Efficient Shadows for Sampled Environment Maps

Aner Ben-Artzi
Columbia University
aner@cs.columbia.edu

Ravi Ramamoorthi
Columbia University
ravir@cs.columbia.edu

Maneesh Agrawala
Microsoft Research
maneesh@graphics.stanford.edu

Abstract

This paper addresses the problem of efficiently calculating shadows from environment maps in the context of ray-tracing. Since accurate rendering of shadows from environment maps requires hundreds of lights, the expensive computation is determining visibility from each pixel to each light direction. We show that coherence in both spatial and angular domains can be used to reduce the number of shadow-rays that need to be traced. Specifically, we use a coarse-to-fine evaluation of the image, predicting visibility by reusing visibility calculations from 4 nearby pixels that have already been evaluated. This simple method allows us to explicitly mark regions of uncertainty in the prediction. By only tracing rays in these and neighboring directions, we are able to reduce the number of shadow-rays traced by up to a factor of 20 while maintaining error rates below 0.01%. For many scenes, our algorithm can add shadowing from hundreds of lights at only twice the cost of rendering without shadows. Sample source code is available online.

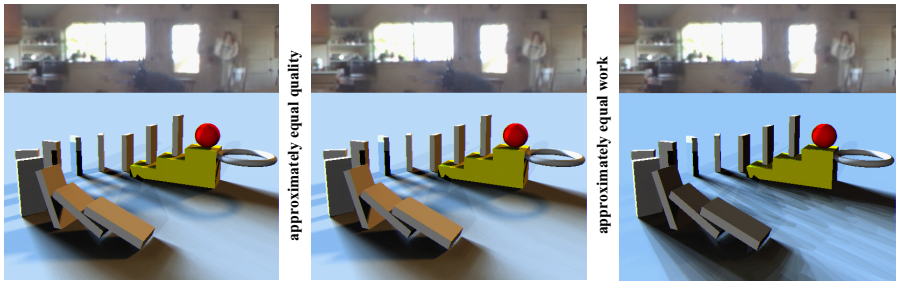

Figure 1: *A scene illuminated by a sampled environment map. The left image is rendered in POV-Ray using shadow-ray tracing to determine light-source visibility for the 400 lights in the scene, as sampled from the environment according to [Agarwal et al. 2003]. The center image uses our Coherence-Based Sampling to render the same scene with a 90% reduction in shadow-rays traced. The right image is again traced in POV Ray, but with a reduced sampling of the environment map (50 lights, again using [Agarwal et al. 2003]) to approximate the number of shadow-rays traced using our method. Note that the lower sampling of the environment map in the right image does not faithfully reproduce the soft shadows.*

1 Introduction

Complex illumination adds realism to computer-generated scenes. Real-world objects hardly ever exist in environments that are well approximated by a small number of lights. We therefore focus on realistic shading and shadowing from environment maps, as first described by [Blinn and Newell 1976], and later by [Miller and Hoffman 1984], [Greene 1986], and [Debevec 1998]. The subtle shadowing effects created by complex illumination are one of the most important visual cues for realistic renderings. Correct shadowing requires solving for the visibility of each light direction with respect to each pixel. This is extremely expensive, since the illumination can come from any direction in the environment map. We leverage recent sampling methods like [Agarwal et al. 2003], [Kollig and Keller 2003], and [Ostromoukhov et al. 2004], that reduce the environment to a few hundred non-uniform directional lights. However, shadow testing for all sources at each pixel is still the bottleneck in a conventional raytracer or other renderer.

In this paper, we show how to calculate the visibility function efficiently by exploiting coherence in both the angular and spatial dimensions. We recognize that the expensive operation when lighting with complex illumination is the tracing of secondary shadow-rays. Thus, in our method, we trace all primary eye-rays and cull only shadow-rays, whenever possible. Our method is closest to that of [Agrawala et al. 2000], who exploited coherence in the visibility of an area light source to reduce the number of shadow-rays cast. We adapt their method to environment maps, and show that modifications to both the spatial sharing and angular sharing of visibility information are required for accurate and efficient visibility sampling in the context of environment lighting. The salient differences from area lighting are the lack of a well-defined boundary, lack of locality, and lack of efficient uniform sampling for environment lighting. By addressing all of these points, we

demonstrate, for the first time, how coherence-based visibility sampling can be used for full-sphere illumination.

We identify two important components of an algorithm estimating visibility by reusing previously calculated visibility, and explicitly tracing new shadow-rays to correct errors in the estimate. By correctly sharing information among pixels, as well as light directions, we show that accurate visibility predictions can be achieved by tracing only about 10% of the shadow-rays that would normally be required by a standard raytracer without loss in accuracy, as shown in Figure 1. Working source code, data, animations, and scripts to recreate all images are available onine at the address listed at the end of this paper.

2 Background

Previous works describe coherence-based methods for rendering scenes illuminated by point or area lights. Most of these techniques find image-space discontinuities and share visibility or shading information within regions bounded by discontinuities, thus reducing the number of primary eye rays (and recursively, the number of shadow-rays). We discuss some of these methods below, and discover that image-space radiance interpolation is not well-suited for complex illumination. Furthermore, an efficient method cannot rely on the presence of a light boundary, the locality of the light(s), nor the ability to uniformly sample the lighting. When these assumptions are broken, existing methods often degenerate to an exhaustive raytracing for both eye- and shadow-rays.

[Guo 1998] was able to efficiently render a scene lit by area light sources by finding radiance discontinuities in image-space. A sparse grid of primary rays was fully evaluated, and radiance discontinuities were discovered by comparing the results of these calculations. The rest of the image pixels are bilinearly interpolated, thus culling the actual primary and secondary rays for those pixels. Unlike area light sources, environment map illumination produces continuous gradients instead of discoverable discontinuities. The method used by [Guo 1998] would incorrectly interpolate large regions of the scene that in fact have smooth, yet complex variations in radiance.

[Bala et al. 2003] also interpolate radiance in image-space, but determine the regions of interpolation by examining the visibility of lights. Each pixel is classified as empty, simple, or complex, depending on whether it contains 0, 1, or more discontinuities of light visibility. Only complex pixels are fully evaluated. When rendering images illuminated by a sampled environment map, nearly every pixel becomes complex. This is due to the presence of hundereds of light sources, several of which exhibit a change in visibility when moving from one pixel to the next.

[Hart et al. 1999] demonstrated an analytic integration method for calculating irradiance due to area light sources. They find all light-blocker pairs, exploiting coherence of these pairings between nearby pixels. The lights are then clipped against the blocker(s) to produce an emissive polygon corresponding to the visible section of the light source. The light-blocker pairs are discovered using uniform sampling of the solid angle subtended by the area light source, taking advantage of the light's locality. For an environment map which subtends the entire sphere of directions, this method degenerates to brute-force visibility sampling.

