EUROGRAPHICS 2005 / M. Alexa and J. Marks
(Guest Editors)

Volume 24 (2005), Number 3

Hierarchical Penumbra Casting

Samuli Laine Timo Aila

Helsinki University of Technology/TML and Hybrid Graphics, Ltd.

Abstract

We present a novel algorithm for rendering physically-based soft shadows in complex scenes. Instead of casting shadow rays, we place both the points to be shaded and the samples of an area light source into separate hierarchies, and compute hierarchically the shadows caused by each occluding triangle. This yields an efficient algorithm with memory requirements independent of the complexity of the scene.

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

1. Introduction

This paper describes a new algorithm for computing physically-based soft shadows from area light sources. Our technique is a generalization of *irregular shadow maps* [AL04, JMB04], and like most previous methods, represents arbitrary light sources as a set of light samples. Our algorithm can be seen as a transpose of ray tracing:

RAY TRACING
for each visible shadow-receiving surface sample r
  for each light sample ℓ
    find triangle that blocks ray $\ell \rightarrow r$

OUR APPROACH
for each triangle T
  find all $\ell \rightarrow r$ rays that are blocked by T

Ray tracing finds the blocker triangles using a hierarchical spatial subdivision, and therefore scales well with respect to the number of triangles. Our method uses two hierarchies simultaneously to quickly find the rays that are blocked by a given triangle, and thus scales well with respect to the output resolution and the number of light samples. As our algorithm processes one triangle at a time, there is no need for building a spatial subdivision for the scene geometry. This is particularly convenient for procedural geometry or for models that are too large to fit into physical memory.

Irregular shadow maps first determine the visible surfaces from the point of view of the camera. Then the visible samples (x,y,z), i.e. the shadow *receiver points*, are transformed to the image plane of the light source, producing sampling points (x',y') and the corresponding light-space depth values z'. The (x',y') correspond exactly to the intersections of shadow rays and the image plane of the light source. Finally, the irregularly spaced points (x',y') are stored into a BSP tree and used as sampling points when the scene is rasterized from the light source. This gives the shadow information exactly at the receiver points, yielding the same result as casting shadow rays. Reflections and semi-transparent shadow receivers are handled using a deep frame buffer, i.e., by storing and processing multiple surfaces per output pixel.

Irregular shadow maps could be directly applied to area lights by executing the algorithm separately for all light samples. However, such a method would suffer from several weaknesses. All operations (projection to the image plane of a light source, building the BSP, processing the triangles) would have to be repeated for every light sample. Furthermore, the same pattern of light samples would have to be used for every pixel, resulting in visible banding.

We build the hierarchy for the receiver points in object space, thus avoiding the projection altogether (Section 3.1). We follow the naming convention of Chin and Feiner [CF92]: the *penumbra volume* of a triangle contains all regions of space that can be either fully (umbra) or partially (penumbra) in shadow with respect to the triangle and an area light source. We construct a static three-level hierarchy for the light samples, and for each triangle, we traverse both the light sample hierarchy and the receiver point hierarchy simultaneously. In practice, this is done by intersect-

ing the penumbra volumes of the triangle with the receiver point hierarchy (Section 3.2). This allows us to quickly reject the receiver points that are not shadowed by the given triangle. The output-sensitivity of the algorithm can be increased with a number of optimizations (Section 3.3). Additionally, the pattern of light samples can be selected separately for each pixel, thus converting banding to less visible noise (Section 3.4).

We analyze the performance and scalability of our algorithm by comparing it against a production-quality ray tracer in Section 4. Discussion and future work, along with a rough complexity analysis, are given in Section 5.

2. Previous Work

This section concentrates on algorithms that create physically-based soft shadows from area light sources. Additionally, a vast amount of literature exists for generating hard shadows from point light sources [WPF90], as well as for non-physical [RSC87] and real-time approximation of soft shadows [HLHS03].

Stochastic ray tracing Stochastic ray tracing algorithms compute shadows by sampling an area light source using shadow rays [CPC84]. 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 [HG86], and the distribution of the samples can be improved by using importance sampling [SWZ96]. With a hierarchical spatial subdivision, the complexity of ray tracing is logarithmic with respect the number of triangles. However, the complexity remains linear with respect to the output resolution and the number of light samples.

Tracing thick rays Amanatides [Ama84] 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 [HH84] 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 [STN87] 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 [GH98] describe a variant of beam tracing that does not clip the beam geometry, but instead subdivides the beam recursively until each subbeam is either fully lit, fully occluded, or a specified subdivision limit is reached. Conceptually, this technique extends ray tracing to process the light samples hierarchically (i.e. sub-linearly), whereas the scalability with respect to output resolution remains linear.

Techniques related to shadow volumes Nishita and Nakamae [NN83] use two shadow volumes [Cro77] for identifying the parts of the scene that lie within the penumbra. Soft shadow computations are performed only for polygons that intersect the penumbra. 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 [CF92] 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. Our method bears some similarity to this technique in the sense that we also utilize penumbra volumes. Tanaka and Takahashi [TT97] propose culling methods for efficiently determining the set of objects that can affect the shadowing of a given point.

View-dependence and discretization The visibility skeleton [DDP97] 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 [Hec92, LTG92] essentially subdivides receiver geometry along shadow boundaries. Back projection algorithms [DF94, SG94] track visibility events to build a data structure for efficiently determining the visible parts of a light source. These techniques have trouble scaling to complex scenes, and the algorithms are also prone to numerical inaccuracies.

When computing soft shadows for a particular viewpoint instead of the whole scene (e.g. radiosity [CW93]), it is unnecessary to consider the entire scene geometry as shadow receivers. Instead, the visible receiver points can be computed in advance [Cro77, Hei91]; this greatly simplifies the subsequent shadow computations. Furthermore, the receiver points can be organized hierarchically in order to exploit the coherence in hard shadow computations [AL04, JMB04, AAM04]. We exploit this observation as a part of our algorithm.

