Progressive Radiance Evaluation Using Directional Coherence Maps

Baining Guo
Intel Corporation*

Abstract

We develop a progressive refinement algorithm that generates an approximate image quickly, then gradually refines it towards the final result. Our algorithm can reconstruct a high-quality image after evaluating only a small percentage of the pixels. For a typical scene, evaluating only 6% of the pixels yields an approximate image that is visually hard to distinguish from an image with all the pixels evaluated. At this low sampling rate, previous techniques such as adaptive stochastic sampling suffer from artifacts including heavily jagged edges, missing object parts, and missing high-frequency details.

A key ingredient of our algorithm is the directional coherence map (DCM), a new technique for handling general radiance discontinuities in a progressive ray tracing framework. Essentially an encoding of the directional coherence in image space, the DCM performs well on discontinuities that are usually considered extremely difficult, e.g. those involving non-polygonal geometry or caused by secondary light sources. Incorporating the DCM into a ray tracing system incurs only a negligible amount of additional computation. More importantly, the DCM uses little memory and thus it preserves the strengths of ray tracing systems in dealing with complex scenes.

We have implemented our algorithm on top of RADIANCE. Our enhanced system can produce high-quality images significantly faster than RADIANCE – sometimes by orders of magnitude. Moreover, when the baseline system becomes less effective as its Monte Carlo components are challenged by difficult lighting configurations, our system will still produce high quality images by redistributing computation to the small percentage of pixels as dictated by the DCM.

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

Keywords: Progressive refinement, image-space discontinuities, directional coherence, radiance evaluation, rendering

1 Introduction

Despite the rapidly increasing power of computers, global illumination is far from being a real-time process. Accurate radiance evaluations often require hours of computation for complex scenes. To balance rendering speed and visual realism, global illumination algorithms often take a "progressive refinement" approach. In the radiosity framework, Cohen [8] extended earlier work by Bergman [2] and developed a progressive refinement algorithm that produces successive approximations, refining continuously towards the final radiosity solution.

As an acceleration strategy, the idea of progressive refinement is readily applicable to radiance evaluation, which must account for all major modes of light transport. One approach to progressive radiance evaluation is to combine progressive radiosity with ray tracing in a multi-pass system (e.g. [6, 27]). Alternatively, we can use a progressive ray tracing algorithm based on Monte Carlo light transport [16]. Compared to the multi-pass approach, progressive ray tracing has several important advantages in real-world applications. In particular, ray tracing uses much less memory than radiosity, while placing fewer restrictions on surface geometry and reflectance models.

This paper develops a radiance evaluation algorithm using progressive ray tracing. Based on the hierarchical integration technique by Kajiya [16], Painter has proposed an adaptive sampling method for progressively refining a ray traced image [23] (see also [37]). One of his important contributions was to recognize that in a progressive ray tracing system, different sampling strategies apply to the task of locating image features and the task of increasing pixel confidence [23].[1] Our main goal is to capture image features early and generate high-quality images as quickly as possible. A fundamental obstacle facing adaptive sampling techniques (including the edge following methods for anti-aliasing [3, 12, 38]) is that these techniques cannot produce high-quality images before densely sampling all the high-frequency details. We overcome this obstacle with knowledge about discontinuities.

Researchers have been aware of the importance of discontinuities for decades, and investigations in this area have led to algorithms for shadows (e.g. [10, 7, 5, 30, 32, 28, 11]) and discontinuity meshing (e.g. [14, 20]). The discontinuities computed by these object-space algorithms may be projected and inserted into an image-plane discontinuity mesh (IPDM), as was proposed by Pighin [24]. The IPDM produces dramatically better shadows from early on, but there are problems, including difficulties in handling non-polygonal geometry and discontinuities caused by secondary sources, as well as the substantial space requirements for discontinuity computations with a complex scene. A big source of inefficiency in the existing discontinuity algorithms is that they try to locate *potential* discontinuities instead of the actual discontinuities. In addition, these object-space algorithms cannot deal with the view-dependent specular components of radiance discontinuities.

A key ingredient of our algorithm is the *directional coherence map* (DCM), a new technique for handling general radiance discontinuities in a progressive ray tracing framework. The DCM includes

*Microcomputer Research Labs, RN6-35, 2200 Mission College Blvd, Santa Clara, CA 95052, email: baining_guo@ccm.sc.intel.com

[1] The task of increasing pixel confidence has attracted extensive research in the context of anti-aliasing [13]. We do not develop new anti-aliasing techniques; instead we construct our progressive renderer on top of a baseline ray tracer such that the pixel confidences of our system depend on the sampling strategy of the baseline system. By building our progressive renderer this way, we know in advance that we will capture the same set of features as the baseline system does.

two main components: an adaptive partition of the image plane into square blocks, such that each block is simple enough to have at most one discontinuity edge, and an estimation of the orientation of the discontinuity edge in each block. The DCM helps us to capture radiance discontinuities through a finite element approximation to the radiance function, with the finite elements on each block oriented in accordance with the orientation of the discontinuity within that block.

In facing the challenge of treating general radiance discontinuities, our overall strategy is to combine object-space and image-space data. Specifically, we efficiently obtain object boundaries in the image using the Z-buffer hardware [36]. By doing so we benefit from the *a priori* knowledge of the scene. We extract other discontinuities from densely evaluated pixels on the block boundaries in the DCM. Extracting discontinuity information from radiance samples is a unique feature of our algorithm. This feature not only allows us to treat several types of discontinuities ignored by previous algorithms, but also saves us time by focusing on the *actual* discontinuities.

We have implemented our progressive rendering algorithm using RADIANCE [33] as the baseline system. To render images of comparable quality, our system typically takes $1/4$ to $1/15$ of the time needed by RADIANCE. When their Monte Carlo components are challenged by difficult lighting configurations, RADIANCE and other ray tracing systems for global illumination [27] will become less effective. In this situation, our system can still produce high quality images by reallocating computational resources to increase the accuracy of the small percentage of pixels needed by the DCM. Fig. 12 demonstrates this capability. The DCM performs well on a variety of discontinuities, including those that can be handled by existing discontinuity algorithms (see, e.g., Fig. 10 with fine shadows cast by polygonal occluders) and those that cannot (see, e.g., Fig. 13 with closely packed surfaces which are both curved and specular). Finally, compared to Painter's successful system based on progressive adaptive sampling [23], our system generates far better images for the same amount of computation time, as Fig. 2 (c) and Fig. 4 show.

The remainder of the paper is organized as follows. In Section 2, we give a high level overview of the progressive rendering pipeline in our system. Section 3 describes the treatment of a block with simple discontinuities, which serves as the foundation of the DCM and is based on the least discrepancy direction and oriented finite elements. Section 4 discusses the initialization and refinements of the DCM, detailing steps for partitioning the image plane into blocks and tests for determining if a block has only simple discontinuities. Section 5 provides the details of our implementation. Results are presented in Section 6, followed in Section 7 by conclusions and suggestions for future work.

2 System Overview

Our progressive rendering system relies on a baseline ray tracing system for pixel radiance evaluations. This type of ray tracing system was first proposed by Kajiya [16], building on earlier work by Cook [9] and Whitted [35]. For simplicity, we assume the baseline system generates an anti-aliased image by filtering an enlarged *work image* of super-sample resolution (this assumption can be relaxed to allow baseline ray tracers that collect super-samples for individual pixels of the output image). Our progressive rendering system augments the baseline system with a component responsible for deciding where to sample and how to reconstruct an approximate image on user demand.

The rendering pipeline of our system, shown in Fig. 1, has two main stages. The first is the regular subdivision stage, in which we use a quadtree to partition the image plane into small blocks. We refer to these blocks as elementary blocks. To perform the regular

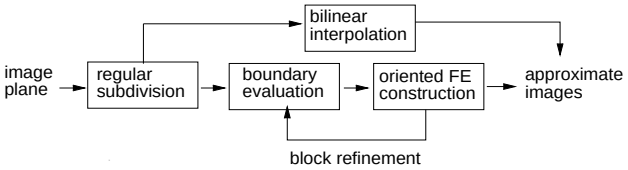

Figure 1: The rendering pipeline of our system. The approximate image is available any time, on demand. When all blocks are of the pixel size, the iterative block refinement terminates with the approximate image output as the final image.

subdivision, we start with the entire image plane as the root block and recursively subdivide each block in four until the current block has become an elementary one. During the regular subdivision, the four corners of each block are sampled, and an approximate image may be created for display at any time by interpolating the corner values. Fig. 2 (a) shows an example image at the end of the regular subdivision stage.