[Agrawala et al. 2000] achieve their speedup by assuming that visibility of the light source is similar for adjacent pixels. They determine visibility of regularly spaced samples of the area light source, and then share that visibility information with the subsequent pixel of the scanline order. Errors are corrected by explicitly tracing new shadow-rays to light samples that lie along visibility boundaries (and recursively flooding the shadow traces when errors are detected). Locality of the area light source and a well-defined edge ensured that the number of samples was small, and boundaries were well-defined.

Our work also relates to much recent effort on real-time rendering of static scenes illuminated by environment maps. Precomputed radiance transfer methods such as [Ng et al. 2003] and [Sloan et al. 2002] have a different goal of compactly representing and reusing the results of exhaustive calculations. They perform a lengthy precomputation step during which standard raytracing is used to compute the radiance transferred to all parts of a static scene. In contrast, we focus on computing the visibility of an arbitrary scene more efficiently. PRT methods may benefit from using our algorithm in the raytracer they employ for the precomputation stage.

3 Preliminaries

In this section, we briefly desctibe how our method relates to rendering engines that solve the reflection equation. The exit radiance at a point, x, in the direction ω_0, is given by

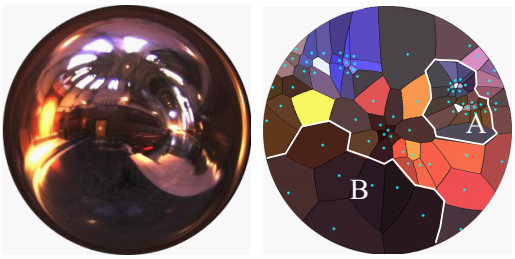

Figure 2: *The environment map for Grace Cathedral (left) is sampled into 82 directional lights. Each light is surrounded by the directions nearest to it, as defined by the Voronoi diagram (right). Note that the sampling is irregular, with region* A *being densely sampled and region* B *containing sparse samples.*

$$L(\mathbf{x}, \omega_0) = \int_{\Omega_{2\pi}} f_r(\omega, \omega_0) L(\omega) V(\mathbf{x}, \omega)(\omega \cdot n) d\omega,$$

where $\mathbf{x}$ is a location in the scene with a BRDF of f_r, n is the normal, and ω is a direction in the upper hemisphere, $\Omega_{2\pi}$. $V(\mathbf{x}, \omega)$ is the 4D, binary visibility function of the scene. The *light-space* of each location, $\mathbf{x}_i$, is the set of directions in the upper hemisphere, $\Omega_{2\pi}$, centered at $\mathbf{x}_i$. $V(\mathbf{x}_i, \omega)$ assigns to each direction in the light-space of $\mathbf{x}_i$, a binary value indicating whether occluding geometry exists in that direction, or not. [Agarwal et al. 2003] demonstrated that the integral above can be turned into an explicit sum over a set of well-chosen directions $\{\omega_i\}$, with associated regions or cells, as shown in Figure 2 *right*. An efficient approximation of several hundred directional lights can be found for most environment maps, and we assume that such a technique has been used. Often, the hardest part of solving for $L(\mathbf{x}, \omega_0)$ is computing the visibility term, $V(\mathbf{x}, \omega)$. Our work concentrates on making this visibility calculation more efficient. Any system which can substitute its computation of $V(\mathbf{x}, \omega)$ in the above equation can make use of the methods described in this paper. We demonstrate that a sparse set of samples in $\{\mathbf{x}\} \times \{\omega\}$ can be used to reconstruct $V(\mathbf{x}, \omega)$ with a high degree of accuracy.

Given a set of directional lights, we partition the set of all directions into regions that are identified with each light, as seen in Figure 2 *right*. This is done by uniformly sampling a unit disk and projecting each sample onto the unit sphere to obtain a direction. The sample is then assigned the ID (visualized as a color) of the light that is closest to the obtained direction, thus constructing the Voronoi diagram. To improve the resolution of regions near the edge of the disk, a transformation such as parabolic mapping ([Heidrich and Seidel 1998]) should be used. In our experience, even for environment maps sampled at 400 lights, with the smallest cells near the horizon, a 513×513 pixel Voronoi diagram suffices. (As presented here, only directions where $y > 0$ appear in the diagram. The extension to the full sphere with a second map is straightforward.) We will see that a notion of adjacency in the light-space is needed. Since the lights are not evenly spaced, we identify all the neighbors of a light region by searching the Voronoi diagram. We found that the average connectivity of lights in our sampled environments is 6. Because we are using environment maps that represent distant light sources, the Voronoi diagram, and the list of neighbors for each light cell is precomputed once for a particular environment.

4 Framework Components

Leveraging coherence to estimate visibility occurs in two phases. First, we construct an approximate notion of visibility based on previous calculations (Section 4.1). Then we must clean up this estimate by identifying and correcting regions that may have been predicted incorrectly (Section 4.2). In this section, we present several choices for compoennts that provide these funcitons. The next section (Secion 5) will analyze the possible combinations to determine the best algorthim for both correctness and efficiency. For the rest of the discussion we will refer to a pixel, and the scene location intersected by an eye-ray through that pixel, interchangeably.

4.1 Estimating Visibility

Our goal is to reuse the points of geometry that represent intersections between shadow-rays and objects in the scene. To predict the visibility at one spatial location, we consider the occluding geometry discovered while calculating visibility at nearby locations, thus taking advantage of spatial coherence. We must choose which previously evaluated pixels

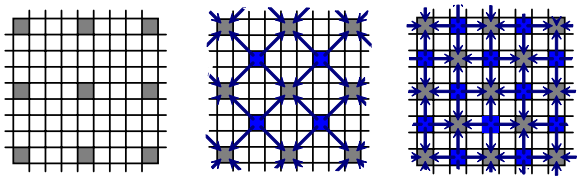

Figure 3: **Grid-Based Evaluation.** *First, the image is fully evaluated at regular intervals. This produces a coarse grid (left). Next, the center (blue) of every 4 pixels is evaluated based on its surrounding 4 neighbors (gray). All 4 are equidistant from their respective finer-gird-level pixel (center). Again, a pixel is evaluated at the center of 4 previously evaluated pixels (right). At the end of this step, the image has been regularly sampled at a finer spaceing. This becomes the new coarse level, and the process repeats.*

will be most helpful when attempting to reuse visibility information. We explore the options of scanline and grid-based pixel evaluations.

