triID, cellID)$. The length of the array is given by the sum of the $cellsN$ variable over all triangles. The location in the array of pairs

$(triID, cellID)$ where a triangle needs to write its pairs according to its 2-D AABB is given by the prefix sum of the $cellsN$ variable over all triangles.

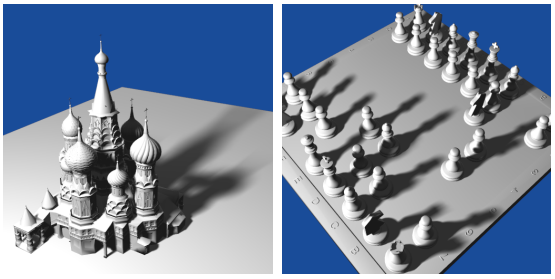
So far we have established a mapping from triangles to cells through the array of pairs $(triID, cellID)$. The mapping needs to be inverted to be able to retrieve the triangles that potentially block a given cell. The inversion is achieved by first sorting the array on $cellID's$. This places all $(triID, cellID)$ pairs for a given cell contiguously. However, one also needs to know the indices (b, e) where the pairs for each cell begin and end. This is computed in parallel over all pairs of the sorted array $(triID, cellID)$. Given a pixel sample p, let (i, j) be the grid cell where it projects, and let (b, e) be the begin and end indices for cell (i, j). Then the potentially blocking triangles of p are found in the sorted array of pairs $(triID, cellID)$ from index b to e.

At Step 4, a visibility bit mask is computed for each pixel sample by rendering the triangles assigned to the pixel sample with a camera that has the pixel sample as its eye and the light as its image plane. A triangle is rasterized by checking the location of each light sample with respect to the three edge planes. An edge plane is defined by a triangle edge and the eye of the camera that computes the bitmask. The fraction of occluded bits is trivially computed for each bit mask to finalize the shadow computation at step 5.

If the light is a point light source, the eye of camera C corresponds to the point light source, the projections of the shadow volumes of the triangles are triangles (i.e. there are only 3 extrusion points F at step 3.3), and the single shadow bit is known for each pixel sample after Step 3 (i.e. Steps 4 and 5 are not necessary).

4.3. Performance

We tested our technique on several scenes: *Spider* (41K triangles, Fig. 1, top), *Bird Nest* (67K triangles, Fig. 1, middle), *Church* (74K triangles, Fig. 7a), *Chess* (201K triangles, Fig. 7b), *Dragon* (81K triangles, Fig. 8) and *Garden* (683K triangles, Fig. 1, bottom). All performance measurements reported in this paper were recorded on a 3.5GHz Intel(R) Core(TM) i7-4770K CPU PC with 16 GB of RAM and an NVIDIA GeForce GTX 780 graphics card.



(a) *Church* (b) *Chess*

Figure 7: Additional scenes used to test our method.

Performance variation with algorithm parameters

Table 1 gives the average frame rate for our test scenes for various output image resolutions. The light visibility bitmask resolution is 16x16 and the grid resolution is 128x128. For 1,024x1,024 resolution our method achieves frame rates between 11.3 and 40.8. The *Bird Nest* scene is more challenging since on average 41% are soft shadow pixels, whereas for the other six scenes the percentage is between 11% and 14%.

Table 1: Frame rate [fps] for various output resolutions.

Output res.	512 x512	1024 x1024	1280 x1280
Chess	29.8	19.9	11.8
Church	36.2	17.1	12.3
Dragon	38.7	15.6	11.9
Bird Nest	26.6	11.3	8.8
Spider	75.6	40.8	27.7
Garden	21.4	11.4	9.0

An analysis of the performance of each step of our method reveals that 80% of the frame time is spent in rasterization. Table 2 gives a breakdown of the frame time for each of the steps of our algorithm. We have implemented an alternative rasterization approach based on a 2-D lookup table (LUT) [ED07, SEA08]. Table 3 gives the performance of our method with LUT rasterization and the speedup achieved over using edge plane rasterization.

