Prefiltered Cross-Section Occluders

Timothy Condon, Bruce Walter, Kavita Bala & Donald Greenberg
Cornell University
Ithaca, NY

***Abstract*—We introduce an impostor-based visibility technique to provide approximate, average visibility for point-to-cluster and cluster-to-cluster visibility queries. Processing each object in a view-independent manner allows us to generate a mip-map-like hierarchy of approximate visibility information that can report approximate average visibility over conical shafts or conical shaft frustums. We demonstrate how these impostors can be used to approximate smooth soft shadow calculation, and improve the quality of lightcuts and multi-dimensional lightcuts. In addition, we describe how to use approximate visibility of shafts to estimate a tighter error bound for the lightcuts visibility term, yielding speedup by reducing over-aggressive cut refinement.**

***Keywords*-approximate occlusion, visibility, shadowing, impostors**

I. Introduction

The performance of visibility queries plays a significant role in the total cost of most rendering algorithms; in some cases, it dominates all other considerations. A variety of techniques exist to accelerate this important element of rendering technology, but most techniques compute (a) exact, (b) binary (fully occluded or fully visible), and (c) point-to-point occlusion. While previous methods have weakened some subset of these requirements, we introduce a technique distinguished in that it relaxes all three assumptions. Recognizing that accurate visibility may be unnecessary in some situations, we seek to use approximate visibility computations where possible, including over long distances or wide angles. This is accomplished by supporting fractional visibility values between source and receiver, rather than purely binary decisions. Finally, we support point-to-cluster and cluster-to-cluster queries, averaging the visibility over all rays that lie inside the frustum from source to receiver. Such shaft-based visibility calculation can be applied to a wide variety of rendering problems, including soft shadowing, estimating visibility bounds, and smoothing artifacts and aliasing.

Our method encodes visibility over space, direction, and frustum size as a set of oriented cross-sections and corresponding texture map hierarchies. Each texture map hierarchy stores visibility in a particular direction, prefiltered by increasingly large kernels. To query shaft visibility of a cross section, we determine the appropriate level in the filter map hierarchy and look up the texture value at the point of intersection. The aggregate visibility of a shaft passing through the impostor is a linear combination of several such cross section queries oriented nearly orthogonal to the shaft.

In lightcuts, our approximation achieves speedups in the range of 10% to 300%, in addition to reducing noise for highly complex scenes.

II. Related Work

Visibility computation and shadow calculation represent a vast body of work, and a thorough review is beyond the scope of this paper; we limit our discussion to only those topics most relevant to our method. For a comprehensive survey, consult the detailed surveys by [1], [2], and [3].

The use of ray casting to compute occlusion dates to the use of shadow rays in Whitted-style ray tracing [4]. The introduction of distributed ray tracing [5] achieved soft shadowing effects by stochastically sampling rays and averaging their results. This implicit visibility shaft between area lights and sample points is made explicit in variants of ray tracing that attempt to leverage the spatial coherence of nearby rays, including shadow volumes [6], cone tracing [7], pencil tracing [8], beam tracing [9] [10], and packet tracing [11] [12].

Shadow maps [13] offers a different paradigm based on rasterizing depth from the viewpoint of a light on to a texture, then querying the texture to determine occlusion. Applying a hierarchy of textures in the spirit of mip-mapping [14] can correct for viewpoint-distortion of shadow maps, as in adaptive shadow mapping [15]. The GPU has made hardware-based shadow mapping feasible [16], and can be extended to support soft shadow effects as well [17], [18]. Techniques based on convolution can expedite soft shadow map generation [19]; convolving a shadow map of occluding geometry with the shape of a light source produces a corresponding soft shadow map.

The cost of computing visibility usually relates to the complexity of scene geometry; therefore, many methods attempt to simplify or approximate complex geometry. Billboard clouds [20] reduces 3D models to representative planes mapped with textures. Dynamic ambient occlusion [21] approximates a mesh with a hierarchy of disks in real time applications. Another real-time method approximates geometry with splatted spheres, using spherical harmonic exponentiation to compensate for overlap [22]. In work most similar to ours, complex aggregate geometry is approximated using a prefiltered volumetric technique [23]. Each node in a bounding volume hierarchy contains opacity information for each orthographic direction. Shafts - represented by a centerline and angle - can be evaluated by recursively descending through the hierarchy until the ratio of BVH element projected solid angle to cone projected solid angle falls below a threshold. Opacity information is accumulated until the shaft clears the object, or else fully occluded. The resulting algorithm reduces noise and cost as compared to stochastic ray tracing.

III. Technique

We begin by describing the interface of our technique and some brief motivating examples. We then discuss the creation and usage of our impostor's data structures. Next, we explain the algorithm to evaluate shaft visibility, and conclude with further details regarding shaft construction, visibility query evaluation, and tuning parameters.

