Piecewise Constant Conic Sections for Accelerated Volume Density Rendering

R. B. Hougs and A. M. Day

School of Information Systems, University of East Anglia, Norwich, England

Abstract

In order to accelerate the rendering of volumetric shadows, we propose a new technique which builds sets of conic volumes to approximate the shape of shadows in a participating medium. The novelty of our approach is notably the construction of the cone-sets, which are built with no knowledge of the underlying geometry of the scene. Instead, information collected during the construction of a global photon map is used to derive an estimate of the outline of the shadows in three dimensions. This information is then used in several different ways to speed up the rendering pass. The method shares many of the advantages proposed by photon maps such as viewpoint independence and decoupling from the geometry of the scene.

1. Introduction

This article is concerned with speeding up, as well as approximating the result of rendering a scene containing participating media. The underlying global illumination framework is assumed to use photon maps [6, 8]. The concept of using data collected during the photon tracing pass of that method to approximate and estimate shadows *on surfaces* has already been presented [7]. However, shadows in volumetric media present a quite different problem: firstly, the bucket-extended kd-tree data structure [2] normally used to store the positions of photons in the scene cannot be used to represent the three-dimensional regions of space occupied by volumetric shadows. Second, it is important that the approximation of the volumetric shadows does not end up being *slower* than using a standard adaptive ray-marching technique. Finally, it is desirable that the volumetric shadow approximation maintain some of the prime advantages of photon maps, namely a decoupling from the underlying scene geometry and viewpoint independence.

The use of constant conic sections, as described in this article, fulfills many of these requirements. Conic sections can either be used independently or in conjunction with other occlusion-estimation methods such as shadow buffers ([20], see section 2.2.2). They are not subject to aliasing in the same way shadow buffers are, use little storage space and do not require any modifications to an already existing photon mapping framework in order to function. They are also relatively simple to implement.

This article introduces the ideas behind constant conic section: a brief introduction to photon maps and participating media rendering leads into a concise overview of previous research in the area. An outline and explanation of our method follows, with enough information that implementing the method in a global illumination renderer should be straightforward. Then, various results of the technique for some standard and not-so-standard test scenes are presented. Finally, a discussion of the advantages and disadvantages of the proposal lead to suggestions for further extensions.

2. Background

2.1. The problem setting

The goal of photorealistic rendering is of course to determine an accurate estimate of the radiant flux arriving at every point on the image plane of a film (e.g. see [10]) or simulated eye (e.g. see [17]). We do not have the space here to provide an overview of all the numerous solutions proposed to this problem — for an excellent summary of these see [18]. In order to estimate the incoming flux at the image plane, we generally need to be able to estimate the light field of the scene we are simulating. This in turn implies the ability to compute all the different phenomena which affect the passage of light.

Different rendering algorithms are generally well suited for some of these phenomena and perform badly for others.

Of these phenomena, for the purpose of this article we are specifically concerned with so-called *volumetric density objects* (VDOs) [3]. More specifically yet, we concentrate on modelling the structure of the light field inside a VDO; that is, we are not concerned here with the details of realistic scattering and absorption (such details can be found in [14, 15]) or with simulating the density distribution within the VDO.

2.2. Previous work

The specific purpose of the research presented here is to find, for a given object, a way of representing the *volume* of space in which an object occludes light from a given lightsource. The aim of the research is to improve the speed and accuracy of rendering scenes containing VDOs. Our survey of previous work will therefore be divided into two sections: the first reviews models for representing the extents of shadows in space or on surfaces; the second briefly reviews different approaches to modeling and rendering VDOs.

2.2.1. Modeling shadows on surfaces

Starting with the "simple" case of representing shadows on surfaces, we observe that such a problem reduces to representing an arbitrarily shaped area in two dimensions (the coordinate system of the surface). How to deduce the extents of that surface? Several methods are known:

- *projection techniques* project the shape of objects onto the surface in question, with the point of projection being the lightsource. This approach works well for primitive shapes or triangle meshes (where vertices can be projected individually), but for shapes with a more complex definition, the projection can be difficult to determine. The techniques does not work well with area light sources (how to project a shape relative to an area?).
- *particle tracing* methods can be extended to encode the area of a shadow. Using shadow photons [7] with the photon map technique [6] is a good example of this. Using shadow photons, rendering time can be cut dramatically for some scenes. For a further description of this approach see section 3.1.

