

# Voxel Occlusion Testing: A Shadow Determination Accelerator for Ray Tracing

Andrew Woo †  
John Amanatides ‡

Department of Computer Science  
University of Toronto  
Toronto, Ontario  
M5S 1A4

## 1. Abstract

A shadow determination accelerator for ray tracing is presented. It is built on top of the uniform voxel traversal grid structure. The accelerator proves to be rather efficient, requires little additional memory and the worst case scenario per shadow determination just reduces down to traditional voxel traversal. It can also be extended to model linear, area lights, as well as atmospheric shadows.

**Keywords:** grid, intersection culling, occlusion, penumbra, ray tracing, shadows, umbra, voxel traversal.

## 2. Introduction

The presence of shadows in a scene is important in conveying realism and aiding depth perception [Woo89b]. It is seen as a comparative darkness within an illuminated area caused by the interception of light by another object. The dark region produced can provide information such as the approximate shape and relative proximity of the intercepting object(s). It can also indicate the approximate location, intensity and size of the light sources.

Hard shadow determination algorithms for opaque surfaces have been categorized into six general classes: shadow generation during scanout [App68] [Bouk70], shadow volumes [Crow77] [Brot84] [Berg86] [Max86], shadow volume binary space partition tree [Chin89], area subdivision [Nish74] [Athe78], depth buffer [Will78] [Hour85] [Reev87], and ray tracing [App68] [Gold71] [Whit80]. The first four approaches have the general constraint that only planar polygons can be easily handled. The depth buffer approach [Will78] does not have this constraint but introduces additional aliasing artifacts. However, the aliasing artifacts have been reduced by filtering, as in the work done by [Hour85] [Reev87].

† Andrew can be reached at SAS Institute Canada Inc., 225 Duncan Mill Road, Don Mills, Ontario, Canada, M3B 3K9.

‡ John can be reached at Department of Computer Science, York University, North York, Ontario, M3J 1P3.

## 3. Shadow Determination in Ray Tracing

*Ray casting* [App68] [Gold71] was introduced as an elegant method for visibility calculations and a simple shadow determination approach without the previous shadow algorithm limitations. *Ray tracing*, as popularized by Whitted [Whit80], uses the ray casting technique to model reflections and refractions.

The principle behind shadow determination in ray tracing is simple. A shadow ray is shot from the visible point (point to be shaded) to the light source. If the shadow ray intersects any objects between the visible point and the light source, then it is in shadow; otherwise it is not. Each intersection check is very floating-point intensive and the naive ray tracing approach requires such checks with all objects in the scene. Since each ray-surface intersection check is very expensive, there is a need to calculate a small candidate set of objects that can possibly intersect the ray to reduce computation time. This is referred to as *intersection culling*.

## 4. Intersection Culling Algorithms

There has been a great deal of research in this aspect of ray tracing acceleration. One class of intersection culling algorithms uses spatial subdivision, of which there are two general approaches: *voxel traversal* [Glas84] [Fuji86] [Aman87] [Snyd87], and *hierarchical bounding volumes* [Rubi80] [Kay86] [Gold87].

Since the new shadow determination accelerator is built on top of the uniform voxel traversal grid structure \*, it is necessary to go over the fundamentals of voxel traversal. Space encompassing all objects is placed in a grid of unit cubes called *voxels*. Each voxel contains a list of all objects which reside in that voxel. Each ray traverses the grid in order and tests for intersection only with objects residing in the voxels traversed (in order), until an intersection is found or the ray has completely traversed the grid.

\* There are basically two variations of voxel traversal: variable sized voxels using hierarchical data structures [Glas84] [Fuji86], and uniformly sized voxels using a grid structure [Fuji86] [Aman87] [Snyd87]. Uniform voxel traversal is used in the implementation of the accelerator.

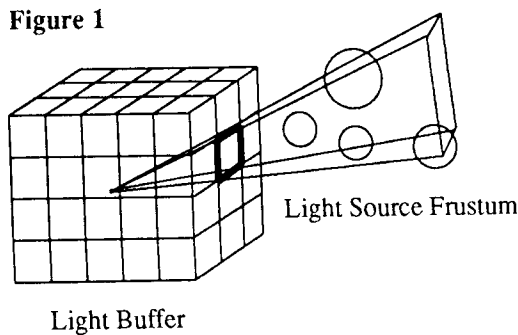
Voxel traversal culls all ray types (including shadow rays) and tends to provide small candidate set of objects that a ray needs to intersect with. However, many of the intersections performed when applied to shadow rays are still unnecessary and savings related to shadow calculations can prove to be valuable. Thus shadow ray cullers need to be looked into.