A. *Interface*

The most common form of visibility computation for rendering algorithms involves intersecting the line segment between two points against the geometry of the scene. If an intersection is detected, the points are mutually occluded; otherwise, they are mutually visible. Many applications require testing visibility between regions with some finite extent, however, and in such cases, a single ray test is insufficient. Most commonly, we can randomly select a point in each region, test the visibility of the line segment between the two points and average over many such queries. A large number of ray queries is necessary to prevent aliasing and noise, making this method very expensive. Our goal is to augment the common functionality of ray-based point-to-point visibility tests with shaft-based tests capable of evaluating approximate visibility between large regions of space.

Given a conical shaft, our algorithm attempts to return a single fractional value for the average (approximate) occlusion over the solid angle of the shaft. Many visibility tasks can be expressed in terms of shafts, so the availability of a primitive shaft-based query offers great utility. Potential uses include finding visibility between a surface point and an area light, determining form factors between two patches, calculating visibility between two clusters of points, and testing the occlusion over a solid angle (such as when evaluating visibility between a surface point and a region of an environment map). Our method is agnostic to how or why the shaft was created; it simply takes a conical shaft with a particular direction, angle, and interval defining the cone frustum and returns a floating point value representing average visibility.

B. *Data Structures*

Traditional ray acceleration structures achieve soft effects by densely sampling sharp geometry. However, another alternative is to blur the geometry, then make a single query against this ”soft” geometry. There are several ways one might go about blurring geometry; one option is to convert the geometry to a volume, then blur this volumetric data directly. The cost of storing complex volumetric data can be prohibitive, though, and most geometry is composed of triangular meshes. Instead, we represent the geometric data as a set of cross section silhouette textures mapped to oriented planes distributed through the volume of space occupied by the geometry to be approximated. In practice, we use 16 such oriented planes to represent each object.

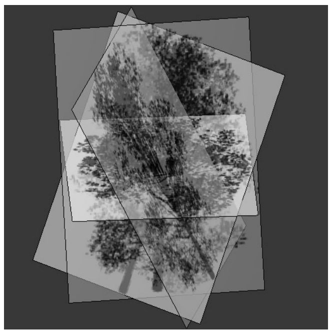


Figure 1. Impostors consist of oriented planes mapped with textures

The first level in the blur map is an orthographic projection of the mesh onto the plane, while each successive level blurs the first level by an increasingly large kernel size. The blur map encodes the visibility function across the silhouette of the object in the direction of the plane's normal for various fixed shaft sizes. This impostor thus sparsely encodes visibility over direction, position, and frustum size; given a particular shaft, we sample over these dimensions in order to determine some approximate average visibility for the shaft.

1) Precomputation: Creating the impostor is a view-independent preprocess. We generate the cross section planes with concentric mapping in order to space their normals evenly over the hemisphere. Since visibility is symmetric from source-to-receiver and receiver-to-source, we can use a single plane to represent occlusion in both the positive and negative normal directions. The billboards are arranged in a radial manner such that they intersect at the center of the object's bounding box (see 1).

Cross section creation begins with rendering an orthographic silhouette of the object with a camera facing the plane in the direction of the normal. The resolution of the image can be modified to trade memory for quality. This image essentially encodes a point-to-point visibility function at evenly sampled points. We use an orthographic projection rather than perspective for two reasons; first, it is desirable to enforce convergence to the correct solution as distance increases, and second, we want the texture to be symmetric in both the positive and negative normal directions. Any perspective projection would violate these two principles.

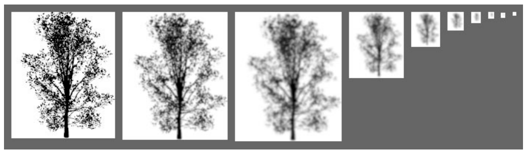


Figure 2. Each level in the filter map is blurred by increasingly large kernels, then downsized to save memory

The initial orthographic projection forms the base (level 0) of the blur map. It also includes a two-dimensional bounding box to map texture map coordinates into world space on the plane. This extent is clipped tightly to the edges of the silhouette in order to minimize the silhouette's surface area in world space. Subsequent maps in the hierarchy are formed by convolving with truncated Gaussian filters of increasingly larger size; at level i, the filter size is $(2^i + 1) \times (2^i + 1)$. In order to accommodate the blur filter on the edges of the silhouette, we increase both the size of the silhouette by the diameter of the filter in all directions and the mapping factor from texture coordinates into world space on the plane. To save memory, we downsample based on filter size; the resolution of a map at level i is reduced by a scaling factor $f = \min(1.0, 1.0/2^{i-2})$. The purpose of downsizing is purely to reduce memory usage and improve locality; the method can be adapted to textures of any size. For this reason, we downsize conservatively, leaving higher resolution maps at full resolution and only downsizing by a factor much smaller than the blur kernel.

