EUROGRAPHICS 2006 / E. Gröller and L. Szirmay-Kalos
(Guest Editors)

Volume 25 (2006), Number 3

An Improved Physically-Based Soft Shadow Volume Algorithm

Jaakko Lehtinen[1,2] Samuli Laine[1,3] Timo Aila[1,3]

[1] Helsinki University of Technology [2] Remedy Entertainment, Ltd. [3] Hybrid Graphics, Ltd.

Abstract

We identify and analyze several performance problems in a state-of-the-art physically-based soft shadow volume algorithm, and present an improved method that alleviates these problems by replacing an overly conservative spatial acceleration structure by a more efficient one. The new technique consistently outperforms both the previous method and a ray tracing-based reference solution in several realistic situations while retaining the correctness of the solution and other desirable characteristics of the previous method. These include the unintrusiveness of the original algorithm, meaning that our method can be used as a black-box shadow solver in any offline renderer without requiring multiple passes over the image or other special accommodation. We achieve speedup factors from 1.6 *to* 12.3 *when compared to the previous method.*

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

1 Introduction

Soft shadows are no doubt one of the key features that distinguish realistic images from those that look unnatural. An ever-increasing number of algorithms have been proposed for the task, but the computation of physically-based soft shadows remains costly, and is avoided in production whenever possible. In practice, soft shadow effects are most often produced using approximate methods or pure hacks.

Proper ("physically correct") determination of a soft shadow requires solving for the exact visibility relations between a pixel being shaded and all points of an area light source. This visibility function is included in an integrand that is evaluated over the light source; this is called the reflectance equation. Most methods discretize the reflectance equation using some sort of stochastic quadrature by placing a large number of sample points on the light source where the integrand (including visibility) is evaluated. Still, the determination whether or not a line-of-sight exists between the point being shaded and possibly several hundreds of these sample points remains slow using traditional methods.

Because of the above-mentioned difficulty, a significant number of soft shadow algorithms are *approximate*; they produce an image that has soft shadows, but they are not based on true visibility but rather on some simplified assumption. There are a number of special cases where approximate algorithms produce physically correct results, but these are exceptions rather than the rule. Furthermore, measuring and predicting the quality of approximations has re-

Figure 1: *Physically-based soft shadows in a 900k triangle scene. Here the previous soft shadow volume method of Laine et al. [LAA*05] is slower than the reference ray tracer, while we achieve a speedup factor of* 12.3 *over the previous method and a factor of* 6.8 *over the reference ray tracer.*

ceived little attention. In contrast, this work focuses on the determination of correct visibility and thus the method presented herein is *not approximate*: It will produce the correct image, bar the noise inherent in any method that evaluates integrals using a limited number of point samples.

Until recently, efficient determination of such physically-based soft shadows was an unsolved problem in most cases of interest, and the best solution has been to bite the bullet and sample the visibility for each light sample using a ray tracer. Despite the often-cited logarithmic complexity of ray tracers, the cost of a shadow query increases linearly with the number of light source samples. Keeping the amount of work at a tolerable level requires, in practice, an adaptive anti-aliasing scheme for avoiding shadow tests where they are not needed, and, of course, using as few light source samples as possible. For instance, texturing helps masking the possible noise in the shadows, thus enabling the use of fewer samples. Also, a voting scheme may be used for skipping work; for instance, if the first 50% of the shadow rays agree, the rest can be skipped. This of course compromises, strictly speaking, the correctness of the result.

Recently, Laine et al. [LAA*05] introduced a novel soft shadow algorithm that takes a fundamentally different approach. We review the method in more detail in Section 2.1. Briefly, the algorithm is based on tracking *visibility events* between a planar area light source and the point being shaded. This technique trades the brute-force per-ray visibility computation to computing the *depth complexity* of each light sample w.r.t. the point being shaded using silhouette information. As demonstrated by the authors, this is often significantly less costly than directly sampling the visibility for each sample. However, the spatial acceleration structure used in the algorithm has several weaknesses that render the algorithm overly sensitive to issues such as the size of the scene and the relative orientations of the light source and the camera. In fact, we show that the algorithm can be "broken" by some very simple cases, in the sense that its performance degrades to the level of the reference ray tracer or even below that. Our present algorithm is a direct successor of this method, but because of a more suitable spatial data structure, performs better than the original algorithm (and the reference ray-tracer) in all our test cases.

Contributions. We identify and analyze the performance problems of the hemicube-like data structure used by Laine et al. in their original soft shadow volume algorithm for identifying the penumbra wedges that potentially affect a point being shaded. Then we present a novel data structure for replacing the hemicube, and demonstrate by several examples that it outperforms the original technique in several situations, particularly in cases where the original method has the greatest difficulties (Figure 1). We stress that our new method has never performed worse than the original method in any of the hundreds of test renderings made during the preparation of this work.

2 Previous Work

This section reviews algorithms that create physically-based soft shadows from area light sources. Additionally, a vast amount of literature exists for generation of approximate soft shadows [RSC87, HLHS03] and hard shadows from point lights [WPF90].

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 variants of importance sampling [SWZ96].

