Efficient Shadows from Sampled Environment Maps

Columbia University Tech Report CUCS-025-04 http://www.cs.columbia.edu/cg/

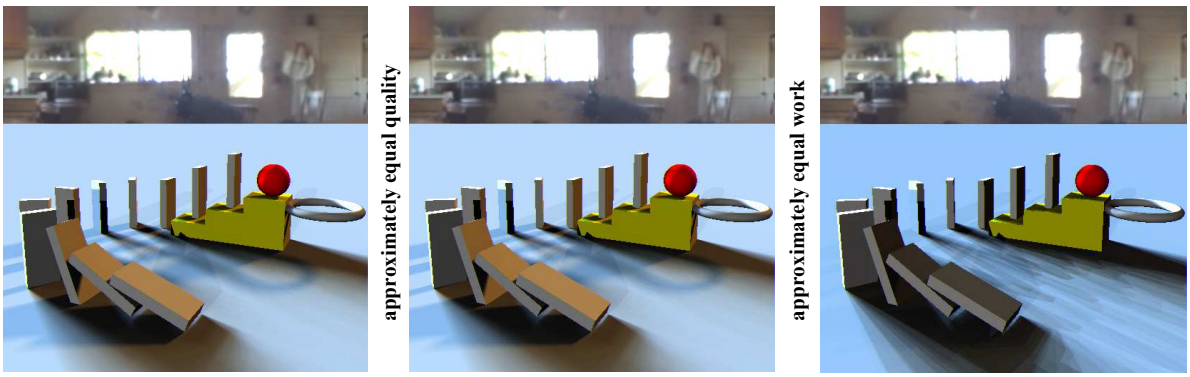
Aner Ben-Artzi
Columbia University
www.cs.columbia.edu/~aner/

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. 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, such as by ray-tracing. 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 four 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 twice the cost of rendering without shadows.



60 million shadow rays
standard ray-tracing

6 million shadow rays
Coherence-Based Sampling

7½ million shadow rays
standard ray-tracing

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 [ARBJ03]. 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 [ARBJ03]) 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. In this paper, we focus on realistic lighting from environment maps, as first described in [BlNe76], and later in [MiHo84,Gre86,Deb98]. One of the most important visual cues is the subtle shadowing effect that cannot be approximated with simple, hard shadows. Correct shadowing requires solving for the visibility of each pixel with respect to each light direction. This is extremely expensive, since the illumination can come from any direction in the environment map. We leverage recent sampling methods like [ARBJ03, KoKe03], that reduce the environment to a few hundred directional lights. Shadow testing for all sources at each pixel is still the bottleneck in a conventional raytracer or other renderer.

In this paper, we show that the visibility function can be efficiently calculated by exploiting coherence in both the angular and spatial dimensions. Existing work, such as [Guo98, BWG03, HDG99], makes use of coherence for scenes illuminated by point or area lights. Most of these techniques find discontinuities in image-space, and share visibility within regions bounded by discontinuities, thus reducing the number of primary rays needed. Such methods are difficult to use for sampling environment map illumination because hard discontinuities are not discernible in the image. Often, the result of hundreds of shadow-ray visibility calculations are combined in every pixel. For example, [Guo98] relies on discontinuities in the image color, which do not exist in the soft shadows produced by sampled environment maps. To correctly apply [Guo98] to environment maps, only a small number of light samples could be used at once. The amount of work done by the algorithm would grow linearly with the number of lights used to sample the environment. [HDG99] stores blocker-light pairs at each pixel. This method would require hundreds of pairings for sampled environment maps. The expensive operation when lighting with complex illumination is the tracing of secondary shadow rays.

Our work is closest to that of [ARH00], who exploited coherence in the visibility of an area light source to reduce

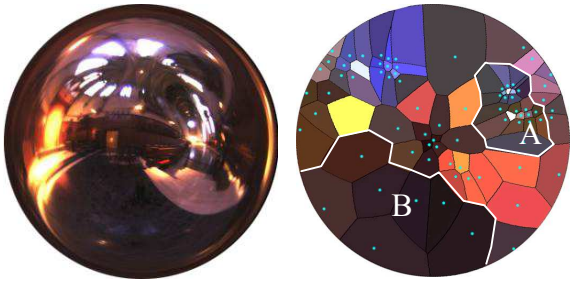
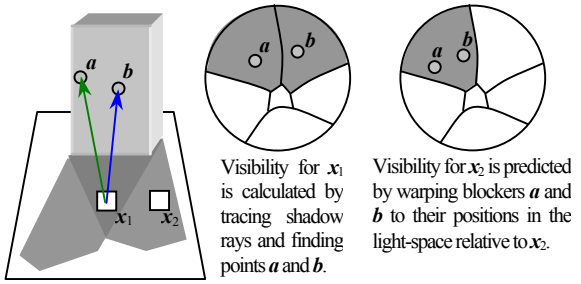


Figure 2. *The environment map for Grace Cathedral (left) is sampled into 82 lights. Each light represents a principal direction and is surrounded by the directions nearest to it, as defined by the Voronoi diagram (right). Note that the sampling is irregular, with region A being densely sampled and region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



Visibility for $\boldsymbol{x}_1$ is calculated by tracing shadow rays and finding points ***a*** and ***b***.

Visibility for $\boldsymbol{x}_2$ is predicted by warping blockers ***a*** and ***b*** to their positions in the light-space relative to $\boldsymbol{x}_2$.

Figure 3. Warping Blockers. *The visibility for $\boldsymbol{x}_2$ is predicted by reusing blocker points **a** and **b**, discovered earlier when the visibility for $\boldsymbol{x}_1$ was computed. The visibility for $\boldsymbol{x}_2$ will be used in subsequent predictions at other locations – only blocker **b** will be used since it is the closest to the actual light direction.*

the number of shadow rays cast. For small area sources, one can assume the source is localized, with a well-defined boundary. They use scanline pixel evaluation to determine visibility at regularly spaced samples of an area light source. Errors are corrected by boundary flooding in the light-space. We compare a direct adaptation of this method to environment maps and show that both the evaluation order and flooding need to be modified for accurate and efficient visibility sampling. [HDG99] uses an analytic method that also leverages the locality of area light sources. They uniformly samples the solid area subtended by a light source, relying on its locality. For full-sphere environment maps, this reduces to an exhaustive search over the whole upper hemisphere. Area light sources are also amenable to uniform partitioning and sampling. On the other hand, an environment map represents illumination from the entire sphere of incoming directions. Efficiently sampling an environment map requires highly non-uniform partitioning of the sphere of incident directions. In this paper, we present a simple method for efficient coherence-based evaluation of visibility for environment maps.