C. Algorithm

Given a shaft and an approximate impostor, computing an estimate of visibility involves selecting a set of candidate planes, intersecting the shaft centerline with each, looking up the occlusion value in the appropriate levels of the blur map, and finally weighting the contributions of each plane to compute a final value.

Each plane is assigned a similarity measure equal to the absolute value of the dot product between direction of the shaft's centerline and the plane's normal. All planes whose similarity is greater than some threshold (in practice, 0.85) are considered candidate planes. For each candidate plane, we intersect the centerline of the shaft against the plane and calculate the barycentric coordinates for the point of intersection. We determine the area of the shaft's cross section at the distance of intersection; this can be found from the angle of the shaft and the distance to the intersected plane. We then interpolate between blur map levels based on how closely the shaft cross section area matches the blur kernel area on the plane in world space, and bilinearly interpolate over each map. Thus, the value reported by a single cross section plane is trilinearly interpolated over barycentric coordinates and blur map levels. The final visibility value of the shaft is a linear combination of each candidate plane's contribution weighted by its similarity.

When testing a shaft against several approximate impostors, we assume that the underlying geometry is uncorrelated, and multiply the contribution of each independent impostor to accumulate the visibility of the entire shaft.

1) Shaft Construction: For our purposes, a visibility shaft needs to be able to support two operations. First, we need to be able to retrieve a ray that defines the centerline of the shaft, consisting of an origin o, a direction d, and an interval that indicates the region over which the shaft's visibility is to be tested (t_{start}, t_{end}). Second, we need to be able to determine the area of the cross section at any given parametric distance t. We store the centerline explicitly in all cases; for conical shafts, we use an angular measure from the centerline ray to the outer edge of the shaft, while for cyndrillical shafts (most commonly, degenerate point-to-point cylinders), we simply store the (constant) cross section area. A point-to-point query has an angle of zero, while a point-to-cluster query is a conical shaft whose apex is the point and whose base is centered on the cluster. A cluster-to-cluster query is a frustum of a cone such that the wide end of the frustum is positioned around the larger cluster, the narrow end around the smaller cluster, and the virtual apex of the cone (and origin of the ray) is located some distance behind the smaller cluster (see 3).

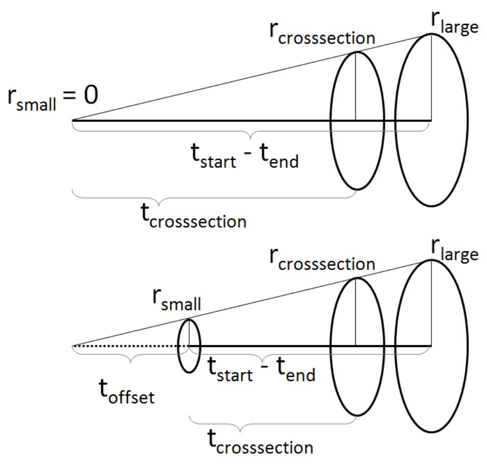


Figure 3. Top: point-to-cluster shaft, Bottom: cluster-to-cluster shaft

To generate a shaft between a source and destination cluster, we compute a cluster "radius" for source (r_{sender}) and destination ($r_{receiver}$), equal to half the diagonal of each box. Let t_{max} be the distance between the sender and receiver. If the clusters are the same size (that is, $r_{source} = r_{receiver}$), then the shaft's cross section is $r_{crosssection} = r_{sender} = r_{receiver}$, and its centerline extends from the center of the source cluster to the center of the destination cluster over the interval $(0, t_{max})$.

If the smaller cluster is a point, the centerline ray can be computed between the point and the center of the larger cluster, over the interval $(0, t_{max})$. Using similar triangles, we observe that for a particular distance t and the shaft's radius r, $\frac{r}{t} = \frac{r_{sender}}{t_{start} - t_{end}}$. Then, we can compute the shaft's cross section area a for a particular t with $a = \pi(t * \frac{r_{sender}}{t_{start} - t_{end}})^2$.

If both clusters have a finite extent, we take advantage of the symmetry of visibility and assume the shaft is directed from the smaller cluster toward the larger; regardless of which is the actual sender, the result will be the same. We take $r_{small} = min(r_{source}, r_{receiver})$ and $r_{large} = max(r_{source}, r_{receiver})$; the direction of the centerline d will simply be the normalized vector from the center of the smaller cluster to the center of the larger. Its origin is positioned such that the angle from the centerline ray to the edge of the source cluster is equal to the angle from the ray to the edge of the destination cluster - that is, we create similar triangles between the centerline ray and each of the clusters, with the cluster radius direction orthogonal to the ray direction. Given that the center of the smaller cluster is the point p, the ray's origin $o = p - d * t_{offset}$, where $t_{offset} = t_{crosssection} * \frac{r_{small}}{r_{large} - r_{small}}$ and $t_{crosssection}$ is the distance to the point at which the centerline intersects the plane. In order to compensate for the virtual origin behind the smaller cluster, the centerline ray interval is $(t_{offset}, t_{large} + t_{offset})$