2.2.2. Modeling shadow volumes

For the more complex case of encoding information about *volumes* of shadows, the following methods have been proposed:

- a *shadow buffer* [20] stores information about the objects "visible" from a lightsource, as well as the distance of those objects from the light source. It then becomes fast to check whether an object is potentially in shadow or not. The major downside of this algorithm is that it cannot handle area light sources, and it is subject to aliasing, due to the discrete nature of the buffer. A recent improvement on this algorithm is the *deep shadow map* [11], which stores compressed, prefiltered exctinction functions at every depth map pixel. Even this method, however, cannot handle area light sources.
- hierarchical representations of space, such as *voxels*, *quad-trees* or *octrees* allow for the delimitation of space into shadowed/non-shadowed areas in a very direct and hierarchical way. A recent example of the use of these techniques to represent occluded volumes of space is [16]. Problems with such data structures is that they are potentially very memory intensive, and they are generally required to be axis-aligned, a requirement which may conflict with the orientation of shadow volumes in the scene.
- "*projection volumes*" can be constructed, which are either primitives or boundary representations of the shape of the shadow volume. Intersection tests can then be done against such shapes to determine the boundary of the shadow region. However, such projection volumes can be difficult to construct, and may comprise many polygons if the occluding shape is complex.

2.3. Rendering VDOs

One of the main application areas of our technique is for detecting the boundaries of shadow regions in space, which is of particular importance when rendering environments with participating media. We therefore propose a brief review of rendering methods applicable to VDOs.

2.3.1. Light transfer in participating media

Flow of light in a volumetric density object is characterised by two kinds of events, absorption and scattering. The former stops the flow, whereas the latter changes its direction. The combined action of scattering and absorption over the length of a ray is described by attenuation. What attenuation cannot describe is in-scattering, that is, light being scattered into the direction of the sampling point. Estimating the in-scattering can be very computationally expensive. Volume photon maps [8] can help in this situation by providing an estimate of in-scattering contributed by light paths which have been scattered at least once. However, the authors of [8] recommend the direct calculation of in-scattered light arriving directly from the light source; this will naturally be the largest contributor of flux. Sampling this in-scattered light is generally done by taking samples along the ray to estimate

$$L_{is} = \int_0^1 b(\gamma)\sigma(0,t)L_i(t)dt \qquad (1)$$

where $L_i(t)$ is the incoming light from the lightsource at point t along the ray, $\sigma(0,t)$ is the attenuation occurring between the sampling point and the ray origin and $b(\gamma)$ is the scattering coefficient of the VDO (γ being the angle between the light source ray direction and the sampling ray direction).

2.3.2. Path tracing and its variants

Path tracing techniques use various sampling schemes to estimate the flux arriving at the image plane. The most important factor determining the performance of such techniques is the choice of sampling points in the space of the scene. Hence most improvements to the basic path tracing algorithm [9] are concerned with finding the areas in the scene where the biggest variations in the light field occur. Although improvements such as bidirectional path tracing and Metropolis light sampling [19, 13] can greatly improve this process, their underlying framework still relies on a simulation of the light flow between points in space. Consequently, they still need other methods of getting an estimate of the in-scattering that occurs between two such points.

In other words, finding a good approximation to equation 1 is still critical in estimating the flux arriving at a sample point in the scene. In order to capture the intervals of high discrepancy along the ray (i.e. those intervals where many samples should be taken) the most commonly used technique is ray-marching [4]. Although ray-marching is attractive by its simplicity and elegance, it has no knowledge of where the discrepancies lie — instead it has to "feel" its way toward such areas, which requires much computation, and may even miss them entirely.

If the ray-marching algorithm could be given some knowledge of where to look for discrepancies, its accuracy could improve while reducing rendering time. This is exactly what our technique proposes to do (see section 3.1 for details).

2.3.3. Discretization of space

Data structures which discretize space, such as those outlined in section 2.2.2 can be used to directly represent the light flux within the scene, and hence inherently represent the areas of light and shadow. However, the memory usage and axis-alignment requirements of such methods render them impracticable for the kind of use we are envisaging here.

3. Cone sets

3.1. Overview

If we consider a point light source illuminating a finite object in two dimensions (see figure 1a), it seems clear that the outline of the shadow will be in the shape of a truncated cone. If we now consider an area light source, still in two dimensions, illuminating the same object, and model the light as a collection of point light sources, then the resulting shadow can be approximated as a set of cones of different base widths and apertures (see figure 1b). If we now expand this concept to three dimensions, the result is that an approximate outline of the volumetric shadow can be provided by a set of cones, chosen so that their expansion rates as closely as possible match the expansion rate of the shadow.