Table 3: Frame rate [fps] for various output resolutions with LUT rasterization.

Output res.	512x512 / Speedup	1024x1024 / Speedup	1280x1280 / Speedup
Chess	33.1 / 11%	23.2 / 17%	14.9 / 26%
Church	38.8 / 7%	19.8 / 16%	14.5 / 18%
Dragon	40.9 / 6%	18.2 / 17%	12.7 / 7%
Bird Nest	54.4 / 105%	18.8 / 66%	11.7 / 33%
Spider	99.3 / 31%	49.0 / 20%	33.0 / 19%
Garden	29.8 / 39%	15.8 / 39%	11.2 / 24%

A significant performance gain is obtained at the cost of the small rasterization approximation introduced by the LUT. The maximum approximation error of LUT rasterization is given by the maximum difference between bitmasks stored at neighboring LUT locations. The smallest rasterization approximation error is obtained for a high resolution LUT and for an irregular sampling of the light where no three light samples are collinear. For Table 4 we used a 1,024x1,024 LUT and 256 random light samples that has a

Table 2: Time [ms] of each step of our algorithm.

Scene	Project pixels to the image plane	Prefix sum of the cellsN variable	Relate triangles to cells	Sorting (triID,cellID) pairs on cellID	Bit mask rasterization
Chess	0.05	7.50	2.12	1.98	27.0
Church	0.04	0.83	0.55	2.05	29.1
Dragon	0.07	0.92	0.26	1.95	29.7
Bird Nest	0.07	0.51	0.13	1.36	47.6
Spider	0.07	0.29	0.06	1.06	9.6
Garden	0.06	2.37	0.83	1.92	35.7

maximum neighboring bitmask difference of 9. This translates to average per pixel channel errors between 0.004 and 0.061 for the 7 scenes, a very small error relative to the performance gain brought by LUT rasterization.

Table 4: Per pixel channel errors with LUT rasterization compared to edge plane rasterization.

Scene	Error	
	Max	Average
Chess	4	0.009
Church	2	0.008
Dragon	3	0.020
Bird Nest	5	0.061
Spider	3	0.004
Garden	4	0.010

Table 5 gives the average frame rate for our test scenes for various resolutions of the light visibility bit mask. Output resolution is 512x512, and grid resolution is 128x128. The bit mask resolution only influences the cost of rasterizing the potentially blocking triangles for each pixel sample.

Table 5: Frame rate [fps] for various visibility mask resolutions.

Bit mask res.	4x4	8x8	16x16	32x32
Chess	36.7	35.6	29.8	22.0
Church	51.0	44.3	36.2	21.7
Dragon	48.7	46.2	38.7	24.9
Bird Nest	57.7	55.3	26.5	16.0
Spider	114.6	110.0	75.6	48.4
Garden	34.2	30.1	21.4	15.2

Table 6 gives the average frame rate for our test scenes for various light source sizes. The diagonals of our scenes are 25, 46, 28, 56, 42 and 185, respectively. The center of the light is 25, 24, 25, 26, 26 and 100 away from the center of the scene. Visibility bit mask resolution is 16x16, output resolution is 512x512, and grid resolution is 128x128. The soft shadows obtained with the various light diagonals are shown for the *Dragon* in Fig. 8.

(a) Diagonal=2 (b) Diagonal=3

(c) Diagonal=4 (d) Diagonal=5

Figure 8: Soft shadows with various light source sizes.

Table 6: Frame rate [fps] for various light source sizes.

Light diagonal	1	2	3	4	5
Chess	35.6	29.8	22.2	17.8	13.9
Church	53.8	36.2	21.6	14.3	11.0
Dragon	52.2	38.7	20.9	18.2	13.8
Bird Nest	30.4	26.5	22.3	19.3	14.9
Spider	88.0	75.6	59.1	52.7	45.2
Garden	29.8	21.4	19.0	16.7	14.4

Table 7 gives the average frame rate for our test scenes for various grid resolutions. The visibility mask resolution is 16x16 and the output resolution is 512x512. The lower the grid resolution, the larger the grid cells, the more variability between the sets of blocking triangles for the pixel samples

within grid cells, and the higher the penalty of assigning all pixel samples within a cell the same set of potentially blocking triangles. The higher the grid resolution, the smaller the grid cell, and the higher the number of grid cells to which a triangle has to be assigned, reducing the efficiency of the grid. For our test scenes best performance was achieved with a 128x128 grid.