2) Visibility Queries: The physical size of the cross section must be greater than or equal to the size of original geometry in order to represent the object, but the required size depends on the shaft size. Since only a shaft's centerline ray is actually tested against the geometry, the billboard size must be equal to the size of the object expanded in all directions by the diameter of the visibility shaft at the distance to intersection. Otherwise, a glancing shaft may fail to intersect the plane and report zero occlusion when in fact some portion of the shaft should be occluded. This can lead to a potential discontinuity at the edge of the quad. On the other hand, a shaft may grow arbitrarily large, but using correspondingly large planes becomes prohibitively expensive.

We resolve this problem with Minkowski sums to dynamically resize the mesh's bounding box based on the shaft size. Given the bounding box b of the mesh to be approximated based on the shaft size. Let c be the center of b and let v be the vector direction of the shaft centerline; then $p = c - o$ and $t_{estimate} = p \cdot v$. We can compute the radius r of the cross section of the shaft at distance $t_{estimate}$, then expand b in all directions by $d = 2 * r$ to form a new box against which we test the shaft's centerline. If the ray does not intersect the box, no part of the shaft intersects the approximation, and we can cull its content.

If the shaft does intersect the box, we gather a set of candidate cross sections for further testing. We assign each plane a similarity metric equal to the absolute value of

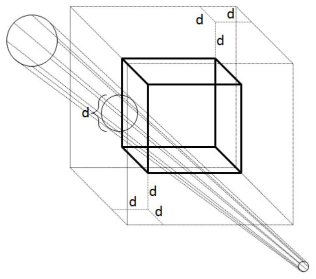


Figure 4. We expand the bounding box in all directions by the diameter of the ray at the point it passes closest to the center of the box

the dot product between the centerline ray direction and the plane's normal. Planes whose similarity exceeds some threshold (in practice, 0.85) are added to the set of candidate planes. Once the candidates have been selected, we assign each plane p_j with similarity s_j a weight w_j such that $w_j = s_j / \sum_{i=1}^{n} s_i$.

The contribution c_j of each plane is computed as follows. First, we intersect the centerline with the plane and determine the distance and barycentric coordinates of intersection. From the distance, we compute the cross section area of of the shaft at the point of intersection. We then find the levels of the blur map which form upper and lower bound on the shaft area. Using the barycentric coordinates and the size scaling factors computed during precomputation, we determine appropriate texture coordinates for each map and look up the visibility. The final contribution c_j is linearly interpolated between these two values based on shaft area.

3) Tuning and Errors: This method supports several parameters for controlling the cost/quality tradeoffs of the approximation. Both the number of cross sections and their resolution can be adjusted. The number of cross sections affects performance directly - at a minimum, similarity must be computed between a shaft and each cross section direction - but the effect of changing cross section resolution is more subtle; devoting more memory to cross section textures increases the memory footprint of the working set and adversely affects cache coherency. In our results, we use 16 cross-sections with a resolution of 256 x 256 for the highest resolution blur map texture.

In order to hide inconsistencies at the transition border from accurate geometry to the approximation, we blend both methods according to a linear fall-off based on distance over some region. However, this requires all rays that originate or terminate in this region to be tested twice. Performance can be improved by shrinking the size of this overlapping region, albeit at the risk of making the transition more obvious.

Because cross sections are generated by orthographic projection, they approach the exact value of the visibility function as the distance approaches infinity. However, for short shafts and wide angles, our approximation fails to capture perspective effects. In addition, while pre-blurring by a Gaussian function in some sense captures an average light source shape and power distribution, it cannot perfectly reproduce the effects of a particular light source or cluster. For example, a long, narrow light or oddly distributed cluster of point lights will produce characteristic shadows we cannot reproduce. This limitation could be partially overcome with anisotropic mip-mapping, but would require tracking information about shaft size and light distribution.

IV. Applications

The applicability of shaft-based visibility is broad; we suggest three specific applications of the method. 1) By replacing point-to-point visibility queries in lightcuts with shaft visibility, we can reduce noise and improve quality. 2) Shaft visibility provides a means to estimate a bound on visibility, and use it tighten the lightcuts refinement criteria. 3) We use shaft-based visibility to compute approximate soft shadow effects.

A. Lightcuts

Instant radiosity [24] computes indirect illumination by tracing particles from direct sources and depositing virtual point lights on surfaces. For a set $\mathbb{S}$ of point lights created in such a manner, the radiance L_p at point p can be computed by summing the intensity I_s of each light source $s \in \mathbb{S}$ multiplied by its material (M_i), geometric (G_i), and visibility (V_i) terms: $L_p(x, \omega) = \sum_{s \in \mathbb{S}} M_s(x, \omega) G_s(x) V_s(x) I_s$.