We identify two important components of an algorithm —reusing visibility calculations to predict the results of tracing shadow rays, and explicitly tracing new shadow rays in regions of uncertainty. The first component involves reusing and possibly warping geometry (in our case, reprojecting shadow ray-geometry intersection points), and is well-studied in image-based rendering (IBR). However, the reuse of sampled geometry alone does not always provide a sufficiently accurate notion of visibility. Hence, the second component is essential—determining regions or light directions prone to errors in the reconstruction. We explicitly trace rays to accurately determine visibility for these directions. Methods such as [Guo98] and [HDG99] share information between neighboring pixels. However, we have found that when many light sources exist, visibility information must be shared between neighboring lights as well.

Based on an evaluation of different alternatives, we develop effective implementations of both components. We use a coarse-to-fine, grid-based evaluation of the image pixels to reuse visibility information, which was previously calculated at 4 nearby locations. We can then reconstruct the visibility of the lights, with regions of uncertainty explicitly marked. We flood this uncertainty to nearby lights until no errors are found. By only tracing in areas of uncertainty, this technique can provide an efficiency improvement in the number of shadow rays required by an order of magnitude. Furthermore, as shown in Figure 1, the resulting images contain almost no perceptible differences from the true images.

Contributions:

- We demonstrate a computationally efficient and simple algorithm for calculating shadows in the presence of arbitrary environment maps, in Section 3.
- We provide experience with variations on the algorithm and give guidance that describes the benefits and drawbacks of possible alternatives, in Section 4.2
- We present well-documented pseudocode for our algorithm, to assist implementers, in Section 4.3.
- The code, data, and scripts to recreate all images in this paper are provided online, as well as some animations.

2 Preliminaries

The radiance at a point, $\boldsymbol{x}$, in the direction $\boldsymbol{\omega}_0$, is given by

$$L(\boldsymbol{x},\boldsymbol{\omega}_0) = \int_{\Omega_{2\pi}} f_r(\boldsymbol{\omega},\boldsymbol{\omega}_0) L(\boldsymbol{\omega}) V(\boldsymbol{x},\boldsymbol{\omega})(\boldsymbol{\omega}\cdot\boldsymbol{n}) d\boldsymbol{\omega},$$

where $\boldsymbol{x}$ is a location in the scene with a *BRDF* of f_r, $\boldsymbol{n}$ is the normal, and $\boldsymbol{\omega}$ is a direction in the upper hemisphere, $\Omega_{2\pi}$. $V(\boldsymbol{x},\boldsymbol{\omega})$ is the 4D, binary visibility function of the scene. The light-space of each location, $\boldsymbol{x}_i$, is the set of directions in the upper hemisphere, $\Omega_{2\pi}$. $V(\boldsymbol{x}_i,\boldsymbol{\omega})$ assigns to each direction in the light-space of $\boldsymbol{x}_i$, a binary value indicating whether occluding geometry exists in that direction, or not. [ARBJ03] demonstrated that the integral above can be turned into an explicit sum over a set of well-chosen directions $\{\boldsymbol{\omega}_i\}$, 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 $L(\boldsymbol{x},\boldsymbol{\omega}_0)$ is computing the visibility term, $V(\boldsymbol{x},\boldsymbol{\omega})$. We demonstrate that a sparse set of samples in $\{\boldsymbol{x}_j\}\times\{\boldsymbol{\omega}_k\}$ can be used to reconstruct $V(\boldsymbol{x}, \boldsymbol{\omega})$ with a high degree of accuracy.

Our goal will be 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 points from visibility calculations in nearby

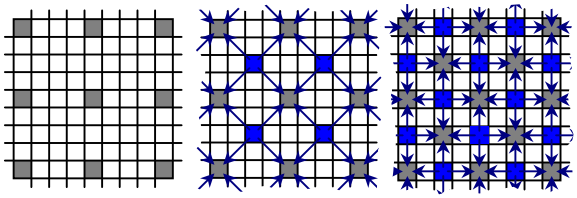


Figure 4. Grid-Based Evaluation. *First the image is fully evaluated at regular intervals. This produces a coarse grid. Next the center (shown in blue) of every 4 pixels is evaluated based on its surrounding 4 neighbors (shown in grey). All four are equidistant from their respective finer-grid-level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

locations. These points can be warped, from the direction relative to the nearby known location, to the direction relative to the new location. Figure 3 shows how the known visibility at $\boldsymbol{x}_1$ can be used to predict the visibility at $\boldsymbol{x}_2$ by using blocker points $\boldsymbol{a}$ and $\boldsymbol{b}$. This warping is very simple—requiring only a few geometric operations. In fact, we will see that in many cases, we can reuse visibility even without performing this warping.

After possibly warping the blockers, we must determine which light cell contains the new direction to each blocker. This is done by projecting the direction to the 2D Voronoi diagram as seen in Figure 2 *right*. Each location in the diagram is labeled with the light for that cell. Care is taken to ensure that the resolution is sufficient to represent each cell faithfully, but even for environment maps sampled at 400 lights with the smallest cells near the horizon, a 513x513 pixel Voronoi diagram suffices. By applying the projection into the Voronoi diagram for only point occluders, the process becomes a simple table lookup.[1]

Our reconstruction introduces sampling artifacts, such as holes, and occluded geometry. Once blockers have been warped to approximate the visibility at a new location, some cells will have a sparse reconstruction of the scene geometry represented by the blockers. The second component of our algorithm will identify these cells as uncertain and flood that uncertainty to neighboring cells. Since the lights are spaced non-uniformly, we turn to the precomputed Voronoi diagram to determine which cells are neighbors. Anywhere two neighboring locations in the diagram are labeled as belonging to different cells, that pair of light cells is identified as neighbors. 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 neighboring light cells is precomputed once, and reused for every location in the image.

3 Algorithm

In this section we will describe how we address the needs of the two main components for correctly predicting visibility by leveraging coherence in $V(\boldsymbol{x},\boldsymbol{\omega})$. After describing the possible solutions to reusing blockers (Section 3.1) and correcting prediction errors (Section 3.2), we introduce a possible optimization for each (Section 3.3). We then test possible combinations for the variations on the two framework components (Section 3.4). This allows us to discover which combinations yield an algorithm that produces few or no errors. In Section 4, we will examine the efficiencies of those candidate algorithms.