4.1.1 Scanline Ordering (S)

When evaluating the pixels in scanline order, we use the previous pixel on the scanline, as well as the two pixels directly above them, on the previous scanline. The reasoning behind this is that visibility changes very little from pixel to pixel, and thus using the immediate neighbors gives a good first approximation to the visibility at the current pixel.

4.1.2 Grid-based Ordering (G)

Scanline ordering has the notable limitation that all of the previously evaluated pixels come from the same general direction (up, and to the left). Therefore, we present an alternative which always allows us to reuse visibility from 4 previously evaluated pixels each one from a different direction. In this case, the pixel location for which visibility is being evaluated is the average of the pixels that are informing the estimation process. To a first approximation, the visibility can also be taken to be the average of the visibility at the 4 nearby pixels. This is similar to the coarse-to-fine subdivision first introduced by [Whitted 1980] for anti-aliasing. The precise pattern (detailed in Figure 3) is chosen here to maximize reuse of information from already-evaluated pixels. We have found that an initial grid spacing between 32 and 8 pixels generally works well for minimal overhead.

4.1.3 Warping (W) and Not Warping (N)

When blockers are distant, the visibility of a light source at a given pixel can be assumed to be the same at a nearby pixel. However, when blockers are close to the surface being shaded, it may be advantageous to warp the direction of a blocker in the nearby pixel's light-space, to a new location in the light-space of the pixel being shaded, as in Figure 4.

Though mathematically correct, warping does not produce a completely accurate reconstruction of blocked regions. As in any image-based rendering (IBR) scheme, warping results in holes, and other geometry reconstruction errors. Estimation errors must be corrected, as described in the next section. We have discovered through empirical testing that this correction method can work well both when warping, and when not warping the blockers from nearby pixels. In both cases, the initial visibility estimation contains enough accurate information to allow for good error correction.

4.1.4 Constructing the Visibility Estimate

The visibility estimate for a pixel is constructed as follows: Each pixel is informed of blockers that were discovered in its neighbors (scanline or grid-based). The direction to these blockers is known, and may be warped to adjust for disparity between the pixel and its informers. At this point the pixel is aware of as many blockers as existed in all of its neighbors, combined. Each blocker is assigned to one of the cells in the light-space of

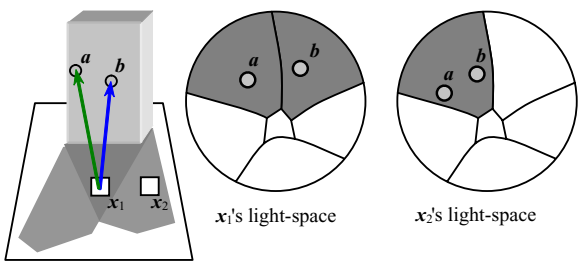

Figure 4: **Warping Blockers.** *Visibility for x_1 is computed by tracing shadow-rays and finding a and b. The visibility for x_2 is predicted by warping blocker points a and b to their positions in the light-space relative to x_2.*

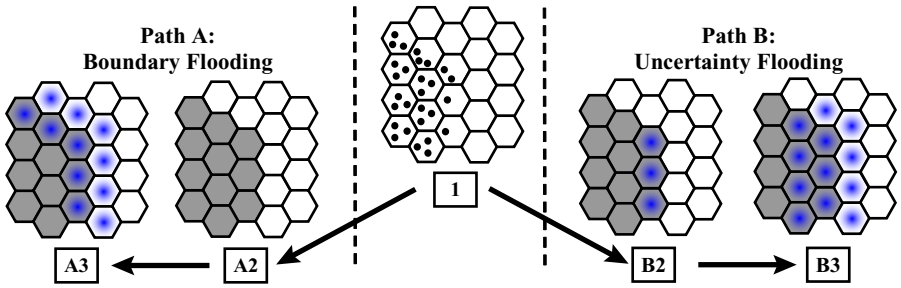

Figure 5: 1: *The blockers (black dots) from 3 neighboring pixels (not shown) are placed in the light-space of the current pixel. Each hexagonal cell represents a cell in the Voronoi diagram that partitions the light-space.* **Path A.** A2: *Light cells which contain more blockers than the threshold (1) are predicted as blocked (grey), while others are predicted as visible (white).* A3: *Light cells whose neighbors' visibility differs are shadow-traced (blue center).* **Path B.** B2. *Light cells that contain many (in this case 3 since we are using 3 previous pixels) blockers are predicted as blocked. Those that contain no blockers are predicted as visible. Light cells containg few (1 or 2) blockers are marked as uncertain and shadow-traced (blue center).* B3. *The neighboring light cells of all uncertain cells are shadow-traced.*

that pixel. (By using the precomputed Voronoi diagram discussed in Section 3, finding the cell corresponding to any direction is a simple table lookup.) Some cells receive many blockers, some few, while others may receive none, as seen in Figure 5.1. The following subsection discusses how to interpret and correct this data.

4.2 Correcting Visibility Estimates

We have shown that spatial visibility coherence among nearby pixels can be used to make educated estimates of visibility. In this section we discuss how angular coherence among nearby directions can be used to correct errors produced by the estimation phase. Once some estimate of visibility exists for the light-space of a pixel, we would like to identify regions of the light-space that may have been estimated incorrectly, and fix them.

4.2.1 Boundary Flooding (B)

Any cell that contains more blockers than a given threshold is assumed to be blocked (Figure 5.A2). Our experience shows that the exact value of this threshold has little effect on the final algorithm, and we set it to 0.99. This estimation of visibility is prone to error near the boundary of cells which are visible, and cells which are blocked. This is because such a boundary is likely to exist, but may not lie exactly where the estimate has placed it. We attempt to correct these errors by explicitly casting shadow-rays in the directions that lie along a boundary (Figure 5.A3). Wherever the shadow-ray indicates that an error had been made in the estimation stage, we recursively shadow trace all the adjacent regions. This process continues until all shadow-rays report back in agreement with the estimate.

4.2.2 Uncertainty Flooding (U)

Instead of guessing that boundaries are prone to error, we can use a more explicit measure of when we are uncertain of the visibility constructed by the estimation phase. Cells that contain many (4 in grid-based, 3 in scanline) blockers are considered blocked because all of the informing pixels agree that it is blocked. Similarly, cells without any blockers are considered visible. Whenever the neighbor pixels disagree on the visibility for any cell of the light-space (indicated by a small number of blockers), we mark it as uncertain (Figure 5.B2). We explicitly trace shadow-rays for these regions, and flood outwards, recursively tracing more shadow-rays when the estimation disagrees with the shadow-ray (Figure 5.B3).

