Soft Shadow Volumes for Ray Tracing

Samuli Laine, Timo Aila, Ulf Assarsson, Jaakko Lehtinen, Tomas Akenine-Möller

Soft Shadow Volumes for Ray Tracing

Samuli Laine*
Helsinki University of Technology
Hybrid Graphics Ltd.

Timo Aila*
Helsinki University of Technology
Hybrid Graphics Ltd.

Ulf Assarsson†
ARTIS, GRAVIR/IMAG INRIA
Chalmers University of Technology

Jaakko Lehtinen*
Helsinki University of Technology
Remedy Entertainment Ltd.

Tomas Akenine-Möller‡
Lund University

Abstract

We present a new, fast algorithm for rendering physically-based soft shadows in ray tracing-based renderers. Our method replaces the hundreds of shadow rays commonly used in stochastic ray tracers with a single shadow ray and a local reconstruction of the visibility function. Compared to tracing the shadow rays, our algorithm produces exactly the same image while executing one to two orders of magnitude faster in the test scenes used. Our first contribution is a two-stage method for quickly determining the silhouette edges that overlap an area light source, as seen from the point to be shaded. Secondly, we show that these partial silhouettes of occluders, along with a single shadow ray, are sufficient for reconstructing the visibility function between the point and the light source.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Shadowing I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Visible line/surface algorithms

Keywords: shadow algorithms, visibility determination

Figure 1: This image was generated using 200 light samples per primary ray. Compared to an optimized ray tracer, our method reduced the shadow computation time from over 30 minutes to less than 2.5 minutes without compromising the image quality.

1 Introduction

This paper describes a new soft shadow volume algorithm that generalizes the earlier penumbra wedge-based methods [Akenine-Möller and Assarsson 2002; Assarsson and Akenine-Möller 2003] to produce physically correct shadows from planar area light sources. The algorithm is targeted to production-quality ray tracers, and generates the same image as stochastic ray tracing [Cook et al. 1984], but substantially faster (Figure 1). Physically-based soft shadows are often avoided in production work due to the prohibitive cost of casting the shadow rays. Instead, crude approximations such as filtered shadow maps are commonly used. We believe the speedups offered by our algorithm enable the practical use of physically-based soft shadows in a wide range of scenes.

The fundamental operation in soft shadow generation is determining the visibility between an area light source and a point, **p**, to be shaded. The operation is illustrated in Figure 2. The *visibility function* indicates, for every position on the surface of a light source, whether or not the sightline from **p** is blocked. Although possible in polygonal scenes, an analytically computed visibility function is usually not worth the computational cost. In realistic rendering, the visible parts of a light source are integrated against a BRDF. This operation often requires Monte Carlo integration, and thus the values of the visibility function are needed only at certain points.

Stochastic ray tracing approximates an area light source with a set of light samples. The required number of samples depends on many parameters, e.g., the size of a light source, size of the penumbra regions in output images, the dynamic range, the frequency content of textures, and the complexity of the visibility function. In our test scenes, 150–1000 samples were enough for obtaining visually pleasing smooth shadows.

The *visibility events* mark all potential changes in the visibility function, and can only occur on the silhouette edges of an opaque polygonal occluder. A *penumbra wedge* is the bounding volume of the penumbra region defined by a silhouette edge. Thus the shadow term of **p** can be affected by an edge only if **p** is inside the corresponding wedge. Throughout the paper, we concentrate on a single area light source.

As our first contribution, we introduce a two-stage method for quickly determining the set of silhouette edges whose projection from **p** to the plane of an area light source overlaps with the light. The first stage constructs a penumbra wedge for all edges that are silhouette edges from at least one point on the light source, and stores the wedges into a hemicube-like [Cohen and Greenberg 1985] data structure, which is centered at and oriented according to the light source (Section 3.1). This operation is carried out in the beginning of a frame. In the second stage, during shading, the list of

*e-mail: {samuli,timo,jaakko}@tml.hut.fi
†e-mail: uffe@ce.chalmers.se
‡e-mail: tam@cs.lth.se

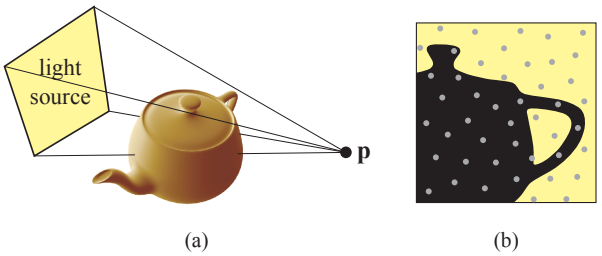

Figure 2: In physically-based soft shadow algorithms the fundamental operation is computing the parts of an area light source that are visible from the point, **p**, to be shaded. (a) A light source, an occluding object and **p**. (b) The light source is viewed from **p**. The non-blocked parts of the light source can be estimated with a sufficient number of samples, shown as gray dots. Visibility of the samples can be solved in two dimensions by projecting the silhouettes of occluders to the surface of the light source. Contrary to previous methods, our algorithm needs only the silhouette edges that overlap with the light source.

wedges that may contain **p** is fetched from the hemicube. The list is conservative, i.e., it includes some wedges that do not contain **p** and some of the corresponding edges are not silhouette edges from **p**. Two simple geometric tests yield the correct set of silhouette edges, as explained in Section 3.2.

As our second contribution, detailed in Section 3.3, we devise a novel technique for reconstructing the visibility function from this local window of silhouette edges. The operation requires only one shadow ray from each **p**.

We compare our algorithm against an optimized stochastic ray tracer in Section 4, and show up to 242-fold performance improvements without compromising the image quality. The new technique supports arbitrary planar light sources, and is also applicable to computing shadows for secondary rays. We revert to casting a reduced set of shadow rays in two automatically detected situations: when the projection of edges could result in numerical inaccuracy ($< 0.1\%$ of pixels; Section 3.3) or when semi-transparent occluders need to be sampled (Section 3.5).

2 Previous Work

This section concentrates primarily on algorithms that create physically-based soft shadows from area light sources, a process that requires evaluating an integral over the visible parts of the light source. In addition, a vast amount of literature exists for generating hard shadows from point light sources [Woo et al. 1990]. A recent survey on approximate real-time soft shadows is given by Hasenfratz et al. [2003]. Several other techniques (e.g. [Reeves et al. 1987]) compute soft shadows by simply filtering hard shadow boundaries. These techniques create plausible soft shadows only in a limited number of cases, but have been widely used because physically-based algorithms have been too slow for practical use.

Soft shadow volumes Using penumbra wedges [Akenine-Möller and Assarsson 2002] is fundamentally more efficient than processing the entire occluders, because the penumbra computations are limited to the silhouette edges whose projection from **p** to the plane of the light source overlaps the light. Also, the number of silhouette edges is typically much smaller than the number of triangles [McGuire 2004].

Earlier approximate soft shadow volume algorithms [Akenine-Möller and Assarsson 2002; Assarsson and Akenine-Möller 2003; Assarsson et al. 2003] have been targeting real-time performance using graphics hardware. However, they have made the simplifying assumption that the silhouette of an object is constant from all **p**. Even though the results are plausible in some cases, the assumption is hardly ever true in practice, and it is easy to construct cases where arbitrarily wrong results are obtained [Assarsson et al. 2003]. One solution to the problem of constant silhouettes is executing a fast silhouette extraction algorithm [Sander et al. 2000; Johnson and Cohen 2001] from every **p**. However, our new algorithm requires only the silhouette edges whose projection from **p** overlaps with the area light source, and thus a faster specialized technique can be derived. Another gross approximation in the previous soft shadow volume algorithms is heuristic occluder fusion: correct processing of penumbra can be guaranteed only when there is a single silhouette edge between **p** and a light source. Furthermore, shadows cannot be computed for secondary rays or semi-transparent surfaces. Our new algorithm suffers from none of these limitations.