Tracing thick rays Heckbert and Hanrahan [HH84] trace beams instead of individual rays. Occlusion is 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. Ghazanfarpour and Hasenfratz [GH98] describe a variant of beam tracing that does not clip the beam geometry, but instead subdivides the beam recursively until a specified subdivision limit is reached or the beam is either fully lit or fully occluded with respect to a single triangle. Pencil tracing [STN87] processes sets 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.

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. Tanaka and Takahashi [TT97] propose culling methods for efficiently determining the set of objects that can affect the shadowing of a given point.

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. Laine et al. [LAA*05] remove these limitations in the context of ray tracing. As our algorithm is an exten-

sion to their work, the algorithm is reviewed more closely in Section 2.1.

Laine and Aila [LA05] transpose the processing order of ray tracing. Instead of searching for a triangle that blocks the current ray, they find all rays that are blocked by the current triangle. This leads to different scalability characteristics and memory requirements compared to ray tracing.

Tracking visibility events 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. Duguet and Drettakis [DD02] present a numerically robust variant. Stark and Riesenfeld [SR00] describe a robust technique for analytic evaluation of irradiance from an uniformly emitting source to a single point. Discontinuity meshing [Hec92, LTG92] 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.

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.

2.1 Soft shadow volumes for ray tracing

The soft shadow volume method of Laine et al. [LAA*05] is based on the idea of tracking the *changes in depth complexity* of the light source samples as seen from the point being shaded. The depth complexity is defined as the number of occluding surfaces pierced by a ray from the point being shaded to the light sample. As a corollary of noting that the depth complexity between the light source and the point being shaded only changes where there is a silhouette edge between the light source and the point being shaded, they observe that silhouette edges (or more precisely, their projections onto the plane of the light source as seen from the point being shaded) are the generalized derivatives of the depth-complexity function. The technique then simply *integrates* these derivatives to yield a relative depth complexity for each light sample. Since the integral is unique only up to a constant, a single reference ray needs to be cast to the light sample that has the lowest relative depth complexity. If the ray is not blocked, all the light samples that share this same relative depth complexity are visible while the rest are occluded, and if the reference ray is blocked, all light samples are occluded.

The algorithm utilizes the *penumbra wedges* of Akenine-Möller and Assarsson [AAM03] for determining the regions of space in which a given edge potentially affects the visibility of the light source. A penumbra wedge due to an a planar area light source and an edge of an occluder is defined as the convex volume of space from which the generating edge overlaps the light source when projected onto the plane of the light source. The wedges are constructed easily by finding the *separating planes* defined by the light source and the edge. Note that the wedges are always convex regardless of the geometry of the occluders, and thus no restrictions on the occluding geometry are imposed. Naturally, wedges need to be constructed only for edges that are silhouettes as seen from some point on the light source, as only they can affect the depth complexity integration.

Laine et al. use a hemicube-like spatial acceleration structure for finding the set of edges whose projections, as seen from the point being shaded, overlap the light source. The hemicube is centered at the light source, and its size chosen so that it encloses the entire scene. All penumbra wedges are intersected with the sides of the hemicube, and each hemicube cell maintains a list of wedges whose footprint overlaps with the cell even partially. The hemicube is encoded using a quadtree for reducing the memory consumption. Then, a set of wedges that may contain the point to be shaded is obtained by projecting the point onto the hemicube using the center of the light source as the center-of-projection and merely returning the list of wedges from the corresponding hemicube cell. This set is conservative in the sense that it is guaranteed to contain at least all relevant wedges. The avid reader is referred to the original article [LAA*05] for reassurance of the fact that the centered projection indeed does not cause any potential wedges to be missed. However, as we demonstrate in the next section, this projection — a flattening of the three-dimensional wedges onto a two-dimensional surface — is a significant cause of inefficiency because the reported wedge sets often contain a large proportion of wedges that do not actually contain the point to be shaded. As our primary algorithmic contribution, we present a more efficient replacement for the hemicube (Section 4).

After the hemicube lookup, the wedges reported by the hemicube are tested, one-by-one, to determine whether or not they actually contain the point to be shaded, and furthermore whether or not the generating edge is a silhouette as seen from the point. The wedges that do not satisfy both of these criteria are discarded. The wedges that pass this test are exactly the silhouette edges that overlap the light source as seen from the point being shaded.

The second-last stage of the algorithm is the integration of the depth complexity using the silhouette edges. Each edge denotes a relative change in depth complexity, and since integration is a linear operation, all the silhouette edges can be processed separately and the results simply summed. An integer depth complexity counter is maintained for each light sample, and these counters are updated as each edge is processed. Finally, after processing all the silhouette edges, it is obvious that only those samples that have the lowest depth complexity can be visible, but because we have only tracked *relative* changes in depth complexity, they might all be blocked. Whether or not this is the case is determined by shooting a single *reference ray* to one of the samples that share the lowest depth complexity.

3 Performance Problems in the Previous Method

This section discusses the weaknesses of the hemicube used by Laine et al. [LAA*05] for detecting the wedges that potentially contain a point being shaded. We visualize all the example cases in 2D for simplicity. In these 2D illustrations the silhouette edge for which a wedge is constructed is denoted by a red point, and the corresponding wedge is visualized as a light green polygon. The reader may think of the illustrations as cross-sections of a 3D situation. The region of space in which the hemicube reports the wedge as a potential candidate is denoted by the darker red polygon. Performance comparisons that quantify the problems are given in Section 5.