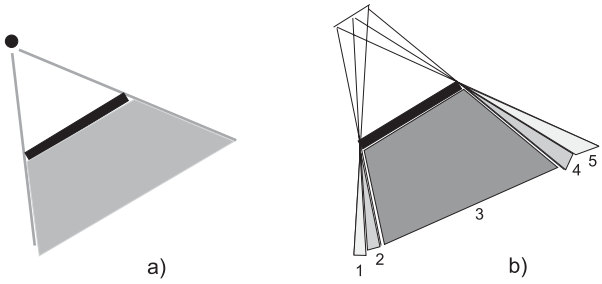

Figure 1: *An occluder object illuminated by a point light source, and the resulting shadow area (figure a); extending this concept to area light sources results in a* set *of shadow areas, shaped like cones with varying base radius and aperture (figure b).*

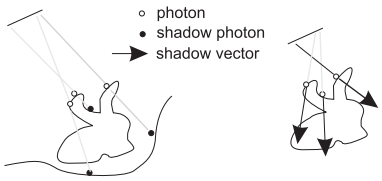

Figure 2: *Shadow photons are stored on the next surface "beyond" the photon intersection point (shown to the left). Shadow vectors are stored on the "underside" of the object intersected by the original photon path (shown here to the right).*

In order to build such a set of cones, we need some information about the *absence* of light flux within the scene: such information is available at the time of construction of the photon map. This idea was exploited in [7] to store so-called *shadow photons* on the surfaces of objects situated "behind" those surfaces being intersected by photons (see figure 2a). This allows for the renderer to determine with some degree of accuracy when and when not to send visibility rays toward light sources. We would like to extend this concept to three dimensions, from surfaces to volumes. Our solution to this problem is to store so-called shadow vectors, not on the surface below the intersected one, but rather on the *backside* of the intersected surface (see figure 2b). We store the intersection point and outgoing direction, as well as the object that cause the shadow vector to be added in a kd-tree, similar to the usual photon map approach.

The resulting set of shadow vectors encode information about the geometric structure of the shadow behind the given object. How now to store and access this information? Our solution is to group this set of vectors into several cones with different characteristics. Using several passes as described below, we can group most of the available vectors into such cones. Because the concentration of points will follow the general outline of the object, the generated cones

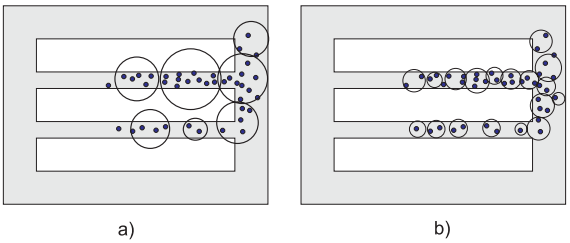

Figure 3: *An example of a difficult case for the algorithm. If we naïvely generate cones based on point densities, the cone bases often end up "spilling over" the edges of the shape as illustrated in figure a. In figure b a better distribution is shown, with more cones but little over-spilling.*

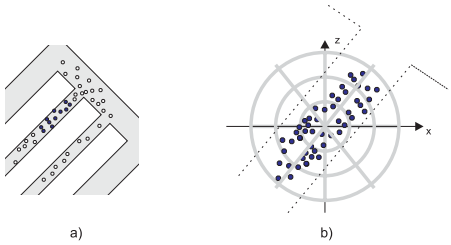

Figure 4: *As described in section 3.2.1, we determine a set K of points from which to construct the cone. These coordinates are transformed into a two-dimensional plane. The points are then partitioned into bins depending on their angular sector. As can be see in figure b, in this case the cone would be generated with a radius corresponding to the innermost circle, since the second one already contains empty sectors, a sign that we have found a discrepancy in the underlying geometrical structure.*

should automatically provide an approximation of the three-dimensional shape of the shadow. Every cone data structure contains information about the geometry of the cone, as well a pointer to the occluder associated with the cone. These cones can be enclosed inside e.g. a bounding cone for faster visibility testing in the next step.

In the subsequent rendering step, the information encoded by the cones can be utilised in various ways. Firstly, if for a given ray no shadow cones belonging to a certain object are intersected, then that object does not need to be tested for occlusion at any point along the ray. If the set of shadow cones *is* intersected, then we have a bound on the segment of the ray in which we need to test for occlusion — outside of that segment, no intersection tests with that object are necessary (neither are any tests inside, if we are not interested in capturing light scattering in the penumbra-delimited zone).

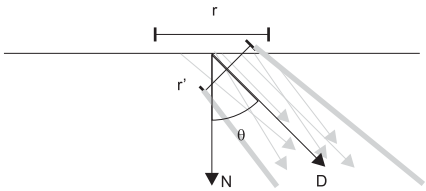

Figure 5: *The correct base radius of the cone is given by multiplying the original radius (r in the figure) by* $\cos\theta$ *to compensate for the projected base radius* r' *(projected along* $\vec{D}$ *in the figure) being smaller than r.*

3.2. Constructing the cone set

In order to construct the cone set, we assume the shadow volume caused by a differential area dA_L of a light source *not* illuminating the volume behind a differential area dA of an occluder object can be modelled by an infinitesimally thin cone with aperture $d\omega$ (we use notation similar to that of solid angle theory). The base radius of the cone is $dR = dA$. Then the shadow volume created by the light source L is

$$\int_{A_L}\int_{A} V(dA, dA_L)\; dA dA_L \qquad (2)$$

where $V(dA, dA_L)$ is the volume of the shadow cone caused by dA blocking the light from L.

