Direct Illumination with Lazy Visibility Evaluation

David Hart Philip Dutré Donald P. Greenberg

Program of Computer Graphics[*]
Cornell University

Abstract

In this paper we present a technique for computing the direct lighting in a three-dimensional scene containing area light sources. Our method correctly handles partial visibility between luminaires and receivers, and is able to efficiently generate accurate soft shadows in scenes modeled with general bidirectional reflectance distribution functions. In most current algorithms, the form factor between a light source and receiver is computed using a stochastic ray casting approach which evaluates partial visibility. Such an approach often leads to noisy artifacts or aliasing problems. Generating significantly more rays is often the only solution to improving image quality. Our approach first stores visibility information in the image plane, using lazy evaluation of the visibility function. In a second phase, illumination values for each pixel are generated, using analytic or stochastic integration. Soft shadows and other shading effects are generated with high accuracy in less time than with existing shading algorithms. Coherence in specific blocker-light source relationships across the image plane is exploited to amortize the cost of analytic form factor calculations. By storing information in the image plane, our method is currently designed for generating a single image, and is thus view-dependent.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - color, shading, shadowing.

Additional Keywords: Rendering, Illumination Effects, Monte Carlo Techniques, Shadow Algorithms, Visibility Determination.

1 INTRODUCTION

In order to compute the direct illumination in a three-dimensional scene, it is necessary to be able to determine the visibility between any surface point and an area light source. Since changes in visibility between a surface point and an area light source are the cause of penumbra and umbra regions on the receiver surface, an efficient processing of the visibility function is often the key for rendering fast and accurate soft shadows. Once the visibility has been processed, the illumination can then be computed by integrating the incoming radiance function due to the light source, taking into account the bidirectional reflectance distribution function at the receiving surface, as expressed by the rendering equation [10].

In this paper we split the direct illumination computation into two distinct phases. A first phase evaluates the visibility function; the second phase computes the actual illumination by evaluating the rendering equation.

As contrasted with some existing techniques for computing soft shadows from direct illumination (e.g. [6]), we do not construct a complete discontinuity mesh in object space. Constructing such a mesh is computationally expensive, and many of the resulting mesh elements do not affect the final image quality. Moreover, implementing a robust algorithm that handles all possible cases often leads to numerical difficulties.

Our visibility pass instead detects, for each pixel in the image, any blocker-light source pairs which affect the illumination for the surface point visible through that pixel. We find such pairs by casting one or more shadow rays from the visible point towards the light source. Once such a pair is found, we exploit coherence by examining adjacent pixels to determine whether the same blocker-light source pair shows up for their respective visible points. The resulting image-space occluder- or blocker-map encodes the necessary visibility events that will be used in the next phase of the algorithm.

The second phase computes the illumination for each visible point. For each point, the stored blockers are clipped against the light source. The remaining light source area defines the integration domain for the illumination integral. Analytic or stochastic integration can then be employed to compute the actual radiance values to be attributed to each pixel.

The algorithm, therefore, passes over all pixels in the image twice and can be summarized as follows:

First pass. (Construction of the blocker-map):
- Compute the nearest point visible through each pixel using a classic ray-casting technique. By casting shadow rays to each light source, blockers are detected and relevant blocker-light source pairs are stored with the pixel.
- Project each blocker onto the light source, to determine whether the same blocker-light source pair appears in neighboring pixels. Neighboring pixels are found by using a flood-fill algorithm.

Second pass. (Evaluation of the illumination):
- For each pixel, the stored blockers are projected and clipped onto their paired light sources using the visible surface point through the pixel as the center of projection.
- Given the clipped area light sources, the illumination is computed to acquire a radiance value for each pixel. Integration can be done analytically, stochastically, or by any other suitable method.

The main advantage of our approach is that we do not need an elaborate examination of all possible visibility events. Blocker-

[*]Program of Computer Graphics, 580 Rhodes Hall, Cornell University, Ithaca NY 14850, USA. URL: http://www.graphics.cornell.edu/

light source pairs are included in the blocker-map only if detected by a shadow-ray. If detected, coherence is exploited by flood-filling the blocker-light source pair to neighboring pixels. This procedure replaces the construction of a discontinuity mesh or a visibility skeleton. We store no visibility information that will not be needed during the illumination computations.

2 PREVIOUS WORK

The generation of shadows from area light sources is a long standing problem in computer graphics. Several algorithms for generating sharp shadows due to point light sources have been published. A good survey of such shadow algorithms can be found in Woo et al. [24].

Stochastic ray tracing algorithms [5,15] compute the direct illumination shadow at any point by sampling the area of the light source with shadow rays. Various optimization techniques based on importance sampling [16] can be used, but the fact remains that the shadow rays still have to evaluate both illumination and visibility.

Radiosity algorithms [4,17] intrinsically compute soft shadows as part of the full global illumination solution. Discontinuity meshing [11] provides an accurate way of computing soft shadows, but requires substantial amounts of computation time. Moreover, discontinuity meshing is usually driven by an exhaustive algorithm, as each possible discontinuity is considered as a potential candidate for subdividing the mesh. A constructed discontinuity mesh can also be used in pixel-based ray tracing algorithms. Drettakis and Fiume [6], constructed the complete discontinuity mesh, after which the exact illumination for a surface point visible through a pixel is computed analytically. Similar ideas for computing shadow boundaries are presented by Stewart and Ghali [19].

The visibility skeleton [7] encodes all possible visibility events that cause discontinuities in visibility or shading at considerable computational cost. It can be used to compute accurate illumination due to area light sources for any surface point in the scene.

