# GEARS: A General and Efficient Algorithm for Rendering Shadows

Lili Wang, Ze Wang, Yulong Shi and Voicu Popescu

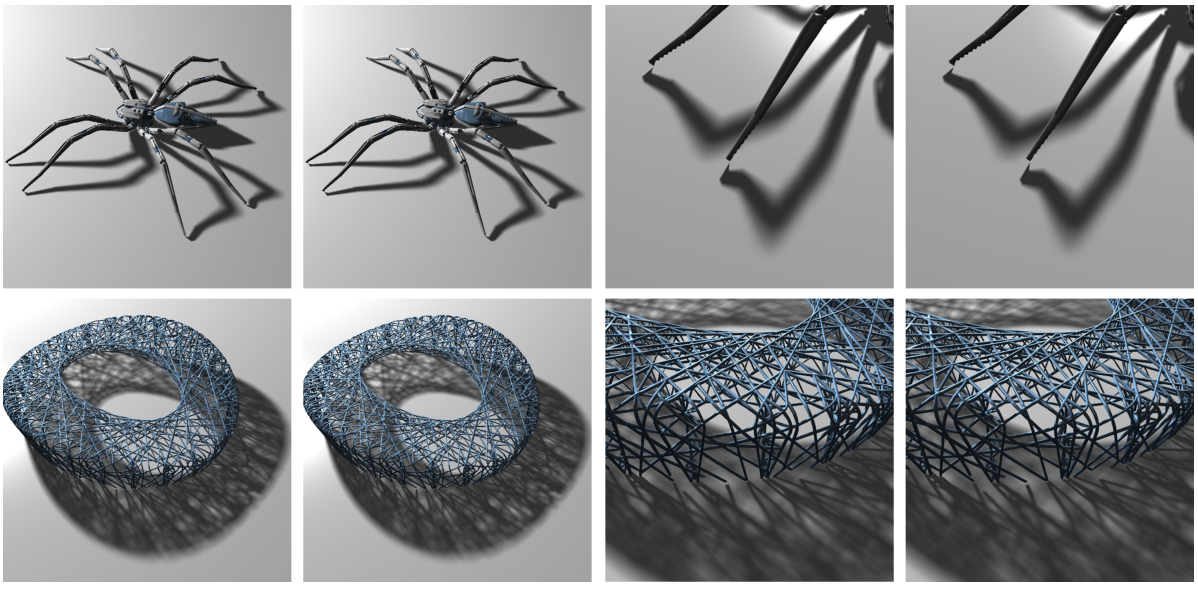


Fig. 1: *Soft shadows rendered with our method (left for each pair) and with ray tracing (right for each pair), for comparison. The average frame rate for our method vs. ray tracing is 15.7fps vs. 0.5fps for the Bird Nest scene and 32.7fps vs. 2.7fps for the Spider scene.*

**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. The algorithm computes for each output image pixel 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.

**Index Terms**—Soft shadows, interactive rendering, hard shadows.

## 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 uniformly with $k$x$k$ light samples, and an output image of resolution $w$x$h$. The challenge is to compute whether there is a direct line of sight from each of the $k$x$k$ light samples to each of the $w$x$h$ pixel samples.

One approach is to trace $k$x$k$ rays from each of the $w$x$h$ pixels. Considering all $w$x$h$ x $k$x$k$ x $N$ possible ray-triangle pairs is prohibitively expensive and an acceleration scheme is needed to avoid considering most ray-triangle pairs that do not yield an intersection. Another approach is to render the scene $k$x$k$ times to obtain $k$x$k$ conventional shadow maps that allow approximating visibility from light samples to pixel samples. 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,


- *L. Wang, Z. Wang and Y. Shi are with the State Key Laboratory of Virtual Reality Technology and Systems, School of Computer Science and Engineering, Beihang University, Beijing China, 100191. E-mail: wanglily@buaa.edu.cn*
- *V. Popescu is with the Computer Science Department, Purdue University, West Lafayette, Indiana, USA, IN 47907. E-mail: popescu@purdue.edu*

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 might block the light as seen from $P$. The set of triangles rendered for a pixel sample is conservative, in the sense that it contains all triangles that block the light. 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. The projection of a shadow volume of a triangle is approximated conservatively, which ensures that a pixel sample is assigned all triangles that block the light as seen from the pixel sample. Our method is (1) accurate, (2) efficient, and (3) general. We also refer the reader to the accompanying video.

(1) Our method is accurate in the sense that it accurately assesses visibility from each pixel sample to each light sample. All approximations employed are conservative. 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 visibility bit mask resolution (i.e. 16x16).