Stochastic ray tracing Stochastic ray tracing algorithms compute shadows by sampling an area light source using shadow rays [Cook et al. 1984]. In order to get smooth and temporally coherent penumbra regions, hundreds of shadow rays are often needed for each point to be shaded. The intersection tests can be accelerated by employing variants of shadow cache [Haines and Greenberg 1986], and the distribution of the samples can be improved by using importance sampling [Shirley et al. 1996]. Unfortunately, each light source sample must be tested separately, even when all samples are occluded by the same surface.

Hart et al. [1999] propose a two-pass algorithm for reducing the number of shadow rays by utilizing image-space coherence. In the first pass, a small number of blocker-light pairs are stored for each pixel. In the second pass, the stored blockers are used for computing the visible parts of the light source. Impressive speedups are reported. However, because the algorithm does not consider all occluding polygons, correct results cannot be guaranteed.

Tracing thick rays Amanatides [1984] parameterizes rays with a spread angle using cones. The technique can approximate soft shadows by tracing a single cone from a surface point to an area light source. Amanatides uses a heuristic approximation for modelling the occlusion inside a cone.

Heckbert and Hanrahan [1984] describe a related technique of beam tracing in polygonal environments. Occlusion is correctly taken into account by clipping the beam with the occluding geometry. In highly tessellated scenes, the beam geometry quickly becomes prohibitively complex, and the performance degrades due to lost coherence. Pencil tracing [Shinya et al. 1987] is a similar technique, where a pencil is a set of rays in the vicinity of a given ray. It handles refractions more accurately than beam tracing, and also provides error tolerance analysis in an elegant manner. Ghazanfarpour and Hasenfratz [1998] describe a variant of beam tracing that does not clip the beam geometry, but instead subdivides the beam recursively until each sub-beam is either fully lit, fully occluded, or a specified subdivision limit is reached.

Techniques related to shadow volumes Nishita and Nakamae [1983] use two shadow volumes [Crow 1977] for identifying the parts of the scene that lie within the penumbra. Soft shadow computations are performed for polygons that intersect the penumbra. All silhouette edges of the shadow casters are projected onto the light source and clipped to its borders. Finally, irradiance is computed using an exact analytic formula. Shadow casters must be decomposed into sets of convex polyhedra which limits the practicality of the approach.

Chin and Feiner [1992] construct separate BSP trees for the scene, for the umbra volume and for the outer penumbra volume. Shadow receivers are then classified into three regions: fully lit, umbra, and penumbra. An analytic shadow term is computed by

traversing the BSP tree of the scene and clipping away the occluded parts of the polygonal light source.

Tanaka and Takahashi [1997] use a ray orientation-based spatial subdivision for culling most objects that do not contribute to the shadow term. The remaining set of occluders is further optimized by testing if the bounding volume of an occluder intersects the pyramid defined by the point and the light source. Finally, all silhouette edges of occluders are projected to the surface of the light source, and the visible parts of the light source are computed analytically using polygon clipping.

Visibility events The visibility skeleton [Durand et al. 1997] finds and stores all visibility events that cause discontinuities in visibility or shading. Illumination due to an area light source can be accurately computed for any point in the scene, but unfortunately the preprocessing and storage requirements are substantial.

Discontinuity meshing [Heckbert 1992; Lischinski et al. 1992] essentially subdivides receiver geometry along shadow boundaries. Back projection algorithms [Drettakis and Fiume 1994; Stewart and Ghali 1994] track visibility events to build a data structure for efficiently determining the visible parts of a light source. Both algorithms were demonstrated with scenes no larger than approximately 600 polygons. Also, the algorithms are prone to numerical inaccuracies.

Miscellaneous techniques Radiosity algorithms [Cohen and Wallace 1993] compute a diffuse global illumination solution that also includes soft shadows from direct lighting. Precomputed radiance transfer [Sloan et al. 2002] allows interactive rendering of soft shadows after substantial preprocessing.

Soler and Sillion [1998] approximate soft shadows using convolution in a hierarchical error-bounded algorithm. Agrawala et al. [2000] present an image-based soft shadow algorithm that uses layered attenuation maps for fast approximations. A coherent ray tracer is used for generating higher-quality images. Parker et al. [1998] render soft shadows at interactive rates in a parallel ray tracer by using only a single sample per pixel and "soft-edged" objects. Their algorithm is very fast, but not physically-based.

Bala et al. [2003] use an edge-and-point image (EPI) to provide sparse sampling of a scene. Important discontinuities, such as silhouette edges and shadow boundaries, are stored as edges in the EPI. Using the EPI, an image can be reconstructed using a specialized filter. For area light sources, both umbral and penumbral edges are detected and stored as edges in the EPI. When taking into account these edges and a sparse point sampling, soft shadows can be rendered.

BRDFs Analytic determination of irradiance from an area light source is possible only in a few special cases. If the visible parts of the light source are polygons and the emission function is constant over the source, the area integral may be converted into an integral over the boundaries of the polygon by using Stokes' theorem. Nishita and Nakamae [1985] compute direct illumination analytically for diffuse BRDFs, and Arvo [1995a] describes how to handle receiver BRDFs consisting of a linear combination of Phong-lobes.

Arbitrary receiver BRDFs can be handled using Monte Carlo integration. The emission function is sampled in a number of random points in the visible parts of the light source, and a weighted average of the samples gives an estimate of the illumination. Stratified sampling of the solid angle subtended by a polygonal light source [Arvo 1995b] can be used to reduce the noise.

3 Soft Shadow Volumes for Ray Tracing

This section describes our new soft shadow volume algorithm in detail. A pseudocode for the algorithm is given in Figure 3, and the following explanation refers to the line numbers in the pseudocode. In a preprocessing stage (CONSTRUCT-HEMICUBE) we extract silhouette edge information from the entire scene and store it into a static acceleration structure. Then, for each point **p** to be shaded, we compute the visibility between **p** and a set of samples on the surface of the light source (SHADOW-QUERY). The light samples are the targets of the shadow ray queries that our algorithm effectively replaces.

The construction of the acceleration structure is described in Section 3.1, and Section 3.2 discusses its use for finding the silhouette edges between **p** and the light source during a shadow query. In Section 3.3 we show how the silhouette edges are used for determining the visibility between **p** and the light samples. An important optimization to the algorithm is given in Section 3.4, and an extension for handling semi-transparent occluders in Section 3.5.

```
CONSTRUCT-HEMICUBE
1   for each edge E
2     if E is a potential silhouette edge from the light source
3       W ← wedge planes of E
4       add W into all hemicube cells that overlap W
5     end if
6   end for

SHADOW-QUERY(point p)
7   clear depth complexity counters of light samples
8   p' ← p projected onto the surface of the hemicube
9   L_W ← list of wedges from hemicube at p'
10  for each wedge W in L_W
11    E ← edge associated with W
12    if p is inside W and E is a silhouette edge from p
13      project E onto the surface of the light source
14      update depth complexity counters of light samples
15    end if
16  end for
17  cast a shadow ray to a light sample with lowest depth comp.
18  if ray is blocked
19    return all light samples are hidden
20  else
21    return light samples with lowest depth comp. are visible
22  end if
```

Figure 3: A pseudocode description of the two functions that constitute our new soft shadow volume algorithm. CONSTRUCT-HEMICUBE is executed once for every frame and it builds an acceleration structure for finding silhouette edges in shadow queries. SHADOW-QUERY is executed for every point to be shaded. It determines which point samples on the light source are visible from the query point. The algorithm is discussed in detail in Section 3.