3.2.1. Partitioning the vector sets

We wish to approximate the shadow volume described by equation 2 with as few cones as possible, without compromising the quality of the approximation. Given a set S of shadow vectors, we need to find the $S_1 \ldots S_N$ different subsets of S which will make up the N shadow volume cones.

The main problem we need to overcome is that we wish for the technique to work with no knowledge of the geometry of the object which caused the shadow vectors to be generated. At the same time, the shadow cones should provide a good estimate of the outline of the shadow; in particular, we need to avoid situations like the one in figure 3 where cones are generated in such a way that they miss or misrepresent the shape of the occluding object.

We have found the following approach to work well. We begin by selecting the first point in S, S_1, and query the kd-tree for the set K of nearest neighbours to this point. Then, beginning with S_1, we collect the vectors in K, going radially outwards from S_1, into "bins" according to an angular two-dimensional partitioning (see figure 4). Note that this assumes that the points in K all lie in the same plane; this will not always be true, particularly for highly irregular shapes with a low shadow vector density. In this case, smaller cones should be generated; we show how to detect

and handle this case in section 3.2.3. Once all points within radius r have been collected into bins, we calculate a rough estimate of the variance from bin to bin. If this variance is below a given threshold, we continue the process with the points in the disc of outer radius $2r$ and inner radius r, and so forth. If the variance is *above* the threshold in the disc with radii $[(k-1)r, kr]$, we stop the process and add a cone with base radius $(k-1)r$. The points contained within that radius are then removed from S, and the process resumes with the first point in the updated set. The process is illustrated in figure 4. The final base radius of the cone is calculated as $r_b = (k-1)r\cos\theta$ where θ is the angle between the estimated normal of the surface and the direction vector $\vec{V_M}$ of the cone (see section 3.2.2). This is necessary because the projected area of the cone base onto the surface is proportional to $\cos\theta$, a well-known result from radiometry. See figure 5.

After the last vector in the set is reached, any remaining shadow vectors are discarded from the set. Although this makes the algorithm less precise, the fine details captured by the remaining vectors is either not likely to show up in the final image, or is already mostly accounted for by surrounding cones. Four parameters control the cone determination process: p_1, the number of vectors to use to build K; p_2 the number of angular subdivisions to use; p_3, the number of points per "sector test" (which determines the values of $r_1 \ldots r_k$) and p_4, the minimum number of shadow vectors per cone. By varying these parameters, the user can control how the cones are generated.

3.2.2. Determining the other cone attributes

To calculate the expansion rate of a cone, we first determine the *mean vector* $\vec{V_M}$ which approximates the "average" direction of all the shadow vectors within S_i:

$$\vec{V_M} = \frac{1}{N}\sum_{i=1}^{N}\vec{V_i} \tag{3}$$

where N is the number of vectors in S. We then calculate the maximal deviation Θ from $\vec{V_M}$:

```
THETA=0
FOR i=0 TO N
    D=DOTPRODUCT(VSET[I],VM)
    IF D > THETA THEN
THETA=ARCCOS(D)
    ENDIF
NEXT i
```

We then determine the expansion rate of the cone by simple trigonometry to be $\tan\Theta$. Note that this method will not detect the case where the vectors converge — it assumes that they diverge (and hence cause the cone to expand). Another way of calculating the expansion rate of the cone would be to calculate the ratio between the base radius of the cone, and the radius of the disc described by the ends of the shadow vectors.

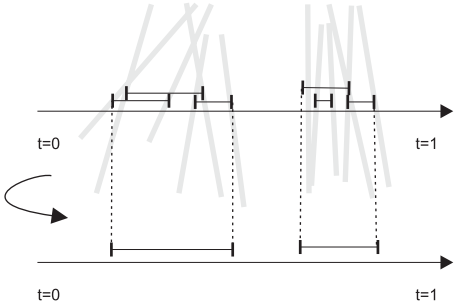

Figure 6: *Intersecting the sampling ray with the cones in its path results in a (possibly empty) set of intersection intervals. These intervals are then grouped together using a simple method, described in section 3.3.1, to obtain the final set of intervals.*

A cone data structure can also contain a pointer to the object whose shadow vectors cause the cone's construction. In this case, the renderer could quickly determine which object to test against if testing for light source occlusion. For scenes with many potential occluders, this would constitute a major speed increase since the renderer could quickly discard irrelevant occluders.