5 Analyzing Possible Component Combinations

So far, we have developed four variations of the first phase: estimating visibility. We can use scanline evaluation (S) or grid-based evaluation (G), and for each of these, we may choose to warp (W) or not warp (N) the blockers from nearby pixels. We have also developed two variations for the second phase: correcting the estimate. We can trace shadow-rays at the visibility boundaries (B) or at the explicitly marked uncertainty regions (U). We performed comprehensive evaluations of all eight possibilities, and will now present the findings of the components' usefulness, both on their own, and as they interact with the other variations. We believe these evaluations can offer insights for future work in coherence-based sampling.

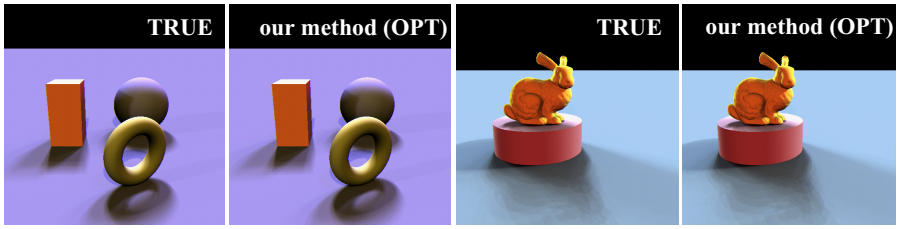

Figure 6: *Shapes and bunny, lit by 200 lights. OPT is indistinguishable from TRUE.*

5.1 The Scenes

We analyze the results of different combinations of the framework components for 4 scenes shown in Figures 1, 6, and 8. All tests were run on 513×513 images. The scenes contain only diffuse, non-textured objects to maximize the visibility of cast shadows. The lighting environments are high-dynamic range light probes from www.debevec.org, sampled into point lights according to [Agarwal et al. 2003]. We enhance the visualization of shadows by increasing the intensity of lights in small regions. Each scene uses a different environment map. The ***shapes*** scene shows simple objects with both smoothly varying and discontinuous curvature, with curved cast shadows. Its environment contains several tight clusters of bright lights. The ***bunny*** represents a scene with a common combination of simple parametric objects and complex curved geometry with self-shadowing. The ***dominos*** show a scene with many separate objects that interact, resulting in fairly detailed shadows, as might be found in a typical animation. The bunny and dominos environments contain small lights near the horizon, as well as non-uniform lights from above. The ***plant*** scene includes detailed geometry that casts very intricate shadows with low coherence. Its environment is relatively uniform. A representative example of the data we collected for these scenes is shown in Figure 7 — a close-up of the shapes scene.

5.2 Correctness Analysis (Scanline vs. Grid)

By examining Figure 7, we notice that only one of the combinations which uses scanline evaluation order was able to produce reasonable results: SWB. For boundary flooding, the change in visibility may occur away from any existing boundaries. This is exactly the case that the torus reveals, and the reason its shadow is filled in Figure 7:SNB. For uncertainty flooding, scanline's informing pixels are too close together to have very different notions of the scene geometry, and no correction is initiated in the form of uncertainty flooding. Hence, there can be staircasing artifacts as seen in Figure 7:SNU. Even with warping, uncertainty flooding from the three pixels immediately to the left and above does not suffice, although it performs somewhat better, as seen in Figure 7:SWU. Thus, we must use warping and boundary flooding in scanline algorithms. It is interesting to note that this corresponds closely to the approach of [Agrawala et al. 2000].

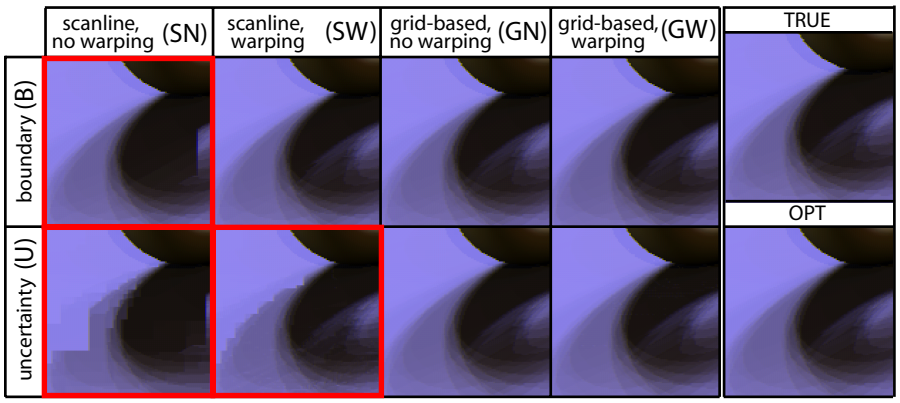

Figure 7: **Error Ananlysis.** *The above table shows possible combinations of the two components of our framework as described in Section 4. The* TRUE *image is provided for comparison. This is a close-up of the torus shadow in the shapes scene, as illuminated by a 100 light approximation. The* OPT *result, as described in Section 6, is shown in the bottom-right. No perceptible errors exist. Significant errors are visible in those combinations outlined in red, whereas the other renderings contain very minor errors, only noticable under high magnification.*

	shapes				bunny				plant				dominos			
lights:	50	100	200	400	50	100	200	400	50	100	200	400	50	100	200	400
TRUE	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
SWB	40.5	34.3	26.3	22.4	52.4	42.6	33.6	25.9	63.2	51.6	44.3	39.7	54.5	43.6	36.0	28.8
GWB	22.0	17.9	15.1	12.4	21.5	16.6	13.5	11.6	37.8	34.5	32.6	30.5	31.6	28.2	24.9	21.8
GNB	21.7	17.1	13.8	10.6	21.2	15.8	12.0	9.3	37.7	34.6	32.7	30.6	30.3	25.7	21.9	18.0
GWU	10.4	10.7	11.0	11.3	9.1	9.4	10.2	11.5	33.3	32.4	32.3	31.1	20.4	22.3	22.2	22.6
GNU	9.4	8.8	8.3	7.5	8.1	7.7	7.2	6.7	35.3	33.8	33.4	31.9	17.3	16.9	16.3	15.3
OPT	4.8	4.6	4.5	4.3	4.6	4.4	4.1	3.9	21.4	20.9	20.8	20.2	9.4	9.2	9.1	8.8