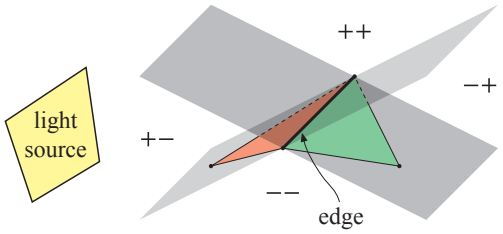

Figure 4: Determining silhouette edges from the light source. The planes of two triangles connected to an edge define four subspaces. If the light source lies entirely inside the $--$ or $++$ subspace, the edge cannot be a silhouette edge. Otherwise, the edge is a potential silhouette edge, as in the case depicted in the figure.

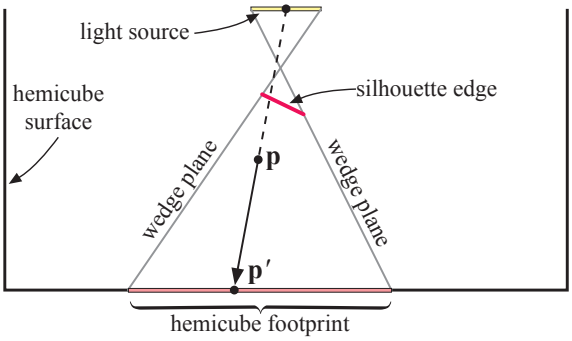

Figure 5: 2D illustration of how the hemicube footprints of penumbra wedges can be used for deciding whether the corresponding silhouette edge may overlap the light source from a given point. Wedge planes are determined according to the light source and the silhouette edge. The intersection of the wedge and the surface of the hemicube is the hemicube footprint of the wedge. To determine if point **p** may be inside the wedge, the point is projected onto the surface of the hemicube from the center of the light source. If the projected point $\mathbf{p}'$ is inside the hemicube footprint of the wedge, point **p** may be inside the wedge. Otherwise point **p** is guaranteed to be outside the wedge, and consequently the silhouette edge does not overlap the light source from **p**.

3.1 Acceleration structure

This section introduces the acceleration structure that is used for quickly finding the relevant silhouette edges for a given point **p**. The structure is a variant of the hemicube [Cohen and Greenberg 1985] with every face represented as a multiresolution grid. Additionally, our hemicube has physical dimensions, as will become apparent later in this section.

We observe that in order to contribute to the silhouette of an occluder between point **p** and the light source, an edge E must satisfy three criteria:

1. E is a silhouette edge from some point on the light source
2. E overlaps the light source from **p**
3. E is a silhouette edge from **p**

Criterion 2 is equivalent to requiring that point **p** is inside the exact *penumbra wedge* constructed from E. Our acceleration structure gives a conservative list of edges that satisfy the first two criteria.

First, we enumerate all edges in the scene, and for each edge we determine whether it is a potential silhouette edge from any point on the light source or not (criterion 1, Lines 1–2 in Figure 3). This is accomplished by classifying the vertices of the light source according to the planes of the two triangles connected to the edge, as illustrated in Figure 4. If there is only a single triangle connected to an edge, it must always be considered as a potential silhouette edge. Edges that do not satisfy criterion 1 according to this test are not added to the hemicube.

Next, penumbra wedges are constructed for the potential silhouette edges (Line 3) using a robust algorithm [Assarsson and Akenine-Möller 2003]. The *hemicube footprint* of each wedge is *conservatively* rasterized into the hemicube, which is centered and oriented according to the light source, and its size is chosen so that it encloses the entire scene (Figure 5). The hemicube footprint of a wedge is obtained by intersecting the wedge with the surface of the hemicube, and thus the size of the footprint depends on the physical dimensions of the hemicube.

After rasterization, each cell of the hemicube contains a list of wedges whose footprint intersects that cell (Line 4). The hemicube can then be used for determining the set of wedges that may contain the given point **p**, as explained in detail in Figure 5. Note that the test is conservative and valid only for points **p** that are located inside the hemicube.

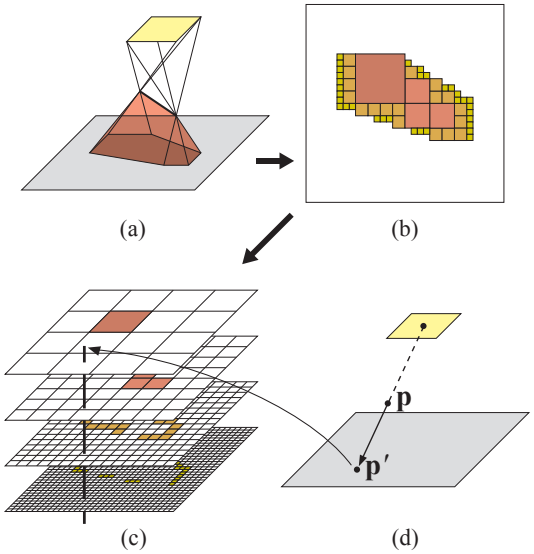

Figure 6: The hemicube footprints of wedges are stored into multiresolution grids to conserve memory. (a) A penumbra wedge is projected onto the bottom face of the hemicube (other faces not shown). (b) A conservative hierarchical representation of the footprint is determined so that the discretized footprint encloses the actual footprint completely. (c) The identifier of the wedge is added into the corresponding cells in a multiresolution grid. (d) During lookup, we collect the wedges from all levels of the multiresolution grid at the projected point $\mathbf{p}'$, shown as vertical line in (c).

A typical resolution for the hemicube is 128×128 on the bottom face and 128×64 on the side faces. The hemicube resolution is a tradeoff between memory consumption and execution speed; the finite resolution never causes artifacts because a wedge is listed in all cells that are even partially covered by the wedge.

In practice, the memory consumption of the hemicube may grow to an unacceptable level if the resolution of the hemicube is high and many footprints are rasterized. To conserve memory, we represent each face of the hemicube as a multiresolution grid, as illustrated in Figure 6. In this approach the footprint is stored adaptively on multiple levels of the multiresolution grid. During lookup, we simply collect the wedge lists of all levels of the multiresolution grid at the lookup position.

3.2 Finding silhouette edges from a point

We now turn to the execution of the soft shadow query. The query is issued for a given point **p** and it determines which light samples are visible from point **p**. The first step of the query determines which silhouette edges satisfy all three criteria listed in Section 3.1.

The hemicube footprints of all wedges have been rasterized into the hemicube in the preprocessing stage. Now we can quickly find the set of wedges that may contain a given point **p**, since every cell in the hemicube contains a list of wedges whose footprint intersects the cell (Lines 8–9 in Figure 3). This list of wedges corresponds to the edges that satisfy criterion 1 and may satisfy criterion 2 listed in Section 3.1. The list is conservative for two reasons: the hemicube footprints are rasterized conservatively and the point-inside-footprint test gives a conservative result for criterion 2. However, no edges are falsely omitted, and thus no artifacts emerge regardless of the resolution of the hemicube.