3.2.3. Handling curved surfaces

In the previous discussions, it has been assumed that the points in K all lie in the same plane. This is not necessarily true — for surfaces with abrupt, frequent changes in curvature the assumption that a small group of points will always be distributed "more or less" in the same plane may not be valid. If handling such a situation is required, one could simply, as part of the process described in section 3.2.1 for every point under consideration, verify the variance of the y-coordinate. If this variance becomes too big, it would be a sign that the assumption of planarity is not valid for this set of points. If such a case arises, a simple solution would be to either use more photons to obtain a denser point distribution (resulting in less local variance), or to decrease the value of K (see section 3.2.1).

3.3. Using the cone set in the rendering pass

Having constructed a cone set C for every object in the scene which is illuminated by direct lighting, we now need to use this information in the rendering pass.

3.3.1. Determining cone intersections

For a given parametric ray segment $\mathbf{R}(t) = \vec{P} + \vec{V}t$ described by $t \in [0,1]$, we would like to know over which intervals of t the path to the light source is not occluded, over which intervals it is partially occluded, and where it is fully occluded. Generally, the cone sets will be fairly conservative

in their estimate of occlusion volumes: points lying outside of a cone are almost certain of having a clear view of the light source. Just how conservative the estimate is can be adjusted by making the cone's expansion rate be a fraction of Θ.

As a first step, any intersections of the ray with the cone set are calculated (appendix A presents the mathematics for doing so). Because it is fast to determine the point of intersection of a ray and a cone, this step is fast. The result is a collection of intervals (see figure 5). These intervals are sorted in increasing order (we have found bubble-sort to work well, since the number of intersections is generally fairly small) and collected, so that any overlapping intervals are merged together.

The result is a set of intervals I_s indicating possible shadowing by an object, and a set of intervals I_c indicating areas where no occlusions occurs. This information can be used in various ways by the renderer. A straightforward application is to simply not do any occlusion testing in the I_c intervals, which in itself will significantly speed up the rendering process. This is the only speedup we currently use in our renderer. We can also vary the sampling density according to interval boundaries: within an interval from I_s we take many samples close to the edges of the interval where the variation in intensity is likely to be large, and fewer samples close to the centre of the interval.

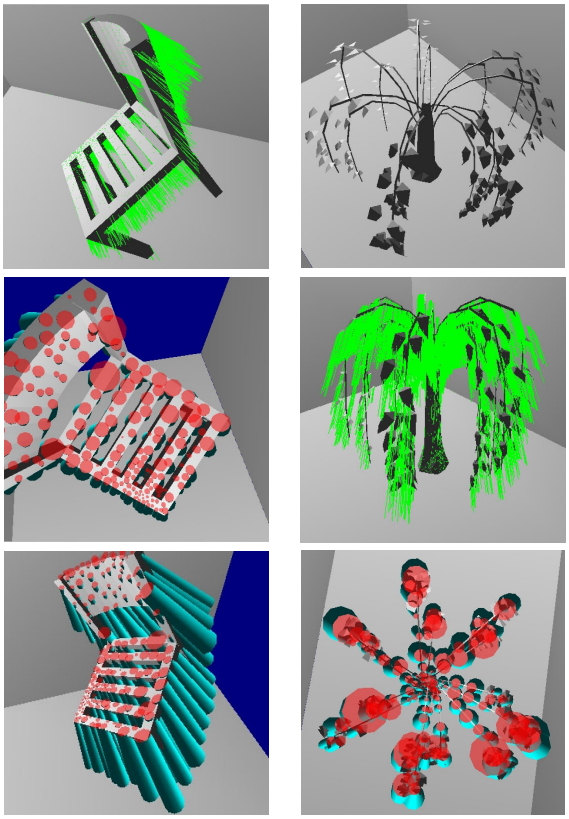

Figure 7: *Two difficult geometric models for the algorithm. In both cases, the shadow cones are shown in blue (they have been truncated for clarity of presentation), and their bases are superimposed in red, to estimate how accurately the cone set captures the underlying geometry of the structure. Shadow vectors are shown in green. As can be seen in the top left picture, which is from a vantage point close to the light source, the fine geometry of the chair's seat has been captured quite successfully — both in terms of the initial radius of the cones (only a few overspill significantly), as well as in terms of the expansion rate of the cones. Parameters for both cases were* $p_1 = 100$ $p_2 = 20$ $p_3 = 16$ $p_4 = 20$ *(see section 3.2.1) and the initial shadow vector set comprised approximately 5800 vectors in both cases, generated by the emission of 40000 photons from the light source in the case of the chair, and 80000 in the case of the tree (the increase is due to the very thin, narrow geometry of the tree). Currently the parameters* $p_1 \ldots p_4$ *are determined empirically based on manual experimentation. A future project consists in automating the determination of these parameters (see section 6).*

4. Results

4.1. The test scene

As the main testing environment, we have chosen a standard Cornell box with diffuse walls. We have then inserted a variety of objects into the box. The scene is illuminated by a disc-shaped area light source of finite radius. Although our renderer supports global illumination algorithms such as (volume, surface and shadow) photon maps, all rendered images only show direct lighting. This is partly to keep computation time down, partly because we wish to emphasize the volume shadows, which would be less distinct if multiple scattering were taken into account. The VDO is simulated using the Blasi-Schlick phase function approximation to Rayleigh scattering [3], and the medium is assumed to be homogeneous (simulating an inhomogeneous medium would make no difference to the algorithm, but would blur out the volume shadows).

