Accelerating Ray Shooting Through Aggressive 5D Visibility Preprocessing

Adrian Sharpe Matthew Hampton Shaun Nirenstein James Gain Edwin Blake*
Department of Computer Science
University of Cape Town

Abstract

We present a new approach to accelerating general ray shooting. Our technique uses a five-dimensional ray space partition and is based on the classic ray-classification algorithm. Where the original algorithm evaluates intersection candidates at run-time, our solution evaluates them as a preprocess.

The offline nature of our solution allows for an adaptive subdivision of ray space. The advantage being, that it allows for the placement of a user set upper bound on the number of primitives intersected.

The candidate sets produced account for occlusion, thereby reducing memory requirements and accelerating the ray shooting process. A novel algorithm which exploits graphics hardware is used to evaluate the candidate sets. It is the treatment of occlusion that allows for the practical precomputation of the ray space partition. This algorithm is called aggressive since it is optimal (no invisible primitives are included), but may result in false exclusion of visible primitives. Error is minimised through the adaptive sampling.

CR Categories: I.3.3 [Picture/Image Generation]: Photorealism I.3.5 [Computational Geometry and Object Modelling]: Ray Shooting I.3.7 [Three-Dimensional Graphics and Realism]

Keywords: Ray shooting, ray casting, ray tracing, global illumination, visibility

1 Introduction

Ray shooting is a simple but computationally expensive task that is fundamental to many algorithms in computer graphics. Stated simply, the problem is to find the nearest intersection of a ray with a set of geometric primitives, with respect to the ray origin. Ray shooting is used to calculate form factors for radiosity, to create photon maps, and is, of course, the central component of both classic and Monte-Carlo ray tracing. In this paper we present a new approach to accelerating ray shooting.

A naïve approach requires that every ray is tested against every geometric primitive in the scene. In order to reduce the number of ray-primitive intersection tests required, researchers have developed a range of acceleration schemes. The most popular schemes use spatial subdivisions, such as binary space partition (BSP) trees or uniform grids. The goal of such schemes, is to trivially reject those parts of the scene which cannot be intersected by a given ray.

*{asharpe, mhampton, snirenst, jgain, edwin}@cs.uct.ac.za

With the same objective in mind, we propose an approach based on Arvo and Kirk's [1987] *ray classification* technique. The original algorithm subdivides the scene using a 5D partition of ray space[1]. The algorithm *classifies* the query ray into a 5D cell and returns a candidate set of primitives associated with this cell. Our key contributions are:

1. An algorithm which fully preprocesses the scene, effectively precomputing all candidate sets. This allows for more efficient ray shooting after a once-off preprocess.
2. An upper bound on the candidate set size at run-time. This may be used to obtain upper time bounds for run-time rendering.
3. A new technique for computing the candidate sets. Graphics hardware is exploited to accelerate the preprocess. Adaptive sampling is used to minimise error.
4. We present a method which accounts for occlusion within the candidate sets. This increases the performance of the ray shooting by trivially rejecting *all* hidden primitives, while simultaneously decreasing the memory requirements (to feasible levels) for a full preprocess.

We begin with a brief discussion of related literature. In Section 3 we discuss the 5D processing of ray space. In this section, our subdivision algorithm is presented, and our technique for generating candidate sets is described. In Section 4 we present some preliminary results.

2 Previous Work

Since Rubin and Whitted [1980] introduced bounding volumes, there has been much research devoted to ray-shooting acceleration schemes. Most methods use 3D spatial subdivision in conjunction with a ray traversal algorithm. Each cell or voxel contains a list of the primitives that are fully or partially contained within it. The ray traversal algorithm traverses the cells along a ray in order. The list of the nearest cell is tested first, so that if an intersection point is found within the boundary of the cell, no further cells need be tested. However, if no intersection is found, or the intersection point is outside the cell's boundary, the primitive list within the next cell must be tested.