A space subdivision based on ray directions emanating from the light sources was proposed by Tanaka and Takahashi [21], allowing a fast detection of possible blockers for a surface point to be shaded. Casting rays from surface points to detect blockers is also proposed in other algorithms for computing soft shadows [20,23].

Soler and Sillion [18] presented a method for interactive soft shadows based on convolution operators and using scan-conversion hardware. Although their method is restricted in computing exact soft shadows in a limited number of cases, the results are quite convincing and can be applied in real-time.

3 DIRECT ILLUMINATION

3.1 Rendering Equation

In order to compute the direct illumination in a scene, we have to integrate the incoming radiance at a surface point according to the rendering equation [10]. The exitant radiance $L(x \to \Theta)$ leaving a surface point x in a direction Θ, due to direct illumination from the light sources, is given by:

$$L(x \to \Theta) = \int_A L_e(y \to x) f_r(\Theta \leftrightarrow \overline{xy}) \frac{\cos\Theta \cos\overline{yx}}{r_{xy}^2} V(x,y) d\mu_y \quad (1)$$

where $L_e(y \to x)$ is the emitted radiance of the light source from a surface point y to x, A is the area of all light sources, $d\mu_y$ is a differential surface area around y, $f_r(\Theta \leftrightarrow \overline{xy})$ is the bidirectional reflectance distribution function (BRDF) at x, $\cos\Theta$ is the cosine of the angle between Θ and the surface normal at x, $\overline{xy}$ is the direction vector connecting x and y, r_{xy} is the distance between x and y, and $V(x, y)$ is the visibility function, having a value of 1 if x and y are mutually visible, 0 otherwise.

By summing over light sources explicitly, and by folding the visibility function into the integration domain, equation (1) can also be written as:

$$L(x \to \Theta) = \sum_{i=1}^{N_L} \int_{A_i} L_{ei}(y \to x) f_r(\Theta \leftrightarrow \overline{xy}) \frac{\cos\Theta \cos\overline{yx}}{r_{xy}^2} d\mu_y \quad (2)$$

where N_L is the number of different light sources, and A_i is the visible part of the light source as seen from point x. The use of this equation implies that we can determine A_i, before carrying out the actual integration.

3.2 Analytic Integration

We now describe a set of conditions for which equation (2) can be worked out analytically:

- The luminaires are a (disjoint) set of polygons;
- The exitant radiance is a constant for a given light source ($L_e(x \to \Theta) = L_{ei}$);
- The receiving surface is diffuse.

Using Stoke's theorem, the continuous integral over the area A_i can be converted into an integral over the boundaries of the polygonal light source [12,3]:

$$L(x \to \Theta) = \frac{f_r}{2} \sum_{i=1}^{N_L} L_{ei} \sum_{j=1}^{E_i} N \cdot \Gamma_j \quad (3)$$

where E_i is the number of boundaries for light source i and N is the normal vector at x. Γ_j is a vector with magnitude equal to the angle gamma, as illustrated in figure 1, whose direction is given by the cross-product of vectors R_j and R_{j+1}. A more detailed derivation of this formula can be found in [9].

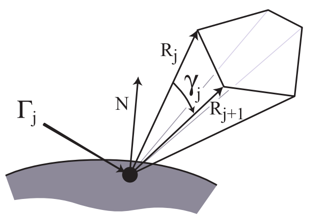


Figure 1: Geometry for analytically evaluating illumination.

Equation (3) is only valid for the visible part of the light source, A_i. In the event where occlusion occurs between x and the light source, and if we have a way to remove the occluded parts from the light source so that only non-occluded parts remain, we can successfully use the above closed form to compute the radiance exactly.

If the receiving surface is not diffuse, an analytic computation of the direct illumination is more difficult. Arvo [1] gave a derivation that makes it possible to integrate equation (2) if the BRDF is composed of a linear combination of Phong-lobes [13]. This method also transforms the area integral into a line integral. Again, this method assumes the integration only takes place over the unoccluded parts of the light source.

3.3 Monte Carlo Integration

Monte Carlo integration techniques can also be used to compute the direct illumination [16], regardless of the type of BRDF. A number of sample points are generated over the area of the light sources, and the integrand has to be evaluated for each point. By taking the weighted average of these evaluations, an unbiased estimator for the direct illumination is obtained. Since the visibility function is part of the integrand, a fraction of the generated samples will evaluate to zero causing significant noise in the image. A reduction of the integration domain to the visible parts of the light sources would decrease noise significantly, using the same number of sampling points.

Moreover, the integration domain can be transformed from the area of the light sources to the solid angle subtended by the light sources on the hemisphere around x. This reduces noise even further since the inverse distance and one cosine factor are folded into the integration variable.

The overall effect of using domain reduction and solid angle sampling is that we can achieve the same image quality with a reduced number of shadow rays, therefore speeding up the rendering algorithm.

4 ALGORITHM

4.1 Construction of the blocker-map.

The purpose of the first pass, which constructs the blocker-map, is to identify relevant blocker-light source pairs for each pixel. We use a combination of shadow rays and a flood-fill algorithm in the image plane to identify the necessary pairs for each pixel. A ray is cast through the center of each pixel in order to find the nearest visible point on a surface. The location of this visible point, along with its surface normal, is stored with the pixel for future reference.

Once the visible point is found, a number of shadow rays starting from this point are generated for each light source by using a uniform sampling function over the solid angle subtended by the light source as seen from the visible point [2]. If one of these rays hits an intervening object, this blocker-light source pair is stored in the blocker-map for that pixel. A blocker will be found with a probability proportional to its subtended solid angle covering the subtended solid angle of the light source. Figures 2a and 2b illustrate the identification and storage of the blockers in the blocker-map.