4.2. Cone construction

As the main test case for the cone construction algorithm, we have chosen an occluder object with geometrical features that are difficult to capture for the technique. The chair depicted in figure 7 is mostly made up of thin, straight features at right angles to each other — a case which is quite difficult to handle for cylindrical/circular shapes such as cones. As

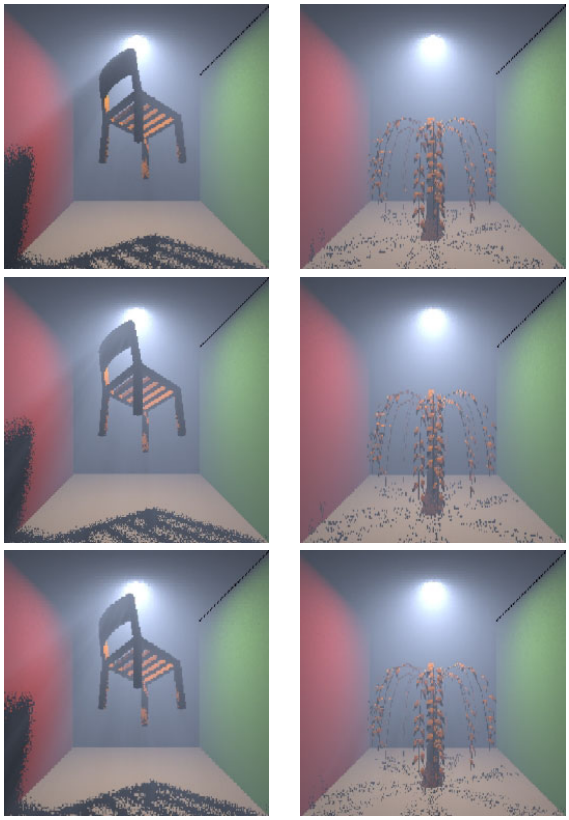

Figure 8: *Rendering the tree and chair models. The first row shows the reference images, rendered using brute force path tracing. The second rows shows the "intermediate" setting, where occlusion testing is only done in the intervals covered by the cones (see section 3.3.1). The third row shows fast approach, which simply consists of not taking any samples within the cone-intersection intervals.*

can be seen in the figure, where the base radii of the cones are shown as red circles, it is rare that a cone base goes beyond the border of the geometry of the chair. The tree is even more difficult: consisting of long, very thin branches and sparse clusters of leaves, it would be a challenging geometry to handle for any rendering method. However, even using the same parameters as for the chair, the constructed cone set follows the outlines of the branches and leaves reasonably well — it certainly doesn't seem to miss any important features of the tree.

4.3. Rendering

Figure 8 shows the results of rendering the scene with the chair object and the tree object, respectively. Rendering was done at an image resolution of 256 by 256 pixels, with 7 spectral samples. In all cases, sampling was done using a simple ray marcher which recursively super-samples the intensity along the ray if a too large discrepancy is detected. The back of the box has been set to black, so the fine variations in intensity in the VDO can better be percieved. All renderings were done on a multiprocessor Compaq Alpha DS20 (only one processor was used for the computations). The timings for the different figures is shown in table 1; all timings were conducted using the UNIX 'time' command. In the table, "fast" refers to a rendering approach where the light source is sampled in unoccluded intervals (with no occlusion test), and no samples are taken in occluded intervals. "Medium" refers to a sampling strategy whereby the light source is sampled evenly in both occluded and unoccluded intervals, but where occlusion testing (i.e., whether the light source is blocked or not from a given sample point) is only done in the occluded intervals.

timings	chair	tree
fast	106s	117s
medium	282s	298s
brute force	858s	580s

Table 1: *Approximate rendering times for the images shown in figure 8.*

As can be seen from the table, in terms of processing time, both the "fast" and "medium" settings are much faster than brute force path tracing [9]. As can be seen from the rendered images, the algorithm manages to capture many of the subtle variations in shading through the VDO — in some cases better so than the path tracing algorithm. On the other hand, the cone-based methods tend to cause an overestimation of the scattered light in some areas (around the top back of the chair model in particular). This seems to be mainly due to the bases of the cones near the surface of the seat being spaced apart too much. The parameters described in section 3.2.1 can be varied, however, to produce different results.