Miscellaneous techniques Soler and Sillion [SS98] approximate soft shadows using convolution, and present a hierarchical algorithm that drives the approximation error below a threshold value. Agrawala et al. [ARHM00] 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. Bala et al. [BWG03] approximate soft shadows by computing the shadowed illumination in a sparse set of points, and then filtering the output image by taking into account important discontinuities such as shadow boundaries. Parker et al. [PSS98] 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.

Assarsson and Akenine-Möller [AAM03] describe an approximate soft shadow volume algorithm, which offers real-time performance in simple scenes. Two gross approximations are made: assumption that the silhouette of an object is constant from all receiver points, and a heuristic occluder fusion method.

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 [NN85] compute direct illumination analytically for diffuse BRDFs, and Arvo [Arv95] 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.

3. Algorithm

In this section, we present our shadow computation algorithm in detail. First, we define the terms and symbols in Section 3.1 and proceed by giving a simplified version of the algorithm in Section 3.2. Next, we discuss a number of important optimizations in Section 3.3. Finally, extensions to the algorithm are presented in Section 3.4.

3.1. Terminology

Let us consider a planar area light source L. We define the light source using a planar, convex bounding polygon Δ_L and a set of *light samples* ℓ_i that lie inside the bounding polygon. For now, we assume that the same set of light samples is used throughout the computation. In reality, this would cause banding artifacts, and a solution is presented later in Section 3.4.

In order to avoid processing each light sample separately, we group the light samples spatially into *light sample groups*, denoted G_k, each of which contains a number of light samples ℓ_i. For each G_k, we also compute a convex bounding polygon Δ_{G_k} that lies on the plane of the light source. This creates a three-level hierarchy for the light samples: on the top level is the entire light source L, on the middle level are the light sample groups G_k, and on the bottom level are the individual light samples ℓ_i.

Symbol	Meaning
T	blocker triangle
L	area light source
ℓ_i	light sample
G_k	light sample group
Δ	polygon
Δ_L	bounding polygon of the light source
Δ_{G_k}	bounding polygon of a light sample group
V_L	penumbra volume defined by Δ_L and T
V_{G_k}	penumbra volume defined by Δ_{G_k} and T
V_{ℓ_i}	hard shadow volume defined by ℓ_i and T
r_j	receiver point
N	node in the hierarchy of receiver points
B_N	bounding box of node N

Table 1: *List of symbols used in Section 3.*

Since we process each shadow-casting *blocker triangle* separately, we need to consider only a single blocker triangle T at a time. We construct penumbra volumes for the two top levels of the light sample hierarchy, namely for the entire light source L and for the light sample groups G_k. In addition, we construct hard shadow volumes for the light samples ℓ_i. These volumes are illustrated in Figure 1. The penumbra volume defined by the blocker triangle T and the bounding polygon Δ_L of the light source is denoted V_L. This penumbra volume encloses all points that may be at least partially shadowed by T. The penumbra volumes defined by T and the bounding polygons Δ_{G_k} of light sample groups G_k are denoted V_{G_k}. Thus, penumbra volume V_{G_k} encloses all points where T may cast shadows from light samples in group G_k. Finally, the hard shadow volumes defined by T and a light sample ℓ_i are denoted V_{ℓ_i}. A hard shadow volume V_{ℓ_i} thus encloses exactly the points that are in shadow from light sample ℓ_i.

The points in the scene that need shadow information are called *receiver points*, and denoted r_j. Thus, the purpose of the algorithm is to determine which $\ell_i \rightarrow r_j$ relations are blocked by blocker triangles. The receiver points are placed into a bounding volume hierarchy, where axis-aligned boxes are used as the bounding volumes. A node in the hierarchy is denoted N, and its bounding box B_N.

The symbols are summarized in Table 1.

3.2. Basic Algorithm

Rendering the scene with shadows computed using our algorithm consists of the following steps:

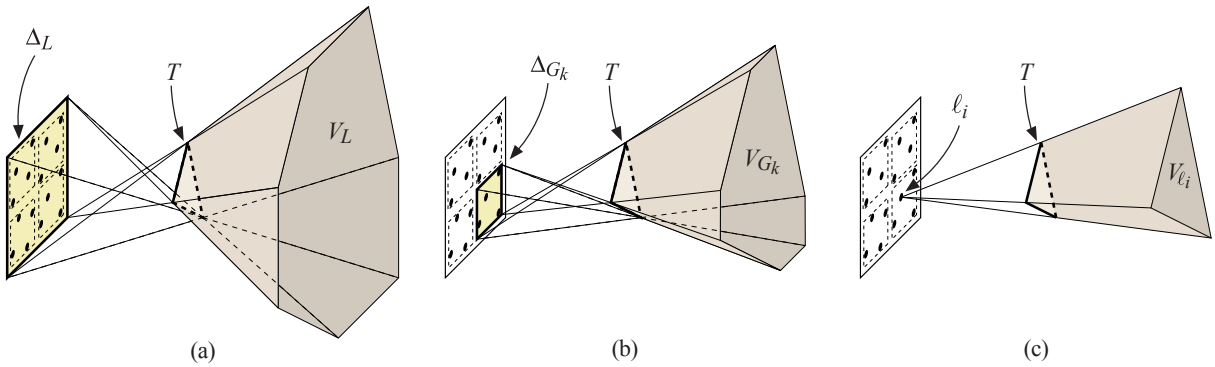


Figure 1: *The volumes associated with the three levels of the light sample hierarchy. In this illustration, the light source consists of 16 light samples grouped into four light sample groups. (a) The main penumbra volume V_L is defined by the bounding polygon Δ_L of the light source and the blocker triangle T. (b) For each light sample group G_k, a penumbra volume V_{G_k} is constructed based on the bounding polygon Δ_{G_k} of the light sample group and the blocker triangle T. (c) For each light sample ℓ_i, a hard shadow volume V_{ℓ_i} is constructed, based on the location of ℓ_i and the blocker triangle T.*

- Rasterize the scene, store depth values of visible surfaces
- Using the depth values, construct receiver points r_j
- Construct the receiver point hierarchy
- Construct the light sample groups G_k
- Process all blocker triangles
- Perform shading

First, the scene is rasterized from the viewpoint of the camera, and the depth values of visible surfaces are stored into a deep depth buffer. The visible samples are then transformed into world space producing receiver points r_j.