3.1 Predicting Visibility (Evaluation Order)

The main question to begin with is which nearby locations we use to identify blockers or geometry for the current location. We look at two image-based orders of evaluation–scanline and grid-based. Every pixel can be uniquely identified with the location in the scene intersected by the eye-ray that corresponds to that pixel. We will refer to scene locations and their corresponding pixels interchangeably. The blockers at a pixel are those discovered when computing the lights' visibility at the corresponding scene location.

One approach is to evaluate the image in scanline order. For each pixel $\boldsymbol{x}_{i+1}$, we use blockers from the previous pixel $\boldsymbol{x}_i$, (i.e. the values of $V(\boldsymbol{x}_i, \boldsymbol{\omega}_k)$, $1 \le k \le s$, where s is the number of lights used to illuminate the scene.) When $V(\boldsymbol{x}_i,\boldsymbol{\omega}_k)$ is blocked, we assume we also know the distance to the blocker point. We can then warp those blockers to reconstruct an approximation to the geometry seen at the current pixel, $\boldsymbol{x}_{i+1}$. If any blocker warps onto the cell corresponding to some light, m, we predict that $V(\boldsymbol{x}_{i+1}, \boldsymbol{\omega}_m)$ is blocked. If no blockers land in the cell of light m, $V(\boldsymbol{x}_{i+1},\boldsymbol{\omega}_m)$ is predicted visible. Subsequent predictions continue to use the blocker which warped nearest to the principal direction $\boldsymbol{\omega}_m$. Scanline evaluation of the image closely corresponds to the approach used by [ARH00] for small area lights. To prime the scanline evaluation, the pixels in the leftmost column of the image are fully shadow-traced. For 513x513 images, this requires fully tracing about 0.2% of image pixels.

An alternative approach is to evaluate the image in a coarse-to-fine, grid-based order as shown in Figure 4. The blockers from each of the 4 neighbors (not immediate) are warped into the local light-space for the current pixel. Cells that contain many (four or more) warped blockers are marked as blocked. Cells that contain no warped blockers are marked visible. If a cell contains few (one, two, or three) warped blockers, it is marked as uncertain, and a shadow-ray is explicitly traced to determine visibility. To prime the grid-based evaluation, the coarsest level of the grid is composed of pixels that are 16 pixels apart. For each of these pixels (comprising less than 0.5% of the image), all shadow rays are traced.

In both scanline and grid-based algorithms, the nearby locations used for prediction may correspond to different objects. Where the nearby locations are not on the same object as the current location, we have found that there is a significant loss in coherence, and we fully shadow-trace 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 7, this results in about 1-3% of the shadow rays.

[1] As presented here, only directions where $y \ge 0$ appear in the Voronoi diagram. An extension to the full sphere is trivial.

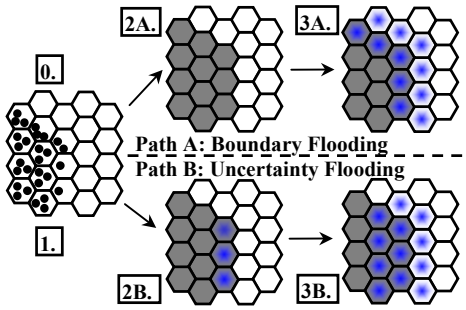


Figure 5b. Boundary Flooding vs. Uncertainty Flooding – *Boundary flooding considers any light on the boundary between blocked and visible to be a low-confidence prediction. Uncertainty flooding considers any light for which there is no consensus to be a low-confidence prediction. In both methods, the neighbors of low-confidence lights are shadow-traced. If the trace reveals that a prediction is wrong, the shadow-tracing floods to neighbors of that light; until all shadow-traces return the same visibility as the prediction. [Agrawala et.al. 2000] use this for boundary flooding.* ***Both algorithms begin the same way: 0.*** *The visibility for all of the lights has been determined at 3 pixels (not shown). We will now use that information to predict the visibility for lights at new pixel.* ***1.*** *For simplicity, we will not consider warping. Each light in our current pixel (as represented by its hexagonal cell in the Voronoi diagram), will receive a blocker (black dots) every time the corresponding light is blocked in one of the previously evaluated pixels. If warping were used, blockers would be warped onto the appropriate light, instead of always being added to the same light.* ***Boundary flooding proceeds with Path A: 2A.*** *Lights cells that contain any blocker(s) are predicted as blocked. Others are predicted as visible.* ***3A.*** *Light cells whose neighbors' visibility differs are shadow-traced (blue center).* ***Uncertainty flooding proceeds with Path B: 2B.*** *Lights cells that contain 3 blockers (since we are using 3 previous pixels) are predicted as blocked. Those that contain no blockers are predicted as visible. Light cells containing 1 or 2 blockers are marked as uncertain and shadow-traced (blue center).* ***3B.*** *The neighboring light cells of all uncertain cells are shadow-traced (blue center).*

3.2 Detecting Uncertain Predictions (Flooding)

When the blockers in an area of densely sampled lights, such as region A of Figure 2, are warped onto an area of sparse lights, such as region B of Figure 2, prediction errors will occur. Cells where the principal direction and most of the cell is visible may receive many blockers clustered in a small region of the cell, and would be incorrectly predicted as blocked. Symmetrically, if the warping occurs form a sparse region to a dense one, cells that are actually blocked may receive no blockers and be incorrectly predicted as visible.

In coarse-to-fine evaluation, many of these types of sampling errors are already identified. This is because we warp blockers from four neighbors, essentially predicting a light only if all four neighbors agree, i.e. there are 0 blockers in the cell (light is visible) or 4 blockers (light is blocked). When under or over-sampling occurs, there will often be 1-3 blockers in a light cell, and this light will already be marked as uncertain, thus triggering an explicit trace.

This uncertainty is also taken as an indication that the region around these lights is not properly sampled, and we propagate the uncertainty to neighboring lights as shown in Figure 5a *left*. For these neighboring light cells, we will verify the prediction by explicitly tracing a shadow ray, recursively tracing their neighbors if the prediction differs from the actual shadow ray. We call this procedure *uncertainty flooding* since uncertainty is spread or flooded to neighbors in light-space.