The second stage is an iterative process in which we begin constructing and refining the DCM. In each iteration, we select a subset of blocks as edge blocks and analyze them for discontinuities; blocks not selected simply go through another step of regular subdivision. On each edge block, we densely sample the block boundary (not just the four corners) and subdivide the block into four quads for the next iteration. From the evaluated boundary pixels and some additional object-space data, we infer the discontinuities on each edge block and record the information into the DCM. With this information, an oriented finite element approximation is constructed on the block. The oriented finite elements on edge blocks and the bilinear interpolants on the other blocks may be resampled into an approximate image at the user's request. Figs. 2 (b) and (c) are images from the second stage. Fig. 2 (d) is the final image.

3 Blocks with Simple Discontinuities

Taking a divide-and-conquer approach, the DCM treats discontinuities by partitioning the image plane into small blocks so that most blocks are crossed by no more than one discontinuity edge. Moreover, the edge is expected to have small curvature like the example in Fig. 3 (a) as opposed to the corner in Fig. 3 (b). Since the effectiveness of the DCM depends on its performance on blocks with simple discontinuities, we wish to capture such discontinuities using a small number of samples. Traditionally, adaptive sampling techniques [22, 23, 13] are used to reduce the amount of sampling needed to capture features or discontinuities. Adaptive sampling is effective in that it significantly reduces sampling in areas away from discontinuities. However, adaptive sampling is not completely satisfactory for us because it does not produce good images unless the discontinuity areas have been densely sampled. In this section, we explore an alternative based on least discrepancy directions and oriented finite elements.

3.1 Discontinuity Characteristics

A simple way to capture a discontinuity edge within a block is to build a mathematical model for the edge. Since the block is small, the edge can be regarded as straight, and we can model its behavior by locating the endpoints on the block boundary. This is essentially the approach we take, even though the basic idea is modified in several ways to accommodate the special properties of image data.

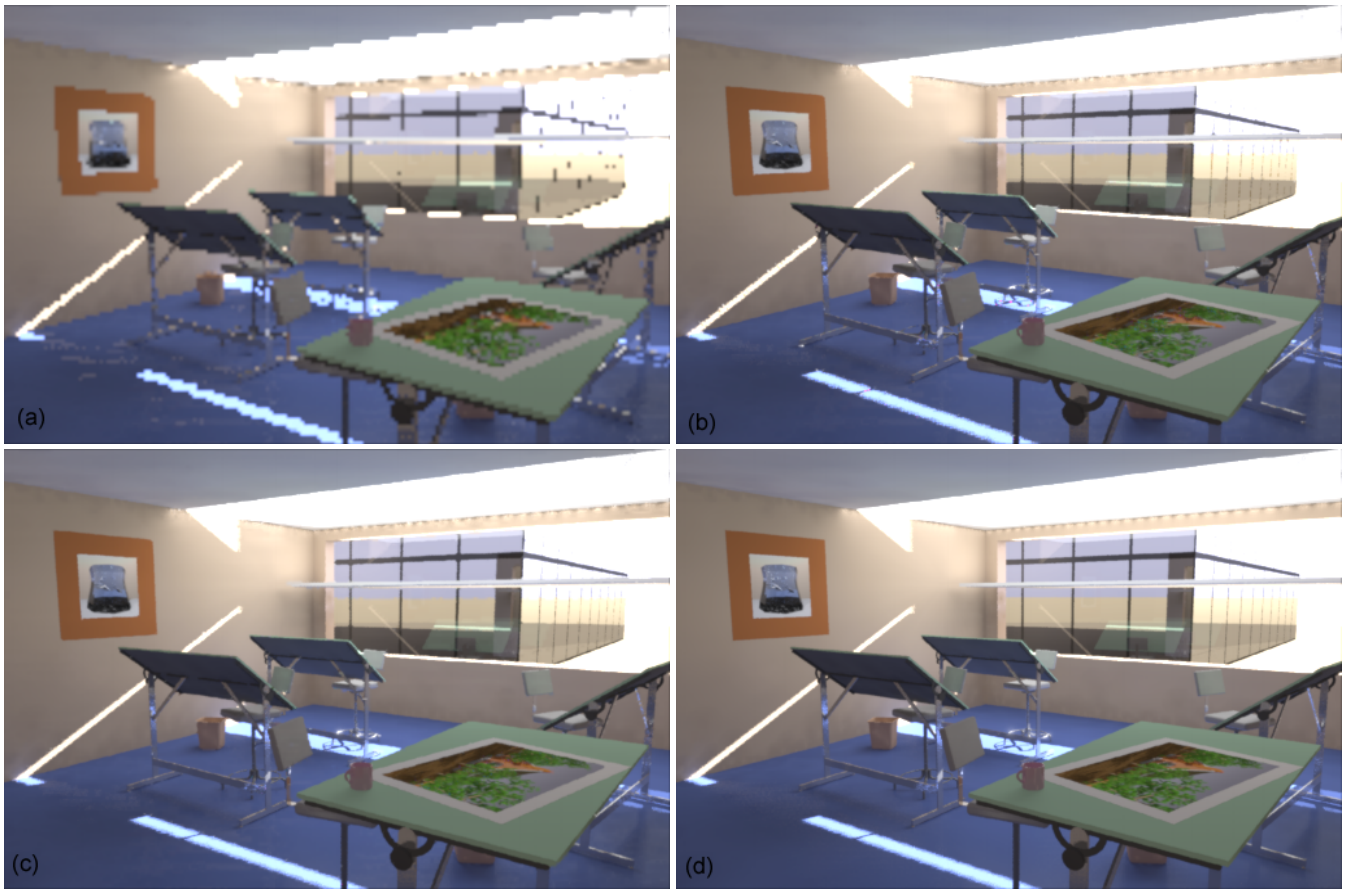

Figure 2: Progressive renderings of an office scene lit by sunlight transferred through a light shelf. (a) The approximate image at the end of the regular subdivision, with 1.6% evaluated pixels located at the corners of the 8×8 blocks in the work image. (b) The approximate image after boundary evaluations for all 8×8 edge blocks in the work image, with 5% of pixels evaluated. (c) The approximate image after evaluating about 6% of the pixels, whose locations are shown in Fig. 5 (bottom left). (d) The final image as rendered by the baseline RADIANCE system. The scene model was supplied courtesy of Greg W. Larson.

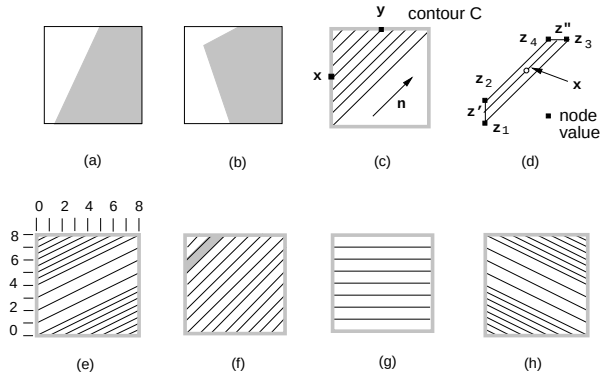

Figure 3: Discontinuity analysis on a block. The discontinuity in (a) is considered simple whereas that in (b) is not, because of the corner. The geometry for the least-discrepancy direction is given in (c). In (d) we show the construction of a typical bilinear element $f_e(\mathbf{x})$ on a quadrilateral $Q = [\mathbf{z}_1\mathbf{z}_2\mathbf{z}_3\mathbf{z}_4]$ with known node values $f_n(\mathbf{z}_i),\ i = 1:4$. Essentially this construction is a Gouraud interpolation with the scanline rotated to be parallel with the least-discrepancy direction. Note that (d) is a zoomed version of the shaded element in (f). Finally, oriented finite elements are shown in (e) through (h) for four different orientations.