## 5. Shadow Ray Culling

Dealing only with opaque objects and hard shadow generation, shadow determination is just a binary decision. It is not concerned with information about object(s) that occlude it. Thus many intersection checks can be avoided. There have been only two attempts at shadow ray culling in ray tracing: *light buffer* and *hybrid shadow testing*.

### 5.1. The Light Buffer

An observation made by Haines et al. [Hain86] is that, in many scenes, shadow determination may dominate the ray tracing processing time. This is especially true when multiple light sources are involved. The *light buffer* [Hain86] is constructed to lower the processing cost of shadow determination.



The light buffer consists of six grid planes forming a box surrounding the light source origin (point light source is assumed). The cells of the buffer contain information on the smallest *full occlusion* (completely fills cell) distance from the light source, and sorted approximate depth values of candidate occlusion objects obtained by projecting objects onto the surface cells during preprocessing. See figure 1, where the four spheres within the light frustum are stored in the cell's data structure of candidate occlusion objects.

For each intersected point in question, if the depth value of the corresponding cell is greater than the smallest full occlusion distance, then the intersection point is in shadow. Similarly, if the cell is void of projected objects, then the surface is not in shadow. Otherwise, shadow determination requires intersection tests with candidate occlusion objects of the cell, performed in order with respect to the depth values until either an intersection hit is found (in shadow) or the depth value of the candidate occluding object is greater than the intersection point (not in shadow).

Haines et al. report a substantial improvement in the shadow determination phase. However, large memory space

is required:  $O(N^2n)$  per light source, where  $N \times N$  is the resolution of one grid plane of the light buffer, and  $n$  is the total number of objects. The preprocessing is also expensive:  $O(\max(EnN^2, N^2n \log n))$  per light source, where  $E$  is the average number of edges per polygon,  $EN^2$  accounts for the scan-conversion cost per object, and  $n \log n$  represents the sorting required per cell.

### 5.2. Hybrid Shadow Testing (HST)

*Hybrid Shadow Testing* (HST) [Eo89] is another approach to accelerate shadow determination. It applies a shadow polygon approach [Crow77] [Brot84] [Berg86] [Max86] and voxel traversal [Glas84] [Fuji86] [Aman87] [Snyd87]. The shadow polygons are placed in the voxel traversal grid structure, and the shadow count is kept up-to-date as the ray is traversed. No shadow rays are generated for this scheme, but intersection calculations with the shadow polygons, if encountered in a traversed voxel, are necessary. When the closest intersected surface is found, the shadow count is checked. If the count is 0, then the surface is not in shadow; otherwise it is in shadow.

Eo et al. [Eo89] realize that the number of shadow polygons are large. Thus each ray may potentially need to intersect many shadow polygons. As such, the traditional shadow ray approach applying voxel traversal is used if many such shadow polygons need to be intersected. The storage and runtime complexity for the HST algorithm is  $O(EN^3n)$  per light source, where  $E$  is the average number of edges per polygon,  $n$  is the number of objects in the scene and  $N \times N \times N$  is the resolution of the grid structure.

## 6. A New Shadow Determination Accelerator

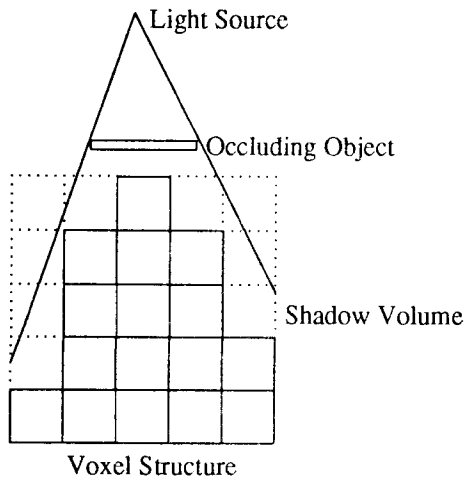
The new shadow determination accelerator, *voxel occlusion testing* [Woo89a], possesses favourable properties in that it uses a data structure already built for fast ray tracing, requires little additional storage, models directional and extended light sources, and has a good worst case scenario.

The implementation of the accelerator is built on top of uniform voxel traversal, i.e. each voxel is the same size. Given the uniform grid structure, an extra 2-bit field needs to be added per light source in each voxel. Its value indicates the level of opaque occlusion of the voxel with respect to each light source: *full*, *null* and *complicated occlusion*. Note that the remainder of the process description assumes only one light source, though the process is to be performed for each light source in the scene.