Table 7: Frame rate [fps] for various grid resolutions.

Grid res.	256x256	128x128	64x64	32x32
Chess	17.9	29.8	28.5	31.5
Church	24.3	36.2	29.0	28.4
Dragon	31.2	38.7	27.7	21.9
Bird Nest	26.0	26.5	24.5	20.9
Spider	68.4	75.6	71.3	65.2
Garden	21.0	21.4	20.3	17.6

Grid efficiency

We investigate grid efficiency in terms of load balancing and in terms of the quality of the approximation of the set of blocking triangles for pixel samples.

Typical maximum and average number of pixel samples per grid cell are shown in Table 8. The grid resolution is 128x128, and output resolution is 512x512. Since the maximum number of pixel samples per grid cell is large both in an absolute and in a relative sense (i.e. it is 10 to 18 times the average), an approach, like alias-free shadow maps [SEA08], that processes a small and fixed number of pixel samples per cell for each rendering pass is inefficient. Our method processes all pixel samples in a second pass in parallel, and is significantly less sensitive to the load balancing of the grid. Table 8 also reports typical maximum and average number of triangles per grid cell.

Table 8: Number of triangles and of pixel samples per grid cell.

Scene	Triangles		Pixel Samples	
	Max	Average	Max	Average
Chess	1,438	66	129	11
Church	2,265	72	165	10
Dragon	2,986	65	232	16
Bird Nest	536	27	308	16
Spider	522	8	280	16
Garden	4,454	66	201	16

Finally we have measured the performance of the grid as a tool for assigning blocking triangles to pixel samples. Perfect performance would assign a triangle to a pixel sample only if the triangle blocks at least one light sample as seen from the pixel sample. Our method is conservative, in the sense that a pixel sample is assigned all its blocking triangles, but the set of blocking triangles is overestimated. Table 9 shows that at least 10% and as many as 33% of the potentially blocking triangles found using the grid are actually blocking triangles.

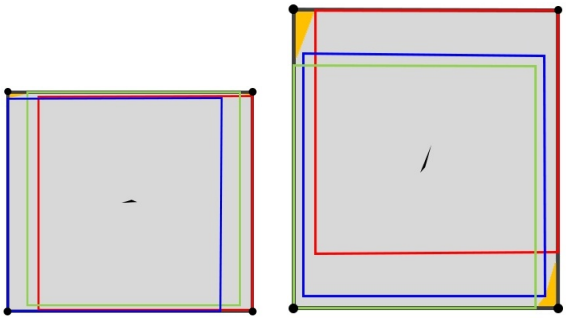


Figure 9: Visualizations of actual projections of triangle shadow volumes onto grid plane. The difference (orange) between the convex hull (grey) and the AABB are small, as predicted by Fig. 4.

Table 9: Triangle to pixel sample assignments [x1,000].

Scene	Necessary	Total	Percentage
Chess	6,993	28,860	24%
Church	11,603	44,233	26%
Dragon	19,074	57,831	33%
Bird Nest	9,941	36,077	28%
Spider	1,688	8,126	21%
Garden	4,920	48,788	10%

Our method overestimates the set of blocking triangles for a pixel sample because of three approximations.

(1) A triangle is assigned to a grid cell if the 2-D AABB of the projection of the triangle's shadow volume intersects the grid cell. The first approximation is the use of the AABB instead of the actual projection. As described in Section 3.1 (Fig. 4), we expect this approximation to be very good. We quantify the quality of the approximation by comparing the number of pixel samples covered by the 2-D AABB to the number of pixels covered by the 2-D convex hull of the projection of the shadow volume (see Step 3.3 in Section 3.1). We have found that, on average, the 2-D AABB only covers an additional 1.13%, 1.5%, 0.31%, 2.2%, 0.96% and 1.65% pixel samples for the *Chess*, *Church*, *Dragon*, *Bird Nest*, *Spider* and *Garden* scenes, respectively. Fig. 9 shows examples of actual projections of shadow volumes and highlights the small difference between the 2-D AABB and the 2-D convex hull. This indicates that the potential benefit of using a convex hull instead of the AABB is small.