For standard scanline evaluation, the visibility at each pixel is predicted from the visibility of only the pixel to its immediate left. Therefore there is no notion of uncertainty, and we cannot use uncertainty flooding. We may modify the scanline evaluation to use the three previously evaluated pixels above and to the left of the current pixel. Under this scheme, 3 blockers would indicate a blocked light, none would indicate a visible light, and 1 or 2 blockers would indicate uncertainty. As discussed in Section 3.4 and evidenced by Figure 6, this approach does not eliminate visual artifacts. However, it is reasonable to assume that uncertainties occur at boundaries between blocked and unblocked lights. In this case, occluded geometry may rise out of these boundaries as the scan progresses. This is the approach taken by [ARH00] (where they also included the boundary of the area light source itself), and we call it *boundary flooding* (see Figure 5a *right*). Since the illumination spans the entire sphere, many visibility boundaries will exist, resulting in flooding to most of the lights. Also, in an environment map, the horizon should be treated as the edge of the light source, and therefore all lights near the horizon must always be traced for boundary flooding. The horizon of an environment map is the set of lights that neighbor lights below the horizon at a given location. When using environment maps, boundary flooding often does unnecessary work since not all visibility boundaries are uncertain, as shown in Figure 5a.

3.3 Optimizations

No Warping. Sometimes warping does not need to be performed explicitly. This is true when blockers are relatively far compared to the distance between the geometry in the pixels from which we are predicting. In these cases, warping will only move a blocker a small amount in the light-space, often landing it back in the same cell. By checking the 4 neighbors of a pixel without warping the blockers, we only shadow-trace lights for which all 4 neighbors do not agree. This is equivalent to finding an image-space shadow boundary for that light. With the help of uncertainty flooding, we can use this technique even when the blockers would warp a significant amount. One advantage of not warping is that we eliminate the dependence on accurate depth information for the blockers. This can be useful when rendering engines use optimizations in the intersection of shadow rays by stopping as soon as an intersection is found, without discovering the true depth of the first blocker.

Restricted Flooding. More than half of the shadow rays traced by our algorithms are triggered by flooding. Both boundary and uncertainty flooding are effective methods for correcting prediction errors. In general, we will see that uncertainty flooding is more efficient than

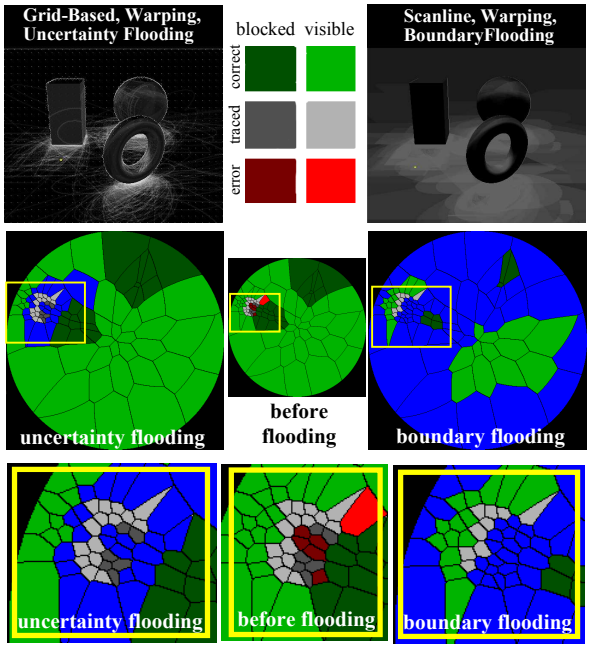


Figure 5a. ***Uncertainty Flooding vs. Boundary Flooding***
Legend: *The central image shows what the light space of a typical pixel might look like after warping blockers from nearby pixels. The dark regions of each color (green, gray, red) represent blocked lights. For boundary flooding, all that matters are the boundaries between a light region of any color and a dark region of any color. Gray regions had few blockers warp into them, and are marked as uncertain. Green and red regions were predicted correctly, or in error, respectively. Blue represents regions that were traced during flooding.*
Top: *Images showing the number of shadow rays traced at each pixel for the shapes scene illuminated by 50 lights for Grid uncertainty flooding and scanline boundary flooding. As can be seen by the brighter intensities on the right, boundary flooding does more work than uncertainty flooding.*
Middle: *The light-space of a representative pixel is examined before flooding (center). The light-space is then seen after uncertainty flooding (left), and after boundary flooding (right). Both methods eliminate the errors (red). Lights traced due to flooding are shown in blue.*
Bottom: *Closeups showing areas of uncertainty (gray). These uncertainties arise due to the common case of coarsely sampled blockers warping to densely sampled lights. Uncertainty flooding (left) only traces near this region. Boundary flooding (right) traces this region along with other visibility boundaries and the horizon, doing significantly more work. (For boundary flooding, we may assume the gray cells were predicted correctly).*

boundary flooding. However, the best solution is obtained by flooding at the intersection of uncertainty flooding regions and boundary flooding regions. The areas marked as uncertain are traced just as before. Instead of flooding the uncertainty to all of the neighbors, it is flooded only to those whose prediction differs from the result of the shadow ray. This method traces about 10-20% less shadow rays as compared to strict uncertainty flooding.

In grid-based evaluation, half of the pixels are contained in the finest grid—when pixels are predicted from their immediate neighbors. The image cannot resolve discontinuities in the shadows smaller than a single pixel. Therefore, if all 4 neighbors agree on the visibility of a particular light, the center pixel can only disagree if the shadow discontinuity is smaller than a single pixel. If we ignore sub-pixel shadow discontinuities, we can turn off flooding at the finest grid level, relying entirely on explicitly marked uncertainty to correct any errors. Since half the shadow rays are a result of flooding, this results in about 25% reduction in the number of shadow rays traced.

3.4 Combining the Components

The two main components of *Evaluation Order* and *Uncertainty Detection and Flooding* must be chosen to match each other. The goal is to create an evaluation order that minimizes uncertainty, while at the same time, making areas of uncertainty easy to accurately find and flood. To determine the most effective algorithms, we tested various possible combinations of the components for four scenes, using several light and image resolutions. In particular, we made a comprehensive evaluation of 9 algorithms, which we believe can offer many insights for future work in coherence-based sampling. These included all 8 possible choices of scanline vs grid-based evaluation order, with and without warping, and with boundary or uncertainty flooding. In addition, we included the optimized algorithm (OPT) described below, and selected based on our evaluation of these results. Our first goal was to determine the types of errors exhibited by the various possible approaches to exploit coherence. A representative example of the data we collected is shown in Figure 6 (this is a closeup of the shapes scene shown in Figure 7).