As a preprocessing step following object placement within the voxel data structure, all empty voxels are marked with *null occlusion* and non-empty voxels are marked with *complicated occlusion*. For each object, the shadow umbra [Crow77] is projected down to the relevant voxels. The voxels which reside solely within the shadow umbra are marked with *full occlusion*; the voxels that contain the shadow umbra edges are marked with *complicated occlusion*. Refer to figure 2, where the *dotted* voxels represent *complicated occlusion*, and the remainder of the voxels are marked with *full occlusion*.

sion.

Figure 2



When a ray intersects a surface in a voxel that has either *full* or *null occlusion* (a 2-bit comparison), then any intersected surface in the voxel is ensured to be in shadow or not in shadow, respectively. No shadow ray nor intersection checks need to be performed; this is referred to as *first pass voxel occlusion testing*. If *complicated occlusion* is the voxel occlusion value, then the fastest method is to resort back to voxel traversal (and intersection checks with candidate set of objects) of the shadow ray. However, as each voxel is traversed, its occlusion value is also checked (another 2-bit comparison). If the occlusion value is either *full* or *null occlusion*, then traversal can be halted and the intersected point is ensured to be in shadow of this light source or not, respectively; this is referred to as *intermediate voxel occlusion testing*.

At worst, no known occlusion values will be found during traversal. As a result, considering the negligible 2-bit comparisons, the worst scenario per shadow determination is the same as for voxel traversal. Refer to the below pseudo code for the different shadow determination procedures under voxel traversal and voxel occlusion testing.

/\* Voxel Traversal's Method for Shadow Determination \*/

```
VoxelTraversalShadow (point, light)
{
  voxel = CalculateCurrentVoxel (point);
  ray = GenerateShadowRay (point, light);

  while ((voxel = TraverseNextVoxel (ray)) != outside grid)
    if (IntersectObjectsInVoxel (ray, voxel) == hit)
      return (in shadow);
  return (not in shadow);
}
```

/\* Voxel Occlusion's Method for Shadow Determination \*/

```
VoxelOcclusionShadow (point, light)
{
  /* First Pass Voxel Occlusion Testing */
  voxel = CalculateCurrentVoxel (point);
  if (voxel occlusion value == full occlusion)
    return (in shadow);
  else if (voxel occlusion value == null occlusion)
    return (not in shadow);

  /* Intermediate Voxel Occlusion Testing */
  ray = GenerateShadowRay (point, light);
  while ((voxel = TraverseNextVoxel (ray)) != outside grid)
  {
    if (voxel occlusion value == full occlusion)
      return (in shadow);
    else if (voxel occlusion == null occlusion)
      return (not in shadow);

    if (IntersectObjectsInVoxel (ray, voxel) == hit)
      return (in shadow);
  }
  return (not in shadow);
}
```

## 6.1. Overhead for Preprocessing

The additional storage necessary is  $2N^3$  bits per light source, where  $N \times N \times N$  is the resolution of the grid. Note that the storage necessary is independent of the number of objects.

The preprocessing mainly involves projections of shadow umbrae onto voxels. The silhouettes of the objects are projected, then scan-converted to determine the voxel occlusion values. Thus the complexity is  $O(EnN^3)$ , where  $n$  is the number of objects,  $E$  is the average number of vertices per object and  $EN^3$  accounts for the projection and scan-conversion costs per object. This preprocessing is simple for polygons since the vertices can be projected, then joined to form the projected silhouette. However, the silhouette is more difficult to identify for quadric surfaces. Detailed projection mathematics for polygons and quadric surfaces are discussed in [Woo89a].

## 6.2. Non-Occluded Regions

If the exact definition of *null occlusion* is considered, this occlusion value can never be found in a voxel that contains a surface, even if the surface is completely unoccluded. *Complicated occlusion* will always be found in such voxels and thus first pass occlusion testing will fail. This can be partially solved for the case of only one object residing in the voxel. The voxel can be marked *null one* if the voxel is not already marked with other occlusion values. This indicates *null occlusion* for first pass occlusion testing, and *complicated occlusion* for intermediate occlusion testing. Thus, shadow rays do not need to be generated for voxels that contain only one convex object. If the object were concave, an intersection test against itself needs to be done before the occlusion status is determined.