Since for each shadow-ray, we find at most one blocker, we probably do not find all relevant blockers for a pixel. We assume that the umbra-penumbra region of a single blocker extends over multiple pixels in the image plane. Therefore, once a blocker-light source pair is found, neighboring pixels are examined to check whether the same blocker shows up, using an eight-connect recursive flood-fill mechanism. For each pixel visited during the flood-fill, the blocker is projected onto the light source, using the visible point in that pixel as the center of projection. If a simple test indicates that the two polygons (blocker and light source) overlap, the pair will be added to the blocker-map for the visited pixel. The flood-fill stops if there is no overlap for the current pixel, or if the blocker-light source pair already has been stored for that pixel. Figures 2c, 2d and 2e illustrate this part of the algorithm. Pseudo-code for the construction of the blocker-map is given in Figure 3.

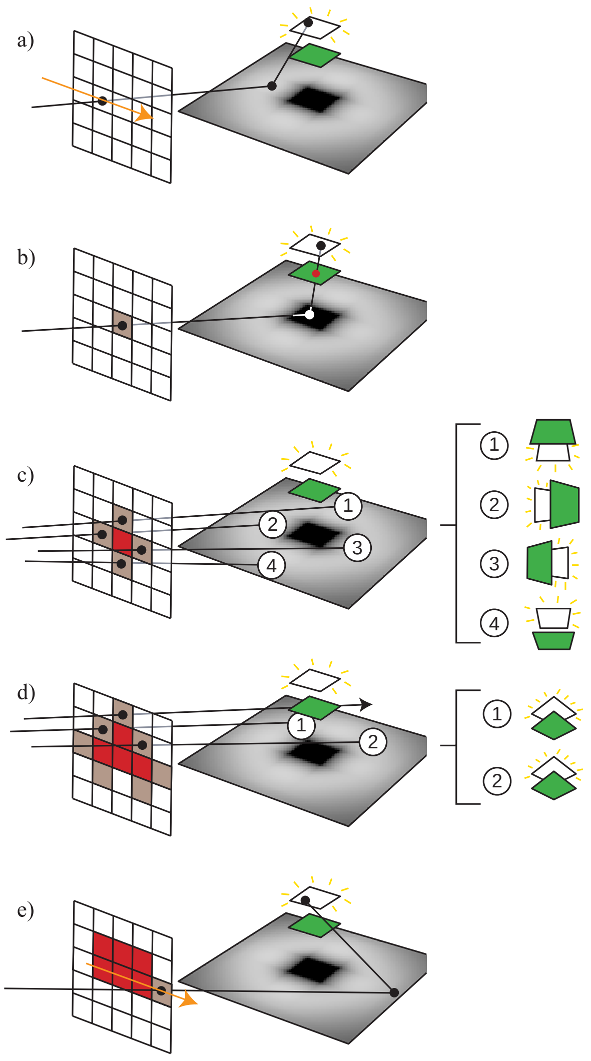


Figure 2: a) A shadow-ray is sent to the area light source, but is not blocked by any blocker. The orange arrow indicates the order in which the pixels are visited. **b)** The shadow-ray generated for this pixel is blocked by the blocker. The blocker-light pair is added to the blocker-map. **c)** A 4-connect flood-fill tests whether the same pair shows up in adjacent pixels. The blocker is projected on the light source plane, and their relative position indicates whether the blocker is added to the blocker-map for the tested pixel (pixels 1, 2 and 3) or the flood-fill is stopped (pixel 4). **d)** The flood-fill continues recursively and stops if the blocker-light source pair was already stored in the blocker-map for that pixel, or if no overlap between projected blocker and light source is detected. **e)** Once the flood-fill has stopped, regular testing is resumed with the next eligible pixel.

Using the flood-fill algorithm makes it sufficient to detect the blocker in any of the pixels covered by the umbra-penumbra region, in order to take it into account for all those pixels during the illumination pass. Note that the blocker-map accumulates all possible blocker-light source pairs, and does not just store the last one encountered.

The result of this first pass is that we have constructed a blocker-map in the image plane. For each pixel, we have stored blocker-

```
floodFillBlocker(int x, y; light l; blocker b)

// if blocker already is stored in pixel, return
if (b,l) is in bMap(x,y) return;

// clip blocker behind light and visible point
cBlocker = clipBlocker(b, l, bMap(x,y).point);

// project the blocker on the light source
pBlocker =
   projectBlocker(cBlocker, l, bMap(x,y).point);

// check if the blocker overlaps the light source

if triangleOverlap(pBlocker, l) then

   // store the pair in the blocker-map
   bMap(x,y).store(b,l);

   // flood-fill the surrounding pixels
   if (x > 1)      floodFillBlocker(x-1,y,l,b);
   if (x < width)  floodFillBlocker(x+1,y,l,b);
   if (y > 1)      floodFillBlocker(x,y-1,l,b);
   if (y < height) floodFillBlocker(x,y+1,l,b);
endif

return;
```

Figure 3: Pseudo-code for the flood-fill algorithm in the image plane.

light source pairs which affect the illumination of the point visible through that pixel. Figure 4 shows what a blocker-map looks like for a simple example. The color values indicate how many blocker-light source pairs are stored in each pixel, with white indicating 15 pairs and black indicating 0. Red regions indicate detected umbra regions (see Discussion). No visibility information is stored in the black pixels, therefore, no clipping or visibility tests are needed during the illumination phase for those areas. The blocker-map can, therefore, also be considered as an indicator of which pixels demand the most amount of work. This indicator could be worthwhile information for parallelizing the algorithm and achieving a good load-balancing amongst different processors.