(2) Our method is efficient because the approximations employed are not only conservative, but they are also tight and fast to compute. 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. This avoids the cost of constructing and using a hierarchical data structure (e.g. kd-tree) to model the non-uniform distribution of pixel sample and blocking triangle data. 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 irregular z-buffer method for rendering hard shadows [1], [2], [3], 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 [4], [5]. In this brief overview, we distinguish between image space and geometry space methods. Image space methods, such as shadow map methods [6], [7], [8], 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 [9], [10], [11]. 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 [12], [13], [9]. For example soft shadows can be simulated by blurring hard shadow edges [14], [15]. 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. [6] 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 [16]. Schwarz and Stamminger [17] 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. [18] 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 [19], 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 [20] naturally supports accurate soft shadows by tracing rays between pixel samples and light points, but peformance is a concern. Laine and Aila [21] 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 [22]. 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. 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. Other limitations of the method include limited scalability with light source area, and poor performance for fragmented meshes, when virtually all triangle edges are silhouette edges. Forest et al. [23] 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 [24] 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. [10] 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 [25] 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. [26] propose alias-free shadow maps (AFSM), a method for rendering accurate hard and soft shadows which, like our method, takes the approach of 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 Algorithm Overview

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 with the following approach:

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 $v_p$.

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

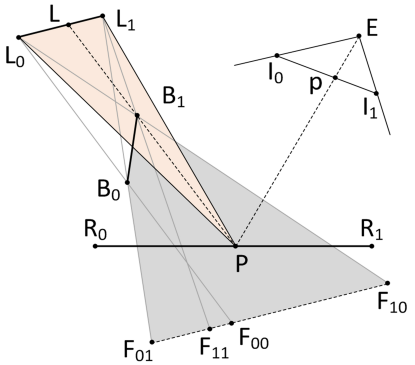



Fig. 2: *Soft shadow computation overview for light $L_0L_1$, output image $I_0I_1$ with viewpoint E, blocker $B_0B_1$, and receiver $R_0R_1$.*

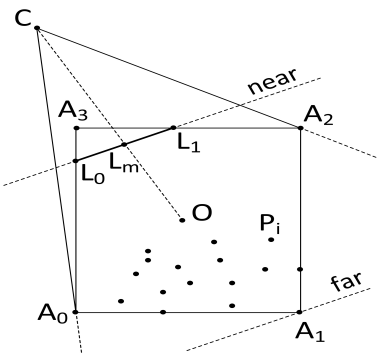



Fig. 3: *Camera used to define and populate grid.*

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 section. 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.

We compute for every pixel sample a conservative but tight approximation of the set of triangles that block the light with the following acceleration scheme.

## 4 Acceleration Scheme

The acceleration scheme finds a superset of the triangles that block the light as seen from each pixel's 3-D sample point. We target fully dynamic scenes where blocker and receiver geometry can move and deform, and where the light rectangle position, orientation, size, and aspect ratio can change from frame to frame. For this, we have developed an acceleration scheme that runs from scratch at the beginning of every frame.

Given a scene $S$, a rectangular area light source $L_0L_1L_2L_3$, and the pixel samples $P$ of the output image, the acceleration scheme computes a regular 2-D grid $G$ that stores at each cell $(u, v)$ a set of pixel samples $P_{uv}$ and a set of potentially blocking triangles $T_{uv}$ for the pixel samples in $P_{uv}$. The acceleration scheme proceeds as follows:

A.1. Construct camera $C$ with grid $G$ as image plane.
A.2. For each pixel sample $P$
  a. Project $P$ with $C$ to $P'$.
  b. $P_{uv} = P_{uv} \cup P$, where $P' \in G(u, v)$.
A.3. For each triangle $T$ in $S$
  a. For each vertex $B_i$ of $T$ and each light vertex $L_j$
    i. Compute 3-D points $F_{ij} = B_i + \| B_i\text{-}L_j \| d_{ij}$.
  b. Project vertices $B_i$ and points $F_{ij}$ with C and compute the 2-D AABB of the projections.
  c. For each $G(u, v)$ touched by the AABB
    i. $T_{uv} = T_{uv} \cup T$.

(A.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 $P$, see $A_0A_1A_2A_3$ in Fig. 3.

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 the rays of camera $C$ approximate the light rays well. Whereas the pinhole $C$ cannot approximate the light rays with high fidelity for a large area light source, 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 [1], [2], [3].

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, blocking triangles are small, which makes that 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

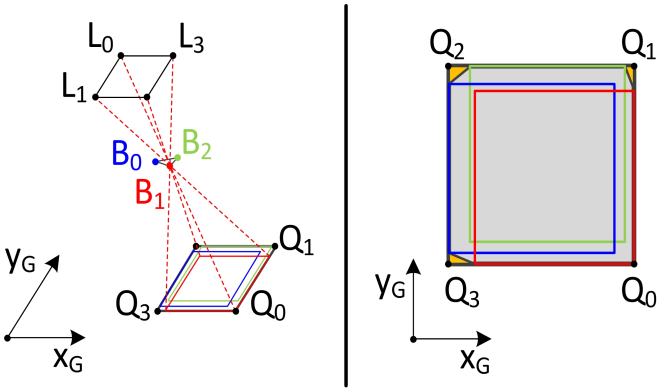



Fig. 4: *Left: projection of shadow volume of triangle $B_0B_1B_2$ onto 2-D grid with axes $x_G$ and $y_G$. Right: 2-D AABB $Q_0Q_1Q_2Q_3$ is a tight approximation of the shadow volume projection.*

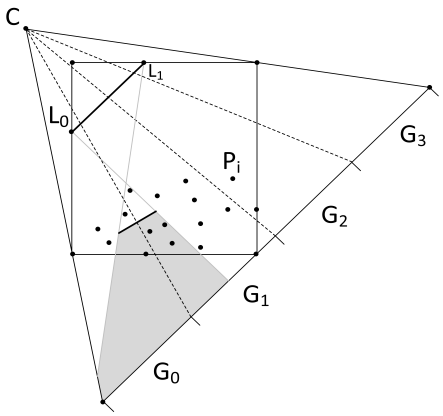



Fig. 5: *Pixel sample and triangle assignment to grid.*

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.