With more populated voxels, it is inevitable that the occlusion value may be marked with *complicated occlusion* (to handle the case of occlusion from objects within the same voxel). For first pass occlusion testing, the ray bounding box approach [Snyd87] is used to reduce intersection checks within the initial voxel. Then hopefully, intermediate voxel occlusion testing can be used to determine the occlusion later on.

### 6.3. Non-Opaque Objects

Shadows as a result of transparent objects are much harder to deal with †. It needs the information of all occluding objects to generate coloured shadows. In addition, due to refraction, a single linear shadow ray is not sufficient to generate the correct results. The compromise commonly taken in ray tracing is to assume no refraction in the transparent occluding objects. This is also the assumption made in this shadow determination accelerator.

A new occlusion value *full transparent occlusion* can be added. However, upon reaching such a voxel, shadow determination cannot be stopped since information about all occluding objects are needed. Only transparent objects need to be checked for intersection. However, this has little advantage and requires the number of bits per voxel to increase to three. It is generally recommended that *complicated occlusion* be marked for the shadow umbrae of transparent objects.

## 7. Implications of the Accelerator

### 7.1. View Independence

In a rendering technique to calculate radiosity [Gora84], view independence can be achieved. Ray tracing is generally a view dependent rendering technique since its illumination calculations are dependent on the ray shot from the eye. But since shadows from opaque objects are only fractional value occlusions multiplied with intensity calculations, they can be view independent if desired. In this paper, voxel occlusion testing provides a view independent approach for shadow calculations in ray tracing. Thus the 2-bit voxel occlusion values need only be generated once and can be retained for future frames in any fly-by animation. In addition, changes in the power or spectral description of any light source require no change in the voxel occlusion values since the fractional occlusions are still the same.

### 7.2. Shadows on Bump-mapped Surfaces

As stated in [Woo89b], shadows on bump-mapped surfaces look perfectly smooth because the shadow ray is not perturbed to reflect the bumpiness. An analytic solution to the perturbation of the shadow ray was proposed in [Woo89b], but is expensive in general. However, the perturbation of the shadow ray only needs to be done near the silhouette of the shadow; the region in the middle usually ends up with the same shadow results. Thus, when the voxel occlusion value is

† Actually, these are not shadows, but filtering of light [Woo89b].

*full occlusion*, it is assumed that it is not near the silhouette region and no work is done to perturb the shadow ray. When the voxel occlusion value is *complicated occlusion*, then the shadow ray is perturbed. This should save some computations in perturbation of the shadow rays.

### 7.3. Voxel Occlusion may not Accelerate

Our testing suggests that voxel occlusion testing should not be used on low resolution images, e.g. 64x64. The preprocessing time may dominate the rendering process. However, the larger the resolution or the higher the sampling rate, the better the improvement of the runtime performance since the preprocessing is independent of resolution and becomes much less dominant compared to the actual rendering process.

Even at reasonable resolutions, this accelerator may not provide accelerated runtime performance over voxel traversal at times (on a single frame basis). Some additional work should go into a quick check whether as to use voxel occlusion testing during the preprocessing stage (on a single frame basis). This can be done since the algorithm is just a simple attachment onto the voxel traversal approach after the grid structure is produced.

A simple solution to this problem is to evaluate the scene after the grid initialization for voxel traversal is done. The equation to be evaluated is presented below, where  $n$  represents the total number of objects, the numerator represents the total region occupied by the objects, and when divided by the grid resolution  $N^2$ , represents a percentage of the total volume used:

$$\frac{1}{n N^2} \sum_{i=1}^n \text{number of voxels occupied by object } i$$

and voxel occlusion testing is not to be used if the above is below some *threshold value* (empirically derived), assuming that the small objects do not project a large shadow umbra. Note that this is only a quick check and may not be valid all the time.

## 8. Testing and Analysis

### 8.1. Some Numerical Results

Testing has been done on a SUN 3/280 with *fpa*, and the algorithm was implemented on a ray tracer at the University of Toronto: *optik*. The voxel traversal implementation in *optik* is described in [Aman87]. All test images are rendered at a resolution of 512x512 with one sample per pixel.

Table 1 illustrates the effectiveness of voxel occlusion testing. A couple of images were taken from the procedural database [Hain87]. Note that in the column *grid*, a grid subdivision level of  $N \times N \times N$  is assumed. In addition, *Voxel* and *Occ* indicate the total CPU time in seconds taken to render the image using voxel traversal and voxel occlusion, respectively. Finally, *%Shad* represents the percentage of shadow rays over all ray types.