Let us now look at the cost a convex hull incurs. As explained in Section 4.2, Step 3.3 is executed by taking two

passes over the scene triangles: the first pass counts the number of grid cells touched by a triangle, and the second pass builds the $(triID, cellID)$ array. When an AABB is used, the AABB is a compact, constant-size encoding of the projection of the triangle shadow volume, which is computed by the first pass and reused by the second pass. The convex hull has a larger and variable number of vertices which have to be stored in order to avoid recomputing the convex hull. Moreover, the convex hull has to be rasterized twice over the grid to first count the number of grid cells touched by a triangle shadow volume, and then to write the $(triID, cellID)$ pairs for each triangle.

(2) The second approximation is that the triangle to grid cell assignment is computed in 2-D and not in 3-D. As described in Section 3.1, we are already optimizing the shadow volume of a triangle by correcting the extrusion to extend only to the farthest sample of all grid cells touched by the unoptimized shadow volume. We have tested modulating the shadow volume extrusion for each individual grid cell, which increased the percentage of necessary assignments by 7, 7, 6, 4, 3, and 4% from the figures reported in Table 8. However, the overall performance decreased as the benefits of fewer unnecessary assignments were outweighed by the cost of modifying and reprojecting the shadow volume for each grid cell.

(3) The third and final reason for unnecessary triangle to pixel sample assignments is the fact that all pixel samples of a grid cell use the same set of potentially blocking triangles. In other words, the set of potentially blocking triangles is computed per grid cell and not per pixel sample. Of course, this approximation can be controlled through the grid resolution. For example, for the *Garden* scene the percentage of necessary assignments increases from 10% to 25% if the grid resolution increases from 128x128 to 512x512. However, as indicated earlier (Table 7), the best performance is obtained for a 128x128 grid.

As described in Section 3.1, we optimize triangle to grid cell assignment by retracting the extrusion of the shadow volume of a triangle up to the depth of the farthest sample in the grid cells touched by an initial shadow volume of the triangle built by extrusion up to the far plane (Figure 5). This optimization brings substantial benefits. The optimization avoids an increase by 223%, 102%, 258%, 269%, 1,278%, and 36% of the number of unnecessary triangle to grid cell assignments.

Compared to the brute force approach of rendering all triangles for each of 512x512 output image pixels, our method renders 512x512x683K/48,788K = 3,670 times fewer triangles for the *Garden* example in Table 7. Compared to the brute force approach of rendering a shadow map for each of 32x32 light points, our method renders 32x32x683K/48,788K = 14 times fewer triangles at a much lower resolution (i.e. 32x32 vs. 512x512).

Comparison to ray tracing

We have compared the performance of our algorithm (GEARS) with accurate, edge-plane based, rasterization to that of NVIDIA's Optix ray tracer. Table 10 shows that frame rates for our algorithm are between 10 and 25 times higher for the same bitmask resolution (i.e. for same shadow quality).

Table 10: Performance comparison between our method and ray tracing for the same light sampling resolution.

Scene	GEARS [fps] (A)	RT static [fps] (B)	Speedup (A)/(B)	RT dyn. [fps] (C)	Speedup (A)/(C)
Chess	29.8	2.91	10	2.52	12
Church	36.2	3.72	10	3.05	12
Dragon	38.7	4.01	10	2.96	13
Bird Nest	26.5	1.07	25	0.95	28
Spider	75.6	5.33	15	3.54	22
Garden	21.4	1.98	11	1.65	13

Whereas Table 10 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 11, to achieve the same performance, ray tracing has to reduce the light sampling resolution considerably. This results in noticeable artifacts as shown in Fig. 10.