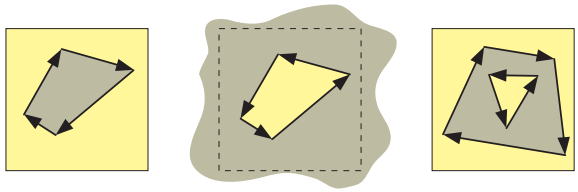

Figure 7: Silhouette edges are sufficient for determining the occlusion. (a) Silhouette loop of a simple occluder. (b) Silhouette loop of a hole in a large occluder. Note that the direction of the silhouette edges is reversed. (c) Complex silhouettes are defined by multiple separate loops.

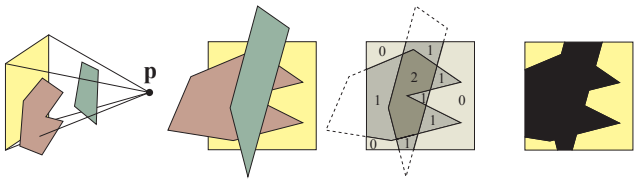

Figure 8: Illustration of the depth complexity function. (a) Two occluders are located between point **p** and the light source. (b) The occluders projected on the light source from **p**. (c) The depth complexity function tells the number of surfaces that overlap a point on the light source. (d) The visibility function is reconstructed from the depth complexity function.

After collecting the list of wedges from the hemicube, we determine which of these are silhouette edges from **p** and overlap the light source (Lines 10–12). Consider a single potential silhouette edge E and the corresponding wedge W. We first test that **p** is indeed inside wedge W, which corresponds to testing that edge E overlaps the light source from **p**. This provides us a definite answer for criterion 2. Next, we test that E is a silhouette edge from **p**. An edge shared by two triangles is a silhouette edge from **p** if exactly one of the triangles is frontfacing when viewed from **p**. If there is only a single triangle connected to an edge, it is always a silhouette edge. Passing both tests indicates that E satisfies all three criteria and is a relevant silhouette edge.

3.3 Local reconstruction of visibility function

Previous methods that use the projection of occluders onto the light source [Nishita and Nakamae 1983; Tanaka and Takahashi 1997] suffer from the fact that entire occluders have to be processed. This can cause significant overhead with complex objects.

We process only the silhouette edges whose projection from **p** overlaps the light source (Lines 12–13). Note that even though the silhouette edges always form closed loops, the projected edges do not necessarily form closed loops since the edges that would be projected outside the light source are discarded. Therefore, polygon filling algorithms cannot be used for directly determining the visible parts of the light source. The silhouette edges are oriented according to the orientation of the surface of the occluder, as illustrated in Figure 7. The correct orientation is obtained from the facings of the triangles connected to the edge as seen from **p**.

In general, the value of the visibility function cannot be evaluated from a local window of silhouette edges, because they indicate only relative changes of the *depth complexity function*. The depth complexity function of a point **s** on the light source is defined as the number of surfaces that a ray from **p** to **s** intersects. We see immediately that the visibility function can be defined in terms of

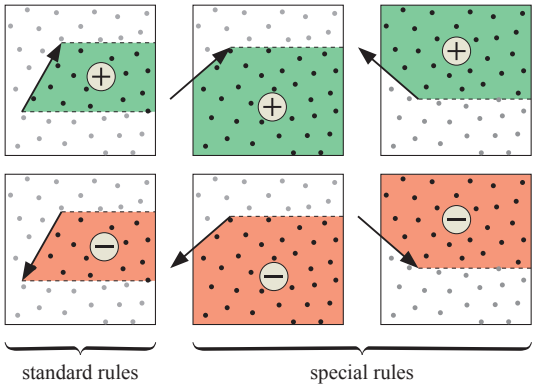

Figure 9: The left-to-right integration rules for updating the depth complexity counters of the light samples. The integration is performed one edge at a time, and the sign of the update is determined by the vertical direction of the edge. Four special rules are needed for silhouette edges that cross the left edge of the light source surface, in order to account for possibly incomplete silhouette loops. No special treatment is needed for silhouette edges that cross the light source boundary elsewhere. With these rules, the relative depth complexities of the light samples are always computed correctly. Determining the appropriate rule is robust because it involves only simple scalar vs. scalar comparisons.

the depth complexity function: if and only if point **s** has zero depth complexity, it is visible. The depth complexity function is illustrated in Figure 8.

Every projected silhouette edge represents a local change in the depth complexity function, and the full set of silhouette edges therefore completely defines the (generalized) derivative of the depth complexity function on the surface of the light source. This enables us to integrate over the derivative, obtaining the depth complexity function, but without the constant of integration. How the constant is determined will be discussed later in this section.

Since integration is a linear operation, it can be performed separately for each projected silhouette edge (Line 14). In addition, we are only interested in the depth complexity at the light samples. Because of this, it suffices to maintain a *depth complexity counter* for each light sample. After all edges have been integrated, these counters tell the relative depth complexities at the light samples. The integration can be performed in many ways, and we chose to integrate from left to right on the surface of the light source. Figure 9 shows the rules that we use for updating the depth complexity counters of the light samples. The magnitude of the adjustment to the depth complexity counters is equal to the number of triangles that share the edge [Bergeron 1986]. Special care must be taken to account for the missing silhouette edges outside the light source. With basic left-to-right integration, incorrect results would be obtained when edges of a projected silhouette loop on the left side of the light source are omitted. This is handled by four special rules that are applied when a projected silhouette edge crosses the left edge of the bounding rectangle of the light source.

Reconstructing the visibility function at the light samples requires solving whether the actual depth complexity of a light sample is zero or nonzero, i.e., solving the constant of integration. In order to accomplish this, we cast a single *reference ray* to a light sample and count the number of surfaces it intersects, yielding the actual depth complexity at that light sample. Given that the relative depth complexities of the light samples have been already determined by integration, we directly obtain the depth complexities of all light samples. Since the depth complexity of the reference ray cannot be negative, we observe that only the

light samples with lowest relative depth complexity can be visible. Because of this, we choose the target for the reference ray from the set of light samples with lowest relative depth complexity and cast a cheaper shadow ray (Line 17). If the shadow ray is occluded, all light samples must be occluded (Lines 18–19). Otherwise, the light samples with lowest depth complexity are visible and all other light samples are occluded (Lines 20–21).

A potential robustness issue arises from tracing the reference ray and performing the depth complexity integration in different coordinate systems. Due to numerical imprecision, an edge can move to another side of a light sample after projection. Casting the reference ray to such a sample would result in incorrect constant of integration, which in turn would yield erroneous actual depth complexities for all light samples. A robust solution is to cast the reference ray to a light sample that is some distance away from all projected edges. Such a sample is called *safe*. We mark the unsafe samples during the integration, and choose the target for the reference ray among the safe light samples with lowest depth complexity. In some cases, all light samples with lowest depth complexity are unsafe. In this situation, we revert to casting a separate shadow ray to every light sample with lowest depth complexity. Typically this happens in less than 0.1% of the pixels. Furthermore, in these cases the number of light samples with the lowest depth complexity is usually small: in our tests, the number of shadow rays cast in the reverted cases was on average less than 1.1. Thus, the cost was almost equal to casting the single reference ray. Proper treatment of these special cases ensures the robustness of our algorithm.