We first focus on scanline evaluation order. If no warping is performed, regardless of the flooding method, incorrect visibility information appears to propagate far further along the scan than is accurate. Hence, there can be staircasing artifacts, and the shadow of the torus is incorrectly filled in, as shown in Figure 6 *top-left*. Even with warping, uncertainty flooding from the three pixels immediately to the left and above does not suffice, although it performs somewhat better. The pixels are too close together to have very different notions of the scene geometry. Hence, uncertainty flooding cannot discover most sampling artifacts, and we still get staircasing artifacts. Thus, we must use warping and boundary flooding in scanline algorithms. It is interesting to note that this corresponds closely to the approach of [ARH00]. Note that flooding along all possible boundaries also does a lot of work that does not correct any errors, as will be seen in the performance numbers of the next section. We next consider the grid-based algorithms. Here, all four possibilities give almost error-free results. The grid-based evaluation combines differing data about visibility at the 4 neighbors to produce a more accurate reconstruction at the pixel of interest. This reconstruction is an accurate starting point for either flooding method, even if we optimize by not warping.

Having discarded the possibility of using the 3 scanline algorithms outlined in red in Figure 6, the final choice is determined by the performance of each of the remaining combinations, as discussed in the next section. The errors in all these algorithms are generally small (<0.1%) either when viewed as absolute mispredictions, or when viewed as the resulting color shift in the image. In some cases, one can see very subtle artifacts upon close examination, which

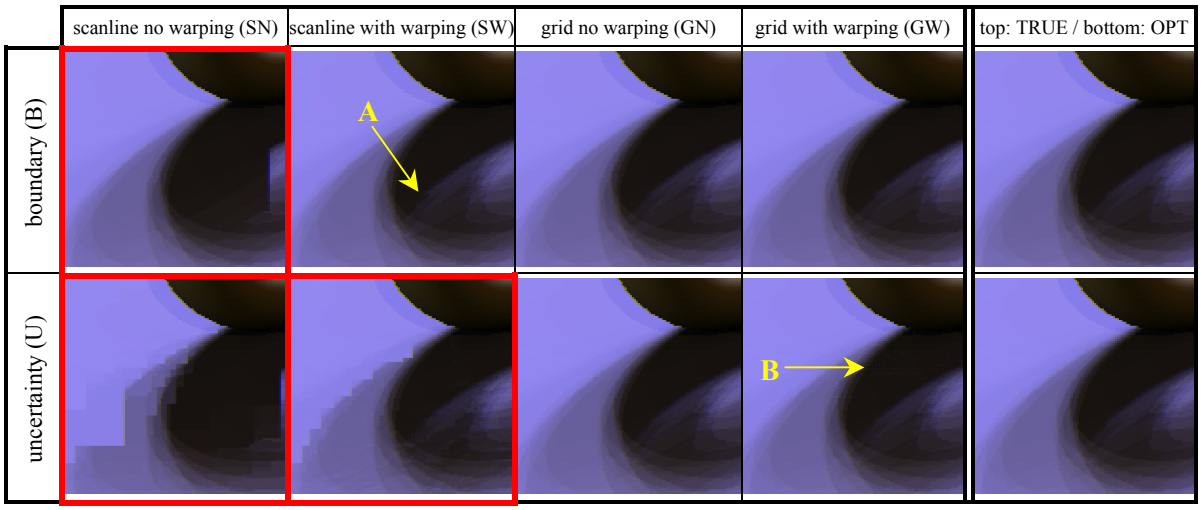


Figure 6. ***Error Analysis.*** *The above table shows possible combinations for the two components of our framework.* ***Scanline no warping*** *produces significant errors with either flooding method, as seen in the red squares on the left. This is an indication that the initial reconstruction of visibility is too erroneous to be corrected.* ***Scanline with warping*** *requires boundary flooding. Otherwise, a staircase pattern occurs because uncertainty flooding is ill-suited for scanline evaluation.* ***Grid no warping*** *works well with both uncertainty flooding and boundary flooding.* ***Grid with warping*** *also produces accurate images. 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 3.4, is shown in the bottom-right. No perceptible errors exist. We also show two subtle artifacts that may arise. The reader is encouraged to zoom into the images to see the regions marked by the arrows. Arrow* ***A*** *shows a difference on the inner edge of the dark shadow. This artifact disappears when a higher sampling of the environment is used. Some lightening can be seen near arrow* ***B*** *where a single strong light was missed by the warping because it subtends a very small solid angle. The overall result in these cases is still faithful to the shadowing that occurs.*

are pointed to in Figure 6. For instance, in the scanline method, there is a minor difference in the inner edge of the dark shadow pointed to by the arrow A. Similarly, in the grid-based method with warping and uncertainty flooding, there is a subtle lightening of a few pixels in the shadow pointed to by the arrow B, due to mispredicting visibility for a single bright light source.

Finally, based on these results, we can pick an optimized algorithm (OPT). The next section will discuss performance, leading to an algorithm that is grid-based, with no warping, and uncertainty flooding. Additionally, we use the flooding restriction described in the previous subsection. This method was selected as the optimized algorithm for its simplicity as well as excellent accuracy and performance.

4 Implementation and Results

We compare the efficiency for components of our framework by examining their performance on several scenes, at varying sampling resolutions of the environment.

4.1 The Scenes

We analyze the results of running the low-error methods on the 4 scenes shown in Figures 1, 7 and 8. All tests were run on 513x513 images. The scenes contain only diffuse, non-textured objects to maximize the visibility of cast shadows. Shadows have been enhanced by increasing the strength of small, powerful light sources. (The images in this pdf are embedded at full resolution, without compression. The reader is invited to zoom in on any image to see details.) The shapes scene shows simple objects with both smoothly varying and discontinuous curvature, with curved shadows cast on the sphere. 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 plant scene includes detailed geometry that casts very intricate shadows with low coherence. Each scene uses a different environment map. The plant's environment is relatively uniform. The shapes environment contains several tight clusters of bright lights. The bunny and dominos environments contain small lights near the