Lightcuts [25] is a scalable VPL algorithm that improves on instant radiosity by building a hierarchy of lights. To shade a pixel, it dynamically computes a partition of lights into clusters by refining a cut through the the light tree. Error bounds can be computed individually for material, geometric, and visibility terms, and the product of these terms bounds the maximum possible error introduced by the node. It is this visibility term which we modulate with our approximation in order to improve performance.

Multidimensional lightcuts [26] extends lighcuts by introducing a gather tree. Gather points can be distributed over time, volume, aperture, and pixel area, providing a means for scalable rendering of motion blur, volume scattering, depth of field, and antialiasing. The refinement process generalizes to a cut on the product graph of the light and gather tree.

Lightcuts offers a natural environment for exercising the full power of our shaft-based visibility approximation. Fundamentally, lightcuts reduces all illumination to a cloud of primitive, point-based lights, then clusters them hierarchically based on position and orientation. In order to shade a point, lightcuts computes a set of light clusters so as to enforce bounded error. Multidimensional lightcuts is based on three kinds of cluster relationships: point-to-point, point-to-cluster, and cluster-to cluster [27]. This presents an ideal environment to leverage the full power of our approximation. We replace the existing point-to-point visibility in lightcuts with shaft-based visibility that matches the size of the light and gather clusters and/or points being evaluated.

1) Smooth Approximate Visibility: When evaluating visibility between a single gather point and a single light, we retain the established practice of performing a ray-based point-to-point query. However, when testing visibility between a single gather point and a cluster of lights, we form a point-to-cluster shaft from the gather point to the representative light, and compute a shaft size based on the size of the light cluster. Likewise, when computing visibility between a cluster of gather points and a cluster of light points, we form a cluster-to-cluster shaft based on the sizes of both the light and gather clusters. Then, we simply weight the contribution of the cluster by the visibility. Though the performance characteristics of this method do not vary widely from the point-to-point method, the errors in illumination are smoothed, and thus less objectionable.

It is possible to use our approximation to replace all point-to-cluster queries, but in practice we limited its usage to indirect lights and environment map lights. This allows us to preserve the sharpness and accuracy of individually discernible shadows, while smoothing and accelerating softer occlusion effects.

2) Visibility Bounds Estimation: We achieve additional improvements by incorporating partial approximate visibility into lightcut's error bound calculation. Although the approximate nature of our method means the error is no longer strictly bounded, the results are nonetheless qualitatively adequate in practice. The maximum error of a node in a cut is computed by separately bounding the geometric, material, and visibility terms, then multiplying the product by the total power of all sources in a cluster. Prior to this work, however, the visibility bound was always set conservatively to one - that is, in the worst case the entire cluster is fully visible. For lightcuts, this is unavoidable; because visibility is calculated only point-to-point from the surface to the light representative, there is no information about occlusion or lack thereof to the rest of the light cluster. Using our method, however, we can approximate the average occlusion between the surface point and the entire light cluster. We could use the visibility estimate directly, but to compensate for the approximate nature of our solution, we add a small offset (0.125 in practice) to account for errors in our visibility calculation. Because lightcuts drives refinement with an upper bound on error, underestimating occlusion adversely affects performance, but overestimating occlusion compromises quality. Thus, offsetting the error by a small amount incur some performance cost, but the improvement in quality is generally worth it.

B. Multidimensional Lightcuts

Adapting our technique to multidimensional lightcuts requires very few changes. The main difference is that instead of individual gather points, we must now handle gather clusters. Thus, we simply replace the point-to-cluster query in lightcuts with a cluster-to-cluster query.

C. Soft Shadows

Traditional soft shadow calculation involves computing visibility between an area light and surface point by sampling rays to points scattered stochastically over the light source. Not only is this process expensive, but the results tend to exhibit noise. Our method allows one to compute approximate average visibility between a surface point and the entire area light. We simply form a point-to-cluster shaft from the surface point to a region of space that surrounds the light source. If the light source is anisotropic or highly irregular, the cluster may be too loose. In such cases, the light can be subdivided into k regular, isotropic subcomponents. We can then form a total of k shafts, one for each subcomponent, test each, and aggregate the results. Not only is the method fast, but the results are smooth even with just one shaft.

V. Results

All results were rendered as a single-threaded application on a 3.33 GHz Intel Xeon CPU. However, adapting the algorithm to be multi-threaded would be straightforward.