3.4 Optimizing the integration

If the occluders are heavily tessellated, many silhouette edges are projected to the light source and integrated. However, in this case most of the silhouette edges are relatively short. We exploit this in order to gain a significant speedup in the integration.

The optimization is based on bucketing the light samples according to their vertical coordinate on the surface of the light source. With uniformly sized buckets, the cost of bucketing the light samples is negligible. When integrating a projected silhouette edge, we need to process only the light samples that are inside the buckets spanned by the edge. If the silhouette edge crosses the left edge of the light source, the vertical extent of integration must be modified according to the special integration rules shown in Figure 9.

After empirical tests, we chose to use $\sqrt{N}$ buckets for N light samples. In practice, this gave an order of magnitude speedup for the integration, since in our test scenes the majority of the silhouette edges spanned only one or two buckets. In addition, we used Intel SSE2 SIMD instructions to process four light samples in parallel. After these optimizations, the integration cost is no longer dominant and our algorithm scales well with the number of light samples, as discussed in Section 4.

3.5 Semi-transparent occluders

Handling semi-transparent, non-refractive occluders requires only minor extensions to the algorithm. Table 1 shows the four possible cases that may occur for each light sample when semi-transparent occluders are present. The idea is to identify the light samples that are occluded by semi-transparent occluders, but not by the opaque ones. The shadow color of these samples is then determined by casting rays.

We solve the opaque and semi-transparent occlusion separately. During the extraction of wedges and construction of hemicube, all edges between opaque and semi-transparent triangles must be considered silhouette edges. This ensures that every silhouette loop corresponds to either an opaque or a semi-transparent surface. Every silhouette edge must also be augmented with a bit that indicates whether it belongs to an opaque or semi-transparent occluder.

Light sample occluded by		Status	Action
opaque	semi-transparent		
no	no	visible	–
no	yes	colored shadow	cast ray
yes	no	occluded	–
yes	yes	occluded	–

Table 1: Four possible cases for the visibility of a light sample in presence of semi-transparent occluders. Only one case requires casting a ray that determines the color of the shadow. In three other cases, the light sample is either fully visible or fully occluded.

The execution of the shadow query begins by determining the occlusion caused by the opaque occluders. This requires running the algorithm so that only the silhouette edges that belong to opaque occluders are considered. If all light samples are blocked by opaque occluders, no further processing is needed. Otherwise, we determine the occlusion caused by the semi-transparent occluders by running the algorithm with only the semi-transparent silhouette edges enabled. This allows us to classify the light samples according to Table 1. Finally, the color of the shadow is sampled by casting a ray to the light samples that are occluded only by semi-transparent occluders.

4 Results

Comparison method We compare our implementation against a heavily optimized commercial ray tracer (Turtle by Illuminate Labs). The ray tracer uses a hierarchical grid [Klimaszewski and Sederberg 1997] as an acceleration structure, traces four rays simultaneously with Intel SSE2 instructions, and implements a variant of the shadow cache [Haines and Greenberg 1986] for optimized intersection tests. Furthermore, the ray tracer uses a heuristic shadow sampling method: if all of the first 50% of shadow rays agree, the rest of the rays are skipped. As a result, most of the computation time is spent in penumbrae.

Scenes and setup We tested the comparison method and our new algorithm in five scenes, shown in Figure 10. The Columns scene is geometrically very simple (580 triangles), but the vast majority of pixels are in penumbra. Formula is more complex (60K), but the penumbrae are relatively small. The modified Sponza Atrium (109K) combines a low-tessellation building with a procedural tree that casts difficult shadows. Robots (1.3M) tests the scalability of the two methods with respect to the number of triangles. Finally, Ring (374K) is a difficult scene that has reflective surfaces and complex geometry. For each of the test scenes, we determined the minimum number of shadow rays that produced visually acceptable results. In Sponza, 150 light samples were sufficient due to the brighter ambient lighting and the masking effect of subtle textures, whereas the other scenes required 200–256 samples. Both methods are able to handle arbitrary planar light sources, but for simplicity, all tested scenes used one rectangular light source. The light samples were distributed according to a jittered grid.

All performance measurements were run on a 3.06GHz Pentium 4 with 1GB of memory. The output resolution of 960×540 is one quarter of the highest HDTV resolution; reduced resolution was used in order to keep the execution time of the comparison method manageable. Adaptive antialiasing was used in all scenes, and for the equal quality tests in Figure 10, the number of primary rays per output pixel was 3.05 in Ring, and 1.03–1.37 in the other scenes. As our method and the comparison method produced the same image, the use of adaptive antialiasing did not affect the reported speedup factors in any way. However, the "comparable time"-images needed more primary rays per pixel, 3.5 for Ring and 1.1–1.5 for the rest of the scenes, due to the substantial amount of noise in the penumbrae.