Working with image data differs fundamentally from working with geometric data. The discontinuity edges computed by discontinuity meshing algorithms have two properties. First, the discontinuities are abstract mathematical lines with no shape or width. Second, discontinuities are either present or absent at a given location. In contrast, image edges have spatial scales (e.g. sharp or fuzzy) and their existence at a given location is modulated by their strength [21] (a weak edge may be just a faint wisp). These properties of image edges create challenges. For example, it is no longer trivial to define the location of the edge. In addition, the fact that image edges have characteristics such as shape and strength requires more information to be estimated with the small amount of sampling at our disposal.

The least discrepancy direction and oriented finite element discussed below form an integrated approach to extracting discontinuity information from image data and modeling the discontinuities in a finite element approximation to the block radiance function. This approach does not explicitly refer to discontinuity locations, and it models edge shape as well as strength. Our discussions on the least-discrepancy direction and oriented finite elements can be easily extended to any convex image blocks, including the non-square blocks often encountered in a quadtree subdivision of the image plane.

3.2 The Least Discrepancy Direction

The least discrepancy direction $\mathbf{m}(B_k)$ of a $k \times k$ block B_k is defined to be the unit vector that minimizes the contour integral

$$d(\mathbf{n}) = \frac{1}{s} \int_C \left(f(\mathbf{x} + t(\mathbf{x})\ \mathbf{n}) - f(\mathbf{x}) \right)^2 ds,$$

where C is the boundary contour of B_k and s is arc length (the reader may observe that the integration actually only needs to extend over half the contour). For a fixed direction $\mathbf{n}$ and a point $\mathbf{x}$ on C, the scalar $t(\mathbf{x})$ is chosen such that the parametric line $\mathbf{y}(t) = \mathbf{x} + t\ \mathbf{n}$ intersects the contour C at $\mathbf{x}$ and $\mathbf{y} = \mathbf{x} + t(\mathbf{x})\ \mathbf{n}$, as is shown Fig. 3 (c). Once the radiance function $f(\mathbf{x})$ is known on the contour C through boundary evaluation, the *directional discrepancy* $d(\mathbf{n})$ is a well-defined function of the direction $\mathbf{n}$.

For implementation, we let $\mathbf{n} = [\cos\theta, \sin\theta]$ and discretize the angular range $0 \le \theta < \pi$ into h different directions $\theta_i = i\pi/h,\ i = 0 : (h-1)$. For each direction $\mathbf{n}_i = [\cos\theta_i, \sin\theta_i]$, the directional discrepancy $d(\mathbf{n}_i)$ is evaluated as

$$d_i = d(\mathbf{n}_i) = \frac{1}{4(k-1)} \sum_{\mathbf{p} \in P} \left(f(\mathbf{p} + t(\mathbf{p})\ \mathbf{n}_i) - f(\mathbf{p}) \right)^2,$$

where P is the set of all pixels in C and $t(\mathbf{p})$ is chosen such that the line $\mathbf{y}(t) = \mathbf{p} + t\ \mathbf{n}_i$ intersects the contour C at $\mathbf{p}$ and $\mathbf{p}' = \mathbf{p} + t(\mathbf{p})\ \mathbf{n}_i$. Even though $\mathbf{p}$ is a pixel location, $\mathbf{p}'$ may not be, in which case $f(\mathbf{p}')$ is linearly interpolated from two adjacent pixel values. From the evaluated sequence $\{d_0, ..., d_{h-1}\}$, we find the minimizer $d_j = \min\{d_0, ..., d_{h-1}\}$ and set the least discrepancy direction $\mathbf{m}(B_k) = \mathbf{n}_j$.

3.3 Oriented Finite Elements

Once the least discrepancy direction is known, the radiance function is approximated by a finite element function whose elements are oriented along the least discrepancy direction. This finite element approximation is a continuous function consisting of bilinear elements (quadratic polynomials). Fig. 3 (e) through (h) describes oriented finite elements for an 8×8 block with $h = 8$ (the angular range $0 \le \theta < \pi$ is discretized into eight different directions). In this case, there are eight different types of oriented finite elements, one for each discretized direction. Only four of them are shown in the figure; the other four are obtained from the ones shown by a 90-degree rotation. The integer values in Fig. 3 (e) mark the locations of the pixel centers on the block boundary. These locations are also the locations for the node values of the finite elements. In Fig. 3 (e) and (h), there are nodes situated halfway between two pixels. For a node of this sort, the node value is taken to be the average of the two adjacent pixels. In general, for an arbitrary θ some nodes of the bilinear elements may not coincide with the pixel locations, and these nodes values are linearly interpolated from the adjacent pixels.

To compare the quality of images generated using oriented finite elements and Painter's adaptive stochastic sampling method [23], Fig. 4 (bottom) examines zoomed views of the same region in Fig. 2 (c) and Fig. 4 (top) (both images are filtered down from their work images using a Gaussian). For the same sampling rate, the DCM already produces a high-quality image while Painter's method suffers from artifacts including heavily jagged edges, missing objects parts, and missing high-frequency details. Fig. 5 shows the sampling patterns of Painter's method and the DCM. Also compare the zoomed views in Fig. 4 with the zoomed sampling patterns in Fig. 5 (the zoomed sampling pattern of the DCM does not include the extra samples needed for the first-order test described in Section 4.3, but the extra sampling is included in the 6% sampling rate reported).

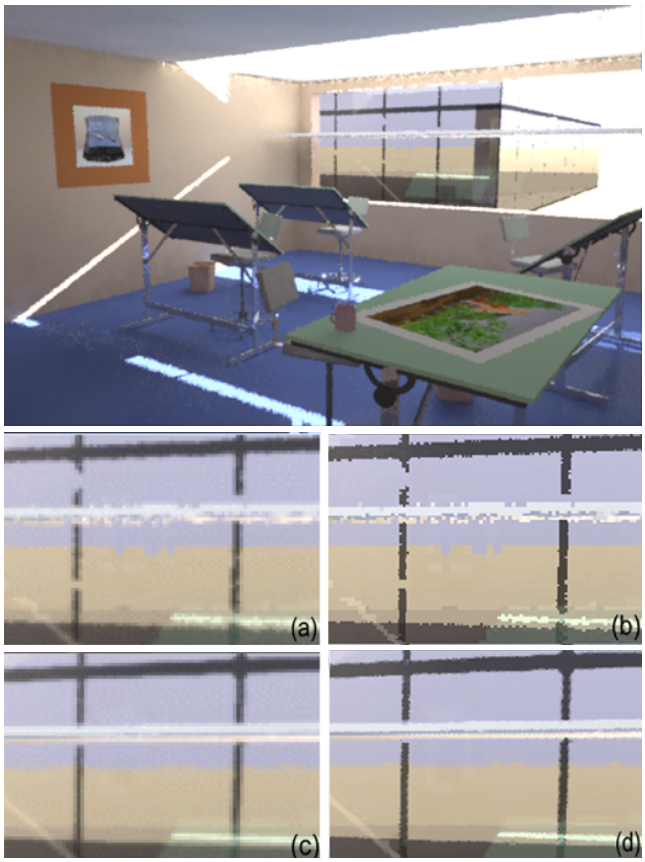

Figure 4: Comparison between the DCM and adaptive stochastic sampling with the office example. The top image, rendered by sampling 6% of the pixels using Painter's adaptive stochastic sampling technique, should be compared with the 6% DCM image in Fig. 2 (c). The bottom image contains four zoomed views of the same region: 6% Painter image in (a) with its work image in (b), and 6% DCM image in (c) with its work image in (d).

Notice that Painter's method very gracefully locates the features, but the number of samples at our disposal is just too small to make this method effective. We have also compared the DCM with adaptive super-sampling [35] and stratified sampling [19]. Painter's method performs much better than the other two because its underlying hierarchical integration sampling technique combines the strengths of adaptive and stratified sampling by stratifying samples with strata that are dynamically adjusted as more samples are taken [16].