3.1 Orientation of the Light Source and Scene Size

As the original method places the hemicube around the scene, it is obvious that the orientation of the light source and the size of the scene play a role in its efficiency. Figures 2a and 2b depict a simple case of an axis-aligned light source and a rotated one; it is clear that the hemicube footprints of the wedges grow when the bottom side of the hemicube is no longer coplanar with the floor of the scene. This has the effect that the wedge is reported for a larger number of scene points, indicated by the larger size of the red polygon. Also, increasing the size of the scene causes the footprints to grow due to both the degradation of the relative resolution of the hemicube and the fact that the footprints of the wedges grow larger (but only up to a limit, as noted by Laine et al.) as the distance to the hemicube surface increases. This is illustrated in Figures 2a and 2c. Note that the rendered image would be the same in both cases.

3.2 Depth Complexity

As the hemicube essentially projects the geometry of the wedges onto the sides of the hemicube, all 3D information about the location of the wedge is lost. This is demonstrated already by the fact that the red polygon used for marking the region for which the wedge gets reported always stretches all the way to the center of the light source. This has the unfortunate effect that when rendering an image, complex occluding geometry *behind* the surface being shaded causes extra wedges to be reported for the surface points. This is illustrated in Figure 2d.

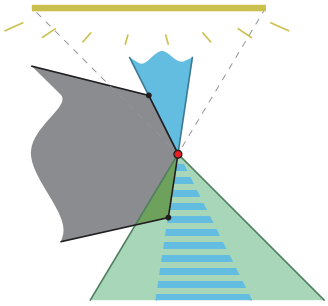

Figure 3: *A 2D cross-section that illustrates silhouette regions. The edge denoted by the red point is a silhouette only in points contained in its silhouette region, denoted by the darker blue region. The penumbra wedge denoted by the lighter green polygon, on the other hand, is the region of space from which the generating edge projects onto the light source. The edge is relevant for depth complexity integration only in receiver points that fall inside both.*

3.3 Silhouette Regions

For an edge shared by two polygons to be relevant in depth complexity integration, it needs to satisfy two criteria: 1) Its projection as seen from the point p being shaded must overlap the light source, and 2) it needs to be a silhouette as seen from p. Points that fall within the penumbra wedge constructed from separating planes satisfy criterion 1. We define the *silhouette region* for an edge shared by two triangles as the region of space where exactly one of the triangles is front-facing; this is the set of points that satisfies criterion 2. In contrast to the penumbra wedge formed by separating planes, the silhouette region is not a simple intersection of halfspaces, but rather an X-shaped "XOR" region of space whose points lie in exactly one of the halfspaces formed by the planes of the triangles associated with the edge. See Figure 3 for an illustration of the relationship of the penumbra wedge and a silhouette region. Edges that belong only to a single polygon cannot be culled using a silhouette region.

Unfortunately for the hemicube, computing the intersection of the penumbra wedge and the silhouette region and then projecting the resulting geometry onto the surface of the hemicube cannot be generally done in a fashion that would preserve the conservatively correct results obtained by the central projection used in the hemicube lookup. Because of this, it is impossible to utilize silhouette regions for tightening the wedge sets reported for query points. This is a major disadvantage that is particularly pronounced in the case of smooth, curved shadow casters, where the silhouette regions are small due to neighboring polygons being almost coplanar. Our new query structure is able to account for the silhouette regions during the query, thus making the query much more efficient.

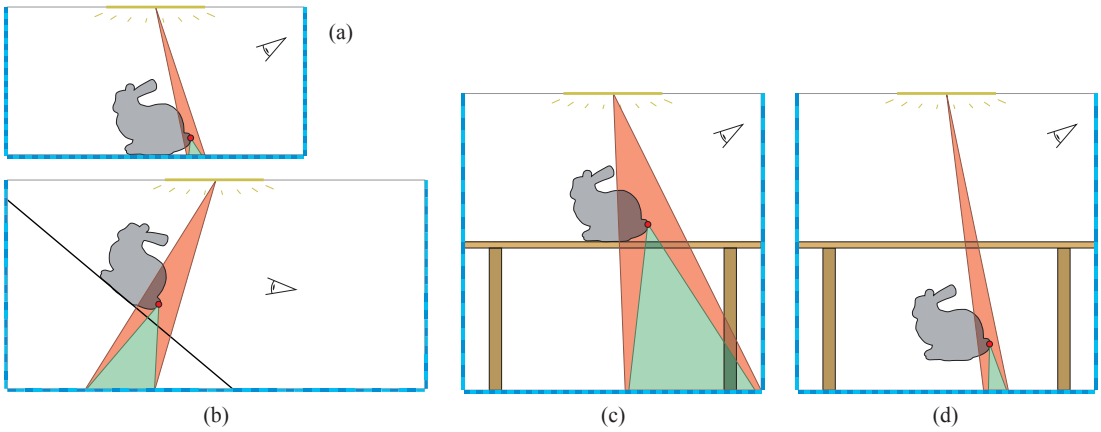

Figure 2: *Illustration of the problems associated with the hemicube structure used by Laine et al. [LAA*05] for detecting penumbra wedges that may contain the point being shaded. The hemicube is denoted by the striped pattern around the scene. A penumbra wedge is denoted by the lighter green polygon, and the darker red polygon depicts the set of scene points for which the hemicube reports the wedge. (a) and (b) Rotating the light source w.r.t. the scene can cause the sizes of wedge footprints on the hemicube to grow, resulting in an overly conservative set of wedges. (c) Increasing the depth of the scene also causes wedge footprints to grow. (d) The hemicube loses all 3D information about the wedge and the query point. None of the wedges generated by the bunny are relevant for the points on the tabletop, but the hemicube reports them nonetheless.*