A. Lightcuts & Multidimensional Lightcuts

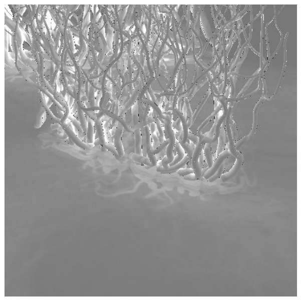
We report results for four scenes: Air War, Tentacles, Forest, and Colonnade. Each scene is lit by directional lights generated by sampling an environment map; the first three scenes use 8000 directional lights, while Colonnade uses 80k directional lights and 10k indirect VPLs. Air War contains 120k triangles spread over 400 airplane models scattered above a plane. Tentacles consists of 100k triangles arranged in four clusters. The forest has six tree models surrounding the bunny, for 300k total triangles. Colonnade consists of an armadillo inside an mostly-enclosed box, except for one open side lined by columns, and totals 200k triangles.

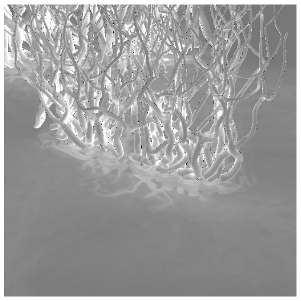
First, replacing point-to-point visibility queries with appropriate cluster-to-cluster queries improves the quality of the final rendered image. Normally, lightcuts must accomplish all soft rendering effects - particularly soft shadows - by averaging over many sharp queries. Simply replacing these ray tests with smooth shaft-based visibility reduces the discontinuity artifacts of lightcuts, particularly in scenes with highly complex visibility.

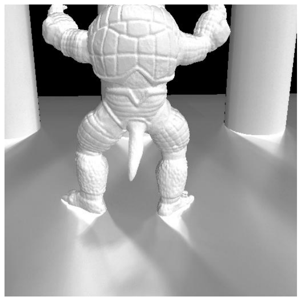
Second, estimating the visibility bound with shafts improves performance. Tighter bounds allows lightcuts to correctly determine which portions of the cut to refine next. Previous versions of lightcuts made no attempt to tightly bound visibility, and while our method does not compute a strict bound, our estimate of the visibility is generally good enough to improve performance without compromising quality. Generally, we see the greatest performance gains for highly occluded regions of the scene; typically, lightcuts does significant refinement for these regions in order to prove no light can reach them, but bounds estimation allows for earlier termination of refinement. Specific runtimes are tabulated in I. All scenes are 512 x 512 with 16x antialiasing, and were rendered with an error threshold of 1% (except Forest, which used an error threshold of 0.5%).

Scene	**Accurate**		**Approximate**		**Speedup**
	Time (sec)	Cut Size	Time (sec)	Cut Size	
Air War	1215.9	388.9	1081.1	301.4	1.12x
Colonnade	3408.9	3777.5	1151.8	1784.2	2.96x
Tentacles	448.0	353.7	378.2	316.3	1.19x
Forest	598.6	399.0	461.1	350.6	1.30x

Table I
Render times and average cut sizes by scene type using multi-dimensional lightcuts with 16x antialiasing