4.2 Illumination Pass

After construction of the blocker-map, a second pass computes the illumination in each pixel. Equation (2) has to be evaluated for the visible point seen through the pixel. As described above, both analytic and Monte Carlo integration benefit from a more elaborate handling of the visibility function, by eliminating the non-visible parts of the light sources from the integration domain. For each pixel, each blocker in the blocker-map is projected onto its paired light source using the visible point as the center of projection. Each light source is clipped accordingly, such that only the visible parts remain.

Integrating the direct illumination can then be done using the reduced integration domain. In general, when using arbitrary BRDFs or non-diffuse luminaires, Monte Carlo integration is the preferred method. A suitable probability density function that samples all visible parts of the light sources produces an unbiased estimator. The variance on this estimator can always be reduced by increasing the number of samples.

Analytic integration is limited to a few cases where the BRDF is diffuse or is described as a linear combination of Phong-lobes. Equation (3), or the methods described in [1], are applied to all visible parts of the light sources. If no blockers are missed in the blocker-map, analytic integration gives the correct answer for the direct illumination.

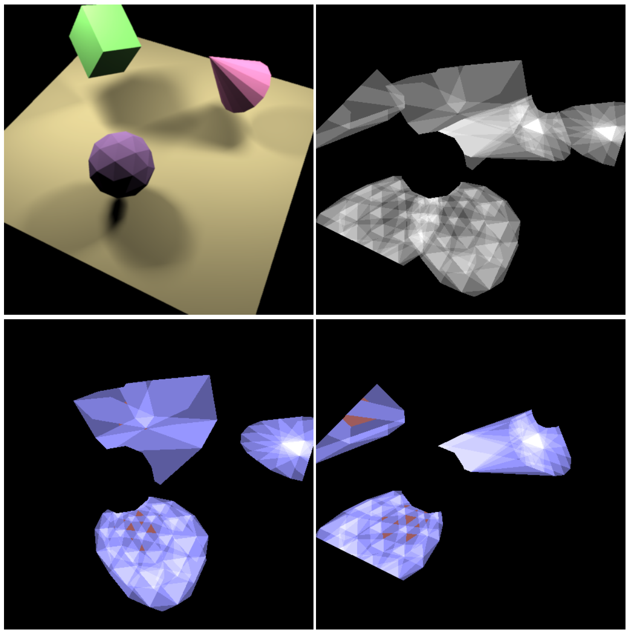


Figure 4: The top-left image shows a simple scene consisting of a ground-plane and several polyhedra. Two light-sources are present, casting soft shadows. The bottom images show the blocker-maps for the two light sources separately, the top-right image shows the combined blocker-map.

4.3 Polygon Clipping

The efficiency and generality of the polygon clipping algorithm determines the overall efficiency of the integration pass. There is a large number of approaches to polygon clipping found in the literature [22]. Many well-known polygon clippers have constraints which make general clipping (e.g. with concave, self-intersecting, or overlapping polygons) difficult or numerically unstable. Our current algorithm constrains the scene to be composed of triangles only, which is possible without loss of generality. The advantage is that we can uniformly sample the projection of a triangle on a hemisphere [2] to compute the illumination integral with Monte Carlo integration.

4.4 Anti-Aliasing

Since the list of blockers constructed for each pixel is only valid for a single surface point visible at the center of that pixel, aliasing artifacts will occur if we compute the illumination only at this single point (e.g. jagged edges when more than one object is visible in a pixel). If more than one ray per pixel is generated for illumination computations as part of an anti-aliasing algorithm, the blocker-light source list stored for this pixel might not be valid for each surface point hit by these rays, since the surface points might be located in very different positions in object space. Nevertheless, we need a strategy that allows us to perform anti-aliasing.

The coherency of the penumbra regions over the image plane can again be exploited. Due to the flood-fill, we know that a blocker is at least valid for the center location of all covered pixels. Pixels located in the middle of the flood-fill area are completely covered with the penumbra region. Therefore, we can assume that the blocker is valid for all surface points seen through such pixels. For pixels located at the edge of the flood-fill area, the penumbra might extend up to the center of the first pixel outside the flood-filled region. Therefore, if we allow the flood-fill algorithm to include the boundary pixels for which the flood-fill test fails, we can safely assume that we have stored all possible blockers in a pixel that are needed for computing the illumination for any point visible through the pixel. This is illustrated in figure 5.

This strategy allows us to generate multiple sample rays for illumination computations, without increasing the number of rays used for constructing the blocker-map. Therefore, a suitable anti-aliasing algorithm can be employed. In the current implementation, adaptive anti-aliasing is carried out in adjacent pixels whose visible points show different polygons with different surface normals or whose illumination computations based on the first point show sufficiently different spectral radiance values. This strategy will capture most aliasing artefacts. Note however, that high-frequency geometry, such as small objects, might be overlooked.

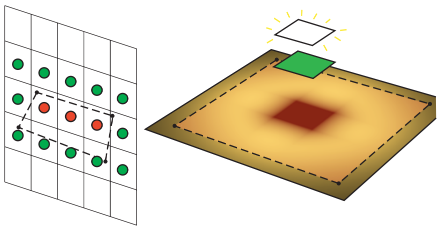


Figure 5: The flood-fill algorithm stores the blocker in the red pixels where the penumbra is detected, and also in the green pixels, where the penumbra might show up.

5 DISCUSSION

Missing blockers. During the construction of the blocker-map, some blockers might be overlooked. Because of the flood-fill algorithm, a blocker is only missed if it is not intersected by any of the shadow rays starting from visible points covered by the penumbra or umbra caused by that blocker. If a blocker is missing, some pixels will be too bright because invisible parts of the light sources are taken into account. By increasing the number of shadow rays, the probability of missing blockers decreases. This implies that the analytic integration method is consistent: the possible error can be made arbitrarily small by generating more shadow rays.