A bounding volume hierarchy is constructed for the receiver points r_j, and a bit mask is allocated for each r_j. These bit masks are used for storing the visibility status of every $\ell_i \rightarrow r_j$ relation. The bit masks are initialized to all zeros, indicating that all $\ell_i \rightarrow r_j$ relations are initially unblocked. We build the bounding volume hierarchy by recursively dividing the receiver points in two groups. As long as there are more than 1K points in a node, we simply split the bounding box of the node in two equal halves. When there are less than 1K points, we find the split position that minimizes the total surface area of the resulting child nodes.

Next, the light sample groups G_k are constructed so that each G_k contains a number of nearby light samples. Constructing the groups is trivial if the light samples are positioned according to a jittered grid distribution. Otherwise, a heuristic grouping algorithm may be applied. After the groups are constructed, a convex bounding polygon Δ_{G_k} is determined for each group G_k.

The next step is processing all blocker triangles in order to find the blocked $\ell_i \rightarrow r_j$ relations. A pseudocode version of the basic algorithm is given in Figure 2. We now examine the algorithm in detail.

Processing a blocker triangle T begins by constructing the penumbra volume V_L defined by T and the bounding poly-

```
procedure PROCESS-BLOCKER(triangle T)
 1  V_L ← MAKE-PENUMBRA-VOLUME(Δ_L, T)
 2  for each G_k do V_{G_k} ← MAKE-PENUMBRA-VOLUME(Δ_{G_k}, T)
 3  for each ℓ_i do V_{ℓ_i} ← MAKE-SHADOW-VOLUME(ℓ_i, T)
 4  active-groups ← every G_k
 5  PROCESS-RECURSIVE(root-node, active-groups)

procedure PROCESS-RECURSIVE(N, active-groups)
 6  if not INTERSECTS(B_N, V_L) then return
 7  active-groups' ← ∅
 8  for each G_k in active-groups do
 9    if INTERSECTS(B_N, V_{G_k}) then add G_k into active-groups'
10  end for
11  if active-groups' = ∅ then return
12  if N is not a leaf node then
13    PROCESS-RECURSIVE(N.left, active-groups')
14    PROCESS-RECURSIVE(N.right, active-groups')
15  else
16    for each r_j in N do
17      if not POINT-IN-VOLUME(r_j, V_L) then continue
18      for each G_k in active-groups' do
19        if not POINT-IN-VOLUME(r_j, V_{G_k}) then continue
20        for each ℓ_i in G_k do
21          if r_j.mask[i] = 1 then continue
22          if POINT-IN-VOLUME(r_j, V_{ℓ_i}) then
23            r_j.mask[i] ← 1
24          end if
25        end for
26      end for
27    end for
28  end if
```

Figure 2: *Pseudocode of the basic algorithm. A detailed explanation of the algorithm is given in Section 3.2. The* **continue** *keyword is borrowed from C language and it causes the next iteration in the nearest enclosing* **for** *loop to be started immediately.*

gon Δ_L of the entire light source (Line 1), as well as the penumbra volumes V_{G_k} defined by T and the bounding polygons Δ_{G_k} of the light sample groups (Line 2). A penumbra volume consists of the *separating planes* between the defining polygon Δ and the blocker triangle T. These planes are found by enumerating every vertex-edge pair between Δ and T, and selecting the planes that have all vertices of Δ on one side and all vertices of T on the other side. The number of separating planes is at most equal to the total number of edges in Δ and T. In addition, if the plane of T does not intersect Δ, the plane of T is added to cap the penumbra volume. The hard shadow volumes V_{ℓ_i} are then constructed for each light sample ℓ_i (Line 3). These always consist of four planes: three defined by ℓ_i and the edges of T, and the plane of T. The volumes are illustrated in Figure 1 and used extensively in the following discussion.

During the recursive hierarchy traversal, we maintain a list of *active* light sample groups. A light sample group G_k is active at node N if triangle T may block at least some of the relations $\ell_i \rightarrow r_j$, where $\ell_i \in G_k$ and r_j is under node N. This is possible only if the penumbra volume V_{G_k} intersects the bounding box B_N at least partially. Initially, all light sample groups are active (Line 4).

Traversing the hierarchy begins from the root node (Line 5). When entering hierarchy node N, we first test if the bounding box B_N of the node intersects penumbra volume V_L. If not, the traversal is terminated (Line 6) since triangle T cannot cast shadows to any receiver point under node N. Otherwise, we construct a tighter list of active light sample groups so that only the groups G_k whose penumbra volume V_{G_k} intersects B_N are included (Lines 7–10). Again, if there is no intersection, triangle T cannot cast shadows from light sample group G_k to any receiver point under node N. If no active groups remain, the traversal is terminated (Line 11).

If node N is not a leaf node, the traversal continues with the tighter list of active light sample groups (Lines 13 and 14). Otherwise, the receiver points r_j in the node are tested for potential shadowing by triangle T. We immediately skip the further processing of r_j that are outside penumbra volume V_L (Line 17). Then, the light sample groups G_k whose corresponding penumbra volume V_{G_k} does not contain r_j are skipped (Line 19). Also, the relations that are already blocked by the earlier triangles need no further consideration (Line 21). For the remaining light samples ℓ_i, we test if triangle T blocks relation $\ell_i \rightarrow r_j$ by testing if r_j is inside the hard shadow volume V_{ℓ_i} (Line 22). Since V_{ℓ_i} always has four planes, this test involves four dot products and these can be executed in parallel with Intel SSE SIMD instructions, as is done in our implementation. If the test indicates that r_j is inside V_{ℓ_i}, the relation is marked as blocked in the bit mask associated with r_j (Line 23).