***Table 1:* Percentage of Shadow-Rays Traced.** *The entries for the table indicate the percentage of shadow-rays where $N \cdot L > 0$ that were traced by each method. For each scene an environment map was chosen, and then sampled as approximately 50, 100, 200, and 400 lights.*

environments sampled at 200 lights	shapes		bunny		plant		domino	
	sec	speedup	sec	speedup	sec	speedup	sec	speedup
primary rays only (EYE)	4.0	N/A	4.0	N/A	4.4	N/A	4.2	N/A
ground truth (TRUE)	75.3	1.0	80.8	1.0	152.3	1.0	124.9	1.0
scanline, warping, boundary (SWB)	34.9	2.2	54.8	1.5	120.3	1.3	67.1	1.9
grid-based, warping, boundary (GWB)	25.2	3.0	36.7	2.2	113.2	1.3	55.7	2.2
grid-based, no warping, boundary (GNB)	16.8	4.5	23.2	3.5	95.1	1.6	35.4	3.5
grid-based, warping, uncertainty (GWU)	19.6	3.8	30.6	2.6	109.0	1.3	47.1	2.6
grid-based, no warping, uncertainty (GNU)	11.2	6.7	16.7	4.8	94.7	1.6	23.8	5.2
optimized algorithm (OPT)	7.1	10.5	10.7	7.6	58.3	2.6	14.7	8.5

***Table 2:* Timings.** *The wall-clock timings for the full rendering with 200 lights, shown here, are related to, but not equivalent to the reduction in work as measured in shadow-rays traced (see Table 1).*

All four grid-based possibilities (Figure 7:G--) give almost error-free results. The coarse-to-fine nature of grid-based evaluation is robust in the presence of different scales of visibility events. In addition, by using the grid-based approach, boundary and uncertainty regions are marked conservatively enough to allow the correction stage to fix nearly all the errors of the estimate. The details of each case are discussed in the next subsection.

5.3 Performance Analysis

The previous subsection concluded that three of the scanline algorithms are error-prone (outlined in red in Figure 7), and we do not consider them further. We now examine the efficiency of each of the remaining combinations (SWB, GWB, GWU, GNB, GNU). Tables 1 and 2 compare the performance for each of the four scenes. Table 1 reports the percentage of shadow-rays traced for each method, at different light sampling resolutions of the scenes' environments (50, 100, 200, 400). Table 2 shows the actual running times for rendering each scene with 200 lights, as timed on an Intel Xeon 3.06GHz computer, running Windows XP. The times reported in Table 2 include the (relatively small) time for the initial setup of the raytracer and tracing just the EYE rays for each pixel, as well as performing the diffuse shading calculation. The errors in the five remaining algorithms are generally small (<0.1%) either when measured as absolute mispredictions, or when viewed as the resulting color shift in the image. All of these algorithms enable significant reductions in the number of shadow-rays traced, with negligible loss in quality.

5.3.1 To Warp or Not to Warp

The motivation for warping blockers into the local frame of a pixel is that it may produce a better approximation of the occluding scene geometry. This turns out to be counterproductive for non-uniformly distributed samples of the light-space. Consider Figure 2 *right*. When the blockers in an area of densely sampled lights (region A), are warped onto an area of sparse lights (region B), large cells in region B that are mostly visible may receive a large number of blockers, clustered in a corner of the cell. These would be incorrectly estimated as blocked. Symmetrically, if warping occurs form a sparse region to a dense one, cells that are actually blocked may not receive any blockers and be erroneously estimated as visible. When the spacing of the light samples is not uniform, warping can introduce more errors in the visibility estimation, requiring more work to be done in the correction phase.

When comparing G**W**U with G**N**U and G**W**B with G**N**B, Table 1 shows that the non-warping algorithms perform slightly better for all but the plant scene. When comparing the warping algorithms (G**W**B, G**W**U) to the equivalent non-warping algorithms (G**N**B, G**N**U) in Table 2, we can see a significant speedup in the wall-clock timings. Not warping

requires fewer instructions since a simple count of the blockers in each cell suffices, whereas warping must perform the 3D transformation of the blockers and project them into the new cells. One additional advantage of not warping is that we eliminate the dependence on accurate depth information for the blockers. This can be useful when rendering engines optimize the intersection test of shadow-rays by stopping as soon as an intersection is found, without discovering the true depth of the closest blocker. (To obtain controlled comparisions, we did not turn on this optimization.)

5.3.2 Boundary Flooding vs. Uncertainty Flooding

Since the illumination from an environment map spans the entire sphere, many visibility boundaries can exist, resulting in flooding to most of the lights when boundary flooding is used. Uncertainty flooding has the advantage of being restricted to those areas which are visible at some neighboring pixels, and blocked at others. It is particularly well suited to combat the sampling discrepancies described in the previous subsection (see Section 5.3.1 and Figure 2). When under- or over-sampling of the light-space occurs, some cells in that region receive few (1-3) blockers, and trigger explicit shadow traces for their neighbors. Marking cells as uncertain is equivalent to finding image-space discontinuities for individual shadows. If such a discontinuity passes through the area surrounded by the informing pixels, that light is marked as uncertain in the current pixel. In essence, uncertainty flooding concentrates on regions with image-space discontinuities, while boundary flooding concentrates on light-space discontinuities.

Looking at Table 1, we see that in all cases, uncertainty flooding performs about twice as well as the corresponding boundary flooding algorithm, at the lower resolutions. In particular, the first row (shapes50), shows GW**B** doing 22% of the work and GW**U** doing only 10%. Similarly, GN**B** does 22% of the work, while GN**U** does only 9%. This indicates that marking boundaries as regions that require shadow-tracing is an overly-conservative estimate. In our experience, not all visibility boundaries are error-prone.

Since the work done by boundary flooding is proportional to the area of the light-space occupied by boundaries, its performance is strongly dependent on the sampling rate of the environment. Notice that for shapes, bunny, and dominos, boundary flooding algorithms degrade by a factor of 2 as the light resolution is decreased from 400 to 50. On the other hand, the performance of algorithms based on uncertainty flooding is relatively independent of the number of lights.

5.4 Discussion of Analysis

The examples shown in Figure 7 indicate that scanline evaluation order is much more constrained than grid-based. This is primarily because the small increments as the scanline progresses are sensitive to the scale of changes that may occur in visibility. We see in the results outlined in red in Figure 7, that a missed visibility event propagates along the scanline until some other event triggers its correction via flooding. It is instructive to compare GWB and SWB in Table 1, and notice that GWB is more efficient by about a factor of 2 for most cases. One reason for this is that scanline must consider the lights near the horizon as part of a boundary. New blockers that rise over the horizon as the scanline progresses are missed without flooding on the lights near the horizon of the pixel's local frame. On the other hand, grid-based evaluation incorporates information from all four directions. This enables it to mark new visibility events as regions of uncertainty when they occur between the 4 informing pixels (i.e. near the pixel of interest).

Correctness is achieved when the correction phase is given an accurate guess of where errors might exist in the initial estimate. Efficiency is achieved when the original estimate is good, and indicators of possible error are not overly-conservative. For the algorithms that correctly combine the estimation and correction phases, accuracy is always obtained. In difficult scenes that exhibit low coherence (such as the plant), the algorithms degrade in performance, but still produce accurate results.