The probability of missing blockers or computing insufficient shadows is further diminished by the fact that we locate the nearest polygon intersected by each shadow ray. We are therefore assured that the polygon causing the most significant shadow on a receiver point is stored in the blocker-map. This is no less efficient than a more traditional approach where any intersecting polygon found is sufficient to conclude the receiver point lies in a shadow, since we use a regular voxel-based spatial subdivision technique (with an average of one polygon per voxel), and all polygons in the traversed voxel are tested for intersection [8].

Receiver Surfaces. Because we do not use any form of texture-map to display soft shadows, our method produces soft shadows on any surface type that can be handled by a ray tracing algorithm. The number of receiver surfaces or polygons is not a limiting factor for generating the correct direct illumination and soft shadows.

Small Blockers. Small objects or small polygons causing shadows create small blockers in the blocker-map. Such blockers probably only clip a very small piece of the light source and, therefore, do not affect the shading by a large amount. One might be tempted to dismiss small blockers without computing their influence on the illumination. However, a whole set of small blockers might significantly affect the visibility of a light source, thus they cannot be ignored. Each of these small blockers requires a full clipping operation. In the limit, this is a worst-case scenario for our current algorithm. A possible solution might be to use some form of clustering as in radiosity algorithms, although with loss of exact visibility; or construct the silhouette polygon of a collection of smaller polygons, and use that as the blocker.

Umbra Regions. One straightforward optimization is to make a distinction between penumbra and umbra regions. When testing whether a blocker overlaps with a light source as seen from a visible point, a triangle-triangle ‘‘surround’’ test indicates if the point lies in the umbra or penumbra region. By storing this information in the blocker-map, we know that this particular light source does not have to be included in the illumination computations, and expensive clipping operations are avoided. Also, it prevents other blockers associated with the same light source from being stored in those pixels.

6 RESULTS

We have implemented the algorithm as outlined above, including umbra detection and anti-aliasing using the flood-fill extension. Timings were performed on one Intel Pentium II 400Mhz processor. All images were computed at a resolution of 512x512 pixels.

Figure 6 shows three different views of the interior of the Church of the Year 2000 in Rome, consisting of 64,216 triangles and one large light source, split in two triangles. All surfaces are diffuse. The top row shows an overview of the chapel, looking from the back towards the altar. The light source is located in the upper right, and casts shadows from the benches on the floor. The middle row shows a close-up of the cross at the altar. The third row shows a view from a small anteroom into the main chapel through a number of slats. The light is shining through the slats and causes shadows to fall on the floor of the anteroom. For each viewpoint, three pictures are shown. The middle column shows pictures generated using the blocker-map, and analytic evaluation of the direct illumination. The first and third columns are generated using standard Monte Carlo rendering of the direct lighting term. Visibility is evaluated by casting shadow rays from each visible point to the light sources, and the shadow rays are generated by sampling the solid angles subtended by the light source. The number of shadow rays, as well as the execution times, are mentioned below each picture. The first column executes in roughly equal time compared to the second column; the third column produces roughly equal image quality as the second column. From these comparisons, it is obvious that for an equal execution time, our algorithm generates significantly better quality pictures. The Monte Carlo solutions show significant noise in the soft shadow regions. The special geometrical case of the anteroom can be handled by our algorithm efficiently; all slats are clipped from the light source and, therefore, we have an almost exact visibility computation although the execution time is quite high. The equal quality comparisons indicate that our algorithm, while providing very accurate soft shadows, executes up to 20 times faster compared to the standard Monte Carlo method.

Figure 7 shows a different scene, with a somewhat more complex set-up. The scene consists of 556 polygons and four light sources. The four images on the right are a close-up of the area marked in red, all generated by different algorithms. Picture (a) is generated using the blocker-map and analytic integration; picture (b) used Monte Carlo integration with 4 sample rays and shadow rays per pixel, using the blocker-map to restrict the integration domain. Picture (c) and (d) are generated using standard Monte Carlo integration with 30 and 100 sample rays per pixel respectively. In each of the pictures, one shadow ray was generated per sample ray for each