3.4 Directional Coherence

Why does the least discrepancy direction work? Simply stated, it works because of image-space coherence. According to Sutherland [29], coherence is the degree to which parts of a scene or its projection exhibit local similarities. Often we think of a discontinuity edge as the break of coherence, since image data change abruptly across the edge. However, discontinuities do not break all forms of coherence. Specifically, image data are typically coherent along the direction of the discontinuity edge even if they change abruptly across the edge [21]. Contour-based image coding (e.g. [21]) and more broadly, second generation image coding [18] takes advantage of this form of coherence. For a block with a simple discontinuity edge, the least discrepancy direction is really the direction

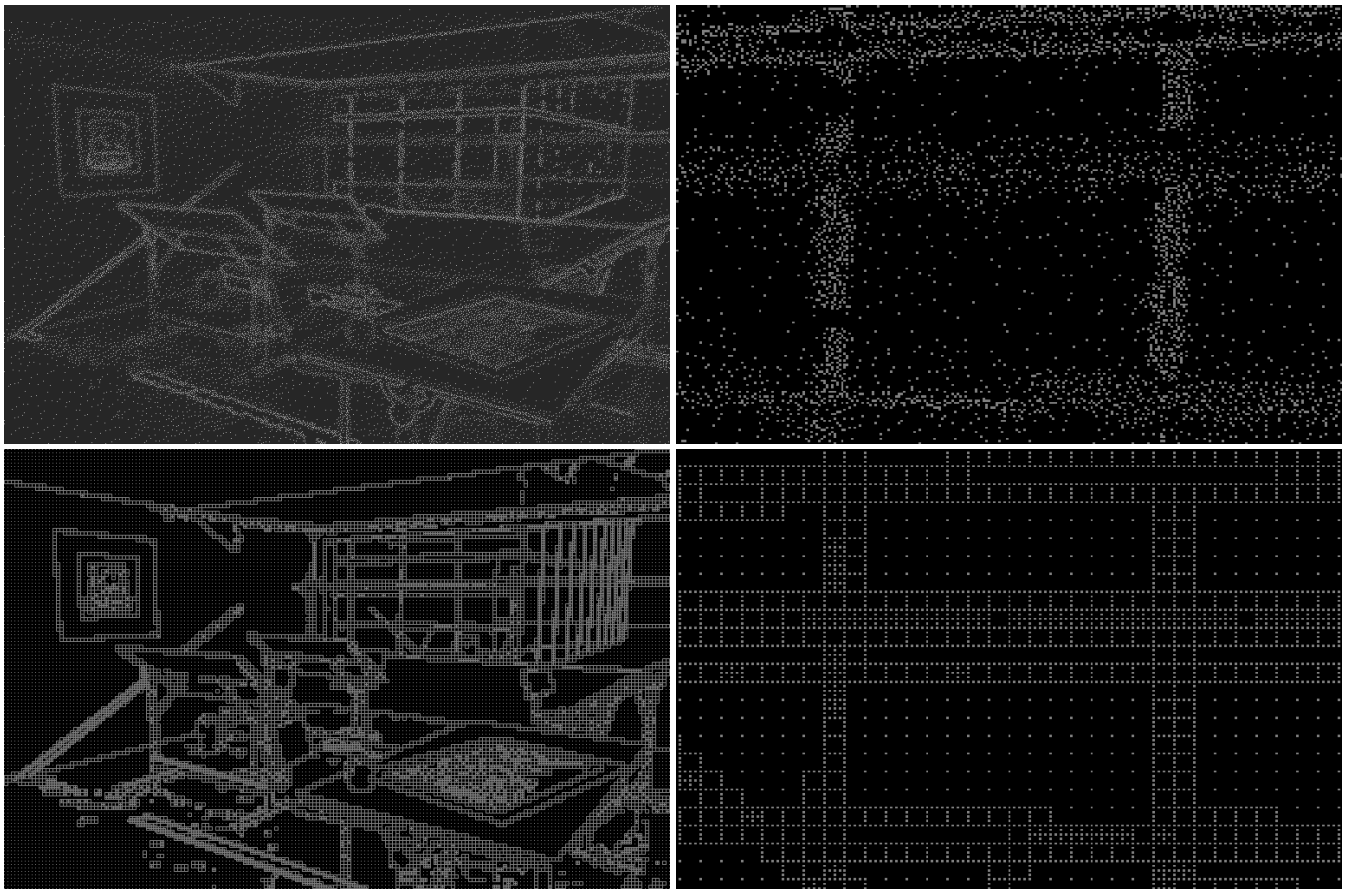

Figure 5: Comparison of the sampling patterns of adaptive stochastic sampling (top row) and the DCM (bottom row). The patterns in the left column are taken from RADIANCE work images described in Section 5. Some of the fine features are shown in zoomed views of the sampling patterns in the right column. These zoomed views correspond to the same region as the zoomed views in Fig. 4.

of maximal coherence as can be inferred from the evaluated boundary pixels. By orienting the finite elements along this direction, we maximize the likelihood of capturing the discontinuity edge along with its characteristics (section 4.3 on block simplicity tests for the treatment of more complex discontinuities such as corners).

4 Coherence Map Construction and Refinements

A DCM consists of a partition of the image plane into square blocks, with some selected blocks having a direction θ ($0 \leq \theta < \pi$) assigned to each of them. These selected blocks are called edge blocks; all other blocks are smooth blocks. The boundary of every edge block is densely evaluated, whereas a smooth block only has its four corners evaluated. To reconstruct an approximate image, we bilinearly interpolate each smooth block and build a finite element approximation on every edge block, with the elements oriented to the direction recorded in the DCM.

The main reason for separating edge and smooth blocks is efficiency. Since it is much more expensive to sample an edge block, we wish to reserve edge blocks for areas with discontinuities. In our block classification procedure, we select edge blocks based on evaluated pixel radiance and object-space data. Thus projections of object boundaries are taken into consideration from the beginning even for small objects. To further reduce the chance of missing blocks with discontinuities, a smooth block is not automatically subdivided into four smooth quads in the block refinement step. Instead, each quad is reevaluated to see if it contains discontinuities.

The DCM is a divide-and-conquer technique: it aims to partition the image plane into blocks with simple structures (i.e., blocks crossed by at most one discontinuity edge). At any given stage of the progressive radiance evaluation, the oriented finite elements produce good results on edge blocks with simple structures; the results on more complex edge blocks are less certain. Naturally, we wish to identify these complex blocks so that we can focus our sampling efforts on them at the next stage of the radiance evaluation. With this goal in mind, we have designed block simplicity tests for edge blocks. An edge block is a simple edge block if it passes these simplicity tests; otherwise it is a complex edge block.

A big concern with the block simplicity tests is that, in general, it is not possible to guarantee that a block is crossed by at most one edge as long as the block interior is not fully sampled. Nevertheless, with the evaluated block boundary we can identify many offending blocks. Our experiments indicate that with carefully designed simplicity tests, we can identify and subdivide sufficiently many offending blocks for the purpose of generating high-quality images. To further avoid mistreating a block with complex interior discontinuities, we constantly reassess the simplicity of a block as more evaluated pixels become available. This verification is done as part of the lazy boundary evaluation described later.

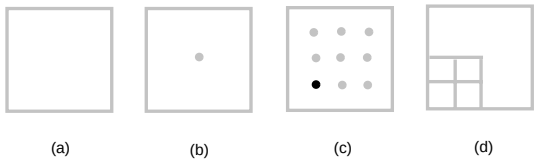

Figure 6: Lazy boundary evaluation on a simple edge block. (a) A simple edge block, with the light grey boundary representing evaluated pixels. (b) The difference between a newly evaluated pixel (marked by the grey dot) and the value predicted by the finite element approximation is within the prescribed tolerance, and no boundary evaluation is invoked. (c) One of the newly evaluated pixels, marked by the black dot, deviates too much from the value predicted by the current finite element approximation. (d) Boundary evaluations are triggered for the surrounding blocks.