After all blocker triangles have been processed, the bit masks of the receiver points tell which light samples ℓ_i are visible to which receiver points r_j. The final step is to perform shading using the computed shadow information. One way of accomplishing this is to rasterize the scene again from the viewpoint of the camera, and fetch the shadow information that corresponds to the visible samples. An alternative implementation might store all the data needed for shading into a deep frame buffer during the first pass.

3.3. Optimizations

There are a number of straightforward optimizations that fit naturally into the basic algorithm discussed above. The aim of these optimizations is to make the algorithm behave in a more output-sensitive fashion both in terms of computation time and memory consumption. There are five main ways for accomplishing this. First, redundant processing of already blocked $\ell_i \rightarrow r_j$ relations can often be terminated early in the traversal. Second, the number of bounding box vs. penumbra volume plane tests can be reduced so that, e.g., when the traversal has proceeded to a node that is completely inside a penumbra volume, further tests can be safely omitted in child nodes. Third, constructing the penumbra volumes V_{G_k} for the light groups and the hard shadow volumes V_{ℓ_i} can be postponed until they are needed, and omitted altogether in many situations. Fourth, the memory consumption can be reduced by on-demand allocation of the bit masks for r_j. Finally, the efficiency of the aforementioned optimizations can be further enhanced by coarsely sorting the blocker triangles according to their occlusion power. In this section, we explain each of these optimizations in detail.

A pseudocode version of the optimized algorithm is given in Figure 3. In the pseudocode, each added line is tagged with the respective optimization by letter A–D. To save space, only a single pseudocode with all the optimizations is given, rather than showing the evolution of the algorithm after each optimization. We recommend investigating the entire pseudocode as a whole only after all optimizations have been covered.

A. Umbra bits The most important optimization involves storing aggregate shadow information into hierarchy nodes and receiver points. We add *group-umbra* and *full-umbra* bits to each hierarchy node N as well as to every receiver point r_j. Whenever all relations $\ell_i \rightarrow r_j$, where $\ell_i \in G_k$, become blocked for some r_j, the group-umbra bit k of r_j is set (Line 72 in Figure 3). In addition, when all group-umbra bits are set in r_j, we set the full-umbra bit of r_j (Line 74). The umbra bits of a leaf node are computed by performing a Boolean AND operation over the corresponding bits of all receiver points r_j in the node (Lines 77–80), and the umbra bits of an internal node are computed similarly by performing a Boolean AND operation over the corresponding bits of its two child nodes (Lines 50–54).

The umbra bits are used for terminating the traversal and for pruning the list of active light sample groups. If node

```
procedure PROCESS-BLOCKER-OPTIMIZED(triangle T)
29 A if umbra-point-count > limit then REBUILD-HIERARCHY()
30   V_L ← MAKE-PENUMBRA-VOLUME(Δ_L, T)
31 C volumes-computed ← false
32 B active-groups ← (G_k, ALL) for every G_k
33 B P_L ← ALL
34 B PROCESS-RECURSIVE(root-node, P_L, active-groups)

procedure PROCESS-RECURSIVE(N, P_L, active-groups)
35 A if N.full-umbra then return
36 B if not INTERSECTS(B_N, V_L, P_L, P_L') then return
37   active-groups' ← ∅
38 B for each (G_k, P_Gk) in active-groups do
39 A   if N.group-umbra[k] then continue
40 C   if volumes-computed then
41 B     if INTERSECTS(B_N, V_Gk, P_Gk, P_Gk') then
42 B       add (G_k, P_Gk') into active-groups'
43       end if
44 BC  else add (G_k, ALL) into active-groups'
45   end for
46   if active-groups' = ∅ then return
47   if N is not a leaf node then
48 B   PROCESS-RECURSIVE(N.left, P_L', active-groups')
49 B   PROCESS-RECURSIVE(N.right, P_L', active-groups')
50 A   for each k do
51 A     N.group-umbra[k] ← N.left.group-umbra[k] ∧
52 A                        N.right.group-umbra[k]
53 A   end for
54 A   N.full-umbra ← N.left.full-umbra ∧ N.right.full-umbra
55   else
56     for each r_j in N do
57 A     if r_j.full-umbra then continue
58 B     if not POINT-IN-VOLUME(r_j, V_L, P_L') then continue
59 D     if not r_j.has-mask then r_j.mask ← NEW-MASK()
60 C     if not volumes-computed then
61 C       volumes-computed ← true
62 C       for each G_k do V_Gk ← MAKE-PENUMBRA-VOL.(Δ_Gk, T)
63 C       for each ℓ_i do V_ℓi ← MAKE-SHADOW-VOLUME(ℓ_i, T)
64 C     end if
65 B     for each (G_k, P_Gk') in active-groups' do
66 A       if r_j.group-umbra[k] then continue
67 B       if not POINT-IN-VOLUME(r_j, V_Gk, P_Gk') then continue
68         for each ℓ_i in G_k do
69           if r_j.mask[i] = 1 then continue
70           if POINT-IN-VOLUME(r_j, V_ℓi) then r_j.mask[i] ← 1
71         end for
72 A       r_j.group-umbra[k] ← ∧ r_j.mask[i], ℓ_i ∈ G_k
73     end for
74 A   r_j.full-umbra ← ∧ r_j.group-umbra[k], ∀k
75 D   if r_j.full-umbra then RELEASE-MASK(r_j.mask)
76   end for
77 A for each k do
78 A   N.group-umbra[k] ← ∧ r_j.group-umbra[k], r_j ∈ N
79 A end for
80 A N.full-umbra ← ∧ r_j.full-umbra, r_j ∈ N
81 end if
```