4 Our Method

As demonstrated in the previous section, the two-dimensional nature of the hemicube causes it to report overly conservative sets of wedges in several cases. Most importantly, its performance is not always predictable; for instance, rotating a light source may affect the shadows only slightly, but can result in a significant penalty in the original algorithm of Laine et al. [LAA*05] through the growth of the footprints of the wedges on the hemicube surface. We address these issues by completely replacing the hemicube and related computations, whereas the rest of the shadow volume algorithm stays fundamentally same as the one described by Laine et al. In particular, the process of projecting the silhouette edges onto the plane of the light source and integrating them is not affected by our improved acceleration structure.

Our novel structure effectively rasterizes the wedges into a hierarchical 3D grid whose cells contain lists of wedges that either intersect or contain the cell. This removes the inefficiency caused by the 2D projection inherent in the hemicube method. The grid is implemented as an axis-aligned 3D BSP tree that covers the scene. If a wedge fully contains a given internal node of the tree, the wedge is stored into this node instead of propagating it down towards the leaves of the tree, as the wedge would cover all the child nodes anyway. Finding the list of potential wedges for a given query point proceeds by walking down the tree, starting from the root node, always continuing to the child node that contains the query point, and collecting the wedges from every BSP node visited during this traversal.

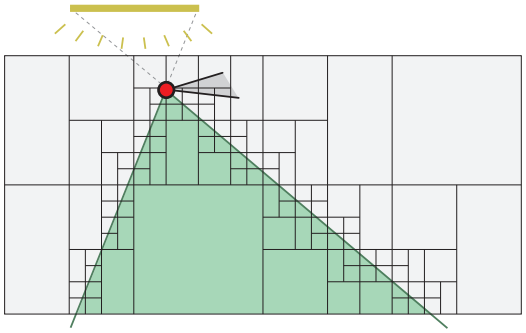

Figure 4: *A 2D cross-section illustrating the "rasterization" of a wedge into a hierarchical grid. The data structure used for determining the regions occupied by a given wedge is an axis-aligned BSP tree, whose nodes keep track of wedges that either intersect or fully contain the node.*

4.1 Lazy Construction of the Wedge BSP Tree

A naïve implementation would construct the wedge BSP tree once as a pre-process. However, this would lead to substantial costs in both memory consumption and processing time, as our BSP is a true three-dimensional structure. In effect, if the wedges occupying the whole volume of the scene were determined, the number of leaf nodes in the BSP tree would grow in $O(l^3)$, where l is the "length" of the scene. This would also be a significant waste because shadow queries are typically issued only on or near the surfaces of the scene, and accurate knowledge of wedge occupancy in free space would have little value in most cases. In order to obtain a more output-sensitive algorithm, we build

the tree lazily, i.e., intermix the construction of the tree with the wedge queries, and *only subdivide the tree in regions that actually receive queries.* This approach has the major advantage that computation and memory resources are not wasted on regions of the scene where shadow queries are not made. This is a significant win, and as our pseudocode in Figure 5 demonstrates, has a simple implementation.

The initialization of our query mechanism simply creates a root node that encloses the whole scene. Note that this implies the reasonable assumption that shadow queries are only issued for points inside the bounding box of the scene. All the wedges found in the initialization stage of the shadow algorithm are initially assigned to the root node. After this the initialization is complete.

When a shadow query for a point p is performed, the function WEDGE-QUERY finds the list of wedges that potentially affect the shadow computation for point p. This function implements our new query structure. The query function recurses down the BSP tree, starting from the root. At each node, the function first checks whether the current node needs to be split, and if so, splits it. If the node has any remaining wedges associated with it, they are appended to the query result — if the current node is an internal node, these are the wedges that fully contain the node, and in case of a leaf node, wedges that partially intersect the node. Then, if the current node is not a leaf, the child node that contains the query point is determined, and the query recurses to that node. See Figure 5 for pseudocode.

When a node is split, all the wedges associated with it that do not fully contain the node are tested against both the child nodes. Wedges that intersect either child node are added to the corresponding list associated with the child node. Note that a wedge may be inserted to both children. Wedges that fully contain the node being split are not propagated down, but are instead left in the node. In case of a single wedge, this process can be thought of as adaptively stopping the subdivision in regions that are fully occupied by the wedge, as illustrated in Figure 4. This, incidentally, is conceptually analogous to the hierarchical 2D rasterization used by Laine et al. for constructing and encoding the hemicube. The test that determines whether or not a wedge intersects or covers a given BSP node is implemented using a fast test that compares an axis-aligned box against an intersection of half-spaces. The test is optimized by maintaining an *active plane mask* for each wedge stored in each BSP node [BEW*98].