5. Limitations of the algorithm

One important limitation of the technique is that, at a certain distance from the apex of each cone, the renderer will start overestimating the portion of occluded space. Although this only helps to ensure that regions of space marked as unoccluded really are unoccluded, for the "fast" rendering method described in section 4.3 regions of the scene may be darkened excessively. This problem could be remedied by simply not using the cone estimate if the cone intersection point occurs too far from the base of the cone.

6. Future work

6.1. Cone set construction

Constructing optimal- or close-to-optimal cone sets which approximate the shape of the shadow volume as closely as possible will likely involve adjusting the parameters $p_1 \ldots p_4$ (described in section 3.2.1). We would like to be able to automatically provide some estimate of the optimal combination of these for a given object and shadow vector density. Finding ways of doing so is a future path of investigation.

6.2. Rendering

For accelerating the rendering step, a cone hierarchy could be constructed, wherein a "bounding" cone would encompass several shadow cones. This would provide a bounding volume to avoid unnecessary intersection tests. Also, some of the suggestions proposed in section 3.3.1 should be implemented to try to find a more optimal way of using the information given by the cone set to direct sampling. Also, an automatic way of assessing the accuracy of the cone set approximation at a given point along the ray could help solve inaccuracies such as those occurring near the back of the chair (see the two lower left images in figure 8).

7. Conclusion

We have presented a method for determining, modeling and rendering occluded portions of space. Our algorithm is designed to work together with the photon map technique, and hence can be implemented directly into a renderer which already supports photon maps. To model the volume, we use a set of primitives with fast intersection tests and very little memory overhead. Our method, like the photon map, does not need any knowledge of the geometry of a scene to function; we have presented a technique for detecting geometrical discrepancies when constructing the cone set, and hence our model can work with very difficult geometrical structures. Furthermore, our model supports area light sources. When rendering scenes comprising volumetric density objects, we can obtain good estimates of the final appearance of the shadowed volumes in a fifth of the time or less that a basic ray marcher would use. Contrary to shadow maps, cone volumes do not directly exhibit aliasing problems, and they can be used with area light sources.

The technique can be used in different ways, depending on the user's aims: either it can generate a fast preview of the scene, or it can help speed up a more accurate rendering. Although it has been presented within the context of rendering scenes with VDOs, it can also be used for occlusion detection for detecting shadows on surfaces.

8. Acknowledgments

We use the Approximate Nearest Neighbour (ANN) library [1] by David Mount and Sunil Arya for kd-tree operations. The tree model is shipped with the tutorials for 3D Studio MAX by Discreet. We use Möller and Trumbore's source code for fast ray-triangle intersection calculations (published in [12]). The first author would like to thank Dr. A. Boswell and Dr. J. Harold for their help with setting up and using the UEA high performance computing facilities.

References

1. S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891–923, 1998. ANN can be downloaded from http://www.cs.umd.edu/ mount/ANN/. 8

2. J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9), September 1975. 1

3. P. Blasi, Bertrand Le Saëc, and Christophe Schlick. A rendering algorithm for distrete volume density objects. *Computer Graphics Forum*, 12(3):201–210, 1993. 2, 6

4. David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and Modeling: a Procedural Approach*. AP Professional, 1994. 3

5. Andrew S. Glassner. *Introduction to Ray Tracing*. Academic Press, 1994. 9

6. Henrik Wann Jensen. Global illumination using photon maps. In Xavier Pueyo and Peter Schröder, editors, *Rendering Techniques '96*, Eurographics, pages 21–30. Springer-Verlag Wien New York, 1996. Proc. 7th Eurographics Rendering Workshop, Porto, Portugal, June 17–19, 1996. 1, 2

7. Henrik Wann Jensen and Niels Jørgen Christensen. Efficiently rendering shadows using the photon map. *Proceedings of Compugraphics '95*, pages 285–291, 1995. 1, 2, 3

8. Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. *Proceedings of SIGGRAPH 98*, pages 311–320, July 1998. ISBN 0-89791-999-8. Held in Orlando, Florida. 1, 2

9. James T. Kajiya. The Rendering Equation. In *Seminal Graphics: Pioneering Efforts That Shaped The Field*. ACM SIGGRAPH, 1998. Reprinted from Computer Graphics Vol. 20, No. 4, August 1986, pp. 143–150. 3, 7

10. Craig Kolb, Pat Hanrahan, and Don Mitchell. A realistic camera model for computer graphics. *Proceedings of SIGGRAPH 95*, pages 317–324, August 1995. ISBN 0-201-84776-0. Held in Los Angeles, California. 1