4.1 Refinement Steps for the DCM

As mentioned in Section 2, the iterations for the DCM construction and refinements begin after the regular subdivision stage has partitioned the image plane into elementary blocks. Each iteration of the DCM refinement takes five steps. First, the block classification step examines the pool of smooth blocks to select edge blocks. For the first iteration, this pool consists of all elementary blocks. In any later iteration, the pool is formed by collecting the four quads subdivided from the smooth blocks in the previous iteration. The classification also marks as edge blocks the four quads of every complex edge block from the previous iteration. In the second step, the boundary evaluation procedure densely samples the block boundary of every edge block. Then, the simplicity test step analyzes every edge block and labels as complex edge blocks those that fail any block simplicity test. The fourth step builds oriented finite elements, and the fifth step subdivides every block into four quads for the next iteration.

The four quads $B_i,\ i = 1:4$ of a simple edge block B from the previous iteration are computed using a lazy boundary evaluation procedure as shown in Fig. 6. Since B is a simple edge block, we already have a finite element approximation $f(\mathbf{x})$ on B. Our experiments indicate that this finite element approximation is usually of very good quality unless some complex structures in the interior of B have gone undetected in the previous iteration. Thus before going through the normal boundary evaluation with B_i, we evaluate the corner pixels of B_i and compare the resulting pixel values with the pixel values predicted by the existing finite element approximation $f(\mathbf{x})$. If the predicted values are within a prescribed tolerance (1% relative error in our system) from the evaluated pixel values, then the simplicity of B is re-confirmed and we continue to use $f(\mathbf{x})$ on the new blocks B_i with small modifications. More specifically, we skip the normal boundary evaluation procedure and substitute $f(\mathbf{x})$ for the pixel values everywhere on the boundary of B_i except at the corners, where the evaluated pixels are used. With the block boundary so obtained, we construct a finite element approximation $f_i(\mathbf{x})$ on B_i with the least discrepancy direction of B. By incorporating the evaluated pixels into $f_i(\mathbf{x})$, we force the approximate image to converge to the final image as rendered by the baseline system.

4.2 Block Classification

A smooth block can be reclassified as an edge block through the following two steps. First, a block contrast value is computed for each block and this value is tested against the prescribed contrast threshold t_c. The block is classified as an edge block if its block contrast value exceeds the threshold t_c. Second, a visible-line rendering of the scene is generated, and every block crossed by a visible line is classified as an edge block.

Contrast Thresholding. For an elementary block with corner luminance values $\{g_1, ..., g_4\}$, the block contrast quantifies the ratio between the average luminance $\bar{g}$ and the deviation Δg from the average. Following Mitchell [22], we compute the block contrast as $\frac{\max - \min}{\max + \min}$, where $\max$ and $\min$ are the maximum and minimum of the corner luminance values $\{g_1, ..., g_4\}$ respectively. The criterion for locating high-frequency details has a significant impact on the effectiveness of the initial edge block selection. The issue here is not the loss of details since the progressive rendering eventually produces the same image as the baseline rendering system. The main concern is whether certain details will appear at earlier stages of the rendering process. In this regard, a criterion based on contrast $\Delta g/\bar{g}$ compares favorably with methods that use deviation Δg alone [19], because the nonlinear human visual sensitivity to the change in light intensity is closely modeled by the contrast $\Delta g/\bar{g}$ rather than just Δg. This logarithmic contrast perception model is the most widely used among other models [15]. In our implementation, we set $t_c = 0.05$.

Computing Visible Lines. The visible lines that we choose include both object boundaries from the scene geometry and their reflections in planar mirrors. These visible lines are efficiently computable through the standard graphics pipeline [36], which supports both polygonal objects and commonly-used curved objects. We use polygon offset to avoid the "stitching" artifact that could result from a naive Z-buffer implementation [36], because stitching turns a solid line into a dotted line and thus allows it to pass through an elementary block undetected. For a curved object the visible-line renderer is instructed to draw the silhouette only [36]. The mirror reflections of visible lines are computed as in [24].

Note that a pure contrast-driven classification can be deceptive for large blocks. To alleviate this problem, we use visible lines to account for object boundaries and at each iteration we reclassify smooth blocks to recover features missed in the previous iteration due to the larger blocks and fewer available samples.

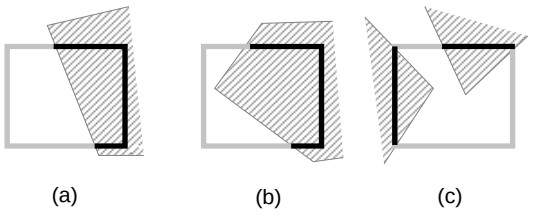

Figure 7: Block simplicity test examples. Case (a) passes both the zero order and first order tests. Case (b) passes the zero order test but not the first order. Case (c) fails both tests. The black and light grey line segments on the block boundary are spans.

4.3 Block Simplicity Tests

The simplicity tests in our system are designed systematically based on the traditional methodology of proof-by-contradiction. First we assume that the block is crossed by at most one edge. From this assumption, we derive facts about the discontinuities on the block boundary. The derived facts can be verified using the known radiance values on the block boundary. The block fails the tests if any contradiction arises. Fig. 7 provides examples for some of the following tests.

Zero Order Test. For a block crossed by a single edge, we should be able to find a luminance threshold t_b, such that the block can be divided into two simply connected regions (connected and having no holes): one for pixels with luminance above the threshold and

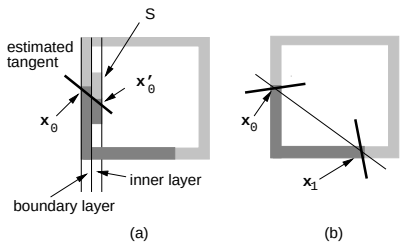

Figure 8: The first order test for block simplicity. (a) The tangent direction for the edge passing through $\mathbf{x}_0$ can be estimated from the transition points $\mathbf{x}_0$ and $\mathbf{x}'_0$ on two adjacent layers. (b) To pass the first order test, the line segment $[\mathbf{x}_0\ \mathbf{x}_1]$ (the thin black line) and the tangent directions at the two transition points $\mathbf{x}_0$ and $\mathbf{x}_1$ (the two thick black lines) must be nearly parallel.

the other for those below. The zero order test identifies the blocks that cannot be so divided, using the evaluated boundary pixels of the block. In our system, we set $t_b = 0.5(b_1 + d_1)$, where b_1 is the highest luminance on the block boundary and d_1 the lowest.

Thresholding with t_b converts the block boundary into a binary pattern consisting of "1-pixels" that exceed t_c and "0-pixels" that do not. Juxtaposed 1-pixels can be collected together to form a "1-span", and likewise 0-pixels form "0-spans" (Fig. 7). Two spans are separated by a transition point, which is defined as a 1-pixel having at least one 0-pixel neighbor. To simplify this binary pattern, we apply a median filter of length three to eliminate spans of one-pixel long. This simplification is needed because of the Monte Carlo component in the baseline ray tracing system, and the median filtering helps to eliminate the spurious spans caused by Monte Carlo noise. To reduce noise effects, we also apply a length-three Bartlett filter to the luminance values on the block boundary before thresholding. The filtered values are only used for computing the spans, not for constructing the block radiance approximation. With spans of length one removed, the number of transition points on the block boundary must be even. If there are more than two transition points, the block fails the zero order test. Otherwise the block survives the test and moves onto the first order test.

First Order Test. A block with no transition points passes the first order test by default. For a block having two transition points, let the transition points be $\mathbf{x}_0$ and $\mathbf{x}_1$. If the block is crossed by a single edge passing through $\mathbf{x}_0$ and $\mathbf{x}_1$, the tangent vectors of the edge at the two points should be close in direction. The first order test computes the tangent vectors at $\mathbf{x}_0$ and $\mathbf{x}_1$ and measures the difference between the tangent directions against a prescribed tangent threshold t_m. If the measured difference exceeds the threshold, the block fails the test. Let θ_{01} be the direction of the line segment $[\mathbf{x}_0\ \mathbf{x}_1]$ that connects $\mathbf{x}_0$ and $\mathbf{x}_1$, whose tangent directions are determined by angles θ_0 and θ_1 respectively. The difference of the tangent directions is measured by $m_{01} = \max(|\theta_0 - \theta_{01}|, |\theta_1 - \theta_{01}|)$. Fig. 8 (b) illustrates the geometry. The threshold t_m is set to 0.05π times the L_2 norm $||\mathbf{x}_1 - \mathbf{x}_0||_2$ in our implementation.