Nodes are always split in the middle along their longest axis — this simple strategy ensures that the resulting query structure is independent of the order of the shadow queries. We considered using information of the query points for guiding the splitting process, for instance keeping a single query point in each node and always splitting in a way that separates two query points, but we concluded that this would result in badly balanced trees.

The criterion that determines whether or not to split a node

```
procedure INIT-BSP(node root_node, wedge list all_wedges)
 root_node.bbox ← scene.bbox
 root_node.wedges ← all_wedges

procedure SPLIT-LEAF(node n, int wedges_above)
 if n.bbox.max_axis_length < ε then return
 if n.wedges.size ≤ 0.25 * wedges_above then return
 construct leaf nodes n.left and n.right
 {n.left.bbox, n.right.bbox} ← split n.bbox along longest axis
 for each w in n.wedges do
   if w does not contain n.bbox entirely then
     remove w from n.wedges
     if w intersects n.left.bbox then add w in n.left.wedges
     if w intersects n.right.bbox then add w in n.right.wedges
   end if
 end for

procedure WEDGE-QUERY(point pt)
 wedge_list ← empty list
 n ← root_node
 repeat
   if n is a leaf node then SPLIT-LEAF(n, wedge_list.size)
   append n.wedges to wedge_list
   if n is a leaf node then return wedge_list
   if pt is inside n.left.bbox then
     n ← n.left
   else
     n ← n.right
   end if
 end repeat
```

Figure 5: *Pseudocode of the combined wedge query and BSP construction.*

is composed of several heuristics. First, if the node has no wedges associated with it, obviously nothing needs to be done. We also limit the recursion by the condition that further subdivision is stopped whenever the number of wedges in a node is less than 25% of the number of wedges already deposited in the nodes *above* the current node. This condition essentially stops overly accurate subdivision of the space in the case of diminishing returns, i.e., where the resulting set of wedges for points in the node could be shrunk no more than by a small amount even in the limit of infinite subdivision. We also saw fit to prevent very deep trees by enforcing a minimum size for the nodes. The threshold used for all results in this paper is $1/200$ of the longest dimension of the scene. We certainly admit that the use of this sort of adjustable constant sounds dubious, but in the course of this work we tried several other stopping criteria, none of which ever resulted in better performance than the one presented here. These criteria that were not incorporated into the final algorithm included 1) always splitting a leaf node after a given number of queries has terminated in it, and 2) only relying on the "less than 25% remaining" rule described above. As use of these criteria consistently resulted in worse performance and higher memory usage, we conclude that our rule-of-thumb size limitation is relatively robust in practice.

4.2 Accounting for Silhouette Regions

As our BSP is a true three-dimensional structure unlike the hemicube used in the original algorithm, it is possible to account for the silhouette region of the edge that generates the penumbra wedge. As a result the wedge is listed only in nodes that intersect both the wedge and its associated silhouette region. This is implemented in the geometric test that determines whether or not a wedge intersects a BSP node. The active plane mask (which is used for tracking the halfspaces that fully contain the node) is augmented with four extra bits that indicate the status of the node with respect to the two triangle planes — the two bits per plane indicate whether or not the node is fully inside or fully outside the corresponding plane. As with the separating planes that form the wedge, these bits are used for skipping the tests when the result has been confirmed for a node and hence for all its children as well. As demonstrated by our results in Section 5, accounting for the silhouette regions increases the efficiency of our method significantly.

5 Results and Discussion

This section discusses the results obtained using our novel acceleration structure. We used four scenes of increasing geometric complexity, lit by relatively large area light sources. Figure 6 shows the renderings used for performance measurements. We tested our improved method against both the method of Laine et al. [LAA*05] that uses the hemicube for determining potential wedges, and a relatively well-optimized ray tracer implemented using cache-efficient data structures. As the main point of this paper is to prove the superiority of our method over the previous technique of Laine et al., we chose not to go through the extra trouble of comparing against a commercial tracer. The implementation of the method of Laine et al. was the same as that used by the authors in their original paper. The same ray tracer was used by both the comparison method and our new algorithm for tracing the reference rays, and it was used for computing the reference results as well. Its performance was 231k-684k shadow rays per second when computing the reference solutions.

The tests were performed on a PC with a 2.8GHz Pentium 4 processor and 2GB of memory. All timings are in seconds, and only the time taken by shadow queries is reported; the time taken by the tracing of primary rays and the computation of illumination based on the visibilities of light samples is excluded, as these are the same for all three methods. All the images were rendered in $16:9$ aspect ratio in 960×540 resolution. A simple two-pass adaptive anti-alias scheme was used with all algorithms. The anti-aliasing resulted in 9-34% additional primary rays. All renderings were made using 256 samples on the light source. The light samples were distributed in a jittered grid, and a separate pattern was randomly chosen for each pixel from a fixed set of 64 different patterns. As all algorithms – ours, that of Laine et al. and the reference method – produce the same image except for the effect of the random light sampling patterns, we only present one image of each scene.