11. Tom Lokovic and Eric Veach. Deep shadow maps. In

Proceedings of SIGGRAPH 2000, pages 385–392, july 2000. 2

12. Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of graphics tools*, 2(1):21–28, 1998. 8
13. Mark Pauly, Thomas Kollig, and Alexander Keller. Metropolis light transport for participating media. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 11–22, June 2000. ISBN 3-211-83535-0. 3
14. Holly E. Rushmeier. Rendering Participating Media: Problems and Solutions from Application Areas (revised edition). In Sakas, Shirley, and Mueller, editors, *Photorealistic Rendering Techniques*, Proceedings of the Fifth EUROGRAPHICS Rendering Workshop. Springer-Verlag, 1994. 2
15. Holly E. Rushmeier. *A Basic Guide to Global Illumination (course 5)*. SIGGRAPH '98 Course Notes. SIGGRAPH '98 Course Notes, 1998. 2
16. Gernot Schaufler, Julie Dorsey, Xavier Decoret, and François X. Sillion. Conservative volumetric visibility with occluder fusion. *Proceedings of SIGGRAPH 2000*, pages 229–238, July 2000. ISBN 1-58113-208-5. 2
17. Peter Shirley, Changyaw Wang, and Kurt Zimmerman. Monte carlo techniques for direct lighting calculations. *ACM Transactions on Graphics*, 15(1):1–36, January 1996. ISSN 0730-0301. 1
18. László Szirmay-Kalos. *Monte-Carlo Methods in Global Illumination*. Winter School of Computer Graphics, 1999. 1
19. Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *Proceedings of SIGGRAPH 97*, pages 65–76, 1997. 3
20. Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of SIGGRAPH 87*, pages 270–274, 1978. 1, 2

Appendix A: Calculating intersections with cones of varying aperture

The canonical equation of a cone is usually given as [5]:

$$x^2 + y^2 = z^2 \tag{4}$$

This formulation describes a cone of half-aperture $\theta = \pi/4$, centered on the z-axis. We would like to reformulate equation 4 so it can describe a cone of any half-aperture $0 < \theta < \pi/2$. We assume this requires a constant factor F, which varies as a function of θ:

$$x^2 + y^2 = Fz^2 \Leftrightarrow F = \frac{x^2 + y^2}{z^2} \tag{5}$$

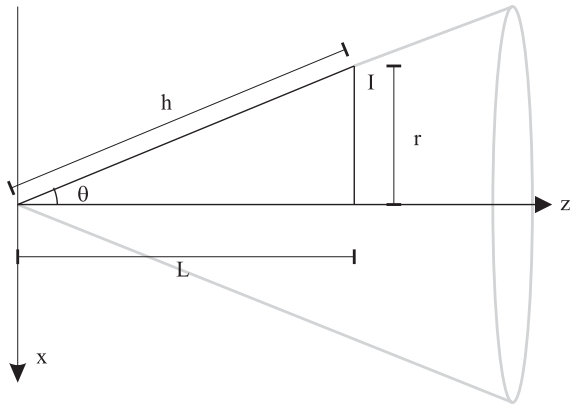

Figure 9: *Illustration of the notation used in appendix A.*

Looking at figure 9, we can observe a few properties of the geometry of a cone. Firstly, L and r are related by simple trigonometry:

$$\tan\theta = \frac{r}{L} \Leftrightarrow L = \frac{r}{\tan\theta} \tag{6}$$

Second, a cone is rotationally symmetrical about its central axis, which in the case of figure 9 is the z-axis. Hence, if we solve F for the two-dimensional case presented in figure 9, the result will be valid for any three-dimensional case. Therefore, we can reformulate equation 5, setting $y = 0$; in this case, x corresponds to the radius of the cone at any point on its surface (r in figure 9) and z corresponds to L in the figure. Substituting this into equation 5 yields

$$F = \frac{r^2 + 0}{L^2} \tag{7}$$

Substituting equation 6 into equation 7 results in the following equation:

$$F = \frac{r^2}{\frac{r^2}{\tan\theta}} = \tan^2\theta \tag{8}$$

So the equation we are looking for is:

$$x^2 + y^2 = z^2 \tan^2\theta \tag{9}$$

Substituting the parametric ray equation (as described in section 3.3.1) into equation 9 and collecting for t gives the following quadratic equation:

$$At^2 + Bt + C = 0 \tag{10}$$

with

$$\begin{aligned} A &= V_x^2 + V_y^2 - V_z^2 \tan^2\theta \\ B &= 2(P_x V_x + P_y V_y - P_z V_z \tan^2\theta) \\ C &= P_x^2 + P_y^2 - P_z^2 \tan^2\theta \end{aligned}$$

Solving equation 10 for a given ray gives the usual three possibilities: two roots in case two intersections are found,

one root in case the ray grazes the cone, and no roots if no intersection occurs. Note that $\tan^2\theta$ can be precalculated for a given cone.