These schemes perform reasonably well in the average case (since a ray is more likely to first intersect the near primitives that are tested first). However, a ray may still be tested against a number of cells and their contained primitives before an intersection is found.

Arvo and Kirk [1987] describe a subdivision of 5D ray space, that eliminates this costly traversal of cells. A 5D *cell* is defined by a parallelepiped of ray origins in 3D (which we will call the *origin box*) and a range of ray directions. It has a natural manifestation in 3D as a *beam*. If a beam does not intersect a primitive then no ray in the 5D cell can either. This property is used to find a relatively small *candidate set* for each cell.

[1]Each ray can be uniquely defined by an origin in 3D space along with two spherical angles for direction

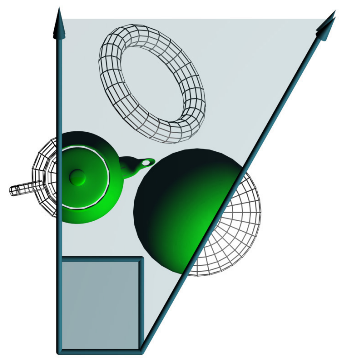

Figure 1: A beam that intersects many polygons, but few are possibly visible from the origin box due to occlusion. The polygons in the candidate set are solid, while polygons that are occluded or are outside the beam are shown in wire frame.

A ray need only be tested against the candidate set of the cell that contains it. Classifying a ray requires at most one tree traversal (this is reduced further in practice by using a caching scheme). In the context of [Arvo and Kirk 1987], the purpose of the 5D subdivision is to function as a caching mechanism. The candidate sets are evaluated lazily, the premise being that the cost of the candidate set computation would be amortised over successive "cache hits" (i.e. successive rays with the same classification).

Unfortunately this method is very memory intensive because of the deep level of subdivision required before candidate sets are sufficiently small [Simiakakis and Day 1994]. This is further exacerbated by the highly conservative nature of these candidate sets. The implementation presented by Arvo and Kirk is conservative, firstly, because occluded primitives are not culled from the candidate sets (see Figure 1), and secondly, because the beam used is a conservative overestimate of the optimal beam. This results in primitives outside of the beam, being included as candidates.

Arvo and Kirk compensate for their conservative approach by *truncating* the candidate sets. Primitives that are further than a specified distance from the origin box are omitted. The beam is capped by a truncation plane and if a ray does not intersect any primitives in the candidate set, it is projected onto the truncation plane and reclassified. This can be effective in many cases, but the initial advantage of a single tree traversal and candidate set per ray is lost. It would be better to evaluate which primitives are occluded and remove them.

Nirenstein *et al.* [2001] describe an aggressive visibility technique that uses *visibility cubes* to determine visibility from 3D regions (in all directions). A visibility cube (strongly related to the radiosity hemi-cube [Cohen and Greenberg 1985]) is created by treating each face of a tiny cube surrounding the view point as an independent depth buffer. The scene is rendered onto the buffers using a distinct colour index for each polygon. The buffers are read back and the indices present represent the polygons visible from that point. The rendering can be accelerated with modern graphics hardware. See [Nirenstein et al. 2001] for details.

An *aggressive* visibility solution is optimum in that no invisible primitives are included, but may result in false exclusion of visible primitives. The technique samples visibility adaptively over the surface of each cell while building a k-d tree of visibility cells. Nirenstein *et al.* successfully accelerate the rendering of large scenes (millions of polygons) with acceptable error.

Our technique is similar to that of Sudarsky and Gotsman [1999], however, their subdivision scheme is optimised for visibility culling, rather than ray shooting. Furthermore, their sampling does not exploit graphics hardware (thereby compromising either time or accuracy), nor does it effectively use the information of previously cast rays to minimise error.