(A.2) Step 2 assigns pixel samples to grid cells. 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.

(A.3) Step 3 assigns potentially blocking triangles to grid cells. For this, the current triangle $T$ is first extruded by computing a 3-D point for each triangle vertex/light vertex pair, a total of 3 x 4 = 12 points $F_{ij}$ (Step 3.a). Point $F_{ij}$ is on the line connecting triangle vertex $B_i$ and light vertex $L_j$, at a distance $d_{ij}$ from $B_i$ and away from the light; $d_{ij}$ is chosen such that $F_{ij}$ lie on 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 $F_{10}$. At Step 3.b, the 3 vertices $B_i$ and the 12 points $F_{ij}$ are projected with $C$ and the 2-D AABB of the 15 projections is computed. Finally, $T$ is assigned to all grid cells touched by the AABB (Step 3.c). In Fig. 5 the triangle whose shadow volume is shown shaded in grey is assigned to grid cells $G_0$ and $G_1$. The shadow volume of a triangle is by definition the union of all lines connecting a point inside the triangle to a point inside the light rectangle, clipped by the triangle plane to the half space away from the light. Step 3 does not compute an approximation of the triangle's shadow volume, but rather an approximation of the projection of the shadow volume. The approximation is conservative, see proof in Appendix.

As a result of the algorithm above, a pixel sample is assigned to exactly one grid cell. Not all triangles are assigned to a cell, as not all are blocking triangles. A triangle can be assigned to more than one grid cell, as the same triangle can block the light for two or more pixel samples stored at different cells. All pixel samples in a cell are assigned the same set of potentially blocking triangles.

## 5 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.

### 5.1 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). A quantiative analysis of the difference between an image rendered with our method and one rendered with ray tracing revealed identical pixel values except for 3 pixels whose intensity differed by 1, which we attribute to numerical precision differences between the two methods.

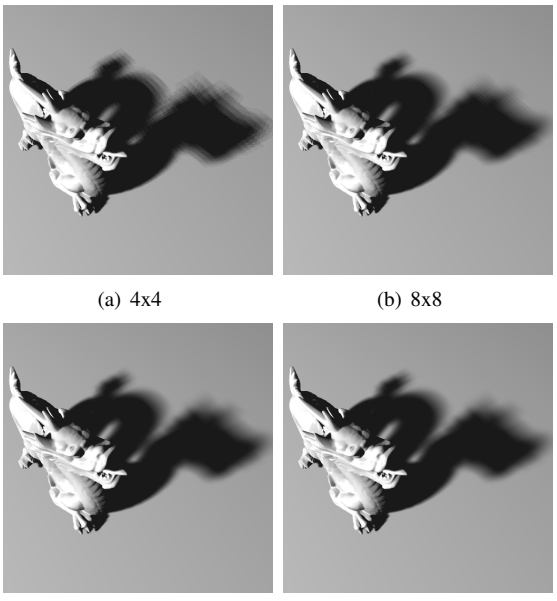
The only approximation made by our method that influences quality is the resolution of the visibility masks (Fig. 6). Whereas a resolution of 4x4 is insufficient, 8x8 produces good results, and there is virtually no improvement beyond 16x16.

### 5.2 Implementation overview

Referring back to the algorithm overview given in Section 3, step 1 (rendering the scene preliminarily, without shadows) and step 2 (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 4.

Steps A.1 and A.2 are executed simultaneously: camera $C$ is constructed, the pixel samples stored in the floating point texture are projected onto the image plane of $C$, the grid $G$ is defined to cover the 2-D AABB of the projections, and pixel samples are assigned to the cells of $G$ where they project. The resolution of $G$ is an input parameter. Step A.3 is executed by first computing the grid cells touched by the projection of the shadow volume of each



(a) 4x4 (b) 8x8

(c) 16x16 (d) 32x32

Fig. 6: Quality dependence on resolution of visibility masks.