A challenging technical problem in the first order test is the efficient computation of tangent vectors at the transition points. We have developed a technique that estimates tangent directions at the cost of a few additional pixel radiance evaluations. To estimate the angle θ_0 at $\mathbf{x}_0$, we evaluate the luminance function at pixels along a short line segment S next to $\mathbf{x}_0$ on the inner layer of pixels, as is shown in Fig. 8 (a). The luminance threshold t_b and a length-three median filter are applied to these additional luminance values to extend the binary pattern from the boundary layer to the line segment S. The evaluation process starts from the pixel next to $\mathbf{x}_0$ in the inner layer and elongates S in both directions, stopping as soon as the transition point $\mathbf{x}'_0$ corresponding to $\mathbf{x}_0$ is found on S. At pixel resolution, the transition points $\mathbf{x}'_0$ and $\mathbf{x}_0$ determine the tangent angle θ_0.

Binarizing images to extract geometric patterns is not new; researchers have used this technique in the field of video coding [15]. An example is the geometric-structure-based directional filtering proposed by Zeng [39]. Even though the patterns extracted in his work are complex and uncertain, he has successfully used these patterns to improve block-based video coding at low bit rates.

Object Test. We enforce the constraint that no more than two object tags are allowed on the boundary of a block. In addition, the object tags also form object spans similar to the spans in the zero order test, and only two object spans are allowed on the block boundary. A block violating these constraints will not be considered simple. The object tags are returned by the baseline ray tracer as by-products of pixel radiance evaluations.

If a block fails any of the above block simplicity tests, it is labeled as a complex edge block. In the next iteration of block refinement, we devote more sampling to a complex edge block B_c by performing real (i.e. not lazy) boundary evaluations on the four quads subdivided from B_c.

4.4 Discussion

The DCM is useful because it allows us to generate high-quality images from a small percentage of evaluated pixels. Unfortunately, the danger of serious approximation errors also grows with the number of unevaluated pixels. For this reason, it is desirable to have a technique that uses known data (e.g. scene geometry and evaluated pixels) to bound the errors, possibly with the help of some analytical formulation [31, 26, 1]. We have not derived such a technique. Instead, we have built two simple principles into the DCM construction. First, we always incorporate the newly evaluated pixels into the DCM and never overwrite them. As a result, the approximate image is guaranteed to converge to the final image as rendered by the baseline system, and all approximation errors are thus eliminated eventually. The second principle is that we regularly re-examine our previous decisions in partitioning blocks to detect errors: smooth blocks are tested for discontinuities and simple edge blocks are probed for complex structures. These error detections are done using newly evaluated pixels as part of the block classification and lazy boundary evaluation.

In practice, failure to detect discontinuities means the delay of high-quality images. In this regard, the DCM performs better with object boundaries than shadows and highlights, which can go undetected with larger blocks. In fact, small highlights and shadows in the block interior will certainly go undetected until the block is subdivided. Fig. 2 (c) contains errors of this sort (e.g. the cup on the table). These performance problems often have solutions, albeit at additional cost. For example, one way to improve on shadows is to include shadow edges as in [24].

5 Implementation

We have implemented our progressive rendering algorithm using the RADIANCE system developed by Ward [33]. RADIANCE is a physics-based lighting simulation system that has gained considerable reputation because of its successful use in real-world projects [33]. Our progressive rendering system allows the user to examine the approximate image any time during the rendering process. When their Monte Carlo components are challenged by difficult lighting configurations, RADIANCE and other ray tracing systems for global illumination [27] will become less effective. In this sort of situation, our system uses the confidence relocation technique described in Section 6.2 to generate images of high visual quality.

Progressive Rendering with RADIANCE. In the baseline RADIANCE system, a work image is first generated at a super-sample

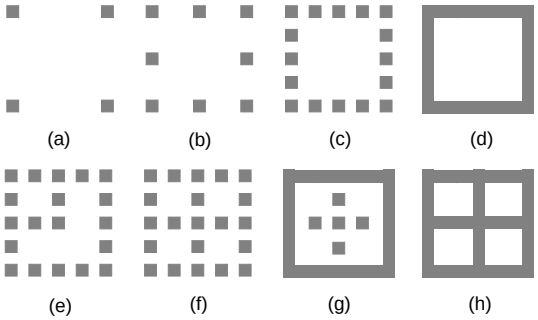

Figure 9: Boundary evaluation procedures for a 8×8 block. The small grey squares stand for evaluated pixels. Other pixels on the block boundary are linearly interpolated. For a simple edge block, we follow the sequence (a), (b), (c), and (d), which evenly distributes evaluated pixels on the block boundary. For a complex edge block, we also need to distribute evaluated pixels in the block interior, and we follow the sequence (a), (b), (c), (e), (f), (g), and (h).

resolution by a hybrid deterministic and Monte Carlo ray tracing program. This work image is then filtered down to the resolution of the output image for anti-aliasing. To generate high-quality images with penumbra from a scene with fine geometric details, as is the case with our examples, the work image is 3×3 times as large as the output image, and every pixel is evaluated through ray tracing.

For pixel radiance evaluation, our progressive rendering system uses exactly the same calculation parameters as the baseline system. Like the baseline system, our progressive system performs the rendering in batch mode. Anytime during the rendering process, the finite element approximation to the radiance function can be assembled "on-demand" in a few seconds, and then re-sampled onto the work image. For the images in this paper, we filtered the work image using the Gauss filter provided with RADIANCE to produce the output image. For the DCM construction, we choose the size of the elementary block to be 8×8 and discretize the angular range $[0, \pi]$ into 8 different directions. All the other DCM-related parameters have been given in the previous sections.

Fig. 9 explains boundary evaluations for 8×8 blocks. The idea is to evenly distribute evaluated pixels in order to allow the construction of oriented finite elements even before every pixel on the block boundary is evaluated. Note that for a complex edge block, we jump from Fig. 9 (c) to (d) because in (c) the distance between evaluated pixels (2) is already smaller than the size of the quads (4) and when this happens we start distributing evaluated pixels on the boundaries of the four quads of a complex edge block.

Generating Better Images by Confidence Relocation. Normally our approximate image progresses towards a final image determined by some given calculation parameters. If this final image suffers from severe artifacts, we must raise the quality standard and progress towards a better final image. This approach is especially relevant to the Monte Carlo components of radiance evaluation. The Monte Carlo computations introduce noise while capturing diffuse interreflections. A common approach to noise reduction is to increase the sampling rates. However, as Rushmeier argues, the sampling rates needed can be impractically high for certain difficult lighting configurations [25].

With the DCM, it is possible to generate images with high visual quality even when the Monte Carlo calculations make it too costly to evaluate every pixel accurately. The basic idea is simple: the Monte Carlo noise is not part of the radiance function by nature, but a manifestation of the limitations of our radiance evaluation techniques. This means that when the pixel values are accurate, they will exhibit the image-space coherence we see in photographs.

image	$L_1(L_2)$ error	time (C)	time (NC)
office 6%	0.008(0.02)	0.9 hrs	8.9 hrs
office final	-	16.5 hrs	38.5 hrs
museum 7%	0.005(0.016)	0.05 hrs	1.2 hrs
museum final	-	1 hrs	5 hrs

Table 1: Progressive rendering statistics. The "time (C)" column lists timings for scenes with cached irradiances, while the "time (NC)" column lists timings for scenes without caching.