6 Optimizations and Implementation

Based on the analysis of the previous section, we determine that GNU is the best combination of the framework components for both accuracy and performance. We also rendered animations for all of our test scenes by rotating the environment to see if any popping artifacts would occur. All of the animations using GNU were indistinguishable form the TRUE animation. We therefore base our optimized algorithm (OPT) on GNU, making it grid-based, without warping, and using uncertainty flooding as the means of correcting errors (Section 6.1). We describe the implementation details for adding a GNU-based algorithm to any standard ray-tracing system in Section 6.2. Finally, we take an in-depth look at the time and memory usage implications of these methods (Section 6.3).

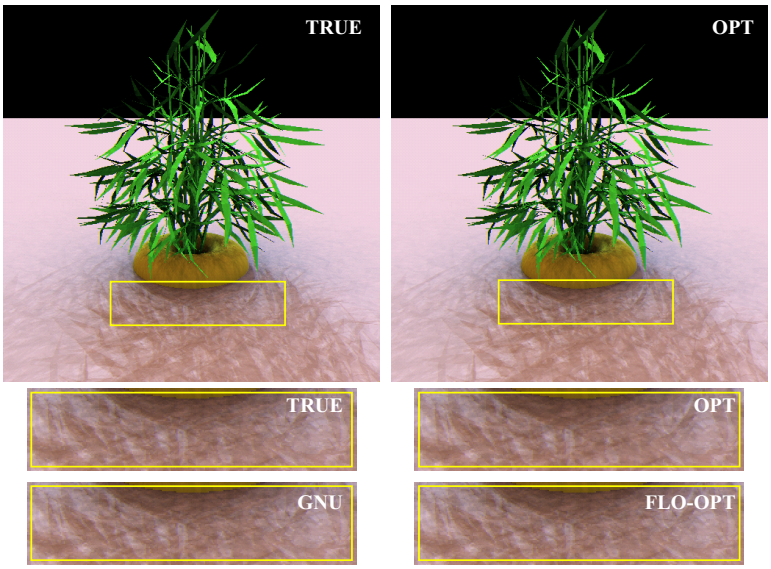

Figure 8: **Plant.** *The plant image, seen here as illuminated by a 100 light approximation of the Kitchen environment, has very detailed shadows with low coherence. Nevertheless, our algorithms produce a very accurate image, as seen in the direct comparisons and closeups.*

6.1 Optimized Algorithm

More than half of the shadow-rays traced by our algorithms are triggered by flooding. We have seen that both boundary and uncertainty flooding are effective methods for correcting prediction errors. A close inspection of the shadow-rays traced due to flooding reveals that many of them agree with the original visibility estimate, and therefore only serve to halt further flooding, without contributing anything to the visibility calculation. We can avoid many such shadow-traces by combining the ideas of uncertainty and boundary flooding. We still shadow-trace all regions of uncertainty as indicated by the original visibility estimate (Figure 5.A2). However, instead of flooding to all neighbors (typically 6) of a cell whose shadow-trace disagrees with the estimate, we only flood to those neighboring cells that also disagree with the shadow-trace result. This means that for uncertainty regions that lie on a border, we only flood onto one side of the border; trying to find where it truly lies. This optimization reduces the number of flood-triggered traces by about 40%. For complex shadows, small artifacts are introduced by this method, as may be visible in the Figure 8:OPT close-up, but we feel that the performance gain is worth the slight difference in image quality. Since animations are more sensitive to small differences, we recommend that this optimization only be used for stills, and omitted in animations. (A comparison of these animations for the shapes scene is available online at the address listed at the end of this paper.)

In an effort to further reduce the number of shadow-rays traced due to flooding, we note that half of the image pixels are contained in the finest grid — when the initial estimate is informed by a pixel's immediate neighbors. If all 4 neighbors agree on the visibility of a particular light, the center pixel can only disagree if there is a shadow discontinuity smaller than a single pixel passing through it. Since the image cannot resolve elements smaller than a single pixel, we ignore these sub-pixel shadow discontinuities and turn off flooding altogether at the grid's finest level only (FLO). We still rely on error correction due to explicit uncertainty marking. This reduces the total number of shadow-rays required to calculate the image by about 25%, without adding any noticeable artifacts to the final image, as seen in the Figure 8:FLO-OPT close-up. This optimization also is safe for animations, as its effects are only noticeable at a sub-pixel level. Sub-pixel sampling for antialiasing is a popular method employed by many rendering engines. When such a feature is used in conjunction with our method, the highest resolution of sampling should be considered to be the native image resolution for the purposes of defining the grid.

Our optimized algorithm (OPT) uses the above two methods of restricting flooding; flooding only to neighbors that disagree with the shadow-ray, and not flooding at the finest grid level. The high quality of OPT can be seen in Figures 1, 7, 6, and 8. The close-ups in Figure 7 and Figure 8 reveal that OPT produces images nearly indistinguishable from the ground-truth (TRUE). Even on the stress-test provided by the plant scene (Figure 8), OPT has only a 0.4% error rate, as measured in actual shadow-ray prediction errors. The

error rate for OPT on the bunny is 0.0026%, meaning only 1 in 38,000 shadow-rays disagree from ground-truth. In terms of visual quality of the images OPT produces accurate results for all the scenes. As visualized in the final images, these errors produce rendering that are nearly identical to the true images when viewed as standard 24 bit RGB images. Adding the above two optimizations typically leads to a 50% reduction in the number of shadow-rays traces, when compared to GNU. As reported in Table 1, this can result in a factor of 20 improvement in efficiency over standard visibility testing in ray-tracers. In Table 2, the timings show that as OPT's performance improves, shadow-tracing is no longer the bottleneck for rendering. In the shapes scene, tracing the primary eye rays (4.0 seconds) takes longer than adding complex shadows to the scene (3.1 seconds).