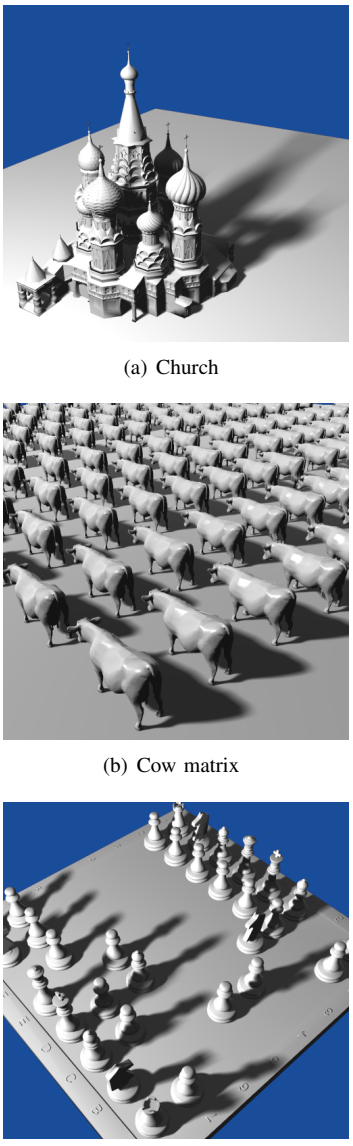
triangle. The number of grid cells touched by each triangle is recorded in an array $A$. For each triangle and for each cell covered by the triangle, a two-tuple $(triID, cellID)$ is stored using the prefix sum of array $A$ as offset. The prefix sum allows storing the mapping from triangles to cells contiguously. The mapping is inverted by radix sorting the tuples based on *cellIDs*. The resulting mapping from cells to triangles, combined with the mapping from cells to pixel samples computed at Step A.2, effectively assigns potentially blocking triangles to pixels. This completes Step 3 of the algorithm (Section 3).

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. 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 A.3), and the single shadow bit is known for each pixel sample after Step 3 (i.e. Steps 4 and 5 are not necessary).

### 5.3 Performance

We tested our technique on several scenes: *Spider* (41K triangles, Fig. 1, top), *BirdNest* (165K triangles, Fig. 1, bottom), *Church* (74K triangles, Fig. 7a), *CowMatrix* (786K triangles, Fig. 7b), *Chess* (201K triangles, Fig. 7c), and *Dragon* (81K triangles, Fig. 8). All performance measurements reported in this paper were recorded on a 3.4GHz Intel(R) Core(TM) i7-2600 CPU PC with 4 GB of RAM and an NVIDIA GeForce GTX 580 graphics card.



(a) Church

(b) Cow matrix

(c) Chess

Fig. 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. Performance is most sensitive with output image resolution for the *BirdNest* and the *Spider* scenes which have many silhouette edges, and consequently the relative complexity of the penumbra regions decreases less with output resolution.

Table 2 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. The rasterization cost is relatively small, as indicated by the relatively small performance penalty for an 8x8 increase of rasterization resolution.

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

| **Output res.** | 256 x256 | 512 x512 | 1024 x1024 | 1280 x1280 |
|---|---|---|---|---|
| *Cow Matrix* | 33.5 | 24.0 | 12.5 | 6.60 |
| *Chess* | 29.1 | 19.7 | 10.3 | 6.10 |
| *Church* | 32.2 | 20.1 | 12.0 | 6.50 |
| *Dragon* | 51.6 | 24.3 | 13.5 | 7.43 |
| *Bird Nest* | 34.9 | 15.7 | 4.6 | 3.16 |
| *Spider* | 60.8 | 32.7 | 14.4 | 11.0 |

TABLE 2: Frame rate [fps] for various visibility mask resolutions.

| **Bit mask res.** | 4x4 | 8x8 | 16x16 | 32x32 |
|---|---|---|---|---|
| *Cow Matrix* | 27.6 | 26.2 | 24 | 20.1 |
| *Chess* | 31.1 | 23.7 | 19.7 | 16.2 |
| *Church* | 34.5 | 26.3 | 20.1 | 15.7 |
| *Dragon* | 36.9 | 32.7 | 24.3 | 16.6 |
| *Bird Nest* | 23.4 | 18.9 | 15.7 | 8.9 |
| *Spider* | 45.8 | 40.3 | 32.7 | 18.6 |

Table 3 gives the average frame rate for our test scenes for various light source sizes. 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.

TABLE 3: Frame rate [fps] for various light source sizes.

| **Light diagonal** | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| *Cow Matrix* | 27.2 | 24 | 21.4 | 19.1 | 17.2 |
| *Chess* | 24.3 | 19.7 | 15.3 | 12.4 | 7.8 |
| *Church* | 31.2 | 20.1 | 15.9 | 10.1 | 7.6 |
| *Dragon* | 35.7 | 24.3 | 18.4 | 12.8 | 9.3 |
| *Bird Nest* | 20.1 | 15.7 | 9.8 | 7.5 | 5.2 |
| *Spider* | 48.7 | 32.7 | 24.3 | 20 | 16.8 |

Table 4 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 4: Frame rate [fps] for various grid resolutions.

| **Grid res.** | 256x256 | 128x128 | 64x64 | 32x32 |
|---|---|---|---|---|
| *Cow Matrix* | 21.1 | 24 | 19.7 | 12.4 |
| *Chess* | 18.2 | 19.7 | 17.2 | 15.3 |
| *Church* | 18.6 | 20.1 | 19.1 | 15.2 |
| *Dragon* | 21.4 | 24.3 | 22.1 | 17.5 |
| *Bird Nest* | 13.1 | 15.7 | 12.2 | 8.7 |
| *Spider* | 29.6 | 32.7 | 28.3 | 23.8 |

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

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

Fig. 8: Soft shadows with various light source sizes.