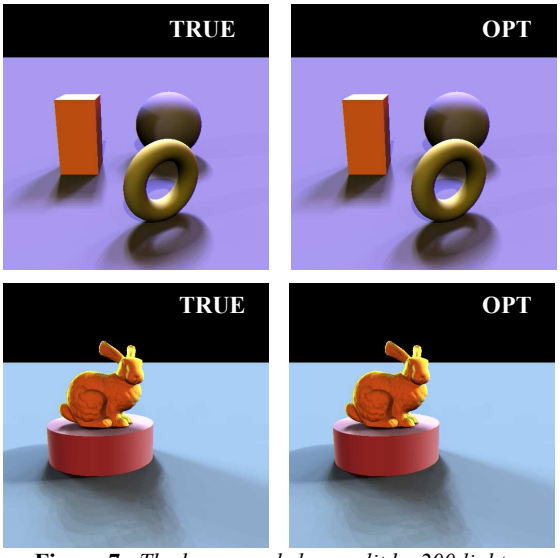


Figure 7. *The bunny and shapes, lit by 200 lights.*

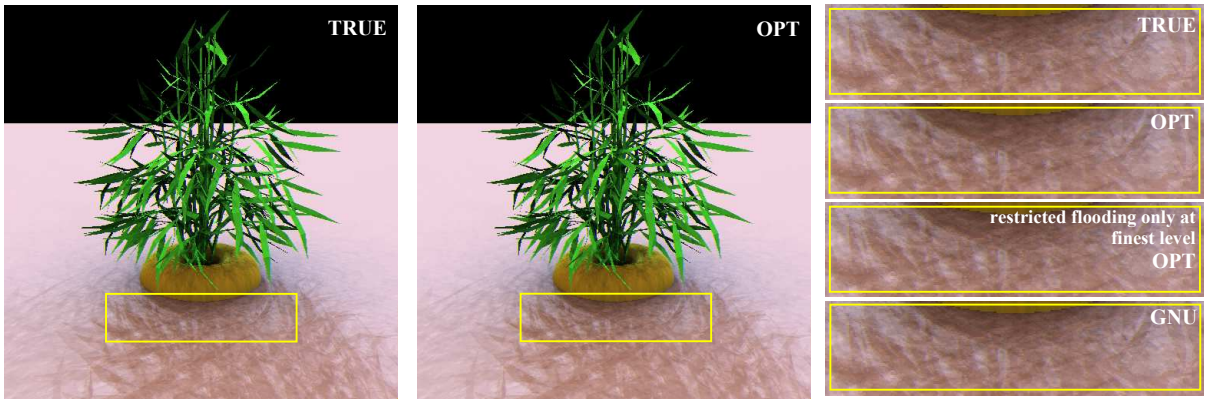


Figure 8. ***Plant.*** *The plant image, seen here as illuminated by a 100 light approximation of the Kitchen environment, has very detailed shadows with low coherence. Nevertheless, our algorithms produce a very accurate image, as seen in the direct comparisons and closeups. Even on very close examination it is difficult to observe any differences.*

horizon, as well as non-uniform lights near the pole. The lighting environments are high-dynamic range light probes courtesy of Paul Debevec (www.debevec.org).

In Tables 1 and 2, we compare the performance of six algorithms—scanline evaluation with warping and boundary flooding (SWB), the 4 possible grid-based (G) algorithms: with warping (W) and no warping (N), as well as boundary (B) and uncertainty (U) flooding, and our optimized algorithm (OPT). In Table 1 we report the percentage of shadow-rays traced for all six methods, at different numbers of lights (50, 100, 200, 400), on the 4 scenes. In Table 2, we report the actual running times for rendering each scene with 200 lights. We also show the speedup relative to fully shadow-tracing the image (TRUE).

We do not report numerical error rates, since we have already eliminated error-prone algorithms from consideration, as shown in Figure 6. In the remaining algorithms, the errors are usually very low for all methods, with mispredictions being less than 0.1% of the number of shadow rays in most cases. For instance, the error rate for OPT on the bunny is 0.0026%, meaning only 1 in 38,000 shadow rays disagree from that obtained using standard ray tracing. Even on the very complicated plant scene, OPT has only a 0.4% error rate (comparable with SWB). In terms of visual quality of the images, we see in the comparisons and closeups of Figure 8 that OPT produces accurate results even on the plant scene. As visualized in the final images, these errors produce renderings that are nearly identical to the true images when viewed as standard 24 bit RGB images.

4.2 Comparing Algorithm Components

By examining the reduction in shadow rays for the various algorithms in Table 1, we can determine what each component contributes to the efficiency gain.

Scanline vs. Grid. We see that GWB outperforms SWB by about a factor of 2 in most cases. Since scanline must consider the lights near the horizon as part of the boundary, it requires more work than grid-based evaluation. New blockers that rise over the horizon are missed without horizon flooding in SWB, causing bright streaks to appear across the image as the scan progresses. On the other hand, grid-based evaluation uses four neighbors, and therefore already has information about blockers that might come out of the horizon. Therefore, no separate flooding on the horizon is required.

Boundary Flooding vs. Uncertainty Flooding. For grid-based evaluation order, we have the option of using uncertainty flooding. Notice that in all cases, uncertainty flooding performs about twice as well as the corresponding boundary flooding algorithm at the lower light resolutions. In particular, the first row of Table 1 (shapes50), shows GWB doing 22% of the work and GWU doing only 10%. Similarly, GNB does 22% of the work, while GNU does only 9%. This indicates that marking all boundaries as regions of uncertainty is an overly-conservative estimate, much as marking the entire horizon for scanline as uncertain was overly-conservative. By explicitly marking regions of uncertainty with the measure of sparse blockers for grid-based evaluation, work is concentrated only in those regions that actually need it. 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 improve by a factor of 2 as the light resolution is increased from 50 to 400. On the other hand, the performance improvement of methods based on uncertainty flooding is relatively independent of the number of lights.

It is instructive to compare GWU and SWB in terms of the work done, to illustrate these differences. First, we note from Table 1 that the grid-based GWU outperforms the scanline SWB by a factor of 2 at higher light resolutions, and often even more at lower resolutions. Figure 5a *top* shows a comparison of the work done on the simple shapes scene. Both methods concentrate work in complex shadowed regions and image shadow boundaries. However, SWB does considerable work in other parts of the scene as well, because boundary flooding is inefficient compared to uncertainty flooding, and because it must always trace the boundary of the visible hemisphere.

Warping vs. No Warping. As indicated in Figure 6, not warping is only a valid option for grid-based evaluation. By not warping, we are assuming coherence in the scene on the order comparable to the grid size being used. Without flooding to correct errors, not warping

	shapes				bunny				plant				dominos			
number of lights:	50	100	200	400	50	100	200	400	50	100	200	400	50	100	200	400
true (TRUE)	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
scan, warp, boundary (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
grid, warp, boundary (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
grid, no warp, boundary (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
grid, warp, uncertainty (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
grid, no warp, uncertainty (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
optimized algorithm (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•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. Notice that the performance of boundary flooding depends on the number of lights.*

assumes that if four pixels lie in the shadow of a light, all of the pixels between them lie in shadow of the same light. Therefore, not warping only performs better than warping when the scene exhibits such coherence. Otherwise, flooding has to correct many errors in the prediction. When comparing GWU with GNU, Table 1 shows that both algorithms perform at the same order of magnitude for all scenes. GNU performs slightly better for shapes, bunny, and dominos—those scenes that contain considerable coherence. In the plant scene, the coherence assumed for not warping is lacking. Therefore GWU performs slightly better than GNU in the plant scene.