3 5D Preprocessing

Our solution to the ray shooting problem is an adaptation of the aggressive visibility technique of Nirenstein *et al.* to choose candidate sets for a 5D subdivision similar, to that of Arvo and Kirk. Using an aggressive technique results in an optimum candidate set being chosen for each 5D cell (with minimal false exclusion). All visibility determination is performed offline as a preprocess.

The algorithm may be summarised as follows:

```
Generate initial set of cells $V$
Set primitive budget to $threshold$
for each $c \in V$ do
    Subdivide( $c$ )
next $c$

procedure Subdivide( $c$ )
    Compute Candidate Set of $c$
    if $ElementsIn(c) \geq threshold$ then
        $\{c_-, c_+\} \leftarrow$ Split($c$)
        Subdivide( $c_-$ )
        Subdivide( $c_+$ )
    end if
end procedure

procedure $\{c_-, c_+\} \leftarrow$ Split($c$)
    Choose 5D splitting hyperplane for $c$ such that:
        $ElementsIn(c_-) + ElementsIn(c_+)$ is minimised
        Return $\{c_-, c_+\}$
end procedure
```

3.1 The Initial Classification

Only those cells whose origins lie within the 3D bounding volume of the scene need be evaluated. For those rays with origins lying outside, but still intersecting the bounding volume, the origin can be moved to the point of intersection and reclassified.

We begin by constructing six cells. The origin box of these cells is the scene bounding box. To specify the angular bounds we note that the direction of any ray can be associated with its *dominant axis*, denoted $+X$, $-X$, $+Y$, $-Y$, $+Z$ or $-Z$. Using this association, Arvo and Kirk [1987] define an isomorphism between the sphere of directions, and the surface of an axially aligned cube. It is convenient to initially partition the direction space by the six dominant axes (discussed in Section 3.3.1). This leads to the six initial cells.

3.2 Adaptive Subdivision

A spatial subdivision of the 5D ray space is used to accelerate ray shooting. At each level of the subdivision the size of the candidate set is reduced. At run-time, this reduces the computation cost of finding the first ray-primitive intersection by reducing the total number of required intersection tests.

The cost of ray intersection becomes $O(\log n + c)$, where n is the number of cells, and c is the user defined candidate set maximum. The logarithmic traversal time, assumes that the tree has been balanced as a post-process. This is hugely beneficial, considering that in practice, $\log n$ is very small, and c may be chosen to be arbitrarily small. No other practical techniques limit the number of intersection tests prior to finding the first ray-primitive intersection.

3.2.1 Separation and Effectiveness

Separation refers to how the primitives are split and shared between the child candidate sets. It is used to measure the effectiveness of a subdivision. An effective subdivision will share very few primitives between the child cell and thus have high separation. The effectiveness of a subdivision is important for deciding how to subdivide a cell, and whether a given subdivision is beneficial.

3.2.2 Subdivision strategies

Naïve subdivision

A subdivision strategy in which each dimension is divided in turn at each level, up to a specified maximum depth, is the simplest to implement and test. It produces a complete k-d tree, where every branch has the same height.

Maximum Candidate Set Size

The naïve subdivision strategy is improved by changing the terminating criteria so that subdivision stops early when the candidate set size drops below a specified user maximum. A maximum depth limit is also provided to stop subdivision from continuing indefinitely. This also produces a k-d tree, however, it is not complete.

Most Separated Dimension

This strategy involves looking at the set of all possible subdivisions and choosing the one that gives the best separation. An analytic solution for this is extremely expensive. However, aggressive visibility sampling can also be used to perform this process. A limited sampling of the 5 possible splitting hyperplanes is used to approximate the separation. The subdivision with the most separation is then chosen and fully sampled. This produces an axis aligned binary space partition tree.