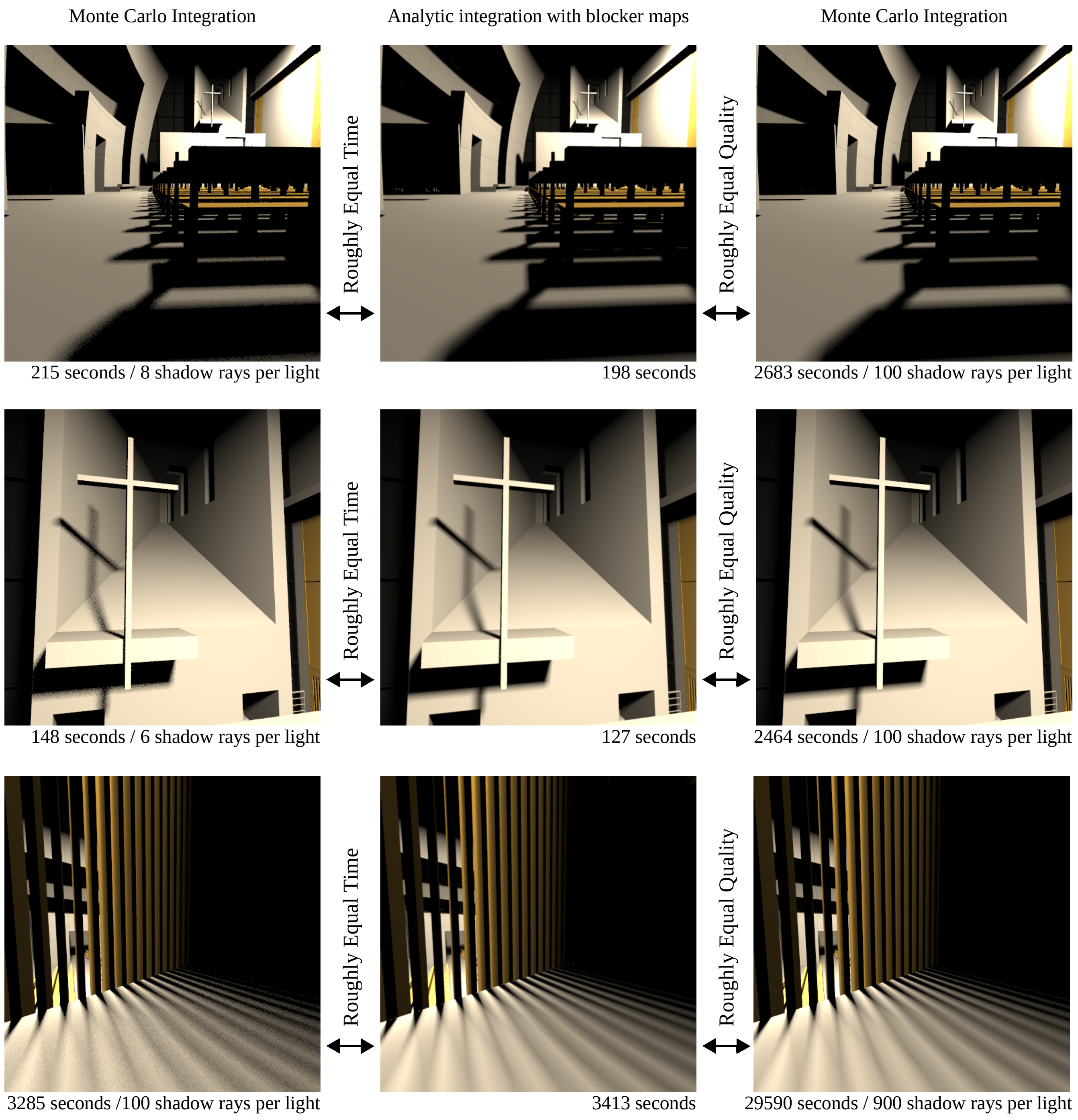
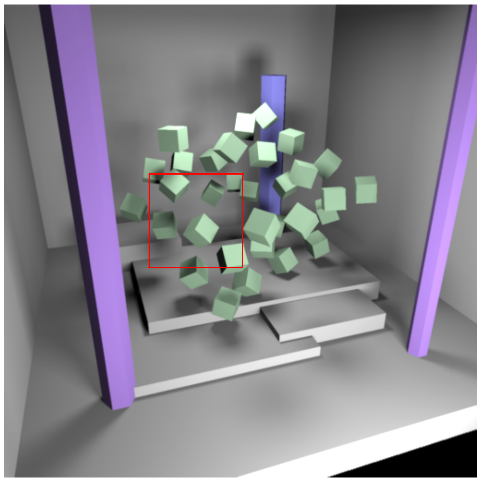
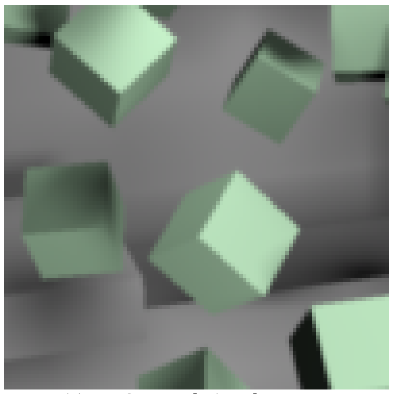

Monte Carlo Integration
Analytic integration with blocker maps
Monte Carlo Integration
Roughly Equal Time
Roughly Equal Quality
215 seconds / 8 shadow rays per light
198 seconds
2683 seconds / 100 shadow rays per light
Roughly Equal Time
Roughly Equal Quality
148 seconds / 6 shadow rays per light
127 seconds
2464 seconds / 100 shadow rays per light
Roughly Equal Time
Roughly Equal Quality
3285 seconds /100 shadow rays per light
3413 seconds
29590 seconds / 900 shadow rays per light

Figure 6: Three different viewpoints for the Church of the Year 2000 in Rome. The middle column is rendered using the blocker map and analytic integration; the left column is generated using standard Monte Carlo integration and shows pictures generated in roughly equal time; the right column shows pictures generated with roughly equal quality.

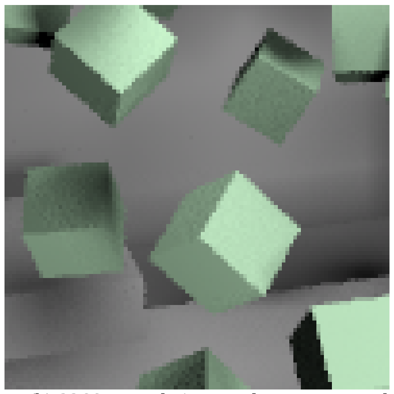
light source. Again it is clear that our algorithm, using analytic integration, provides the best quality in the least time. Picture (c) give an equal time comparison with (a), while (d) provides an equal quality comparison. Picture (b) indicates that Monte Carlo integration benefits from using the blocker-map to reduce the integration domain: the noise on the back wall is almost completely gone, and the only noise that is left on the green cubes is due to one remaining cosine factor in the integral. Note that the execution times for this scene are higher than for the church scene, although the number of polygons in this scene is far less than in the church scene. This is mainly due to the more complex soft shadows and, therefore, the higher number of blockers stored in the blocker-map. Execution times are comparable to the anteroom pictures of the church. Nevertheless, by comparing images (a) and (d), our algorithm provides the same image quality in about one-tenth the time.