Optimized Algorithm. Our optimized algorithm OPT is based on GNU because GNU performs better than other algorithms in most cases, while maintaining negligible error rates, and being extremely simple to implement. We optimize GNU by also adding the restricted flooding optimization, flooding only at the intersection of boundary flooding and uncertainty flooding regions, and by not flooding at the finest level. Flooding only at the intersection introduces artifacts for complex shadows. We feel that the performance gain is worth the slight difference in image quality. If quality is more important that performance, full uncertainty flooding (except at the finest level where there is no flooding) can be used at an increase of about 10-15% in work done, relative to OPT.

Performance. All of our algorithms enable significant reductions in the number of shadow rays traced, with negligible loss in quality. In particular, the optimized algorithm shows a reduction by a factor of 20 on the simpler shapes and bunny scenes, and an order of magnitude on dominos. Even on the plant scene, that has very intricate shadows with significantly less visibility coherence, we reduce the number of shadow rays traced by a factor of 4. Furthermore, the results are very accurate for the algorithms discussed here, even in the plant scene.

4.3 Implementation

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. When the 4 pixels do not agree on a particular light, it is marked as uncertain. All uncertain lights and their neighbors are shadow-traced. If the optimizations are used, only neighbors whose prediction disagrees with the shadow-trace are traced. No flooding to neighbors is performed at the final grid level.

Code for GNU is shown in the following pseudocode:

```
    procedure Calculate-All-Visibilities()
1     for (gridsize←16; gridsize ≥ 1; girdsize /= 2) do
2       foreach pixel, p, in current grid do
3         lights-to-trace ← Ø
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-env-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-env-map do
              vis(p, L) ← Trace-Shadow-Ray(p, L);
            endforeach
          endif
        endforeach
      endfor
```

The procedure Calculate-All-Visibilities is called to determine the visibility of every light at every pixel. After Calculate-All-Visibilities is complete, the BRDF for each object can be used to sum up the contributions of the visible lights. Line 1 refers to the coarse-to-fine grid as described in Figure 4. Line 2 indicates that, given a grid size, we render each pixel in that grid in order. See the code available online for details of the grid-based evaluation order for the pixels. At Line 3, we initialize the set, ***lights-to-trace***, to the empty set. Lights will eventually be added to this set in Line 12, and in Flood-Uncertainty. At the beginning of each pixel evaluation, the set is empty. Line 4 assigns to ***n***[], the four pixels that surround ***p*** from the coarser grid as shown in Figure 4. Line 5 assigns the object ID of the first object hit by a primary ray that is

	shapes				bunny				plant				dominos			
number of lights:	50	100	200	400	50	100	200	400	50	100	200	400	50	100	200	400
true (TRUE)	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
scan, warp, boundary (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
grid, warp, boundary (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
grid, no warp, boundary (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
grid, warp, uncertainty (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
grid, no warp, uncertainty (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
optimized algorithm (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•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. Notice that the performance of boundary flooding depends on the number of lights.*

traced from the camera, through pixel ***p***. The if statement in Line 6 checks to see that the four neighbors from the coarser grid correspond to primary rays that intersect the same object as the primary ray through ***p***. Whenever this is not the case, we are near an object boundary and the if statement fails. In that case, control is passed to Line 13, where each shadow ray is explicitly traced to determine visibility. When all object IDs match, the foreach statement in Line 7 cycles through all of the directional lights generated by sampling the environment map through some method such as [ARBJ03]. Line 8 checks to see if all four neighbors from the coarser grid agree about a particular light's visibility. If they agree, the visibility for that light is predicted to be that same at the current pixel, ***p***. This prediction is assigned in Line 9. If the four neighbors do not agree than we are near a shadow edge for this particular light. In that case, Line 10 marks the visibility of this light at the current pixel as 'uncertain'. This light is added to the set of ***lights-to-trace*** in Line 11. After all the visibility of all lights at ***p*** have been assigned a prediction ('visible' or 'blocked'), or have been deemed 'uncertain', we call Flood-Uncertainty(). The pseudocode for Flood-Uncertainty() is explained below:

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

When Flood-Uncertainty is called in line 12 of Calculate-All-Visibilities, 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. Line 2 shows that the resulting visibility ('visible' or 'blocked') is stored in a temporary variable, ***tempVis***. For the initial lights placed in ***lights-to-trace***, the if statement in Line 3 will always succeeds, since vis(***p, L***) is equal to 'uncertain'. When this happens, Line 4 assigns the correct visibility to the current light. The foreach loop in Lines 5 and 6 also add all of ***L***'s neighbors to the set ***lights-to-trace***. The Voronoi diagram was used earlier to determine which lights are the neighbors of any given light. The newly added neighbors may fall into one of 3 types: 1. Already in the set; in which case the assignment in Line 6 has no effect. 2. Not yet in the set, but already traced; in which case the intersection in Line 1 is unaffected. 3. Not yet in the set, and not yet traced; in which case this newly added neighbor is a candidate for the next iteration of the while loop in Line 1. Neighbors continue to be added until either all lights have been shadow traced (rare), or the if statement in Line 3 fails for all lights in the intersection of Line 1. This second condition indicates that all shadow rays agree with the prediction made in Line 9 of Calculate-All-Visibilities, and the flooding can terminate. Notice that Line 1 must be implemented in a way that the same light does not get picked twice from within the intersection.

4.4 Time and Memory Considerations

Timings. Table 2 shows the running times for the algorithms in Table 1, and the corresponding speedup relative to TRUE. It also includes the (relatively small) time for the initial setup of the raytracer and tracing just the EYE rays or primary rays for each pixel. The speedup shown is for the wall clock running time for the full tracing of the image, including tracing the primary rays and performing the diffuse shading calculation, which we do not attempt to optimize. As the table is read from top to bottom, the algorithms perform better. Towards the bottom (for GNU and OPT), the time to trace primary rays becomes more relevant when evaluating the speedup of the algorithm. For the shapes scene, OPT is able to reduce the time spent on casting shadows below the time spent on tracing the primary rays (the total time is the sum of the two). In this case, we are adding complex image-based lighting for the effective cost of primary ray-tracing the scene once. Even on the complicated plant scene with low coherence, OPT provides a timing speedup by a factor of almost 3.