In particular, directional coherence will be present and we can use the DCM to reconstruct high-quality images from a small percentage of evaluated pixels. Improvement of pixel accuracy is usually achieved by increasing our confidence in the pixel values through variance reduction. When the available computation is more or less uniformly spread over the entire image plane, so is our confidence in the pixel values. If the resulting image is poor, we can try to improve the accuracy of every pixel, but that usually means a dramatic increase in computational costs. An alternative is to improve the accuracy of pixel values in a progressive system based on the DCM. Since the progressive system can generate high-quality images with a small percentage of evaluated pixels, we usually obtain high-quality images without additional computation. When taking the second approach, we relocate the uniformly spread low confidence to concentrated high confidence in the small percentage of pixels needed by the DCM.

In our implementation, we use hierarchical integration [16] to reduce the variance within each pixel, stopping when a confidence interval test passes [23].

Time and Space Considerations. Compared to the cost of pixel radiance evaluations, the time needed for managing the DCM and oriented finite elements is negligible. More specifically, the time complexity of DCM-related construction is no greater than an inverse discrete cosine transform (DCT), which is used for decoding JPEG images [15]. For a typical image included in this paper, generating an approximate image can be generated on the order of seconds as opposed to the few hours needed for evaluating the pixel radiance. The memory requirements are also modest. In addition to the storage needed by the baseline RADIANCE, we need only store a list of edge blocks and a horizontal strip of the work image. The complete work image is stored on the disk and it is retrieved only when the user demands the system to display the current approximate image.

6 Results

All the high-quality approximate images reported in this paper are obtained after the progressive rendering system finished processing all 4×4 edge blocks. At this point, the approximate images become visually hard to distinguish from the final images for most scenes.

6.1 Progressive Rendering

Fig. 2 shows a series of images progressively rendered from an office scene lit by sunlight transferred through a light shelf. This scene was introduced in [33] to demonstrate how RADIANCE preprocesses "virtual" light sources to optimize light calculations for certain difficult environments. Both the baseline and the progressive systems have included that optimization. As is typical with the other scenes we have tested, 6% evaluated pixels allow us to generate a high-quality image that is hard to distinguish visually from the final image. See Fig. 5 for the locations of the 6% samples. Fig. 10

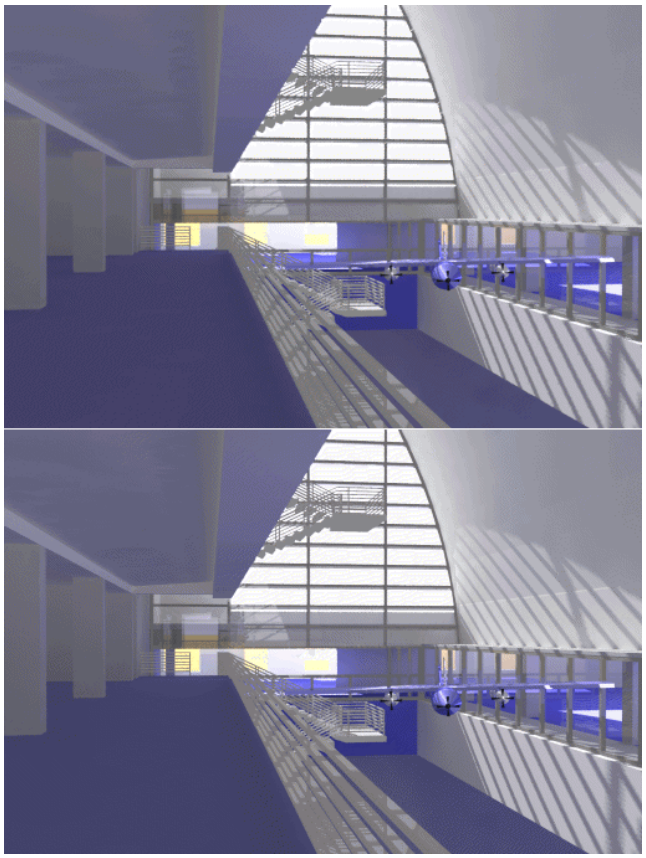

Figure 10: Progressive renderings of a museum scene lit by skylight through the window. Notice the fine shadows cast by the polygonal occluders. The top image is the approximate image after evaluating about 7% of the pixels. The bottom image is the image rendered by the baseline RADIANCE system. The scene model was provided courtesy of Charles Ehrlich.

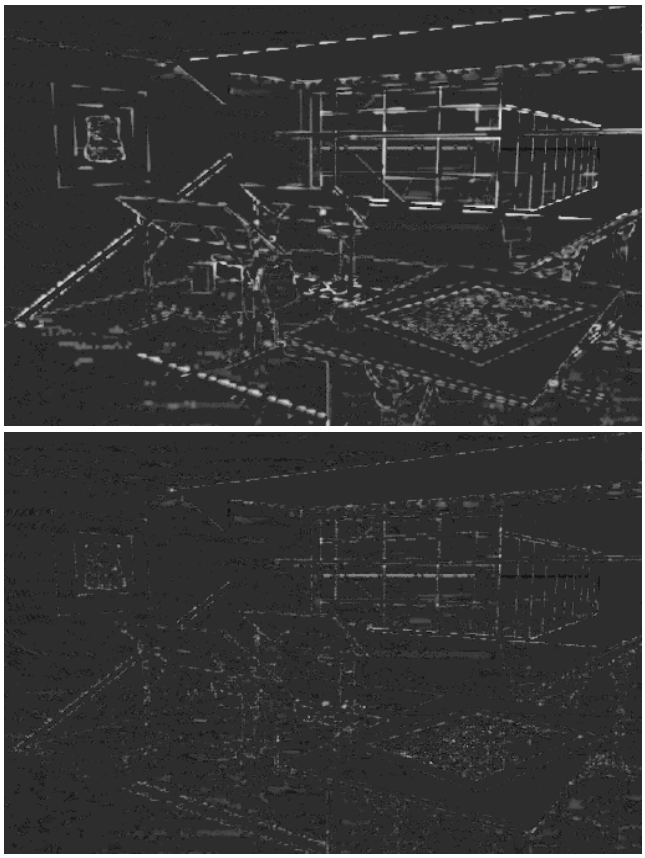

Figure 11: Error distributions for the office example. The top image shows the errors in the approximate image with 1.6% evaluated pixels. The bottom image is for the approximate image with 6% evaluated pixels. Even though the two distributions are taken from very different stages of the rendering process, the most significant errors are clustered around the image discontinuities for both distributions. This attests to the importance of properly handling discontinuities when generating high-quality images.

shows another example, which is a museum lit by skylight through the window.

To quantify the errors in a high-quality approximate image, we subtract it from the final image F to get an error image E. Then we compute the relative error as either $||E||_1/||F||_1$ or $||E||_2/||F||_2$, where $||.||_1$ and $||.||_2$ is the L_1 and L_2 norms respectively. The resulting L_1 and L_2 errors for the two examples are in Table 1.

Table 1 also compares the computation times for the high-quality approximate images and the final images. All timings are taken on a 180 MHz SGI Indy workstation with 64 Mb of main memory. The time indicated for each final image is the time needed by the baseline RADIANCE system. The time for an approximate image includes not only the time for pixel radiance evaluations but also all the computation related to the DCM.

An important factor that affects the timings is the irradiance caching in RADIANCE [34]. The diffuse interreflections in the rendering equation can be calculated using Monte Carlo ray tracing [16], but to reduce the variance to a tolerable level, hundreds of rays must be spawned for each eye-ray that strikes a surface. To avoid invoking this expensive calculation at every pixel, RADIANCE caches indirect diffuse contributions and interpolates them over each surface in the scene. At an early rendering stage, there is little irradiance cached in the scene to interpolate from, and an eye-ray is more expensive to evaluate. As time goes by, more cached values become available, and the radiance evaluation accelerates. We tabulate two timings for each example: one measured with the lighting simulation starting without irradiance cached, and the other with the irradiance cached from a previous rendering from a different viewpoint. Note that even when irradiance caching is not a dominant factor, there is no exact correspondence between the time percentage and the percentage of pixels evaluated. In fact, the time percentage can be smaller than the pixel percentage when there is a large amount of irradiance cached in a scene, as is the case with the "museum" example.

Fig. 11 provides error images at different stages of the rendering process. The intensities of both images have been scaled up to make the errors more visible. As a result, the error images mainly show the error distributions.