**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 5. The grid resolution is 128x128, and output resolution is 512x512. The pixel sample to grid assignment is visualized in Fig. 9. 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, 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 5 also reports typical maximum and average number of triangles per grid cell.

Finally we have measured the performance of the grid as a tool for assigning blocking triangles to pixel samples.

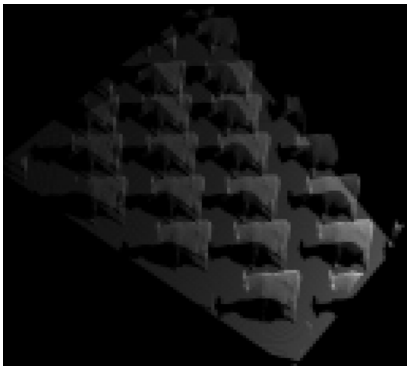


Fig. 9: *Visualization of pixel sample distribution over grid.*

TABLE 5: Number of triangles and of pixel samples per grid cell.

| Scene | Triangles | | Pixel Samples | |
|---|---|---|---|---|
| | Max | Average | Max | Average |
| *Cow Matrix* | 539 | 70 | 278 | 16 |
| *Chess* | 1333 | 54 | 103 | 11 |
| *Church* | 2121 | 54 | 146 | 9 |
| *Dragon* | 2844 | 60 | 283 | 16 |
| *Bird Nest* | 501 | 25 | 243 | 16 |
| *Spider* | 506 | 9 | 244 | 16 |

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 6 shows that at least 11% and as many as 27% of the potentially blocking triangles found using the grid are actually blocking triangles.

TABLE 6: Triangle to pixel sample assignments [x1,000].

| Scene | Necessary | Total | Percentage |
|---|---|---|---|
| *Cow Matrix* | 2,523 | 22,435 | 11% |
| *Chess* | 6,249 | 44,636 | 14% |
| *Church* | 11,554 | 109,748 | 11% |
| *Dragon* | 19,139 | 91,007 | 21% |
| *Bird Nest* | 9,562 | 35,313 | 27% |
| *Spider* | 1,673 | 9,260 | 18% |

Our method overestimates the set of blocking triangles for a pixel sample because of three approximations. 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 4 (Fig. 5), 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 A.3 in Section 4). We have found that, on average, the 2-D AABB only covers an additional 1.75%, 1.13%, 1.5%, 0.31%, 2.2%, and 0.96% pixel samples for the Cow Matrix, Chess, Church, Dragon, Bird Nest, and Spider scenes, respectively. Fig. 10 shows examples of actual projections of shadow volumes and highlights the small difference between the 2-D AABB and the 2-D convex hull. We conclude that the convex hull brings virtually no benefit, which makes the cost of computing and rasterizing the convex hull unwarranted.

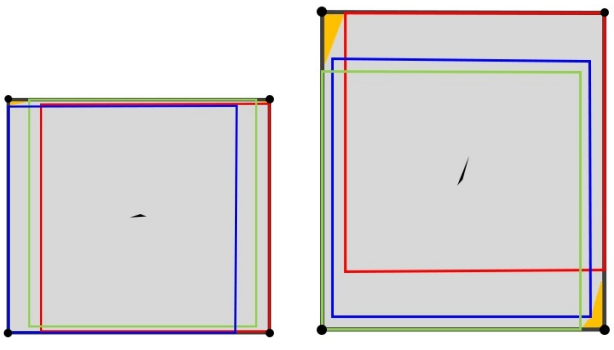


Fig. 10: Visualizations of actual projections of triangle shadow volumes onto grid plane. The difference (orange) between the convex hull (grey) and the AABB are small, as predicted by Fig. 4.

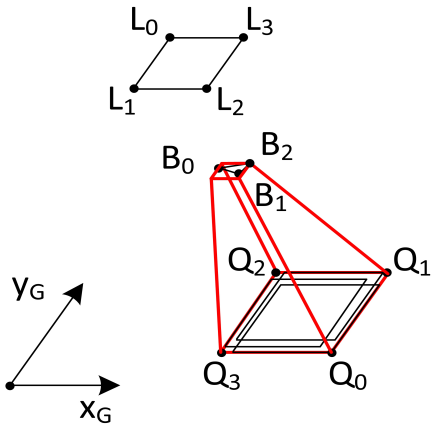



Fig. 11: *Triangle shadow volume approximated conservatively with a frustum (red).*

The second approximation is that the triangle to grid cell assignment is computed in 2-D and not in 3-D. It can happen that even though the projection of a triangle shadow volume intersects a grid cell, there are no grid cell pixel samples that are actually inside the shadow volume. In order to quantify the impact of this approximation we have developed a conservative 3-D approximation of the triangle shadow volume as shown in Fig. 11, which uses the same notations as Fig. 4.