Figure 3: *Pseudocode of the optimized algorithm. A detailed explanation of the algorithm is given in Section 3.3. Letters* A–D *after the line numbers refer to the corresponding optimizations in Section 3.3.*

N has its full-umbra bit set, the traversal can be immediately terminated (Line 35), since all receiver points under it are already in shadow. Similarly, if node N has group-umbra bit k set, the light sample group G_k needs no further consideration in the child nodes of N, as all relations $\ell_i \rightarrow r_j$, where $\ell_i \in G_k$ and r_j is under N, are known to be blocked (Line 39). The same observations apply for avoiding the processing of receiver points r_j (Lines 57 and 66). Since the light samples with full-umbra bits set need no further processing, we rebuild the receiver point hierarchy from scratch when a large enough fraction of receiver points are in umbra (Line 29). This ensures that the hierarchy stays relatively balanced throughout the computation, and also decreases the average height of the hierarchy, i.e., number of traversal steps from the root node to child nodes.

B. Active plane sets The basic algorithm performs many redundant box-vs-plane tests when traversing the hierarchy. Recalling that all penumbra volumes consist of a set of planes, we can skip the majority of these tests by associating *active plane sets* to the penumbra volumes. This is a commonly used technique in e.g. hierarchical view frustum culling [BEW*98]. If bounding box B_N is completely on the positive side, i.e. "inside", of a plane, so are the bounding boxes of all nodes under it. In this case, the plane can be marked inactive and safely ignored while traversing the subtree under N. In particular, when B_N is completely inside a penumbra volume, the associated active plane set becomes empty.

Active plane sets P_L and P_{G_k} are associated with penumbra volumes V_L and V_{G_k}, respectively. The special token ALL refers to a set where all planes are active (Lines 32, 33 and 44). P_L is passed as a parameter in recursion calls (Lines 34, 48 and 49), and P_{G_k} are coupled with the corresponding groups G_k in the *active-groups* list (Lines 32, 38, 42, 44 and 65). We modify the box-vs-plane intersection test (function INTERSECTS in the pseudocode) so that it takes the active plane set of the penumbra volume as input and produces a new plane set as its output (Line 36: $P_L \Rightarrow P_L'$, Line 41: $P_{G_k} \Rightarrow P_{G_k}'$). The POINT-IN-VOLUME tests (Lines 58 and 67) are also modified to consider only the active planes.

C. Lazy penumbra volume and hard shadow volume construction With the above optimizations, the processing of redundant blocker triangles that cast shadows to already shadowed areas is very efficient. With such triangles, most of the time is spent in the construction of penumbra volumes V_{G_k} and hard shadow volumes V_{ℓ_i}. This can be avoided by postponing the construction of these volumes until they are actually needed. Therefore, instead of constructing these volumes before traversing the hierarchy, we merely note that they have not been constructed yet (Line 31). As long as the volumes are not available, we do not prune the list of active light sample groups by box-vs-volume tests (Lines 40

and 44). If we encounter a receiver point where a previously unblocked $\ell_i \rightarrow r_j$ relation may become blocked, we construct the volumes (Lines 60–64).

D. On-demand bit mask allocation It is quite uncommon that all receiver points r_j are simultaneously in penumbra. In most situations, it is possible to conserve memory by allocating the bit mask for r_j only when one is needed. Initially all receiver points are fully lit and thus need no bit masks to store the status of the blocked $\ell_i \rightarrow r_j$ relations. When the hierarchy traversal finds an r_j that is potentially occluded by a blocker triangle, an empty bit mask is allocated if the receiver point does not already have one (Line 59). In addition, since we store the full-umbra bits to r_j (Line 74), we may deallocate the bit mask if the receiver point has all $\ell_i \rightarrow r_j$ relations blocked (Line 75).

Coarse blocker sorting The overall efficiency of the above optimizations can be increased by sorting the blockers coarsely according to their estimated occlusion power. This is beneficial, since it is favorable to find large shadow regions early in the process, thus eliminating the need for processing the affected receiver points later. Especially with receiver point hierarchy rebuilding (Line 29), finding large shadowed regions early reduces the size of the entire problem for the remaining blocker triangles. We perform the sorting by *bucketing* the blocker triangles according to their surface areas. Other criteria might yield more efficient ordering, but we did not investigate this further. Bucketing is a linear-time operation that can be executed entirely out-of-core by using separate files for storing the sorted triangles. We first loop through all blocker triangles and construct a histogram of their areas. We then assign an area range to each bucket and create an empty file for each bucket. Next, we loop through the blocker triangles again, streaming the blocker triangles into the bucket files. By processing the bucket files in order from largest to smallest triangles, we obtain a coarse sorting for the blocker triangles in linear time.

3.4. Extensions

In this section, we consider four extensions to the algorithm. These include a method for suppressing the banding artifacts caused by using the same set of light samples for every receiver point, support for alpha matte textures, adaptive antialiasing, and support for volumetric light sources.

Multiple sets of light samples The most important extension involves using multiple sets of light samples for suppressing the banding artifacts. The optimal solution would be using a different light sample set for every receiver point, as is done by a stochastic ray tracer. This can be approximated by using a limited number of distinct light sample sets. In our test scenes, eight light sample sets were enough for suppressing the banding artifacts. Supporting multiple light sample sets is straightforward and requires only three modifications to the algorithm.

First, the bounding polygons Δ_{G_k} of the light sample sets must be constructed so that they bound the light samples of the respective group in all light sample sets. This ensures that the penumbra volumes V_{G_k}, defined by Δ_{G_k} and T, contain all points that may be shadowed by T regardless of which light sample set is used. Second, the hard shadow volumes V_{ℓ_i} must be constructed separately for each light sample set. Finally, the same light sample set must always be used for the same receiver point r_j. This is accomplished by assigning a light sample set to each r_j and using it consistently in the POINT-IN-VOLUME test (Line 70).