Scene	Wedges reported by query structure		Validated wedges	Acceptance ratio	
	HC	BSP		HC	BSP
Columns	883.3	137.8	62.5	7.1%	45.4%
Racecar	2708.9	489.9	298.8	11.0%	60.1%
Sponza	1470.0	581.2	409.9	27.9%	70.5%
Max	16515.3	443.2	86.9	0.5%	19.6%

Table 3: *Average number of wedges reported* per shadow query *by the hemicube (HC) of Laine et al. and our BSP. Also reported are the average numbers of validated wedges, i.e., those wedges that passed the projection and silhouette tests, and the ratio of accepted wedges to reported wedges.*

As can be seen from Table 1, our new method performs better than the method of Laine et al. and the reference ray tracer in all of our test cases. Speedup factors range from 1.6 to 12.3 over the previous method, and from 2.5 to 15.3 over the reference solver. We note, however, that our test cases have been constructed in ways that display the shortcomings of the previous method — cases can be constructed easily where much larger speedups over the reference solution are obtained by both our new technique and the method of Laine et al., but typically the difference between the two soft shadow volume algorithms is small in such cases. We stress again that our new method has never been outperformed by the old technique.

Table 2 provides a detailed breakdown of the time consumption. By far the largest difference, compared to Laine et al., is the time spent on edge validation. This is a direct result of our new data structure providing a much tighter set of wedges for consideration, as shown in Table 3. The time taken by projection, integration and tracing the reference rays is noticeably higher in the old method. We verified explicitly that both our method and the old algorithm perform exactly the same amount of work in these stages, and conclude that the differences must be attributed to cache pollution caused by the larger wedge sets processed by the old method in the edge validation loop.

The memory consumption of our algorithm is larger than that of the hemicube-based method (see Table 1). However, since the space is subdivided only where queries are made, the footprint is more sensitive to the surface area visible in the picture rather than the sheer volume of the scene. As is the case with the method of Laine et al., the memory consumption scales linearly w.r.t. the number of wedges generated by the blockers. This discourages the use of overly-tessellated scenes and exceptionally large light sources.

Figure 7 contains false-color images that indicate, for each pixel, the amount of *extra* wedges reported by the query structure for both the new method and the hemicube used by the comparison algorithm. These are wedges that were rejected during validation, i.e., they correspond to edges that are not silhouette edges whose projection overlaps the light

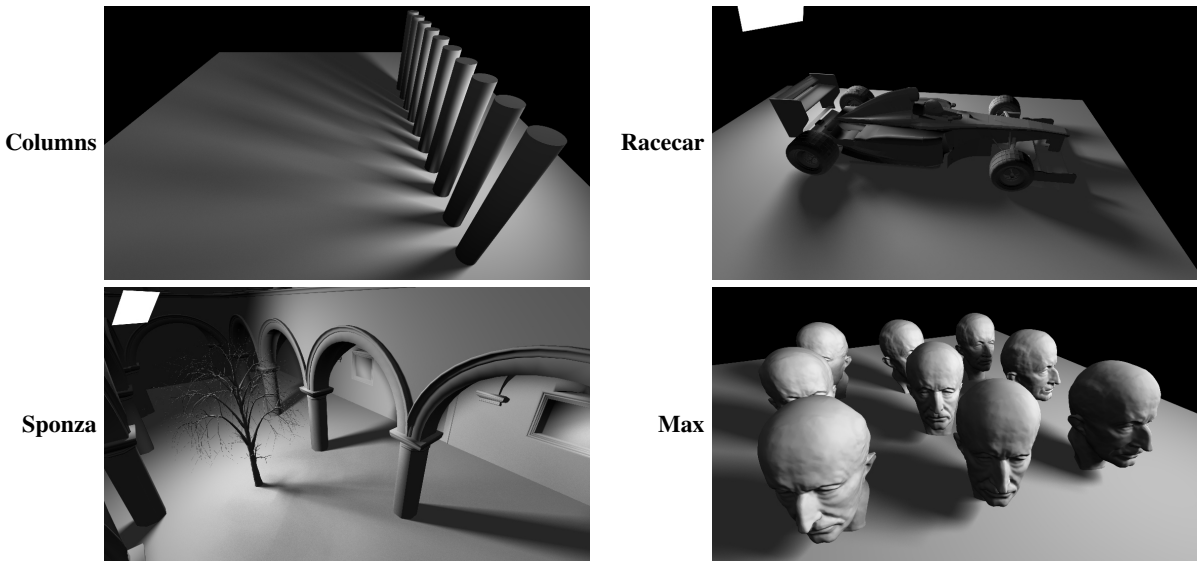

Figure 6: *Test scenes used for performance measurements. See Table 1.*

Scene	#Tris	#Primary rays	Total shadow time			Speedup factor vs.		Memory (MB)	
			Ray tracing	Old	New	Ray tracing	Old	Old	New
Columns	16k	567820	162.79	29.56	10.62	15.3	2.8	9.3	10.9
Racecar	68k	569128	131.03	111.02	52.60	2.5	2.1	13.6	73.7
Sponza	109k	695160	762.85	174.02	110.19	6.9	1.6	10.4	45.2
Max	900k	574208	270.48	490.76	39.80	6.8	12.3	103.5	79.6