The shadow volume is inscribed into a frustum with 6 faces. The bottom face $Q_0Q_1Q_2Q_3$ corresponds to the 2-D AABB of the projection onto the grid. The top face is a 2-D AABB of the projection of the triangle $B_0B_1B_2$ onto a plane parallel to the grid and that passes through the vertex closest to the light, here $B_2$. The edges of the top and bottom faces are pair-wise parallel so the side faces of the frustum are planar. The frustum is a convex polyhedron and it is straight forward to decide whether a 3-D pixel sample is inside the frustum or not. By not assigning a triangle to a grid cell if none of the grid cell pixel samples are inside the frustum approximation of the shadow volume of the triangle, the percentages in the last column of Table 6 become 28, 39, 38, 42, 39, and 28%, for each of the six scenes, respectively. Although the triangle to pixel sample assignment approximation becomes tighter, the frame rate decreases, which indicates that the cost of the 3-D approximation of the shadow volume outweighs its benefit. Therefore, at least in the context of our implementation, the 3-D approximation of the shadow volume remains just a tool for investigating grid efficiency.

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 *CowMatrix* scene the percentage of necessary assignments increases from 11% to 22% if the grid resolution increases from 128x128 to 512x512. However, as indicated earlier (Table 4), the best performance is obtained for a 128x128 grid.

Compared to the brute force approach of rendering all triangles for each of 512x512 output image pixels, our method renders 512x512x786K/22,435K=9,184 times fewer triangles for the *CowMatrix* example in Table 6. Compared to the brute force approach of rendering a shadow map for each of 32x32 light points, our method renders 32x32x768K/22,435K=35 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) to that of NVIDIA's Optix ray tracer; Table 7 shows that frame rates for our algorithm are between 7 and 31 times higher for the same bitmask resolution (i.e. for same shadow quality). This is for static scenes, when the ray tracing's acceleration data structure does not have to be rebuilt, and is thus not included in the frame rate. For dynamic scenes, our performance remains the same, whereas the peformance of ray tracing degrades further.

TABLE 7: 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) |
|---|---|---|---|---|---|
| *Cow Matrix* | 24.0 | 1.70 | 14 | 1.29 | 18 |
| *Chess* | 19.7 | 1.79 | 11 | 1.46 | 14 |
| *Church* | 20.1 | 2.10 | 10 | 1.75 | 12 |
| *Dragon* | 24.3 | 2.56 | 10 | 1.90 | 12 |
| *Bird Nest* | 15.7 | 0.50 | 31 | 0.46 | 34 |
| *Spider* | 32.7 | 2.70 | 12 | 1.80 | 18 |

Whereas Table 8 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. This results in noticeable artifacts as shown in Fig. 12.

TABLE 8: 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 |
|---|---|---|---|
| *Cow Matrix* | 8.3 | 32x32 = 1,024 | 48 |
| *Chess* | 10.5 | 32x32 = 1,024 | 50 |
| *Church* | 11.0 | 32x32 = 1,024 | 90 |
| *Dragon* | 10.0 | 32x32 = 1,024 | 72 |
| *Bird Nest* | 15.7 | 16x16 = 256 | 12 |
| *Spider* | 32.7 | 16x16 = 256 | 18 |

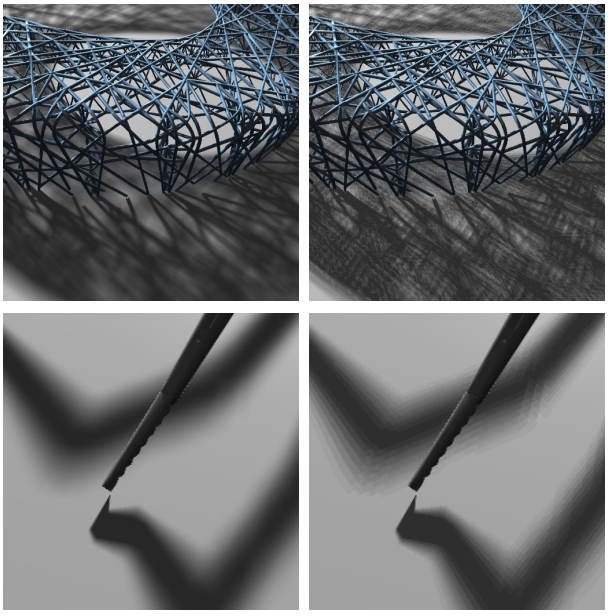


Fig. 12: Quality comparison between GEARS (left) and ray tracing (right) for equal performance.

## 5.4 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 as 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.

## 6 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. We have proven that 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 the Limitations subsection, 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 [27]. 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.

## Appendix A

Given a rectangular light source $L_0L_1L_2L_3$, a triangle with vertices $B_0B_1B_2$, and a camera with eye $C$ and far plane yon also serving as its image plane, let $L$ be a point in $L_0L_1L_2L_3$ and $B$ a point in $B_0B_1B_2$. Let $F_B$ be the intersection of $LB$ with yon, and let $F_{ij}$ be the intersection of $L_jB_i$ with yon ($i$ from 0 to 2 and $j$ from 0 to 3). Furthermore let $F_{CB}$ be the intersection of $CB$ with yon, and $F_{Ci}$ be the intersection of $CB_i$ with yon ($i$ from 0 to 2).