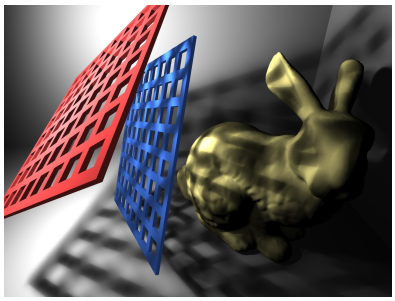
Alpha matte textures In order to support textures with alpha matte, i.e. with both opaque and transparent texels, a texture fetch needs to be performed in addition to the POINT-IN-VOLUME test on Line 70. We have not implemented this in our prototype implementation, and all polygons are considered opaque in our test scenes. The texture must be sampled at the intersection of the $\ell_i \rightarrow r_j$ ray and the blocker triangle. Because of the additional cost of computing the texture sampling location and performing the texture fetch, it is probably best to process the alpha matte textured blocker triangles only after all opaque blocker triangles have been processed. This can be achieved by placing the alpha matte textured triangles in separate buckets during the coarse sorting of the blocker triangles.

Adaptive antialiasing Since our algorithm requires that all receiver points r_j are known before processing the blocker triangles, adaptive antialiasing cannot be performed in a single pass. Instead, a feasible approach is to first render the image with one sample per pixel, then determine the pixels that need antialiasing, and perform a second pass with multiple samples taken at these pixels.

Volumetric light sources Our approach is not limited to planar light sources. Volumetric light sources can be supported simply by replacing the bounding polygons of the light source and the light sample groups with convex bounding volumes. The planes of the penumbra volumes can then be computed exactly as with bounding polygons, since the same method works for convex polygonal volumes as well.

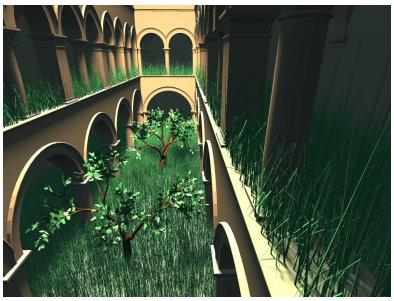
4. Results

We compared our algorithm against a commercial ray tracer (Mental Ray 3.2) in three test scenes shown in Figure 4. The simple GRIDS scene has relatively few triangles, but the penumbra regions are very large. The FLOWERS scene features procedurally generated foliage consisting of a large number of triangles. In SPONZA, a low-polygon mesh is combined with dense procedural grass and two relatively low-polygon trees. All test scenes are illuminated with a single rectangular area light.



GRIDS, 73K triangles

FLOWERS, 903K triangles



SPONZA, 1.27M triangles

Figure 4: *The test scenes used for measuring the shadow computation speed.*

scene	**output resolution**	sort time	hierarchy build time	penumbra cast time	samples mem	hierarchy mem	mask mem	**total time**	**comp. time**	**performance ratio**
GRIDS	640×480	0.3	1.6	50.4	5	3	7	52.3	470.9	9.0
	1280×960	0.3	6.8	131.4	19	12	27	138.5	1863.9	13.5
	2560×1920	0.3	33.1	443.2	75	47	106	476.6	7938.1	16.7
1K smp	1280×960	0.3	6.8	275.1	19	12	108	282.2	7360.5	26.1
4× light	1280×960	0.3	6.8	382.1	19	12	29	389.2	1949.1	5.0
FLOWERS	640×480	13.4	3.1	390.4	5	2	3	406.9	580.3	1.4
	1280×960	13.4	12.8	580.1	19	9	11	606.3	2118.2	3.5
	2560×1920	13.4	57.7	993.6	75	36	43	1064.7	8315.9	7.8
1K smp	1280×960	13.4	12.8	1947.5	19	9	44	1973.7	8245.4	4.2
4× light	1280×960	13.4	12.8	1080.9	19	9	15	1107.1	2293.6	2.1
SPONZA	640×480	6.5	4.8	107.7	5	3	8	119.0	534.4	4.5
	1280×960	6.5	21.1	218.2	19	12	31	245.8	2014.6	8.2
	2560×1920	6.5	120.0	572.5	75	47	122	699.0	7970.7	11.4
1K smp	1280×960	6.5	21.1	650.9	19	12	124	678.5	7890.6	11.6
4× light	1280×960	6.5	17.5	1125.0	19	12	33	1149.0	2249.1	2.0

Table 2: *The results of the benchmark renderings made for the three test scenes. For each scene, five renderings were made with our prototype implementation and the comparison method. All times are in seconds and memory consumption figures are in megabytes. Columns* sort time, hierarchy build time *and* penumbra cast time *refer to the total time taken by coarse blocker sorting, building the receiver point hierarchy (potentially multiple times) and performing the penumbra casting, respectively. Columns* samples mem, hierarchy mem *and* mask mem *refer to the memory usage of the buffer used for storing the receiver points, the peak memory usage of the receiver point hierarchy and the peak memory usage of the bit masks for the receiver points, respectively. The* total time *column tells the total shadow rendering time of our algorithm, and the* comp. time *column tells the shadow rendering time taken by the comparison method. The* performance ratio *is the ratio between the shadow rendering time of the comparison method and the total shadow rendering time of our algorithm. All renderings used 256 samples for sampling the area light source, except those on rows marked with* 1K smp *that used 1024 light samples. In the renderings on rows marked with* 4× light, *the surface area of the light source was quadrupled.*

We performed five renderings for all test scenes. Three of the renderings used 256 light samples at resolutions 640×480, 1280×960 and 2560×1920. In order to measure the scalability of our method with respect to the number of light samples, a 1280×960 rendering was made with 1024 light samples. Finally, the sensitivity of our algorithm with respect to the size of the area light source was measured by performing a 1280×960 rendering with the surface area of the light source quadrupled, using 256 light samples.