Table 11: Light sampling resolution comparison between our method and ray tracing for the same frame rate.

Scene	Frame rate [fps]	GEARS Bitmask res.	Ray tracing N. of light rays
Chess	18.1	32x32 = 1,024	50
Church	13.5	32x32 = 1,024	90
Dragon	17.8	32x32 = 1,024	72
Bird Nest	17.5	16x16 = 256	12
Spider	61.5	16x16 = 256	18
Garden	21.4	16x16 = 256	20

It is true that ray tracing supports irregular light sampling straightforwardly, and irregular light samples achieve higher quality with a smaller number of rays. Our edge plane rasterization is implemented incrementally for a mimimal per light-sample amortized cost, which requires regular light samples. Irregular light samples can be supported with a slightly higher cost (i.e. one dot product versus one add). With LUT rasterization our method supports irregular light sampling at no extra cost and achieves a higher performance. Finally, ray tracing performance was measured in Table 11 for static scenes, when the ray tracing's acceleration data structure does not have to be rebuilt. For dynamic scenes, our performance remains the same, whereas the performance of ray tracing degrades further.

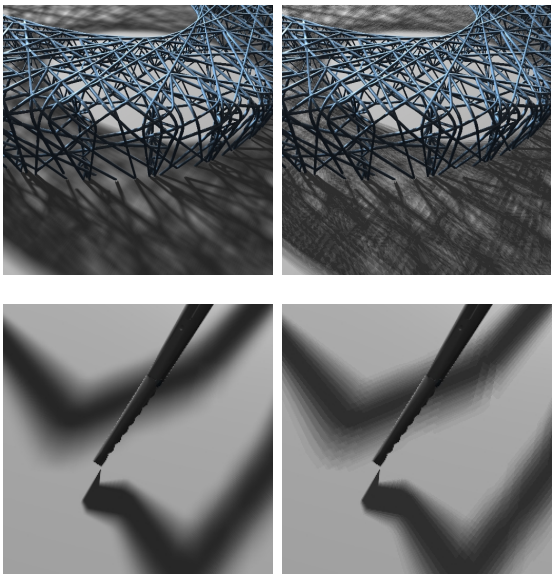


Figure 10: Quality comparison between GEARS (left) and ray tracing (right) for equal performance.

Of course, there are many ray tracing acceleration schemes and many implementations. We have chosen to compare our method primarily to Optix, which is a robust and stable ray tracing platform. The ray tracing platform developed by Aila et al. [AL09] provides faster frame rates in the static scene mode for all our scenes except for the *Spider*. The speedup of our method over Aila's ray tracer is 7.5, 8.5, 9.3, 9.1, 17, and 6.0, for our six scenes in the order they appear in Table 10. Aila's ray tracer is optimized for static scenes and reconstructing the SBVH acceleration data structure is too slow for interactive rates in the context of dynamic scenes. Recent research results [KA13, Kar12], which are expected to be integrated into the next major overhaul of Optix, i.e. Optix Prime, promise better support for dynamic scenes.

4.4. Limitations

Our approach samples the light densely enough for quality penumbra approximations, but the blockers have to be large enough for their projection to be detected in the 16x16 or 32x32 bit masks. This is a fundamental limitation of all approaches based on light visibility bitmasks. Possible solutions include increasing the resolution of the bit masks further with the corresponding performance penalty, or increasing the resolution only for bit masks corresponding to grid cells where thin features project. Thin features could be labeled at input or detected automatically in a conventional z-buffer rendered from the center of the light.

Our approach uses a regular grid, which, compared for example to a quadtree, has the important advantage of simple construction from pixel sample and shadow volume projections. Of course, the potential disadvatage of a regular grid is an inefficient modeling of non-uniform sampling. Our method starts out with a grid matching the 2-D AABB of pixel samples and then discards grid cells where no pixel sample projects. Consequently the grid adapts somewhat to a varying density of pixel samples. In our tests, the additional cost of a quadtree was not warranted. For extreme cases where grid cells get hot enough to hinder performance, a possible solution is to have secondary grid subdivision of the hot grid cells.