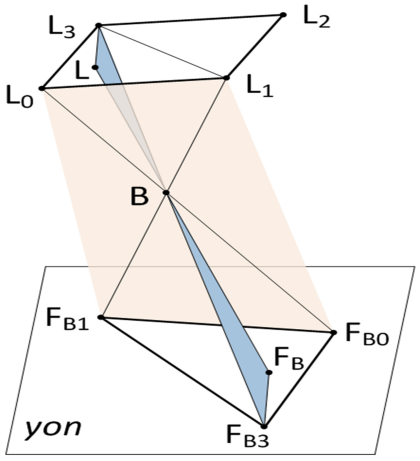



Fig. A1: *Illustration of proof that $F_B$ belongs to $F_{B0}F_{B1}F_{B3}$.*

Then segment $F_BF_{CB}$ is contained in the 2-D axis aligned bounding box $AABB$ of the 15 points $F_{ij}$ and $F_{Ci}$ ($i$ from 0 to 2 and $j$ from 0 to 3).

**Proof**

Since $AABB$ is convex, it is sufficient to show that (1) $AABB$ contains $F_B$ and that (2) $AABB$ contains $F_{CB}$.

(1) Without loss of generality, assume that $L$ is inside triangle $L_0L_1L_3$ (Fig. A1). Let $F_{B0}$, $F_{B1}$, and $F_{B3}$ be the intersection of $L_0B$, $L_1B$, and $L_3B$ with yon, respectively.

We first show that $F_B$ is inside triangle $F_{B0}F_{B1}F_{B3}$. $L$ is inside triangle $L_0L_1L_3$ so $L$ is on the same side of $L_0L_1$ as $L_3$. Consequently $L$ and $L_3$ are on the same side of plane $L_0L_1F_{B0}F_{B1}$ (orange in Fig. A1). Both lines $LF_B$ and $L_3F_{B3}$ intersect plane $L_0L_1F_{B0}F_{B1}$ at $B$, thus $F_B$ and $F_{B3}$ are on the

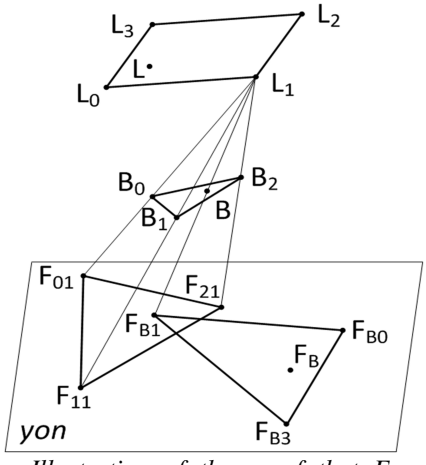



Fig. A2: *Illustration of the proof that $F_{B1}$ belongs to $F_{01}F_{11}F_{21}$.*

same side of plane $L_0L_1F_{B0}F_{B1}$, namely the side opposite to the side where $L$ and $L_3$ are. Consequently $F_B$ and $F_{B3}$ are on the same side of $F_{B0}F_{B1}$ in triangle $F_{B0}F_{B1}F_{B3}$. Similarly it can be shown that $F_B$ and $F_{B0}$ are on the same side of $F_{B1}F_B$ and that $F_B$ and $F_{B1}$ are on the same side of $F_{B3}F_{B0}$. Consequently $F_B$ is inside triangle $F_{B0}F_{B1}F_{B3}$.

Now we show that triangle $F_{B0}F_{B1}F_{B3}$ is inside $AABB$. Since $B$ is inside triangle $B_0B_1B_2$, $F_{B1}$ is inside the pyramid with apex $L_1$ and base $B_0B_1B_2$ (Fig. A2). Consequently $F_{B1}$ is inside triangle $F_{01}F_{11}F_{21}$, and thus inside $AABB$. Similarly it can be shown that $F_{B0}$ and $F_{B3}$ are inside $AABB$ and consequently triangle $F_{B0}F_{B1}F_{B3}$ is inside $AABB$.

(2) Since $B$ is inside triangle $B_0B_1B_2$, $F_{CB}$ is inside the triangle $F_{C0}F_{C1}F_{C2}$, which is the image of triangle $B_0B_1B_2$ (i.e. the yon plane projection of $B_0B_1B_2$ from eye $C$). Since $F_{C0}$, $F_{C1}$, and $F_{C2}$ are part of $AABB$, $F_{CB}$ is part of $AABB$.

This concludes the proof that our approximation of the shadow volume projection is conservative.

## ACKNOWLEDGMENTS

We would like to thank Qijiang Jin for help with the implementation. This work was supported in part by the National Natural Science Foundation of China through Projects 61272349 and 61190121, by the Macao Science and Technology Development Fund through Project 043/2009/A2, by the National High Technology Research and Development Program of China through 863 Program NO. 2011AA010502, and through Beijing Science Technology Star Plan No. 2009B09.

## REFERENCES

[1] G.S. Johnson, J. Lee, C.A. Burns and W.R. Mark, "The Irregular Z-Buffer: Hardware Acceleration for Irregular Data Structures," *ACM Transactions on Graphics*, vol. 24, no. 4, pp. 1462-1482, 2005.

[2] G.S. Johnson, W.R. Mark and C.A. Burns, "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] W. Zhang, "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, 2004.