Only the time taken by shadow computation was accounted for. For the comparison method, this was accomplished by performing the renderings both with and without shadows, and calculating the difference in the rendering times. Based on initial tests, we chose to use 32 light samples per light sample group and 16 receiver points in the leaf nodes of the receiver point hierarchy. The light samples were distributed according to a jittered grid distribution, which enabled straightforward grouping. In addition, we used 8 distinct light sample sets for suppressing the banding artifacts.

Our prototype implementation considers only the geometry of the scene, and thus cannot handle materials with alpha matte texture. Therefore, we changed the materials of all test scenes to opaque white for the renderings with the comparison method, thereby making the results comparable. Of our test scenes, only FLOWERS features a small number of alpha matte textured surfaces. The images in Figure 4 are rendered with original materials that show the structure of the scenes better than the black-and-white renderings produced by the benchmark renderings.

All tests were run on a laptop with 1.6 GHz Intel Pentium M processor and 1.5 GB of memory. The results of the benchmark renderings are shown in Table 2. We see that our algorithm scales much better than the comparison method with respect to the output resolution. This can be expected because of the hierarchical processing of the receiver points r_j. The scalability with respect to the number of samples used for sampling the light source is also good. The performance of our algorithm degrades when the size of the area light source is increased, due to the enlarged size of the penumbra volumes and consequently diminished efficiency of hierarchical culling. Since we effectively compute the shadows analytically until processing the receiver points r_j, the amount of redundant work grows when the light samples ℓ_i are located sparsely. The memory usage of our algorithm is observed to remain at an acceptable level, even though the storage cost of the receiver points is quite high especially with the highest output resolutions.

5. Discussion and Future Work

We have shown in several tests that the performance of our algorithm compared favorably to tracing shadow rays, establishing that tracing shadow rays is not the only realistic option for computing physically-based soft shadows. The proposed method scales well with respect to the number of light samples and the output resolution. Because the entire scene geometry does not need to be considered at any point during the rendering, the method is particularly well suited for rendering soft shadows from procedural geometry or very complex scenes. Since no spatial hierarchy is needed for the scene geometry, our algorithm is able to handle dynamic scenes without the overhead of rebuilding such hierarchy for every frame in an animation sequence. In this section we discuss the algorithm's limitations, potential improvements and extensions, as well as some thoughts on future work.

The computational complexity of our algorithm differs fundamentally from ray tracing. Denoting the number of receiver points, light samples and triangles as R, L and T, respectively, we may consider the computational complexity of both ray tracing and our algorithm. With the very crude assumption that every $\ell_i \rightarrow r_j$ relation is blocked by a separate triangle, ray tracing has execution time of complexity $O(\mathsf{RL}\log\mathsf{T})$, whereas the respective complexity for our algorithm is $O(\mathsf{T}\log\mathsf{R}\log\mathsf{L})$, assuming a light sample hierarchy with $\log\mathsf{L}$ levels. It is however uncertain whether a full light sample hierarchy (instead of a three-level hierarchy) would be beneficial in practice. Shadow caching for ray tracing and the optimizations for our algorithm (Section 3.3) affect the output-sensitivity of both algorithms substantially, making theoretical analysis more complicated. In addition, it is usually the case that most triangles block more than one $\ell_i \rightarrow r_j$ relation, and not all relations become blocked. Both of these benefit our algorithm more than ray tracing. In particular, our algorithm performs practically no work for completely lit receiver points. The memory consumption complexity for ray tracing with a hierarchical acceleration structure is $O(\mathsf{T})$, while for our algorithm it is $O(\mathsf{RL})$, again exhibiting the exact transpose of ray tracing. Because of the different execution time and memory consumption complexities, it is always possible to construct cases where one algorithm performs better than the other.

The algorithm could be directly extended to handle semi-transparent shadow casters, but that would unfortunately require 24 times more mask memory, i.e., an RGB value instead of a single bit for each $\ell_i \rightarrow r_j$ relation. We plan to attack this problem by augmenting each relation with a bit that indicates whether or not the relation is blocked by semi-transparent blockers. The relations blocked only by semi-transparent blockers can then be treated in a second shadow rendering pass, e.g., by processing a certain maximum number of semi-transparent relations at a time or by using shadow rays.

As observed in Section 4, increasing the size of the area light source decreases the performance of our algorithm. This undesirable phenomenon could perhaps be tackled by extending the method of maintaining active light sample groups during the hierarchy traversal to individual light samples, and choosing the appropriate granularity adaptively. The good scalability of our algorithm with respect to the number of light samples partially alleviates the problem already, since more samples are generally needed for larger area light sources [ARBJ03].

It is possible to extend the algorithm to process multiple blocker triangles simultaneously. This would complicate the geometry of the penumbra volumes, but also reduce the number of hierarchy traversals and the number of receiver points that are temporarily classified to be in penumbra. This might improve the performance in many scenes.

The algorithm has two desirable characteristics that might make it feasible for hardware implementation in the future: the processing is performed one triangle at a time and no spatial hierarchy of the scene is required.

Acknowledgements We thank Jaakko Lehtinen, Janne Kontkanen and Lauri Savioja for fruitful discussions. The original Sponza Atrium model by Marko Dabrovic, RNA studio, www.rna.hr. This research was funded by the National Technology Agency of Finland, Bitboys, Hybrid Graphics, Nokia and Remedy Entertainment. Timo Aila was partially supported by ATI.

References

[AAM03] ASSARSSON U., AKENINE-MÖLLER T.: A Geometry-Based Soft Shadow Volume Algorithm using Graphics Hardware. *ACM Transactions on Graphics, 22*, 3 (2003), 511–520.

[AAM04] AILA T., AKENINE-MÖLLER T.: A Hierarchical Shadow Volume Algorithm. In *Graphics Hardware 2004* (2004), pp. 15–23.