3.2.3 Optimisations

For a given scene the cost of the preprocess is largely related to the number of samples taken. We employ a number of simple optimisations to decrease this overhead:

- The candidate set for a child cell is always a subset of the candidate set of its parent. Therefore, to determine visibility for a child cell, only the primitives visible in the parent cell need to be rendered. Fewer primitives results in faster rendering of visibility samples at deeper levels of the subdivision.
- For any cell, the list of pertinent samples necessary for determining visibility is stored with the cell. These samples are reused in child cells, if possible. A sample can only be reused if the current subdivision does not cross its bounds (see Section 3.3.1 and 3.3.2). Reuse can be increased by storing more that just a visibility list with each sample, for example, by breaking the rendering into a grid of lists. The reuse of samples drastically reduces the sampling cost, since only the splitting surface needs to be sampled for any subdivision.
- Caches are also implemented to facilitate the reuse of samples across subdivision branches. This allows samples to be reused on neighbouring cells that do not share a parent (that are in separate branches of the tree). All requests for samples are directed through the cache. A sample is created if it does not already exist. Samples are eliminated from the cache when they are no longer needed.

3.3 Constructing Candidate Sets

The candidate set for a 5D cell is the union of the primitives that intersect the origin box and the primitives visible by rays that originate from the surface of the origin box, with direction bounded by the angular range of the cell In order to determine what is visible from a surface we sample the set of rays that can originate from that surface. This is done using two complementary methods: *point-samples* and *area-samples*. For a point-sample, we sample many rays originating from a single point on the surface (see Nirenstein *et al.* [Nirenstein et al. 2001]). Conversely, for an *area-sample*, all the rays sampled are parallel, but have a different origin (see [Chrysanthou et al. 1998]). Both can be created using graphics hardware. The details are as follows:

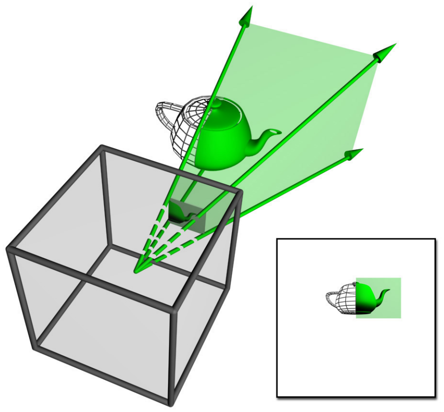

Figure 2: A visibility cube with point-sample frustum restricted to the subset of angles required (uv bounds $(0,0)$ to $(0.6,0.5)$). The inset shows a typical rendering for a full face of a direction cube. The shaded region is all that need be rendered for a restricted point-sample.

3.3.1 Point-samples

Within the context of ray space, the desired point-sample is a restricted visibility cube. The initial six way subdivision of ray space means that only one face of the visibility cube need be rendered for any sample. Furthermore, the 2 angular bounds restrict the portion of the face that has valid samples. To ensure that the entire rendering is useful, the point-sample is created by rendering the scene using a perspective projection, where the frustum reflects the angular bounds of the 5D cell. This is illustrated in Figure 2.

In our implementation, each point-sample reveals what is visible from a point over the full range of the angular bounds of a 5D cell. This is manipulated as a single set of items. As such, one cannot determine in which direction a particular primitive in the set is visible, but merely that it is visible in some direction from the point-sample origin. This has efficiency implications, since a subdivision of one of the angular dimensions requires that all existing point-samples must be discarded. One cannot reuse the sample in this case because the direction information has not been preserved.

The reuse of samples greatly accelerates the performance of this technique, especially for large scenes. Having to discard a sample is an expensive operation. We reduce the effect of this by breaking up a point-sample into a number of sets, each representing an angular region of the rendering. This is done by splitting the rendering into a grid of separate samples on the visibility cube. When an angular subdivision takes place the relevant lists are copied into a new sample. Although this reduces the problem to a certain degree, it does not remove it. The only way to do so is to store the entire rendering for each point-sample. Even with image compression, the memory requirements for this would be excessive.