[4] E. Eisemann, U. Assarsson, M. Schwarz and M. Wimmer, "Casting Shadows in Real Time," *SIGGRAPH Asia 2009 course*, 2009.

[5] J.M. Hasenfratz, M. Lapierre, N. Holzschuch and F.X. Sillion, "A Survey of Real-time Soft Shadows Algorithms," *Computer Graphics Forum*, vol. 22, no. 4, pp. 753-774, 2003.

[6] G. Guennebaud, L. Barthe and M. Paulin, "Real-time Soft Shadow Mapping by Backprojection," *In Eurographics Symposium on Rendering*, pp. 227-234, 2006.

[7] B.G Yang, Z. Dong, J.Q Feng, H.P Seidel and J. Kautz, "Variance Soft Shadow Mapping," *Computer Graphics Forum*, vol. 29, no. 7, pp. 2127-2134, 2010.

[8] L. Shen, G. Guennebaud, B.G. Yang and J.Q Feng, "Predicted Virtual Soft Shadow Maps with High Quality Filtering," *Computer Graphics Forum*, vol. 30, no. 2, pp. 493-502, 2011.

[9] T.A. Moller and U. Assarsson, "Approximate Soft Shadows on Arbitrary Surfaces using Penumbra Wedges," *Eurographics Symposium on Rendering/Eurographics Workshop on Rendering Techniques*, pp. 297-305, Jun. 2002.

[10] G.S. Johnson, W.A. Hunt, A. Hux, W.R. Mark, C.A. Burns and S. Junkins, "Soft Irregular Shadow Mapping: Fast, High-Quality, and Robust Soft Shadows," *ACM Symposium on interactive 3D graphics*, pp. 57-66, 2009.

[11] V. Forest, L. Barthe, G. Guennebaud and M. Paulin, "Soft Textured Shadow Volume," *Computer Graphics Forum*, vol. 28, no. 4, pp. 1111-1120, 2009.

[12] S. Brabec and H.P. Seidel, "Single Sample Soft Shadows Using Depth Maps," *Graphics Interface*, pp. 219-228, May 2002.

[13] J. Arvo, M. Hirvikorpi and J. Tyystjrvi, "Approximate Soft Shadows with an Image-Space Flood-Fill Algorithm," *Computer Graphics Forum*, vol. 23, no. 3, pp. 271-280, 2004.

[14] R. Fernando, "Percentage-closer soft shadows," *ACM SIGGRAPH 2005 Sketches*, Jul. 2005.

[15] M. MohammadBagher, J. Kautz, N. Holzschuch and C. Soler, "Screen-space Percentage-Closer Soft Shadows," *ACM SIGGRAPH 2010 Posters*, pp. 2009-2009, 2010.

[16] G. Guennebaud, L. Barthe and M. Paulin, "High-Quality Adaptive Soft Shadow Mapping," *Computer Graphics Forum*, vol. 26, no. 3, pp. 525-533, 2007.

[17] M. Schwarz and M. Stamminger, "Microquad Soft Shadow Mapping Revisited," *Eurographics 2008 Annex to the Conference Proceedings: Short Papers*, pp. 295-298, 2008.

[18] B.G Yang, J.Q. Feng, G. Guennebaud and X.G Liu, "Packet-based Hierarchal Soft Shadow Mapping," *Computer Graphics Forum*, vol. 28, no. 4, pp. 1121-1130, 2009.

[19] M. Schwarz and M. Stamminger, "Bitmask Soft Shadows," *Computer Graphics Forum*, vol. 26, no. 3, pp. 515-524, 2007.

[20] T. Whitted, "An Improved Illumination Model for Shaded Display," *Communications of The ACM*, vol. 23, no. 6, pp. 343-349, 1979.

[21] S. Laine and T. Aila, "Hierarchical Penumbra Casting," *Computer Graphics Forum*, vol. 24, no. 3, pp. 313-322, 2005.

[22] S. Laine, T. Aila, U. Assarsson, J. Lehtinen and T. Akenine-Moller, "Soft Shadow Volumes for Ray Tracing," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 1156-1165, 2005.

[23] V. Forest, L. Barthe and M. Paulin, "Accurate Shadows by Depth Complexity Sampling," *Computer Graphics Forum*, vol. 27, no. 2, pp. 663-674, 2008.

[24] E. Eisemann and X. Dcoret, "Visibility Sampling on GPU and Applications," *Computer Graphics Forum*, vol. 26, no. 3, pp. 535-544, 2007.

[25] C. Benthin and I. Wald, "Efficient ray traced soft shadows using multi-frusta tracing," *Advances in Computer Graphics Hardware*, pp. 135-144, 2009.

[26] E. Sintorn, E. Eisemann and U. Assarsson, "Sample-based Visibility for Soft Shadows Using Alias-free ShadowMaps," *Computer Graphics Forum*, vol. 27, no. 4, pp. 1285-1292, 2008.

[27] M. McGuire and E. Enderton, "Colored Stochastic Shadow Maps," *Proceedings of the ACM Symposium on Interactive 3D Graphics and Games 2011*, pp. 89-96, 2011.