Table 1: *Total timings, speedup factors and memory consumption for the reference ray tracer, the old method of Laine et al. [LAA*05] and our new method.*

Scene	Initialization		BSP build + query	Edge validation		Projection + integration		Reference ray		Misc time	
	Old	New	New only	Old	New	Old	New	Old	New	Old	New
Columns	1.24	0.04	1.22	21.66	2.88	5.01	4.84	0.76	0.72	0.88	0.91
Racecar	2.02	0.15	6.47	79.99	18.89	26.89	25.42	1.37	0.93	0.75	0.73
Sponza	1.82	0.27	5.49	101.23	38.52	63.37	59.97	6.06	4.44	1.54	1.51
Max	15.21	1.98	10.99	458.78	16.37	10.60	7.43	4.76	2.11	1.40	0.90

Table 2: *Timing statistics breakdown for our new method in comparison to the old method of Laine et al. [LAA*05]. The total time taken by the shadow queries can be found in Table 1.*

source as seen from the point to be shaded. The first column of false-color images visualizes the extra work performed by the old hemicube-based algorithm. The second column indicates the number of wedges reported by our novel 3D structure if the silhouette regions are *not* accounted for. Finally, images in the third column depict the number of extra edges reported by our new query structure. These images, along with timing breakdowns presented in Table 4 clearly show that intersecting the wedges with their silhouette regions results in substantial savings. The color ramp used in the visualizations is shown in Figure 8.

Figure 9 depicts a scene where a race car is lit by an area light source. In the left image the light source is aligned with the ground plane, and in the right image the light source is rotated by 45 degrees along two axes while retaining its center in the same position. In the axis-aligned case the old and new methods both perform well: The new method requires 5.2 seconds and the old method 7.6 seconds. The situation is drastically different in the case of the rotated light source — the new method is essentially unaffected and requires 5.7 seconds in total, while the old method bogs down to 35.9 seconds. The false-color images again depict the number of superfluous wedges reported by the query structure. This test clearly demonstrates the effect of the relative orientation of the light source and the scene, as discussed in Section 3.1.

Figure 8: *The color ramp used in efficiency visualization. Absolute scales are reported in the respective figures.*

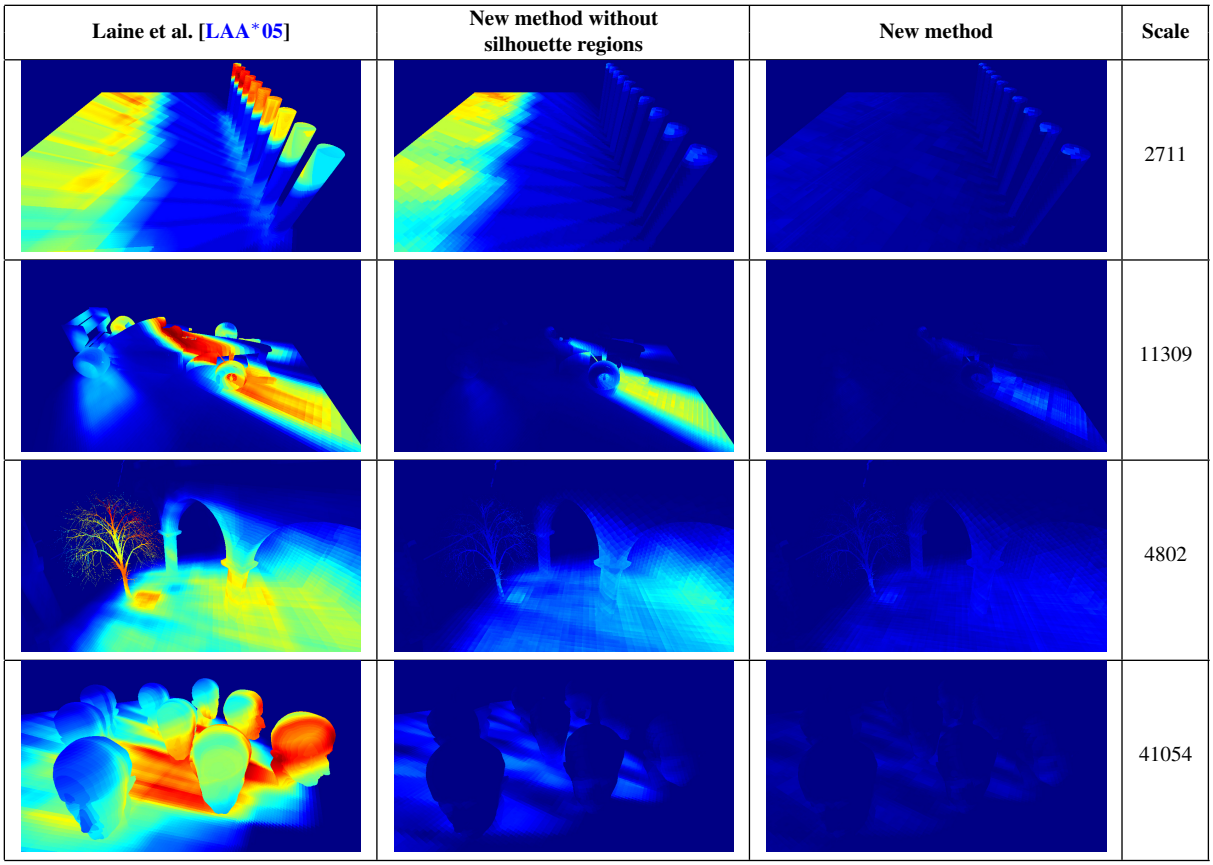

Figure 7: *Visualization of the unnecessary extra work performed by the method of Laine et al. and our method. The second column shows results obtained using our novel query structure* without *taking the silhouette regions into account. The scale denotes the maximum number of extra wedges, indicated by the darkest red in the visualization.*