Point-sampling results in a large number of directional samples for a small number of origins. Improving the coverage of sampled origins, involves the evaluation of more point-samples. An adaptive sampling of the surface is used to achieve this. The four corners of the surface are sampled and then further samples are taken recursively in the manner of a quad-tree. Termination is governed by an error minimising heuristic.

We use a heuristic based on that of Nirenstein *et al.*. Although this heuristic was developed for full directional visibility cubes for determining from-region visibility, it we use it with almost no adaptation for our point-samples[2]. This heuristic uses an image based similarity measure between adjacent samples. The user defines the error threshold. When the number of common items is above the threshold then subdivision stops.

The angular restriction to the visibility cube introduces a number of sampling problems, illustrated in Figure 3. Sampling can leave significant portions of the scene un-sampled. Primitives in these portions will be falsely categorised as not visible if the terminating condition is satisfied for the outer samples. This problem arises because all of the rays sampled by the point-samples are clustered around a few key origins. The problem is further exacerbated when the angular bounds are small, since the resulting gaps become larger. It is particularly significant since the un-sampled regions are near the origin box surface.

In order to improve the coverage of the ray samples and to avoid the sampling gaps that are created by point-sampling, an alternative sampling methodology, termed *area-sampling* is also used.

3.3.2 Area-samples

Rather than sampling a range of ray directions from a single origin, an area-sample shoots parallel rays from a range of origins across the surface. This is achieved through a generalisation of the restricted visibility cube used for point-samples. Point-sampling uses perspective projection, with the centre of projection on the surface and the front clipping plane very near to the surface. The area-samples use an orthographic projection in the specified direction with the sampling surface itself as the front clipping plane.

We adaptively sample visibility from the surface starting with four samples at the four angular extents. However, the original termination heuristic leads to unnecessary oversampling on the four surfaces of the origin box that are orthogonal to the dominant direction. This is because the orthographic projection includes a shearing operation used to achieve the desired ray direction. This shearing is particularly significant for area-samples taken from these surfaces and stretches insignificant polygons such that they contribute too much to the error measure. Scaling the polygons contribution by the shearing factor produces a better weighted sampling.

Area-samples and point-samples have complementary properties. Where point-samples cannot be reused after splitting an angular dimension, area-samples cannot be reused after splitting the x, y or z dimension. The reason is the same: the only information kept with a sample is the list of visible primitives.

There are also sampling problems with area-samples (see figure 3. Where point-samples have un-sampled regions close to the origin box surface, area-samples leave more remote regions unsampled. It is also evident that point-sample gaps are larger in scene volume, when the sampling surface is large, whereas area-sample gaps are larger when the sampling surface is small).

Because of their complementary nature, the best solution uses a combination of point-samples and area-samples. The nature of the sampling problems of each sample type suggests that more area-samples should be used in the early stages of subdivision when the sampled cell surfaces are large. More point-samples should be used deeper in the hierarchy, when the cell surfaces are smaller.

Uniform sampling of origin box surfaces is required for efficiency. Since child cells samples are based on their parent cell's visibility, it does not prove useful to adaptively sample a surface to further depth than the parent did. No extra information can be recovered. Similarly, it is not useful to render the item buffers at the same or higher resolution for the child cells. Instead, the maximum adaptive sampling depth and the sampling resolution can be reduced as the subdivision takes place. This ensures that no time is spent rendering more samples than are absolutely necessary for visibility determination.

[2]The contribution that each polygon makes to the error measure is scaled by the number of sample rays that intersect it. Nirenstein *et al.* use a constant weight.