[AL04] AILA T., LAINE S.: Alias-Free Shadow Maps. In *Proceedings of Eurographics Symposium on Rendering* (2004), pp. 161–166.

[Ama84] AMANATIDES J.: Ray Tracing with Cones. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)* (1984), pp. 129–135.

[ARBJ03] AGARWAL S., RAMAMOORTHI R., BELONGIE S., JENSEN H. W.: Structured importance sampling of environment maps. *ACM Transactions on Graphics 22*, 3 (2003), 605–612.

[ARHM00] AGRAWALA M., RAMAMOORTHI R., HEIRICH A., MOLL L.: Efficient Image-Based Methods for Rendering Soft Shadows. In *Proceedings of ACM SIGGRAPH 2000* (2000), pp. 375–384.

[Arv95] ARVO J.: Applications of Irradiance Tensors to the Simulation of Non-Lambertian Phenomena. In *Proceedings of ACM SIGGRAPH 95* (1995), pp. 335–342.

[BEW*98] BISHOP L., EBERLY D., WHITTED T., FINCH M., SHANTZ M.: Designing a PC Game Engine. *IEEE Comput. Graph. Appl. 18*, 1 (1998), 46–53.

[BWG03] BALA K., WALTER B., GREENBERG D. P.: Combining Edges and Points for Interactive High-Quality Rendering. *ACM Transactions on Graphics 22*, 3 (2003), 631–640.

[CF92] CHIN N., FEINER S.: Fast Object-Precision Shadow Generation for Area Light Source using BSP Trees. In *Symposium on Interactive 3D Graphics archive* (1992), pp. 21–30.

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed Ray Tracing. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)* (1984), pp. 137–145.

[Cro77] CROW F.: Shadow Algorithms for Computer Graphics. In *Computer Graphics (Proceedings of ACM SIGGRAPH 77)* (1977), pp. 242–248.

[CW93] COHEN M. F., WALLACE J. R.: *Radiosity and Realistic Image Synthesis*. Academic Press Professional, 1993.

[DDP97] DURAND F., DRETTAKIS G., PUECH C.: The Visibility Skeleton: A Powerful and Efficient Multi-Purpose Global Visibility Tool. In *Proceedings of ACM SIGGRAPH 97* (1997), pp. 89–100.

[DF94] DRETTAKIS G., FIUME E.: A Fast Shadow Algorithm for Area Light Sources Using Back Projection. In *Proceedings of ACM SIGGRAPH 94* (1994), pp. 223–230.

[GH98] GHAZANFARPOUR D., HASENFRATZ J.-M.: A Beam Tracing with Precise Antialiasing for Polyhedral Scenes. *Computer Graphics 22*, 1 (1998), 103–115.

[Hec92] HECKBERT P.: Discontinuity Meshing for Radiosity. In *Third Eurographics Workshop on Rendering* (1992), pp. 203–215.

[Hei91] HEIDMANN T.: Real shadows, real time. *Iris Universe 18* (1991), 28–31.

[HG86] HAINES E. A., GREENBERG D. P.: The Light Buffer: A Ray Tracer Shadow Testing Accelerator. *IEEE Comput. Graph. Appl. 6*, 9 (1986), 6–16.

[HH84] HECKBERT P., HANRAHAN P.: Beam Tracing Polygonal Objects. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)* (1984), pp. 119–127.

[HLHS03] HASENFRATZ J.-M., LAPIERRE M., HOLZSCHUCH N., SILLION F.: A Survey of Real-Time Soft Shadows Algorithms. *Computer Graphics Forum, 22*, 4 (2003), 753–774.

[JMB04] JOHNSON G. S., MARK W. R., BURNS C. A.: *The Irregular Z-Buffer and its Application to Shadow Mapping*. Tech. rep., The University of Texas at Austin, Department of Computer Sciences, April 2004.

[LTG92] LISCHINSKI D., TAMPIERI F., GREENBERG D. P.: Discontinuity Meshing for Accurate Radiosity. *IEEE Comput. Graph. Appl. 12*, 6 (1992), 25–39.

[NN83] NISHITA T., NAKAMAE E.: Half-Tone Representation of 3-D Objects Illuminated by Area Sources or Polyhedron Sources. In *IEEE Computer Software and Application Conference* (1983), pp. 237–242.

[NN85] NISHITA T., NAKAMAE E.: Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection. In *Computer Graphics (Proceedings of ACM SIGGRAPH 85)* (1985), pp. 23–30.

[PSS98] PARKER S., SHIRLEY P., SMITS B.: *Single Sample Soft Shadows*. Tech. rep., University of Utah, UUCS-98-019, 1998.

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering Antialiased Shadows with Depth Maps. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)* (1987), pp. 283–291.

[SG94] STEWART A. J., GHALI S.: Fast Computation of Shadow Boundaries using Spatial Coherence and Backprojections. In *Proceedings of ACM SIGGRAPH 94* (1994), pp. 231–238.

[SS98] SOLER C., SILLION F. X.: Fast Calculation of Soft Shadow Textures Using Convolution. In *Proceedings of ACM SIGGRAPH 98* (1998), pp. 321–332.

[STN87] SHINYA M., TAKAHASHI T., NAITO S.: Principles and Applications of Pencil Tracing. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)* (1987), pp. 45–54.

[SWZ96] SHIRLEY P., WANG C., ZIMMERMAN K.: Monte Carlo Techniques for Direct Lighting Calculations. *ACM Transactions on Graphics, 15*, 1 (1996), 1–36.

[TT97] TANAKA T., TAKAHASHI T.: Fast Analytic Shading and Shadowing for Area Light Sources. *Computer Graphics Forum, 16*, 3 (1997), 231–240.

[WPF90] WOO A., POULIN P., FOURNIER A.: A Survey of Shadow Algorithms. *IEEE Comput. Graph. Appl. 10*, 6 (1990), 13–32.