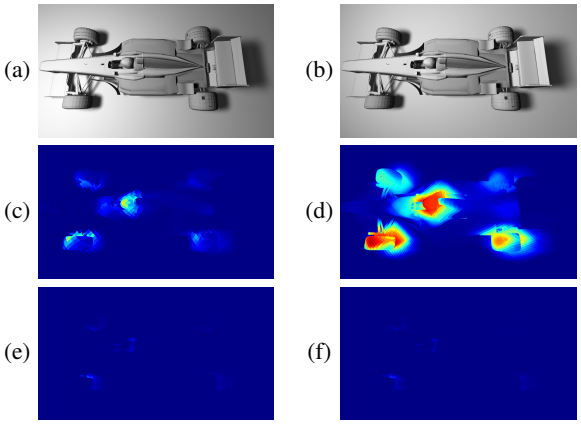

Figure 9: *(a) The scene with an aligned light source. (b) The scene with a rotated light source. (c) and (d) Inefficiency visualization for the method of Laine et al. (e) and (f) Inefficiency visualization for novel method.*

We performed an additional test confirming the superior performance of our 3D query structure in case of complex occluding geometry *behind* the surfaces shown in the image. We set up a situation where the light source is behind the camera that is looking head-on to one of the pillars in the Sponza scene so that the complex tree model lies directly behind the pillar, but only the pillar is visible in the image. The setup is directly analogous to Figure 2d. Our method rendered the image in 3.7 seconds, while the hemicube-based method took 91.8 seconds.

Scene	BSP build + query		Edge validation		Total time	
	w/o SR	SR	w/o SR	SR	w/o SR	SR
Columns	1.8	1.2	9.7	2.9	18.2	10.6
Racecar	8.1	6.5	36.0	18.9	73.0	52.6
Sponza	5.2	5.5	54.7	38.5	128.6	110.2
Max	16.0	11.0	63.4	16.4	94.8	39.8

Table 4: *The effect of accounting for silhouette regions. All times are in seconds, and give the timing breakdown of the relevant portions of the algorithm. The "SR" columns show timings for our method when silhouette regions are properly accounted for, and the "w/o SR" columns show timings when only the penumbra wedges are used.*

6 Discussion and Future Work

By taking up the slack due to the projective nature of the hemicube of Laine et al., our new method brings the soft shadow volume algorithm closer to its theoretical maximal performance. Any query structure cannot, of course, change the fact that larger light sources subtend larger solid angles, and thus the number of relevant wedges generated by them is larger, causing more time to be spent in the integration. However, as we have demonstrated, the improved soft shadow volume algorithm performs well in a larger class of rendering situations than the previous method, and we believe there are many practical cases that would benefit greatly from use of our improved technique.

It would be possible to adapt our method to run in a smaller or perhaps even fixed memory footprint by collapsing branches of the wedge BSP in regions that have not received queries for some time. This would of course be most efficient when the queries are issued in a spatially coherent order, but we believe that for instance typical pixel-order traversal should fulfill this criterion sufficiently well. At the moment we see no other obvious algorithmic improvements, except perhaps treating the light samples fully hierarchically when integrating the visibility events.

Acknowledgments The original Sponza Atrium model by Marko Dabrovic, RNA studio, www.rna.hr. This work was sponsored in part by the Helsinki Graduate School in Computer Science and Engineering, the Academy of Finland, the National Technology Agency of Finland, Anima Vitae, Bitboys, Hybrid Graphics, and Remedy Entertainment.

References

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

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

[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 Trans. Graph. 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* (1992), pp. 21–30.

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

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

[DD02] DUGUET F., DRETTAKIS G.: Robust epsilon visibility. *ACM Trans. Graph. 21*, 3 (2002), 567–575.

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

[DF94] DRETTAKIS G., FIUME E.: A Fast Shadow Algorithm for Area Light Sources Using Back Projection. In *Proc. 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 *Proc. Eurographics Workshop on Rendering* (1992), pp. 203–215.

[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 (Proc. 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.

[LA05] LAINE S., AILA T.: Hierarchical penumbra casting. *Computer Graphics Forum 24*, 3 (2005), 313–322.

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

[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.

[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 (Proc. 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.

[SR00] STARK M. M., RIESENFELD R. F.: Exact Illumination in Polygonal Environments using Vertex Tracing. In *Proc. Eurographics Workshop on Rendering* (2000), pp. 149–160.

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

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

[SWZ96] SHIRLEY P., WANG C., ZIMMERMAN K.: Monte Carlo Techniques for Direct Lighting Calculations. *ACM Trans. Graph. 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.