While the improvements in running time 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 by OPT in tracing shadow rays with 400 lights. We compare actual and theoretical speedups in rows 3 and 4. We can also measure the average time to trace a shadow ray for TRUE and OPT (rows 6 and 7). Since both methods use the same

	shapes		bunny		plant		domino	
	sec	speedup	sec	speedup	sec	speedup	sec	speedup
EYE	4.0	N/A	4.0	N/A	4.4	N/A	4.2	N/A
TRUE	75.3	1.0	80.8	1.0	152.3	1.0	124.9	1.0
SWB	34.9	2.2	54.8	1.5	120.3	1.3	67.1	1.9
GWB	25.2	3.0	36.7	2.2	113.2	1.3	55.7	2.2
GWU	19.6	3.8	30.6	2.6	109.0	1.3	47.1	2.6
GNB	16.8	4.5	23.2	3.5	95.1	1.6	35.4	3.5
GNU	11.2	6.7	16.7	4.8	94.7	1.6	23.8	5.2
OPT	7.1	10.5	10.7	7.6	58.3	2.6	14.7	8.5

Table 2. ***Timings****. The actual timings for the full rendering, shown here, are related to, but not equivalent to the reduction in work as measured in shadow-rays traced. Each scene was lit by a 200-light sampling, and timed on an Intel Xeon 3.06GHz computer, running Windows XP.*

unmodified POV-Ray ray-tracing engine, this is a very controlled experiment. A perhaps surprising result is that all shadow rays are not 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.

A comparison of rows 5 and 8 in Table 3 shows that the actual computational overhead of our algorithm is negligible. The decrease in speedup relative to the theoretical predictions are almost entirely explained by the increased average difficulty of tracing shadow rays for the bunny and plant scenes. 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.

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 513x513 image, memory usage can be reduced to 5MB. If we were to use independent blocks of 64x64, as discussed above, the memory consumption can be reduced to a very insignificant 100KB.

	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.*** *This table shows that the perceived loss in speedup for timing measurements is a result of increased average times to trace a shadow ray. Row 1 shows the time for only the shadow calculations when the image is fully traced. Row 2 shows the time for only the shadow calculations when using OPT. This includes making predictions and tracing shadow rays. Row 3 shows the speedup in the shadow calculations for OPT relative to TRUE. Row 4 shows the reduction in the number of shadow rays that are actually traced. Note that for bunny the time reduction is less than the theoretical reduction based solely on the reduction in shadow rays traced. Row 5 shows the ratio between the two speedups, highlighting the perceived loss in efficiency for bunny and plant. Rows 6 and 7 show the average time for just the operation of tracing a shadow ray. This was calculated by generating a list of all shadow rays traced by each algorithm, and then retracing those rays without any other operations being performed. Row 8 shows the ratio between the average time for tracing a shadow ray in TRUE and the average time for tracing a shadow ray in OPT. Note that for bunny and plant, OPT traces only 'difficult' shadow rays, as explained in Section 4.3. Observe that the two highlighted rows show similar ratios. This confirms that the perceived reduction in efficiency for tracing the bunny and plant scenes is a result of culling away only the 'easy' shadow rays.*

5 Conclusions and Future Work

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 uncertainty. IBR uses the techniques of warping and splatting to recreate geometry. 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 will guarantee that even scenes with weak coherence, such as the plant, will be rendered accurately, though at a lower prediction rate. Our method can typically reduce the number of shadow tests needed by an order of magnitude, with essentially no loss in quality, thus allowing scenes to be efficiently traced under environment map illumination. Due to the artifacts that may occur with OPT, we recommend GNU for rendering animations. Specifically, OPT's optimization of flooding only at intersections of boundaries and uncertainty may not correct error predictions for lights near the horizon. We rendered animation 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 from the TRUE animation.

This paper 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 straightforward 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 mostly mispredicted due to blockers falling on neighbors, but not the light itself. Adding boundary flooding for only these high-energy lights will ensure higher fidelity in the final image. The method presented in this paper also can be useful for speeding up precomputation in methods such as [SHHS03] and [NRH03].

Higher speedups than presented in this paper may be achieved if some tolerance for errors exists. We plan to explore a controlled process for the tradeoff between predictions and errors in future work. More broadly, we want to explore coherence for sampling other high dimensional functions such as the *BRDF*, and more generally in global illumination, especially animations.

6 References

[ARBJ03] AGARWAL, S., RAMAMOORTHI, R., BELONGIE, S., AND JENSEN, H. W.: Structured Importance Sampling of Environment Maps. In *Proceedings of SIGGRAPH 2003*, pages 605–612.

[ARH00] AGRAWALA, M., RAMAMOORTHI, R., HEIRICH, A., AND MOLL, L.: Efficient Image-Based Methods for Rendering Soft Shadows. In *Proceedings of SIGGRAPH 2000*, pages 375–384.

[BlNe76] BLINN, J.F., NEWELL, M.E.: Texture and Reflection in Computer Generated Images. *Communications of ACM 19(10)*: 542-547 (1976)

[BWG03] BALA, K., WALTER, B., GREENBERG, D.: Combining Edges and Points for Interactive High-Quality Rendering. In *Proceedings of SIGGRAPH 2003*, pages 631-640.

[Deb98] DEBEVEC, P.: Rendering synthetic objects into real scenes, *Proceedings of SIGGRAPH 1998*, p.189-198, July 1998

[Gre86] GREENE, N.: Environment Mapping and Other Applications of World Projections, *IEEE Computer Graphics & Applications* Vol. 6, No. 11, 1986, pages 21-29.

[Guo98] GUO, B.: Progressive radiance evaluation using directional coherence maps. In *Proceedings of SIGGRAPH* 1998, pages 255-266.

[HDG99] HART, D., DUTRE, P., GREENBERG, D.: Direct illumination with lazy visibility evaluation. In *Proceedings of SIGGRAPH 1999*, pages 147-154.

[KoKe03] KOLLIG, T., KELLER, A.: Efficient Illumination by High Dynamic Range Images. *Proceedings of Eurographics Rendering Workshop*, 2003, pp.45-50

[MiHo84] MILLER, G., HOFFMAN, C.: Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments, *Proceedings of SIGGRAPH 1984 Advanced Computer Graphics Animation seminar notes*.

[NRH03] NG, R., RAMAMOORTHI R., HANRAHAN, P.: All-Frequency Shadows Using Non-Linear Wavelet Lighting Approximation. *ACM Transactions on Graphics 22, 3*, pp. 379-381

[SHHS03] SLOAN, P., HALL, J., HART, J., SNYDER, J.: Clustered Principal Components for Precomputed Radiance Transfer. *ACM Transactions on Graphics 22, 3*, pp.382-391