(a) Tentacles cut sizes using approximation; maximum corresponds to 5008


(b) Tentacle cut sizes without approximation; maximum corresponds to 5008


(c) Colonnade cut sizes using approximation; maximum corresponds to 4464

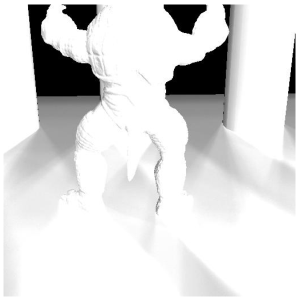
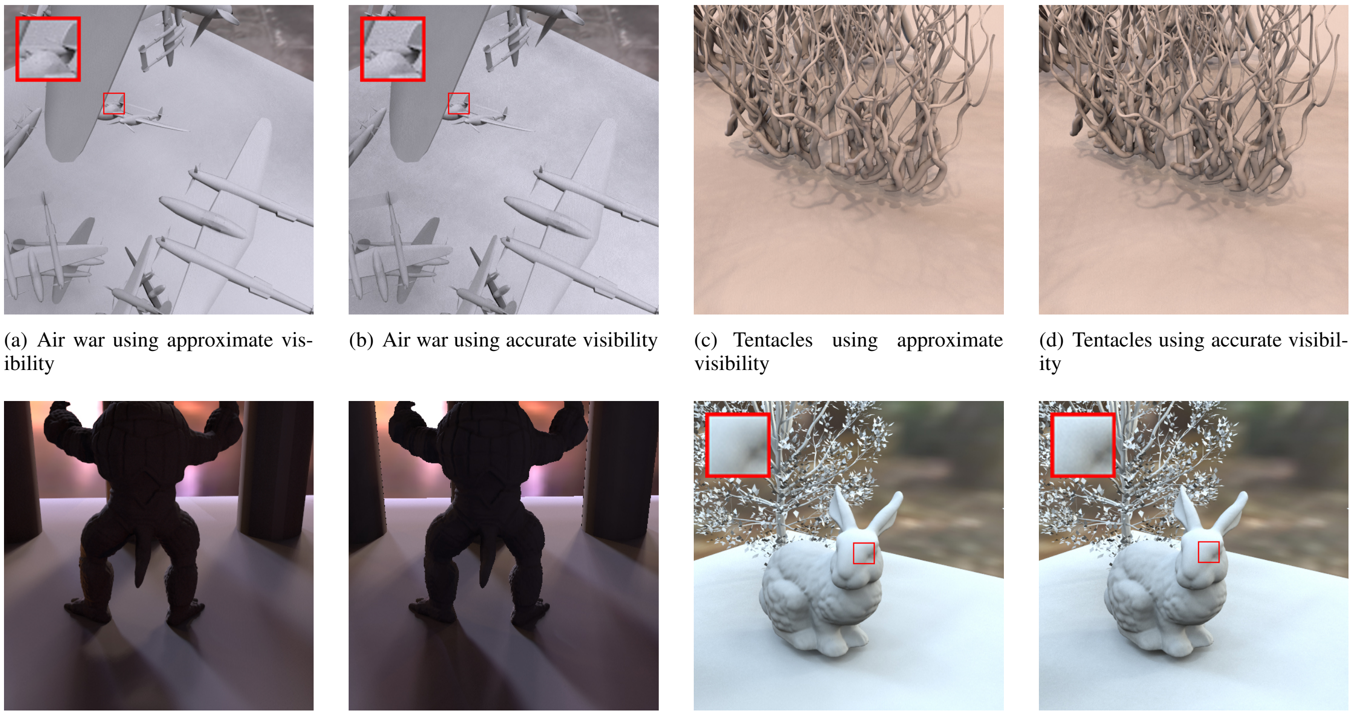

(d) Colonnade cut sizes without approximation; maximum corresponds to 4464

Figure 6. Cut size images

B. Soft Shadows

We demonstrate soft shadows with a scene consisting of a tree on a plane. Testing visibility with 2048 samples per pixel requires 187 seconds and still exhibits noise, while the approximation takes 28 seconds and yields smooth results.

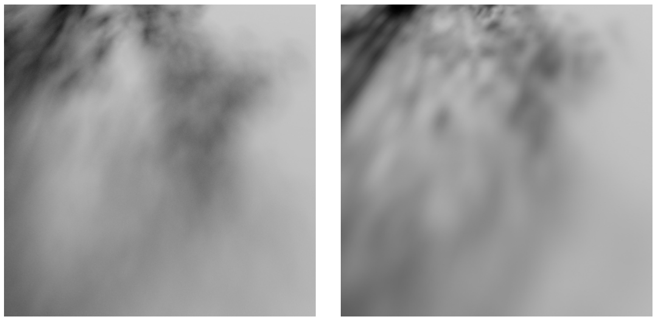


(a) Air war using approximate visibility (b) Air war using accurate visibility (c) Tentacles using approximate visibility (d) Tentacles using accurate visibility

(e) Colonnade using approximate visibility (f) Colonnade using accurate visibility (g) Forest using approximate visibility (h) Forest using accurate visibility

Figure 5. Comparison renders using accurate and approximate visibility

However, perspective and depth-dependent features are lost, and blending between planes introduces additional blurring to the shadow, removing some high-frequency features. While insufficient for final rendering, this may be effective for rapid preview.



(a) Distributed ray tracing with 2048 samples (b) Approximate visibility

Figure 7. Soft shadow comparison

VI. Conclusion

We presented an approximate soft visibility system capable of handling point-to-point, point-to-cluster, and cluster-to-cluster queries. We demonstrated a variety of applications, including soft shadows, bound estimation, and cluster-to-cluster visibility computation.

There are several improvements that could be made to our system. First and foremost, the algorithm would be a natural fit for the GPU, with a corresponding increase in performance. Our method generates one impostor per mesh, but the solution would be substantially more scalable if adapted to a hierarchy over meshes. Oriented planes are distributed radially at the center of objects, but fitting them more closely to the geometry might address some artifacts of our method.

Acknowledgment

This work was supported by research grants provided by Autodesk, Inc. and NSF CPA-G&V 0811680. We would also like to thank the team members of Autodesk Labs Project Showroom for fruitful discussions related to problem specification, usage patterns, and system requirements.

References

[1] A. Woo, P. Poulin, and A. Fournier, “A survey of shadow algorithms,” *IEEE Computer Graphics and Applications*, vol. 10, no. 6, pp. 13–32, Nov. 1990.

[2] F. Durand, “A multidisciplinary survey of visibility,” 2000.