Figure 8 shows that glossy surfaces can also be successfully handled by our algorithm. Both pictures are computed using the blocker-map and analytic evaluation of the illumination. The float-

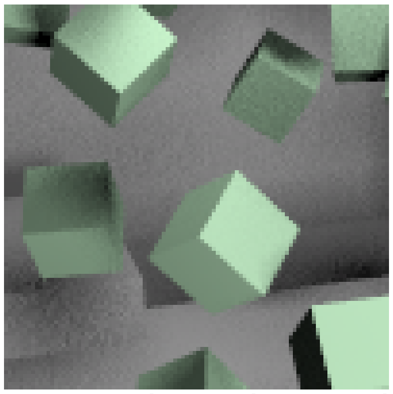




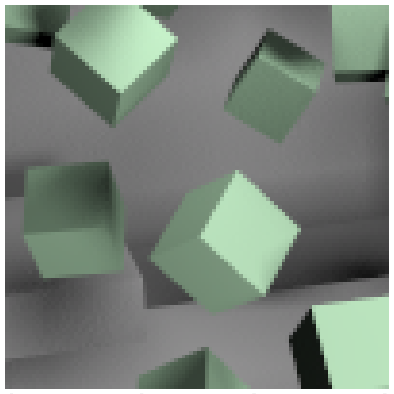
(a) 1256 seconds / analytic integration



(b) 3366 seconds / 4 sample rays per pixel



(c) 1238 seconds / 30 sample rays per pixel



(d) 16526 seconds / 400 sample rays per pixel

Figure 7: The pictures on the right are close-ups from the red area on the left, and are rendered with different algorithms: (a) Analytic integration using the blocker map; (b) Monte Carlo integration using the blocker-map to reduce the integration domain; (c,d) Standard Monte Carlo integration. Pictures (a) and (c) are an equal time comparison, pictures (a) and (d) an equal quality comparison. Picture (b) shows the effect of domain reduction on the Monte Carlo integration.

ing cubes scene on the left has a glossy back wall. One can clearly see the soft shadows cast in the glossy highlight of the light source. In the picture on the right, the SIGGRAPH 99 title floats above a spherical surface. One light source is positioned in front, another placed in the back. Two different shadows are visible: the diffuse shadow caused by the diffuse component of the BRDF, and the specular shadows caused by the glossy component. In the second case, the viewer is inside the Phong-lobe of light reflected from the light source in the back. Although no indirect illumination component is computed, the geometry of where the reflection should occur is handled correctly and shows up as a specular shadow.

7 FUTURE WORK

Even though the algorithm seems to perform well in our test cases, there are areas where improvement can be achieved:

Silhouette Polygons and Clustering. In our current implementation, polygons are considered as blockers without looking at the higher-level object of which they are part. By substituting a collection of polygons by its silhouette polygon, only the silhouette needs to be clipped. It is likely that the complexity will be lowered in this case, although the silhouette needs to be recomputed for every new visible point.

Non-polygonal surfaces. Computing analytic visibility for non-polygonal surfaces (e.g. spline surfaces) could be a very interesting direction for future research. It is conceivable that when a point needs to be shaded, the silhouette of an object could itself be computed as a spline curve. If the clipping pass could handle two-dimensional spline-polygon and spline-spline boolean operations, soft shadows could be computed for smooth surfaces.

Reprojection of blocker maps. Because a blocker map contains per pixel information, the blocker-map can be reprojected to a new viewpoint, using an image-warping function which is common in image-based rendering algorithms. This would open possibilities for faster image generation in animation or interactive sequences.

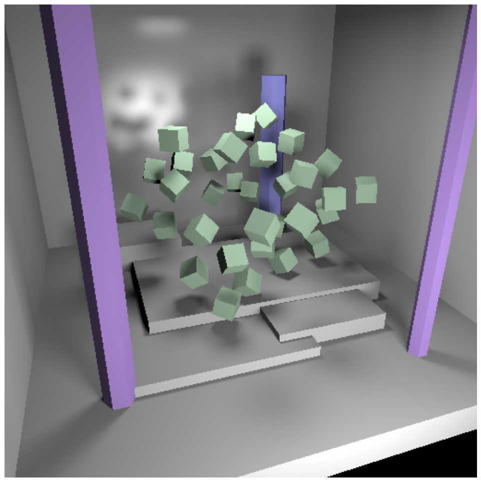
Load balancing & Perception. The blocker-map gives an indication of how much work is needed to render individual pixels. This could be used to achieve better load balancing in parallel rendering algorithms or distribute work according to some perceptual error metric [14].

8 CONCLUSION

We have presented an efficient and fast algorithm to compute accurate direct illumination in a general polygonal environment. Two passes over the image plane are used. A first pass identifies blocker-light source pairs on a per pixel basis. A limited number of shadow rays combined with a flood-fill algorithm in the image plane gives a highly reliable method for detecting the correct blockers. A second pass computes the actual illumination for each pixel by performing analytic or stochastic integration.

Timing results indicate that the algorithm runs faster than more classic Monte Carlo illumination techniques. Although there is a probability that some shadow effects may be missed, there is in general no visible differences between the rendered images with our method and highly accurate reference solutions.

Our results indicate that clipping the light sources with blockers is a viable and acceptable tactic to be used for direct illumination computations or, more generally, for global illumination algorithms.