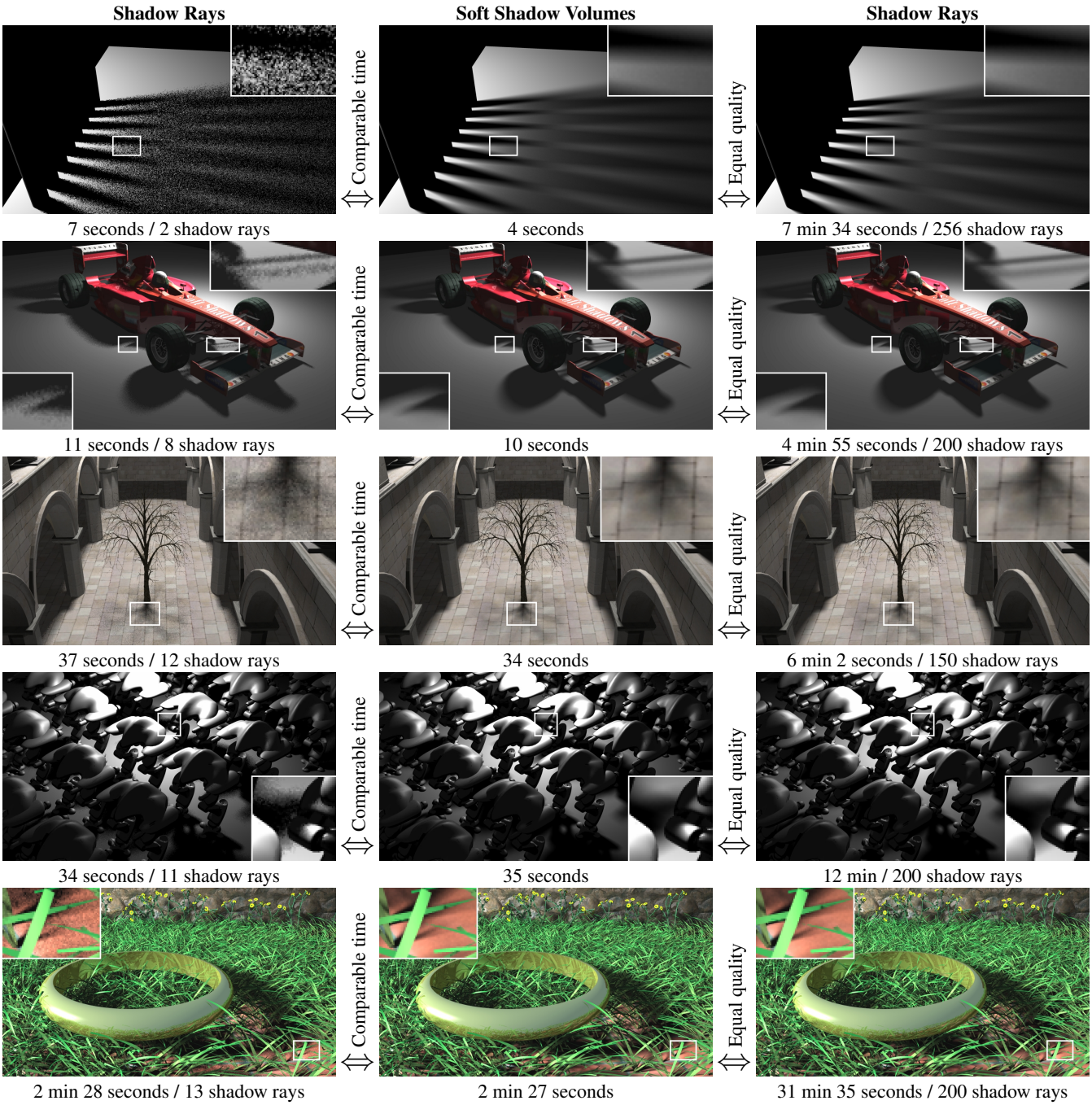

Figure 10: From top to bottom: Columns, Formula, Sponza, Robots and Ring. The timings include the hemicube construction and per-pixel shadow queries for our algorithm, and shadow rays for the comparison method. The results in the center column were rendered using our new soft shadow volume algorithm. On the leftmost column, a roughly equal amount of time was spent for casting shadow rays using the comparison method, whereas the rightmost column shows identical quality obtained using the shadow rays. Further statistics of the scenes are listed in Table 2. The Sponza scene allowed more noise in penumbra regions due to the high level of ambient lighting. This favored the comparison method, since a smaller number of shadow rays (150) was sufficient.

Performance analysis Table 2 lists the performance measurements from our test scenes. The highest speedup of 103 times was obtained in the simple Columns scene, which has large penumbra regions and thus forced the heuristic ray casting in the comparison method to cast more shadow rays than in the other scenes. The Formula and Robots scenes are structurally similar to each other, and thus we conclude from the respective speedup factors, 30 and 21, that our methods scales well with the number of triangles. The lowest speedup factor of 11 was obtained in Sponza and Ring. This is mostly due to the unusually large portion of the total time spent on edge validation. The scenes feature an exceptionally large number of silhouette edges compared to the number of triangles, and thus suffer from lack of coherence among light samples. In all test scenes, the execution time of our algorithm was comparable to trac-

Scene	#Triangles	#Samples	Comp. method	Our method							Speedup factor
			Total time	Hemicube rasterization	Edge validation	Edge projection	Integration	Reference ray	Misc	Total time	
				lines 1–6	8–12	13	14	17	7–9, 18–21		
Columns	580	256	454	0.0	0.7	0.2	0.4	2.3	0.8	4.4	**103**
Formula	60K	200	295	0.5	4.5	0.7	0.8	2.4	1.1	10.0	**30**
Sponza	109K	150	362	1.2	19.8	3.5	3.3	5.7	0.8	34.3	**11**
Robots	1.3M	200	720	7.2	12.8	2.5	2.8	8.2	1.2	34.7	**21**
Ring	374K	200	1898	5.9	96.2	19.7	29.8	25.2	3.7	180.5	**11**

Table 2: Performance results from our test scenes (Figure 10) with a hemicube resolution of 128×128. The timings (in seconds) and speedup factors refer to shadow computations, and exclude the construction of spatial subdivision as well as the casting of primary rays. All computations performed by our algorithm are included in the timings. The line numbers refer to the pseudocode in Figure 3.

Scene	Comp. method	Our method			Speedup factor
	Total time	Integration	Misc	Total	
Columns	1768	1.9	2.2	7.3	**242**
Formula	898	3.2	2.8	13.9	**65**
Sponza	1284	6.6	1.8	39.0	**33**
Robots	2293	5.9	2.7	39.3	**58**
Ring	6808	61.4	7.5	215.9	**32**

Table 3: Performance results with quadrupled light sample count, i.e., 256→1024, 200→800, and 150→600. The rest of the numbers are the same as in Table 2.

Scene	Comp. method	Our method			Speedup factor
	Total time	Hemicube rast.	Edge valid.	Total time	
Columns	454	0.3	0.7	4.7	**97**
Formula	295	2.0	3.8	11.0	**27**
Sponza	362	5.4	15.4	34.1	**11**
Robots	720	25.0	11.1	50.8	**14**
Ring	1898	17.6	51.1	147.1	**13**

Table 4: Performance results with hemicube resolution of 512×512. The rest of the numbers are the same as in Table 2.

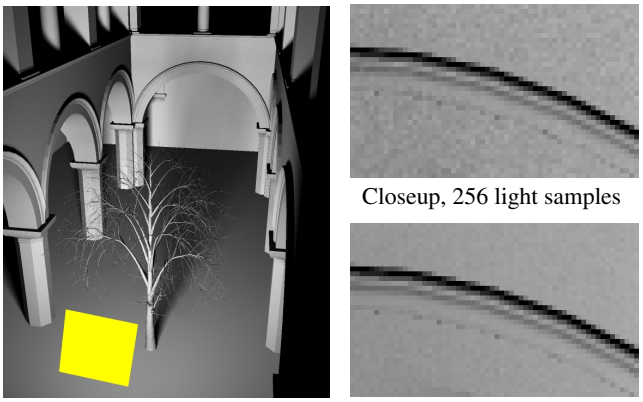

Figure 11: In difficult lighting conditions, 256 light samples are not sufficient for producing smooth penumbra regions. Because our algorithm scales well with respect to the number of light samples, computing the shadows for the setup on the left took only 47% more time with 1024 light samples than with 256 light samples. The samples were distributed according to a jittered grid, and adaptive anti-aliasing was used.

ing only 2–13 shadow rays per primary ray.

There are a number of situations where a large number of light samples is required for obtaining noise-free images. Image composition, tone mapping, textured light sources and high-dynamic range output medium, e.g. film, require more samples per light source. It is a common pitfall to assume that 256 light samples is always sufficient for 8-bit output per color component. When the occlusion between light source and the receiving surface is complex, the sampling of the light source must be viewed as a random process with no correlation between light samples, even when stratified sampling is used. In this case, basic probability theory states that the fraction of visible light samples is a random variable with approximate normal distribution, having standard deviation of $\sqrt{p(1-p)/n}$, where n is the number of light samples and p is the visible fraction of the light source. In the worst case, a 50%-lit surface may thus exhibit noise with standard deviation of $\frac{1}{32}$ with 256 light samples. Based on the confidence intervals of normal distributions, this amounts to one third of shadow calculations producing an estimation error over 3.12%. As illustrated in Figure 11, the error may become clearly visible. With 1024 samples, the amplitude of the noise is halved.

Fortunately, increasing the number of light samples affects the running time of our algorithm only modestly. The integration and misc categories in Table 2 are the only parts of our algorithm that are affected by the number of light source samples. These took less than 30% of our total shadow time. In contrast, the comparison method scaled almost linearly with respect to the number of samples. Therefore the speedup factors are much higher when more light source samples are required. We verified this by quadrupling the number of light samples in all scenes; the results are shown in Table 3. The speedup factors increased roughly by a factor of two to three, to 32–242.