[3] J.-M. Hasenfratz, M. Lapierre, N. Holzschuch, and F. Sillion, "A survey of real-time soft shadows algorithms," in *Eurographics*, Eurographics. Eurographics, 2003, state-of-the-Art Report. [Online]. Available: http://www-imagis.imag.fr/Publications/2003/HLHS03

[4] T. Whitted, "An improved illumination model for shaded display," *CACM*, vol. 23, no. 6, pp. 343–349, 1980.

[5] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," in *SIGGRAPH '84*, 1984, pp. 137–145.

[6] U. Assarsson and T. Akenine-Mller, "A geometry-based soft shadow volume algorithm using graphics hardware," in *ACM Transactions on Graphics*, Jul. 2003.

[7] J. Amanatides, "Ray tracing with cones," in *Computer Graphics (SIGGRAPH '84 Proceedings)*, H. Christiansen, Ed., vol. 18, 1984, pp. 129–135. [Online]. Available: citeseer.nj.nec.com/amanatides84ray.html

[8] M. Shinya, T. Takahashi, and S. Naito, "Principles and applications of pencil tracing," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 45–54, 1987.

[9] P. Heckbert and P. Hanrahan, "Beam tracing polygonal objects," *Computer Graphics (Proc. Siggraph '84)*, vol. 18, no. 3, pp. 119–127, 1984.

[10] R. Overbeck, R. Ramamoorthi, and W. R. Mark, "A Real-time Beam Tracer with Application to Exact Soft Shadows," in *Eurographics Symposium on Rendering*, Jun 2007.

[11] I. Wald, C. Benthin, M. Wagner, and P. Slusallek, "Interactive rendering with coherent ray tracing," in *Proc. of Eurographics*, 2001, pp. 153–164.

[12] R. Overbeck, R. Ramamoorthi, and W. R. Mark, "Large Ray Packets for Real-time Whitted Ray Tracing," in *IEEE/EG Symposium on Interactive Ray Tracing (IRT)*, Aug 2008, pp. 41—-48.

[13] L. Williams, "Casting curved shadows on curved surfaces," in *Computer Graphics (Proceedings of SIGGRAPH 78)*, vol. 12-3, Aug. 1978, pp. 270–274.

[14] ——, "Pyramidal parametrics," in *SIGGRAPH '83*, 1983, pp. 1–11.

[15] R. Fernando, S. Fernandez, K. Bala, and D. P. Greenberg, "Adaptive shadow maps," in *Proceedings of ACM SIGGRAPH 2001*, ser. Computer Graphics Proceedings, Annual Conference Series, Aug. 2001, pp. 387–390.

[16] C. Everitt, A. Rege, and C. Cebenoyan, "Hardware shadow mapping," in *In ACM SIGGRAPH 2002 Tutorial Course #31: Interactive Geometric Computations*, 2002, pp. 38–51.

[17] M. Herf, "Efficient generation of soft shadow textures," Tech. Rep., 1997.

[18] P. S. Heckbert and M. Herf, "Simulating soft shadows with graphics hardware," Tech. Rep., 1997.

[19] C. Soler and F. X. Sillion, "Fast calculation of soft shadow textures using convolution," in *Proceedings of SIGGRAPH 98*, ser. Computer Graphics Proceedings, Annual Conference Series, Jul. 1998, pp. 321–332.

[20] X. Décoret, F. Durand, F. Sillion, and J. Dorsey, "Billboard clouds for extreme model simplification," in *ACM Transactions on Graphics*, 2003.

[21] M. Bunnell, "Dynamic ambient occlusoin and indirect lighting," in *GPU Gems 2*, M. Pharr, Ed., 2005.

[22] Z. Ren, R. Wang, J. Snyder, K. Zhou, X. Liu, B. Sun, P.-P. Sloan, H. Bao, Q. Peng, and B. Guo, "Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation," *SIGGRAPH*, 2006.

[23] D. Lacewell, B. Burley, S. Boulos, and P. Shirley, "Raytracing prefiltered occlusion for aggregate geometry," in *IEEE Symposium on Interactive Raytracing 2008*, 2008.

[24] A. Keller, "Instant radiosity," in *Proceedings of SIGGRAPH 97*, ser. Computer Graphics Proceedings, Annual Conference Series, Aug. 1997, pp. 49–56.

[25] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. P. Greenberg, "Lightcuts: a scalable approach to illumination," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 1098–1107, 2005.

[26] B. Walter, A. Arbree, K. Bala, and D. P. Greenberg, "Multidimensional lightcuts," in *To appear in Proceedings of SIGGRAPH 2006*, ser. Computer Graphics Proceedings, Annual Conference Series, Aug. 2006.

[27] E. Velázquez-Armendáriz, S. Zhao, M. Hašan, B. Walter, and K. Bala, "Automatic bounding of programmable shaders for efficient global illumination," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 1–9, 2009.