Whereas our method computes accurate soft shadows with excellent temporal stability, edges of blocking geometry exhibit some degree of temporal aliasing. The segments in the accompanying video were rendered with supersampling only for the perliminary rendering pass without shadows. This antialiases edges of blocking geometry incorrectly with background pixels whose shading hasn't been yet finalized. Of course, a straight forward solution is to use supersampling over the entire output frame, but supersampling the soft shadows is expensive and it seems unnecessary as soft shadows have low frequencies and are not causing a problem. We will investigate a method for integrating our soft shadow computation with antialiasing, which does not incurr the cost of supersampling the soft shadows by touching up fully lit pixels that neighbor pixels with a shadow value above a threshold.

Finally, our method leverages the coherence of light rays for a given output image sample by computing the sample's visibility mask in feed-forward fashion by projection followed by rasterization. Light rays are coherent as long as the light source is confined to a contiguous region in space. Therefore our method cannot efficiently support a set of disparate point light sources.

5. Conclusions and Future Work

We have presented a general and efficient algorithm for rendering shadows. The algorithm handles robustly fully dynamic scenes modeled with a collection triangles and renders soft shadows accurately. As the light source decreases in size, the algorithm converges to rendering pixel-accurate hard shadows, overcoming the traditional shadow map resolution challenge.

We have shown that a regular grid with a variable and unbounded number of pixel samples and blocking triangles per cell can now be implemented efficiently on graphics hardware and that, in the case of soft shadows, the cost of constructing and querying a hierarchical data structure is not warranted. The 2-D AABB approximation of the shadow volume we employ is conservative, and we have shown that the approximation is also tight. We have analyzed the benefit brought by a convex hull approximation of the projection of

the shadow volume and we have found that such a benefit is very low, substantially outweighed by the additional costs of convex hull construction and rasterization. We have compared our approach to ray tracing and we have shown that our approach has a substantial performance advantage. The visibility rays defined by output image pixel samples and light samples are very coherent, compared to, for example, the rays resulting from specular reflections off reflective surfaces in a scene. Consequently, our feed-forward approach of assigning triangles to pixel samples by projection followed by rasterization of shadow volumes and then of rasterizing blocking triangles onto bitmasks is efficient, and it outperforms the approach of hierarchical partitioning of scene geometry used in ray tracing.

In addition to the possible extensions discussed in Section 4.4, our method can be readily extended to support 2-D area light sources with complex shapes modeled with "transparent" light image pixels and colored shadows cast by transparent blockers [ME11]. Our method can also be extended to 3-D light sources, with two modifications. First, the shadow volume of a triangle needs to be conservatively approximated with eight and not just four rays per vertex, i.e. one ray for each of the corners of a box bounding the light. Second, the visibility map for a pixel should be large enough to encompass the bounding box of the light, and the 3-D light should be rasterized to the visibility map to mark the locations that actually sample the light.

Our paper makes an infrastructure contribution towards solving the general problem of visibility computation, which could prove useful in other contexts such as rendering participating media or occlusion culling for rendering acceleration.

6. Acknowledgements

We would like to thank Ze Wang for help with the implementation. This work is the part of Project 61272349, 61190121 and 61190125 supported by National Natural Science Foundation of China and Project 043/2009/A2 funded by Macao Science and Technology Development Fund, and is also supported by Beijing Science Technology Star Plans and Technology Star Plans (No. 2009B09), by the National High Technology Research and Development Program of China through 863 Program (No. 2013AA01A604).