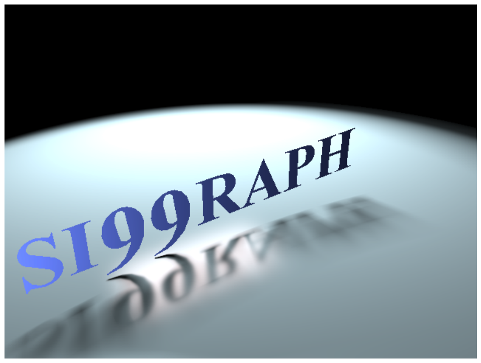


Figure 8: In the top picture, soft shadows are visible in the glossy highlight on the back wall. In the bottom picture, specular shadows are visible on the ground plane.

Acknowledgements

This work was supported by the NSF Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-8920219) and by NSF grant ASC-9523483, and performed on workstations generously donated by the Intel Corporation.

Richard Meier & Partners, Architects, generously provided us with the model of the “Church of the Year 2000”.

References

[1] James Arvo. Applications of irradiance tensors to the simulation of non-lambertian phenomena. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 335-342. ACM SIGGRAPH, Addison Wesley, August 1995.

[2] James Arvo. Stratified sampling of spherical triangles. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 437-438. ACM SIGGRAPH, Addison Wesley, August 1995.

[3] Daniel R. Baum, Holly E. Rushmeier, and James M. Winget. Improving radiosity solutions through the use of analytically determined form-factors. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH 89 Conference Proceedings)*, volume 23, pages 325-334. Addison Wesley, July 1989.

[4] Michael F. Cohen and John R. Wallace. *Radiosity and Realisitc Image Synthesis*. Academic Press Professional, San Diego, CA, 1993.

[5] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed Ray tracing. In *Computer Graphics (SIGGRAPH 84 Conference Proceedings)*, volume 18, pages 137-145. Addison Wesley, July 1984.

[6] George Drettakis and Eugene Fiume. A fast shadow algorithm for area light sources using back-projection. In Andrew Glassner, editor, *SIGGRAPH 94 Conference Proceedings*, Annual Conference Series, pages 223-230. ACM SIGGRAPH, Addison Wesley, July 1994.

[7] Frédo Durand, George Drettakis, and Claude Puech. The visibility skeleton: A powerful and efficient multi-purpose global visibility tool. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 89-100. ACM SIGGRAPH, Addison Wesley, August 1997.

[8] Tanaka T., Fujimoto A. and Iwata K. Arts: Accelerated ray tracing system. *IEEE Computer Graphics and Applications*, 6(6):16-26, April 1986.

[9] Adel F. Sarofim Hotte, Hoyt C. *Radiative Transfer.* McGraw Hill, New York, NY, 1967.

[10] James T. Kajiya. The rendering equation. In David Evans and Russel Athay, editors, *Computer Graphics (SIGGRAPH 86 Conference Proceedings)*, volume 20, pages 143-150. Addison Wesley, August 1986.

[11] Daniel Lischinski, Filippo Tampieri, and Donald P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications*, 12(6):25-39, November 1992.

[12] Tomoyuki Nishita and Eihachiro Nakamae. Continous tone representation of three-dimensional objects taking account of shadows and interreflection. In Brian Barsky, editor, *Computer Graphics (SIGGRAPH 85 Conference Proceedings)*, volume 19, pages 23-30. Addison Wesley, July 1985.

[13] Bui-T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311-317, June 1975.

[14] Mahesh Ramasubramanian, Sumanta N. Pattanaik, and Donald P. Greenberg. A Perceptually Based Physical Error Metric for Realistic Image Synthesis. In Alyn Rockwood, editor, *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series. ACM SIGGRAPH, Addison Wesley, August 1999.

[15] Peter Shirley and Chang Yaw Wang. Distribution ray tracing: Theory and practice. *Proceedings of the Third Eurographics Workshop on Rendering*, pages 33-43, May 1992.

[16] Peter Shirley, Chang Yaw Wang, and Kurt Zimmerman. Monte Carlo methods for direct lighting calculation. *ACM Transactions on Graphics*, January 1996.

[17] François Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufmann, San Francisco, 1994.

[18] Cyril Soler and François X. Sillion. Fast calculation of soft shadow textures using convolution. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference series, pages 321-332. ACM SIGGRAPH, Addison Wesley, July 1998.

[19] James Stewart, Sherif Ghali. Fast Computation of Shadow Boundaries Using Spatial Coherence and Backprojections. In Andrew Glassner, editor, *SIGGRAPH 94 Conference Proceedings*, Annual Conference Series, pages 231-238. ACM SIGGRAPH, Addison Wesley, July 1994.

[20] Wolfgang Stürzlinger. Adaptive Mesh Refinement with Discontinuities for the Radiosity Method. *Photorealistic Rendering Techniques (Proceedings of the 5th Eurographics Workshop on Rendering)*, G. Sakas, P. Shirley, S. Müller (eds.), Springer-Verlag 1995.

[21] Toshimitsu Tanaka, Tokiichiro Takahashi. Fast Analytic Shading and Shadowing for Area Light Sources. *Computer Graphics Forum (Proceedings of Eurographics 1997)*, volume 16, 3.

[22] B.R. Vatti. A generic solution to polygon clipping. *Communications of the ACM*, 35(7):56-63, July 1992.

[23] Christophe Vedel. Computing Illumination from Area Light Sources by Approximate Contour Integration. *Proceedings of Graphics Interface ‘93*, pages 237-244.

[24] Andrew Woo, Pierre Poulin, and Alain Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13-32, November 1990.