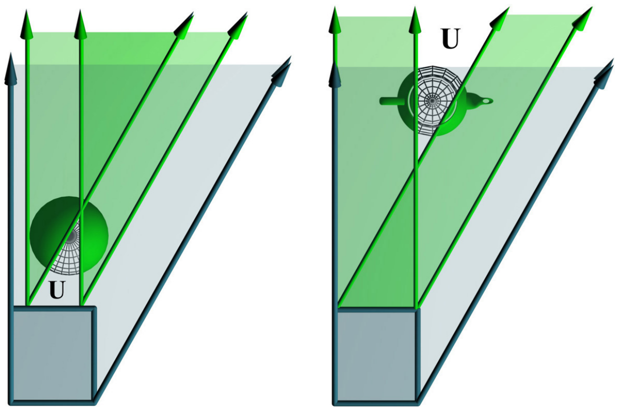

Figure 3: Sampling problems with point-samples (a) and area-samples (b). The polygons in the un-sampled regions (U) will be classified as not visible (rendered in wire frame).

	Scene	Size	Cell Depth	Sample Depth	Sub. Strategy
1	roomsbig	45k	10	6	A
2	roomsbig	45k	15	6	A
3	hilltop	485k	12	6	B

Table 1: Preprocessing input parameters. A is the Maximum Candidate Set Size subdivision strategy while B is the Most Separated Dimension strategy (see Section 3.2.2)

4 Preliminary Results

The techniques presented in this report have been fully implemented. A number of scenes were processed using a dual Pentium 4 1.7Ghz with 1.2GB of RAM and an NVidia G-Force 4 Ti 4600 graphics card. We have evaluated the performance of the preprocessor, the accuracy of the visibility information and the degree of ray shooting acceleration.

4.1 Performance

Table 1 and 2 give the preprocessing parameters and results, respectively, for a number of trials using three different scenes. The length of the preprocess depends mainly on the total number of samples that are rendered. This number is directly related to the user specified *maximum cell depth* and *maximum sample depth* parameters. We use both the Maximum Candidate Set Size (A) and the Most Separated Dimension (B) strategies (see Section 3.2.2).

Both subdivision strategies effectively reduce the average and maximum number of triangles for each candidate set. Trial 5 successfully processed a large scene and reduced the average number of triangles per cell to 0.11% of the original scene size. The maximum number of triangles for a cell was 2.94% of the scene size.

We have found that subdivision strategy B takes significantly longer because of the extra processing involved in choosing the splitting hyperplane. However, this increased preprocessing time is balanced by a reduction in preprocessed data file size. The max-

	Time h:mm	Cells	Tri/Cell Ave.	Tri/Cell Max.	Cell Depth Ave.	Cell Depth Max.	Sample Depth Ave.	Sample Depth Max.	File Size (MB)
1	0:25	5263	625	4944	9.9	10	4.7	6	29
2	1:51	82192	318	2312	14.5	15	5.0	6	143
3	38:43	17972	535	14267	11.9	12	5	6	93

Table 2: Preprocessing results.

	Min.	Max.	Ave.	St. Dev.
1	0.00%	1.79%	0.03%	0.17%
2	0.00%	9.58%	0.22%	0.98%
3	0.00%	82.19%	3.65%	10.78%

Table 3: Error rate statistics for ray traced images.

imum file size was 93Mb, only 2.5 times larger than the scene file. This is compared to Strategy A, whose maximum file size was 50 times bigger than the scene file.

4.2 Accuracy

The accuracy of the preprocessor is evaluated by comparing images produced using the preprocessed visibility files against images produced using a reference renderer. For our evaluation the reference renderer used a bounding volume hierarchy (BVH) [Havran 2000] to find the closest intersection. The reference ray tracer produces an exact image. We calculate the number of incorrect intersections using our 5D visibility information. Table 3 shows the accuracy statistics of the trials.

Trial 1 and 2 produce acceptable rates of error. Trial 3 had a higher average error rate of 3.65%, but this is acceptable considering the size of the scene and the depth of the preprocess. Unfortunately, its maximum error rate was 82.19%, a substantial proportion of the image. This image, and most of the others with error rates above the average, had viewpoints near a falsely excluded polygon. This polygon counted for the majority of the error.