References

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

[AL04] AILA T., LAINE S.: Alias-free shadow maps. *Proceedings of the Fifteenth Eurographics conference on Rendering Techniques* (2004), 161–166. 3

[AL09] AILA T., LAINE S.: Understanding the efficiency of ray traversal on gpus. *Proceedings of the Conference on High Performance Graphics 2009* (2009), 145–149. 11

[BS02] BRABEC S., SEIDEL H.: Single sample soft shadows using depth maps. *Graphics Interface* (2002), 219–228. 3

[BW09] BENTHIN.C., WALD I.: Efficient ray traced soft shadows using multi-frusta tracing. *Advances in Computer Graphics Hardware* (2009), 135–144. 3

[ED07] EISEMANN E., DCORET X.: Visibility sampling on gpu and applications. *Computer Graphics Forum 26*, 3 (2007), 535–544. 3, 7

[ESA11] EISEMANN E., SCHWARZ M., ASSARSSON U.: Real-time shadows. *CRC Press* (2011). 3

[FBGP09] FOREST V., BARTHE L., GUENNEBAUD G., PAULIN M.: Soft textured shadow volume. *Computer Graphics Forum 28*, 4 (2009), 1111–1120. 3

[FBP08] FOREST V., BARTHE L., PAULIN M.: Accurate shadows by depth complexity sampling. *Computer Graphics Forum 27*, 2 (2008), 663–674. 3

[Fer05] FERNANDO R.: Percentage-closer soft shadows. *ACM SIGGRAPH 2005 Sketches* (2005). 3

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

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

[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. 3

[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. 3

[JLBM05] JOHNSON G., LEE J., BURNS C., MARK W.: The irregular Z-buffer: Hardware acceleration for irregular data structures. *ACM Transactions on Graphics 24*, 4 (2005), 1462–1482. 3, 5

[JMB04] JOHNSON G., MARK W., BURNS C.: The irregular z-buffer and its application to shadow mapping. *Technical Report TR-04-09, Department of Computer Sciences, The University of Texas at Austin* (2004). 3, 5

[KA13] KARRAS T., AILA T.: Fast parallel construction of high-quality bounding volume hierarchies. *Proceedings of the 5th High-Performance Graphics Conference* (2013), 89–99. 11

[Kar12] KARRAS T.: Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics* (2012), 33–37. 11

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

[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. 3

[ME11] MCGUIRE M., ENDERTON E.: Colored stochastic shadow maps. *Proceedings of the ACM Symposium on Interactive 3D Graphics and Games* (2011), 89–96. 12

[MKHS10] MOHAMMADBAGHER M., KAUTZ J., HOLZSCHUCH N., SOLER C.: Screen-space percentage-closer soft shadows. *ACM SIGGRAPH 2010 Posters* (2010), 1–1. 3

[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. 3

[SEA08] SINTORN E., EISEMANN E., ASSARSSON U.: Sample-based visibility for soft shadows using alias-free shadowmaps. *Computer Graphics Forum 27*, 4 (2008), 1285–1292. 3, 7, 9

[SGYF11] SHEN L., GUENNEBAUD G., YANG G., FENG J.: Predicted virtual soft shadow maps with high quality filtering. *Computer Graphics Forum 30*, 2 (2011), 493–502. 3

[SS07] SCHWARZ M., STAMMINGER M.: Bitmask soft shadows. *Computer Graphics Forum 26*, 3 (2007), 515–524. 3

[SS08] SCHWARZ M., STAMMINGER M.: Microquad soft shadow mapping revisiteds. *Eurographics 2008 Annex to the Conference Proceedings: Short Papers* (2008), 295–298. 3

[Whi79] WHITTED T.: An improved illumination model for shaded display. *Communications of The ACM 23*, 6 (1979), 343–349. 3

[YDF*10] YANG B., DONG Z., FENG J., SEIDEL H., KAUTZ J.: Variance Soft Shadow Mapping. *Computer Graphics Forum 29*, 7 (2010), 2127–2134. 3

[YFGL09] YANG B., FENG J., GUENNEBAUD G., LIU X.: Packet-based hierarchal soft shadow mapping. *Computer Graphics Forum 28*, 4 (2009), 1121–1130. 3

[ZM10] ZHANG W., MAJDANDZIC I.: Fast triangle rasterization using irregular z-buffer on cuda. *Master of Science Thesis in the Programme of Integrated Electronic System Design, Chalmers University of Technology* (2010). 3, 5