6.2 Implementation

```
    procedure Calculate-All-Visibilities()
 1     for (gridsize = 16; gridsize > 1; gridsize /= 2) do
 2        foreach pixel, p, in current grid do
 3           lights-to-trace ← ∅;
 4           n[1..4] ← Find-Evaluated-Neighbors(p);
 5           object(p) ← Trace-Primary-Ray-At-Pixel(p);
 6           if object(n[1]) == object(n[2]) == object(n[3]) ==
                object(n[4]) == object(p) then
 7              foreach light, L, in sampled-environment-map do
 8                 if vis(n[1], L) == vis(n[2], L) ==
                       vis(n[3], L) == vis(n[4], L) then
 9                    vis(p, L) ← vis[n[1], L);
                   else
10                    vis(p, L) ← 'uncertain';
11                    lights-to-trace ← lights-to-trace ∪ L;
                   endif
                endforeach
12              Flood-Uncertainty(p);
             else
13              foreach light, L, in sampled-environment-map do
14                 vis(p, L) ← Trace-Shadow-Ray(p, L);
                endforeach
             endif
          endforeach
       endfor
```

Our algorithms are all very simple to add to a conventional ray-tracer or other rendering system. We have done so in the context of a widely available ray tracer, POV-Ray. Adding coherence-based sampling requires few changes to the core rendering engine, and uses only a few additional simple data structures for support. Furthermore, in our experience, care need not be taken to ensure that the implementation is engineered precisely. Even if it does not adhere strictly to our method, it will still provide a vast improvement over full ray-tracing. In particular, GNU and OPT are very simple to implement. For each pixel, once the 4 nearby pixels are identified in the grid-based evaluation, a simple check of agreement for each light is performed. If all 4 pixels agree that a light is blocked or visible, it is predicted as blocked or visible, respectively. When the 4 pixels do not agree on a particular light, it is shadow-traced, with possible flooding to neighboring lights. If the optimizations are used, only neighbors whose prediction disagrees with the shadow-trace are traced, and no flooding to neighbors is performed at the final grid level. Code for GNU is shown in `procedure Calculate-All-Visibilities`.

Lines 1–4 describe the use of grid-based evaluation as shown in Figure 3. To prime this process, the coarsest level of the grid is evaluated at a scale of 16 pixels, which we found to be suitable for all of the scenes we tested. For each of these pixels (<0.5% of the image), all shadow-rays are traced. This is comparable to the priming for scanline which requires the top row and left column to be fully evaluated (0.4% for 513x513 images). In lines 5 and 6, we perform a sanity check to make sure that only visibility calculations relating to the same scene object are used for estimation. We have found that there is a significant loss in coherence when the informing pixels are not on the same object as the current pixel. We therefore fully shadow-trace around these silhouette edges to determine visibility. For very complex scenes such as the plant in Figure 8, explicitly tracing in this

situation traces 7% of the shadow-rays. For average scenes like those in Figure 1 and Figure 6, this results in about 1–3% of the shadow-rays.

```
    procedure Flood-Uncertainty(pixel p)
1       while (lights-to-trace ∩ set-of-untraced-lights(p)) ≠ ∅,
         with an L from the above intersection do
2         tempVis ← Trace-Shadow-Ray(p, L);
3         if tempVis ≠ vis(p, L) then
4            vis(p, L) ← tempVis;
5            foreach L's neighbors, N do
6               lights-to-trace ← lights-to-trace ∪ N
             endforeach
          endif
        endwhile
```

In lines 7–11, the initial rough estimate of visibility is formed as shown in Figure 5 steps 1 and A2. In line 12, the error correction phase is called, as described in Section 4.2.2, and diagrammed in Figure 5.A3. The pseudocode for this is shown in `Flood-Uncertainty`.

When `Flood-Uncertainty` is called, the set *lights-to-trace* contains all those lights which were deemed 'uncertain'. None of these have been traced yet. All other lights have had their visibility predicted, but also were not explicitly traced. Line 1 checks to see if there are any lights in *lights-to-trace* which have not yet been shadow-traced. If any such light, L, is found, a shadow-ray is traced from the object intersection for p in the direction of L, at Line 2. At Line 5, the neighbors of L are added to the list of lights to trace. If OPT were being implemented, line 5 would only add those neighbors whose prediction disagreed with *tempVis*. It may seem reasonable to remove a light from *lights-to-trace* once it has been traced, rather than checking against all the untraced lights in Line 1. However, doing so would lead to an infinite loop since Line 6 would add a neighbor, and when that neighbor got traced, it would add the original light.

6.3 Time and Memory Considerations

In this section, we evaluate some of the more subtle (yet interesting) aspects of coherence-based visibility sampling.

6.3.1 Timings

While the improvements in running time reported in Table 2 are significant, in some cases, they are somewhat less than that predicted from the reduction of shadow-rays alone, shown in Table 1. We investigate this further in Table 3, which focuses only on the time spent tracing shadow-rays. Rows 1 and 2 show the wall-clock time for a full rendering of the scenes using both TRUE (trace all shadow-rays) and OPT. We compare actual and theoretical speedups in rows 3 and 4. Row 5 compares these two speedups. Notice that for bunny and plant, the actual timed speedups are not as good as the theoretically predicted speedup. We measured the average time for just the operation of tracing a shadow-ray, and report these in Rows 6 and 7. We do this by generating a list of shadow-rays traced during the rendering with each method. Those shadow-rays are then retraced in the absence of any other operations. Since both methods use the same unmodified POV-Ray ray-tracing engine, this is a very controlled experiment. A perhaps surprising result is

400 light sampling		shapes	bunny	plant	domino
1	sec to trace TRUE	245.50	232.70	378.10	337.50
2	sec to trace OPT	9.40	14.80	118.90	26.20
3	time speedup	26.12X	15.72X	3.18X	12.88X
4	theoretical speedup	23.20X	24.33X	4.94X	11.38X
5	ratio of speedups	1.13	0.65	0.64	1.13
6	usec/ray in TRUE	3.24	3.52	5.62	5.39
7	usec/ray in OPT	3.05	5.82	8.68	4.93
8	ratio of usec/ray	1.06	0.60	0.65	1.09

***Table 3:* Shadow-Ray Timings.** *Row 5 shows the ratio of real speedup based on wall-clock timings (Rows 1 and 2), relative to theoretical speedup based on the number of shadow-rays traced as reported in Table 1. Row 8 shows the ratio of average times to trace a shadow-ray in the unmodified portion of the ray-tracer, with and without our method. A comparison reveals that the perceived loss in speedup is due to the increased average difficulty of tracing a shadow-ray, due to culling of 'easy' shadow-rays.*

that not all shadow-rays are equally expensive. In particular, shadow-rays that are not occluded generally need to perform only the bounding box intersection test. When a ray intersects an object, or grazes the silhouette of an object, an exact point of intersection must be discovered (or proven non-existent). In our algorithm, we optimize away many of the 'easy' shadow tests, tracing primarily the 'difficult' shadow-rays. Hence, the average time to trace a shadow-ray in OPT can exceed that in TRUE. Row 8 of Table 3 shows the ratio of the average time per ray when all rays are traced (TRUE), versus the average time to trace only rays needed by OPT. A comparison to Row 5 reveals that the perceived loss in speedup is due to the increased average difficulty of tracing a shadow-ray. Furthermore, the actual computational overhead of our algorithm is negligible. For the shapes and domino scenes, we achieve the theoretical speedups from Table 1.