The output resolution-independent hemicube rasterization consumed approximately 4–12% of the total shadow computation time when the resolution of the hemicube was set to 128×128 (Table 2). As the relative cost of hemicube construction diminishes in higher output resolutions, our method scales slightly better with respect to output resolution than the comparison method. In further tests, we got an additional $\sim$10% speedup over the comparison method in the highest HDTV resolution.

In most scenes, the edge validation and reference ray dominate the overall shadow computation cost. The cost of edge validation is primarily caused by the conservative hemicube, which lists many more wedges than absolutely necessary. As shown in Table 5, 3–16% of the wedges reported by the hemicube were projected to the light source. In the largest scenes, Robots and Ring, the 128×128 hemicube consumed 12MB of memory. To a certain extent, the list of wedges can be made tighter by increasing the hemicube resolution at the expense of additional memory consumption; the resolution does not affect the correctness of the results.

We investigated how the hemicube resolution affects the memory consumption and execution speed in the test scenes by increasing the resolution of the hemicube to 512×512. As shown in Table 5, the memory consumption roughly quadrupled when compared to a 128×128 hemicube, whereas the total shadow computation time

Scene	Wedges in hemicube	Hemicube resolution	Memory consumption (MB)	Max wedges in cell	Avg wedges in cell	Avg wedges tested/query	% wedges projected
Columns	267	128	0.2	40	3	9.6	15.6
		512	0.8	39	3	9.2	16.3
Formula	21K	128	1.8	4216	23	131.5	3.2
		512	11.6	3957	18	103.2	4.2
Sponza	49K	128	5.6	4235	103	491.5	6.3
		512	26.6	3579	89	402.1	7.7
Robots	193K	128	12.0	2499	147	373.2	4.6
		512	51.5	2206	113	294.4	5.8
Ring	453K	128	12.3	2763	93	1146.7	3.9
		512	53.8	2289	48	543.3	8.3

Table 5: Statistics related to the hemicube and penumbra wedges: number of wedges stored in the hemicube, resolution of the hemicube (largest face), memory consumption, maximum and average number of wedges per hemicube cell, average number of wedges tested in edge validation step per primary ray, and finally the percentage of wedges that pass the edge validation step and are thus projected and integrated.

was affected by less than 20% in all of the scenes except for Robots (Table 4). In general, the increased hemicube resolution slowed down the hemicube rasterization 1.5–4.0-fold, but also reduced the edge validation work. The more accurate hemicube would result in slightly faster execution speed in higher output resolutions when the relative cost of hemicube construction diminishes. We conclude that the selected hemicube resolution affects the execution time of our algorithm only weakly, and 128×128 was sufficient for all of the tested scenes. In the Ring scene, the speedup factor increased from 11 to 13 with the higher hemicube resolution.

The results can be directly generalized to animations because our method optimizes only the execution time, while the resulting image is identical to the one obtained by casting a shadow ray to each light sample.

The significance of the obtained speedup factors is that physically-based shadows from area light sources are finally a practical option for high-quality rendering.

5 Discussion and Future Work

Our new algorithm offers a significant performance improvement in a variety of test scenes. It is also relatively straightforward to implement — our optimized implementation amounts to $\sim$2000 lines of C++. The algorithm can be easily plugged into any existing ray tracer. In this section we discuss its limitations, potential improvements and extensions, as well as some thoughts on future work.

Due to its reliance on silhouette edges, the proposed algorithm is not applicable to direct rendering of implicit or parametric surfaces without tessellation. However, most rendering architectures convert the surfaces to triangles or micro-polygons before rendering.

The scalability of a ray tracer that uses a proper hierarchical spatial subdivision is logarithmic to the number of triangles in the scene. The average number of silhouette edges in a mesh of n triangles is higher, around $n^{0.8}$ [McGuire 2004]. This implies that, at least theoretically, a break-even point should exist so that tracing N shadow rays becomes faster than our algorithm. A scene that consists of random, non-connected triangles is difficult for our algorithm. In such a case, all edges are silhouette edges, and the coherence among sample points is nearly non-existent. This almost holds for the Ring scene, where the number of silhouette edges is greater than the number of triangles. Despite the unfavorable structure of the scene, our algorithm gave an order of magnitude speedup.

A larger light source subtends a larger solid angle, and in general, makes the visibility function between $\mathbf{p}$ and the light source more complex. The cost of casting shadow rays depends only slightly on the size of the light source. However, our algorithm reconstructs the visibility function, and thus larger light sources are generally more expensive to compute than smaller ones. Our experimental results support the theoretical hypothesis that the number of silhouette edges that need to be projected is roughly proportional to the solid angle subtended by the light source. Therefore the proposed method is not suitable for computing shadows from environment maps and other very large light sources. On the other hand, this effect is at least partially cancelled out by the fact that larger light sources often require additional light samples for limiting the noise to an acceptable level, and our method scales well with respect to number of light samples.

A spatially larger hemicube increases the relative sizes of the wedge footprints, and thus reduces the efficiency of our algorithm, but only up to a limit. The theoretical maximum size of a footprint can be determined by first constructing the wedge as usual, and then translating all wedge planes so that they pass through the light center (revisit Figure 5). This can be observed by scaling the light source and shadow-casting geometry down to an infinitesimal size, which is equivalent to having an infinitely large hemicube. The importance of the spatial dimensions of the hemicube depend on the particular configuration of light and objects. We did not investigate this further, and in our experiments the size of the hemicube was simply selected so that it contained the entire scene. An obvious improvement, also in terms of memory consumption, would be to further limit the size by the attenuation range of the light source. Only the wedges inside the attenuation range would need to be stored. Typically the orientation of the hemicube affects its size only slightly, and thus we can assume that the orientation is not critical.

It would be possible to parameterize the hemicube differently so that the set of affecting penumbra wedges becomes less conservative than in the current implementation. However, it is unclear whether this would improve the performance in practice because the redundant wedges are efficiently culled using a geometric point-inside-wedge test. Additionally, it might be fruitful to build the hemicube lazily so that the wedge rasterization would be limited to occluders whose bounding volume partially occludes the light source. This should reduce the memory requirements significantly, and possibly also improve the performance.

When multiple light sources illuminate the same parts of the scene, the number of samples taken per light source can usually be decreased. This favors the comparison method as only a part of our algorithm is affected by the number of light samples. In our test scenes (Figure 10), we can estimate that our method should be faster as long as more than 2–10 light samples per light source are taken. With a large number of light sources, the memory usage of the hemicubes might become an issue (Table 5). In these situations, the hemicube resolution could be automatically decreased.

We have experimented with using lookup tables for the depth complexity integration. The idea was dropped because a precomputed lookup table consumed additional memory, forced us to use the same sample pattern for every pixel, and somewhat surpris-

ingly, resulted in inferior performance due to cache pollution.

We are looking into efficiently supporting importance sampling according to BRDFs and the solid angle subtended by the light source. For these techniques, the samples on the light source need to be positioned according to the properties of the receiver point. Nothing in our algorithm prevents this.

Acknowledgments We would like to thank Ville Miettinen, Jacob Ström, Eric Haines, Lauri Savioja, Jonas Svensson, Janne Kontkanen, and Jussi Räsänen for helpful comments. Sponza Atrium by Marko Dabrovic, RNA studio, www.rna.hr. The graphics group at Helsinki University of Technology was funded by the National Technology Agency of Finland, Bitboys, Hybrid Graphics, Nokia and Remedy Entertainment. Timo was partially supported by ATI, Ulf by Illuminate Labs, and Tomas by the Hans Werthén foundation, Carl Tryggers foundation, Ernhold Lundströms foundation, and later by the Swedish Foundation for Strategic Research.