4.3 Ray Shooting Acceleration

The success of the technique for accelerating ray tracing depends directly on its ability to reduce the number of intersection tests required. Table 4 highlights the performance of the technique. The $\times$ *Speedup* column is relative to the naïve approach (intersecting every ray with every primitive). The speedup using our technique is compared to the speedup using a bounding volume hierarchy (BVH).

For all scenes the number of intersections per ray was vastly reduced compared to a naïve approach to ray shooting. 5D preprocessing successfully reduced the size of the candidate set to a manageable level. While 100 to 600 intersections per ray may not seem impressive, it is important to realise that these figures assume the candidate sets are not further processed (this is also why the BVH performs better on average).

The traversal of the 5D tree requires one comparison per node, thus with a maximum overhead of 15 floating point comparisons the candidate size can be reduced to as little as 0.1% of the total scene

	Size	5D Ave. Int/Ray	5D × Speedup	BVH Ave. Int/Ray	BVH × Speedup
1	45k	594.94	75.51	40.46	1112.21
2	45k	126.20	356.57	40.46	1112.21
3	485k	290.34	1670.45	61.96	7827.63

Table 4: Ray shooting acceleration statistics

size and consequently reduce the number of required intersections to as little as 0.03% of the total scene size. These are promising results, considering that a hybrid of this technique with a BSP tree or bounding volume hierarchy offers the potential for substantial further reductions.

5 Conclusion

We have presented a novel approach to accelerate ray shooting. An offline 5D subdivision preprocess reduces the candidate set for a ray to a manageable size based on both visibility and occlusion. The results (in Table 2) reflect success in bounding both maximum and average ray intersections per ray.

Indeed, we effectively limit the number of intersections per ray to at most 2.5% of the scene size (but on average, less than 1%). This is at the cost of a small error: on average, less than 4% of rays return the incorrect primitive on very large scenes. Generally, the average error rate is closer to 0.3% (Table 3). Furthermore, our technique completes in practical time, even for large scenes.

5.1 Future Work

We intend to evaluate different schemes to post-process the candidate set lists. This should significantly improve the effectiveness of our scheme for accelerating ray shooting. In addition, we would like to employ more effective sampling strategies to reduce error rates, using better error heuristics and an intelligent strategy for combining point- and area-samples. We will continue to investigate performance enhancements to reduce preprocess times, including the possibility of parallelisation.

References

ARVO, J., AND KIRK, D. 1987. Fast ray tracing by ray classification. In *Computer Graphics, Annual Conference Series (SIGGRAPH '87 Proceedings)*, vol. 21, ACM, 55–64.

CHRYSANTHOU, Y., COHEN-OR, D., AND LISCHINSKI, D. 1998. Fast approximate quantitative visibility for complex scenes. In *Computer Graphics International 1998*, IEEE Computer Society, Hannover, Germany.

COHEN, M. F., AND GREENBERG, D. P. 1985. The hemi-cube: a radiosity solution for complex environments. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, ACM Press, 31–40.

HAVRAN, V. 2000. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague.

NIRENSTEIN, S., GAIN, J., AND BLAKE, E. 2001. Aggressive visibility pre-processing with adaptive sampling. Tech. Rep. CS01-01-00, Department of Computer Science, University of Cape Town, http://www.cs.uct.ac.za/Research/CVC/Techrep/CS01-01-00.pdf.

RUBIN, S. M., AND WHITTED, T. 1980. A 3-dimensional representation for fast rendering of complex scenes. In *Computer Graphics, Annual Conference Series (SIGGRAPH '80 Proceedings)*, vol. 14, ACM, 110–116.

SIMIAKAKIS, G., AND DAY, A. M. 1994. Five-dimensional adaptive subdivision for ray tracing. *Computer Graphics Forum 13*, 2, 133–140.

SUDARSKY, O., AND GOTSMAN, C. 1999. Dynamic scene occlusion culling. *IEEE Transactions on Visualization and Computer Graphics 5*, 1, 13–29. ISSN 1077-2626.