We performed a similar analysis of timings for algorithms that include warping. In those algorithms, some extra time (20%) is expended on operations such as blocker retrieval and warping, and light-cell identification through lookup in the Voronoi diagram. The added overhead is partially due to cache misses in the memory. The net performance of the grid-based warping algorithms is still a significant gain over standard ray tracing in most cases, as seen in Table 2.

6.3.2 Memory

The grid-based algorithms that use warping store all blocker points for reuse. The memory usage for 200 lights, for a 513x513 image, corresponds to approximately 160MB, assuming 20 bytes per blocker. When memory usage is important, an additional 3% of the pixels can be traced to split the image into independent blocks of 64x64. The peak memory requirements will then be approximately 2.5MB. This compares favorably with the size of the geometry for complex scenes, such as the bunny model.

For algorithms that do not warp, such as OPT, the memory consumption can be significantly reduced. In this case, we do not need to store information about the blocker, but simply need one bit for storing past visibility calculations. In these cases, for 200 lights in a 513×513 image, memory usage can be reduced to 5MB. If we were to use independent blocks of 64×64, as discussed above, the memory consumption can be reduced to a very insignificant 100KB.

7 Other Considerations

When strong coherence exists in the visibility function, as in most scenes, efficient evaluation can be achieved using very simple coherence-based methods. It is important to couple predictions based on coherence with an appropriate measure for discovering areas of possible error. IBR uses the techniques of warping and splatting to recreate geometry. After experimenting with splatting (not discussed in this paper), we determined that too much information must be gathered to attempt a highly detailed reconstruction. We were encouraged to start with a rough approximation, and work to create appropriate error-correcting functionality. This approach is aided by the fact that in a ray-tracing environment we can benefit from the added ability to introduce new samples wherever we believe they are needed. A good measure of uncertainty guarantees that even scenes with weak coherence, such as the plant, will be rendered accurately, though at a lower prediction rate. It is important to remember that our method is only as accurate as the sampling of the environment map. If tiny objects exist in the scene such that all shadow-rays miss them, the lighting must be sampled more densly to produce the cast shadows from such objects. It is also possible to create a scene that fools our method into believing that all predictions are correct, when they are not. This would require an unnatural collusion between the geometry, lighting, camera position, and image resolution. If there is concern that such a grid of objects exists in the scene, boundary *and* uncertainty flooding can be used. While we tested our method with many types of objects, we did not experiment with particle objects that cast shadows, such as hail. We expect real-world scenes with gravity and solid objects to be rendered accuratly with our method.

The algorithm we presented considers the prediction of each shadow-ray to be equally important. In real scenes, some lights contain more energy than others, and the cosine fall-off term gives further importance to particular lights. It is straight-forward to consider a scheme that catalogs the lights with the most energy for each possible normal direction, and treats those lights specially. Specifically, such lights tend to be small, and thus are more likely to be mispredicted. Adding boundary flooding for only these high-energy lights ensures higher fidelity in the final image. Besides the differing energy of shadow-rays, we saw in Section 6.3.1 that some shadow-rays are more expensive to trace than others. Unfortunately, the work required to trace a ray seems to be inversely proportional

to the predictability of its path. This is understandable, and may be used in the future to set a theoretical lower bound for coherence-based ray-tracing.

Our algorithm also considered accuracy to be much more important than efficiency. We presented methods of combining and optimizing the components in a way that often sacrifices efficiency to maintain accuracy. It may be possible to achieve higher speedups if some tolerance for errors exists. We saw in Section 6.1 that restricting the flooding can improve performance at the cost of some error. An adjustable level of such flood-restriction may provide more control for accuracy/efficiency tradeoff. However, one must be careful of a naive implementation that would result in unwanted poping artifacts, or other large fluctuations in the image.

8 Web Information

Additional material can be found online at http://www.acm.org/jgt/papers/Ben-ArtziEtAl05. All of the data needed to recreate the images in this paper can be downloaded, along with scripts that run the appropriate exacutable for each image. The code used for this project is provided with a switch to use it in conjunction with Pov-Ray. It can also be used as a standalone executable on the scenes in this paper. Animations of a 360^o lighting rotation are also available.

Acknowlegments:

This work was supported in part by grants #0305322 and #0446916 from the National Science Foundation, and an equipment donation from Intel Corporation.

References

Agarwal, S., Ramamoorthi, R., Belongie, S., and Jensen, H. W. 2003. Structured Importance Sampling of Environment Maps. *ACM Transactions on Graphics, Vol. 22*, No. 3 (July), 605–612.

Agrawala, M., Ramamoorthi, R., Heirich, A., and Moll, L. 2000. Efficient Image-Based Methods for Rendering Soft Shadows. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 375–384.

Bala, K., Walter, B. J., and Greenberg, D. P. 2003. Combining Edges and Points for Interactive High-Quality Rendering. *ACM Transactions on Graphics, Vol. 22*, No. 3 (July), 631–640.

Blinn, J. F., and Newell, M. E. 1976. Texture and Reflection in Computer Generated Images. *Communications of the ACM, Vol. 19*, No. 10, 542–547.

Debevec, P. 1998. Rendering Synthetic Objects Into Real Scenes. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, 189–198.

Greene, N. 1986. Environment Mapping and Other Applications of World Projections. *IEEE Computer Graphics & Applications, Vol. 6*, No. 11, 21–29.

Guo, B. 1998. Progressive Radiance Evaluation Using Directional Coherence Maps. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, 255–266.

Hart, D., Dutré, P., and Greenberg, D. P. 1999. Direct Illumination With Lazy Visibility Evaluation. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 147–154.

Heidrich, W., and Seidel, H.-P. 1998. View-independent environment maps. In *1998 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, 39–46.

Kollig, T., and Keller, A. 2003. Efficient Illumination by High Dynamic Range Images. In *Eurographics Symposium on Rendering: 14th Eurographics Workshop on Rendering*, 45–51.

Miller, G. S., and Hoffman, C. R. 1984. Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments. In *Course Notes for Advanced Computer Graphics Animation in Siggraph 84*.

Ng, R., Ramamoorthi, R., and Hanrahan, P. 2003. All-Frequency Shadows Using Non-linear Wavelet Lighting Approximation. *ACM Transactions on Graphics, Vol. 22*, No. 3 (July), 376–381.

Ostromoukhov, V., Donohue, C., and Jodoin, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics, Vol. 23*, No. 3 (Aug.), 488–495.

Sloan, P.-P., Kautz, J., and Snyder, J. 2002. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. *ACM Transactions on Graphics, Vol. 21*, No. 3 (July), 527–536.

Whitted, T. 1980. An Improved Illumination Model for Shaded Display. *Communications of the ACM, Vol. 23*, No. 6, 343–349.