References

AGRAWALA, M., RAMAMOORTHI, R., HEIRICH, A., AND MOLL, L. 2000. Efficient Image-Based Methods for Rendering Soft Shadows. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press, 375–384.

AKENINE-MÖLLER, T., AND ASSARSSON, U. 2002. Approximate Soft Shadows on Arbitrary Surfaces using Penumbra Wedges. In *13th Eurographics Workshop on Rendering*, Eurographics, 297–305.

AMANATIDES, J. 1984. Ray Tracing with Cones. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, ACM Press, 129–135.

ARVO, J. 1995. Applications of Irradiance Tensors to the Simulation of Non-Lambertian Phenomena. In *Proceedings of ACM SIGGRAPH 95*, ACM Press, 335–342.

ARVO, J. 1995. Stratified Sampling of Spherical Triangles. In *Proceedings of ACM SIGGRAPH 95*, ACM Press, 437–438.

ASSARSSON, U., AND AKENINE-MÖLLER, T. 2003. A Geometry-Based Soft Shadow Volume Algorithm using Graphics Hardware. *ACM Transactions on Graphics 22*, 3, 511–520.

ASSARSSON, U., DOUGHERTY, M., MOUNIER, M., AND AKENINE-MÖLLER, T. 2003. An Optimized Soft Shadow Volume Algorithm with Real-Time Performance. In *Graphics Hardware*, ACM SIGGRAPH/Eurographics, 33–40.

BALA, K., WALTER, B., AND GREENBERG, D. P. 2003. Combining Edges and Points for Interactive High-Quality Rendering. *ACM Transactions on Graphics 22*, 3, 631–640.

BERGERON, P. 1986. A General Version of Crow's Shadow Volumes. *IEEE Computer Graphics and Applications 6*, 9, 17–28.

CHIN, N., AND FEINER, S. 1992. Fast Object-Precision Shadow Generation for Area Light Source using BSP Trees. In *Symposium on Interactive 3D Graphics archive*, ACM Press, 21–30.

COHEN, M. F., AND GREENBERG, D. P. 1985. The Hemi-Cube: A Radiosity Solution for Complex Environments. In *Computer Graphics (Proceedings of ACM SIGGRAPH 85)*, ACM Press, 31–40.

COHEN, M. F., AND WALLACE, J. R. 1993. *Radiosity and Realistic Image Synthesis*. Academic Press Professional.

COOK, R. L., PORTER, T., AND CARPENTER, L. 1984. Distributed Ray Tracing. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, ACM Press, 137–145.

CROW, F. 1977. Shadow Algorithms for Computer Graphics. In *Computer Graphics (Proceedings of ACM SIGGRAPH 77)*, ACM Press, 242–248.

DRETTAKIS, G., AND FIUME, E. 1994. A Fast Shadow Algorithm for Area Light Sources Using Back Projection. In *Proceedings of ACM SIGGRAPH 94*, ACM Press, 223–230.

DURAND, F., DRETTAKIS, G., AND PUECH, C. 1997. The Visibility Skeleton: A Powerful and Efficient Multi-Purpose Global Visibility Tool. In *Proceedings of ACM SIGGRAPH 97*, ACM Press, 89–100.

GHAZANFARPOUR, D., AND HASENFRATZ, J.-M. 1998. A Beam Tracing with Precise Antialiasing for Polyhedral Scenes. *Computer Graphics 22*, 1, 103–115.

HAINES, E. A., AND GREENBERG, D. P. 1986. The Light Buffer: A Ray Tracer Shadow Testing Accelerator. *IEEE Computer Graphics and Applications 6*, 9, 6–16.

HART, D., DUTRÉ, P., AND GREENBERG, D. P. 1999. Direct Illumination with Lazy Visibility Evaluation. In *Proceedings of ACM SIGGRAPH 99*, ACM Press, 147–154.

HASENFRATZ, J.-M., LAPIERRE, M., HOLZSCHUCH, N., AND SILLION, F. 2003. A Survey of Real-Time Soft Shadows Algorithms. *Computer Graphics Forum 22*, 4, 753–774.

HECKBERT, P., AND HANRAHAN, P. 1984. Beam Tracing Polygonal Objects. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, ACM Press, 119–127.

HECKBERT, P. 1992. Discontinuity Meshing for Radiosity. In *Third Eurographics Workshop on Rendering*, Eurographics, 203–215.

JOHNSON, D., AND COHEN, E. 2001. Spatialized Normal Cone Hierarchies. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press, 129–134.

KLIMASZEWSKI, K. S., AND SEDERBERG, T. W. 1997. Faster Ray Tracing Using Adaptive Grids. *IEEE Computer Graphics and Applications 17*, 1, 42–51.

LISCHINSKI, D., TAMPIERI, F., AND GREENBERG, D. P. 1992. Discontinuity Meshing for Accurate Radiosity. *IEEE Computer Graphics and Applications, 12*, 6, 25–39.

MCGUIRE, M. 2004. Observations on Silhouette Sizes. *Journal of Graphics Tools 9*, 1, 1–12.

NISHITA, T., AND NAKAMAE, E. 1983. Half-Tone Representation of 3-D Objects Illuminated by Area Sources or Polyhedron Sources. In *IEEE Computer Software and Application Conference*, 237–242.

NISHITA, T., AND NAKAMAE, E. 1985. Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection. In *Computer Graphics (Proceedings of ACM SIGGRAPH 85)*, ACM Press, 23–30.

PARKER, S., SHIRLEY, P., AND SMITS, B. 1998. Single Sample Soft Shadows. Tech. rep., University of Utah, UUCS-98-019.

REEVES, W. T., SALESIN, D. H., AND COOK, R. L. 1987. Rendering Antialiased Shadows with Depth Maps. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, ACM Press, 283–291.

SANDER, P. V., GU, X., GORTLER, S. J., HOPPE, H., AND SNYDER, J. 2000. Silhouette Clipping. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press, 327–334.

SHINYA, M., TAKAHASHI, T., AND NAITO, S. 1987. Principles and Applications of Pencil Tracing. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, ACM Press, 45–54.

SHIRLEY, P., WANG, C., AND ZIMMERMAN, K. 1996. Monte Carlo Techniques for Direct Lighting Calculations. *ACM Transactions on Graphics 15*, 1, 1–36.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. *ACM Transactions on Graphics 21*, 3, 527–536.

SOLER, C., AND SILLION, F. X. 1998. Fast Calculation of Soft Shadow Textures Using Convolution. In *Proceedings of ACM SIGGRAPH 98*, ACM Press, 321–332.

STEWART, A. J., AND GHALI, S. 1994. Fast Computation of Shadow Boundaries using Spatial Coherence and Backprojections. In *Proceedings of ACM SIGGRAPH 94*, ACM Press, 231–238.

TANAKA, T., AND TAKAHASHI, T. 1997. Fast Analytic Shading and Shadowing for Area Light Sources. *Computer Graphics Forum 16*, 3, 231–240.

WOO, A., POULIN, P., AND FOURNIER, A. 1990. A Survey of Shadow Algorithms. *IEEE Computer Graphics and Applications 10*, 6, 13–32.