6.2 Confidence Relocation

Fig. 12 provides an example of generating better images with confidence relocation.[2] Each pixel in the 512×342 image on the left corresponds to 3×3 pixels in the work image, but for the penumbra areas this sampling rate is insufficient. Fig. 12 (a) and (c) show artifacts in the left image with zoomed views of two regions. Fig. 12

[2]For information on luminaires, the reader is referred to the scene model at http://radsite.lbl.gov/radiance/pub/models/bath.tar.Z

Figure 12: The image on the left is rendered using the baseline RADIANCE system by setting the quality to the highest level available. On the right, (a) and (c) are zoomed views of two regions from the left image. Notice the striping artifacts and missing penumbra boundaries. In (b) and (d), we show zoomed views of the same two regions from an image rendered in less time by our progressive renderer (for the complete image see the Conference Proceedings CD-ROM). The scene model was provided courtesy of Greg W. Larson.

(b) and (d) are zoomed views of the same two regions from an image rendered using confidence relocation. The image generated by confidence relocation is also a 512×342 image, filtered down from a work image 3×3 times as large. But this time, instead of evaluating every pixel of the work image, we only evaluate 6.7% of the pixels and we super-sample these pixels to increase pixel confidence. The computation times for the left image of Fig. 12 and the image with confidence relocation are 5.5 hours and 2.7 hours respectively on our SGI Indy with each computation initiated without irradiance caching. When there is caching, the left image takes 1.4 hours and the image generated with confidence relocation takes about $1/7$ of the time. The correctness of the image by confidence relocation has been verified with a benchmark image generated by taking 256 super-samples for each pixel of the final image.

6.3 Discontinuity Varieties

We have tested the DCM on a variety of scenes with very different discontinuities. Fig. 10 is a scene with fine shadows cast by polygonal occluders. For this sort of scene, discontinuity meshing works well. The main advantage of the DCM in this case is the ability to handle large scenes without suffering the storage overhead of meshing.

Fig. 13 is from a scene filled with specular surfaces placed next to each other. Discontinuity meshing cannot cope with this scene well because of the view-dependent specular highlights and curved geometry. Moreover, discontinuity meshing seeks to track down all potential discontinuities prior to radiance evaluation [20], which is extremely difficult to do here due to the closely packed specular surfaces. To render the scene, we incorporate the DCM into Rayshade [17], a well-known system for classical ray tracing [35]. We choose to do so partly because RADIANCE does not handle torus primitives – we also wanted to see how easy it is to incorporate the DCM into a typical ray tracing system. Rayshade collects radiance samples for individual pixels of the output image as follows. First, one sample is collected for each output pixel and a contrast value is computed based on the current pixel and its four neighbors. If high contrast is encountered, the current pixel and the four neighbors are anti-aliased by taking 3×3 samples for each pixel (the default setting). To incorporate the DCM into the system, we first create a work image 3×3 times as large as the output image. Then we carry out DCM-related construction as in our RADIANCE implementation.

The top right image in Fig. 13 is rendered after evaluating 10% of the pixels of the work image. The percentage is higher than usual because the complex interactions between the tightly packed specular surfaces lead to high contrast almost everywhere. As shown in Fig. 13 (bottom left), the high contrast also causes much super-sampling in Rayshade. Fig. 13 (bottom row) compares the Rayshade and DCM sampling patterns in the RADIANCE work image using zoomed views of a chosen region (for clarity, the zoomed sampling pattern for the DCM does not include the extra samples needed for the first-order test described in Section 4.3, but the extra sampling is included in the 10% samples reported). On our Indy workstation, the top image of Fig. 13 takes 10 minutes to render with Rayshade, whereas the bottom image takes 1.4 minutes when the DCM is used.

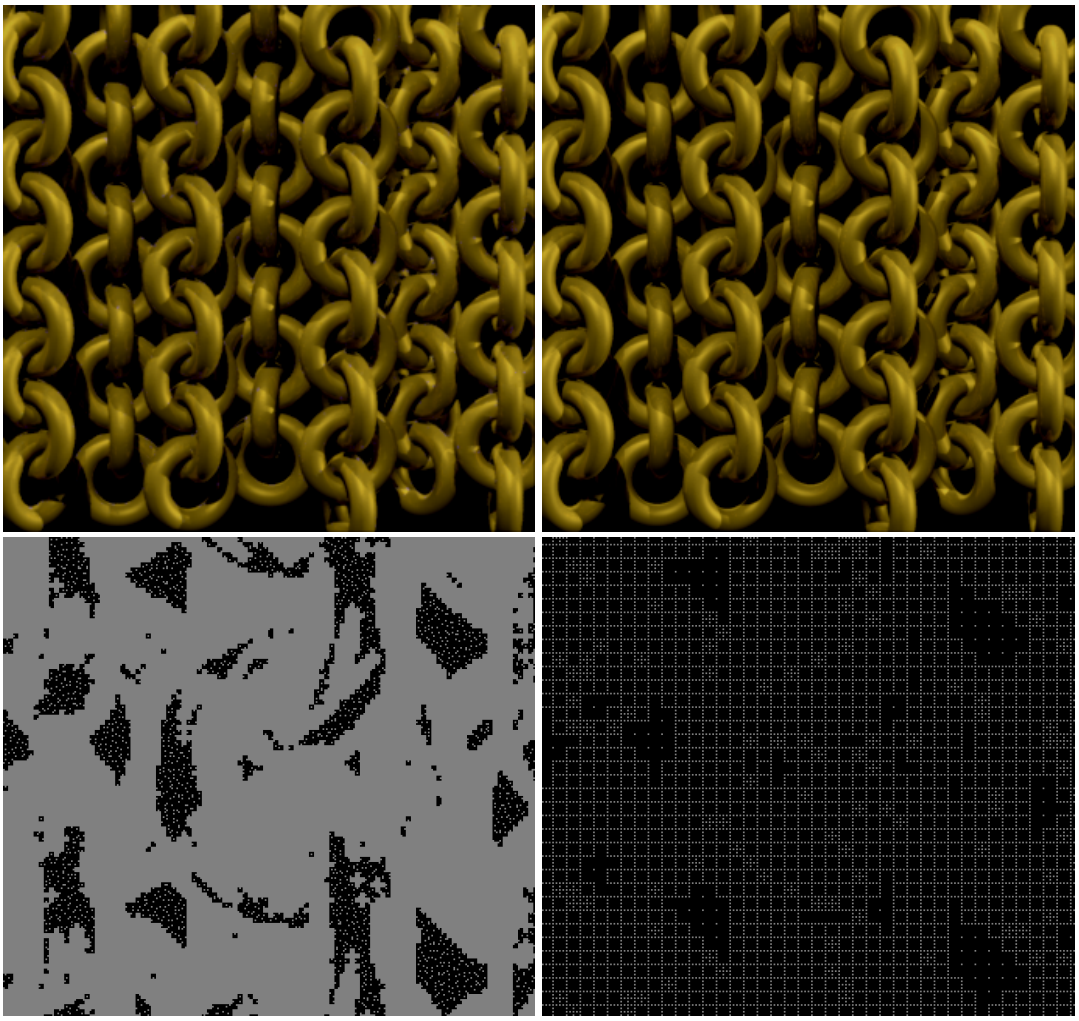

Figure 13: The top left image is a benchmark image rendered by Rayshade. The top right image is rendered using the DCM after a small percentage of pixels have been evaluated. The main purpose of this experiment is to test the DCM's capability in treating discontinuities that cannot be handled by existing discontinuity algorithms. The bottom row provide zoomed views of the sampling patterns for the same region of the work image (left pattern for Rayshade and right for the DCM). The grey dots indicate the locations of the samples. Compared to Rayshade, the DCM performs much less sampling yet can produce an image of similar quality because of the effective treatment of discontinuities. The scene model was provided courtesy of Stuart Warmink.

7 Conclusions

We have presented a progressive refinement algorithm for radiance evaluation, by showing how to handle general radiance discontinuities using a novel technique called the Directional Coherence Map. The DCM subdivides the image plane into blocks with simple discontinuities and captures the discontinuities on each block with the least discrepancy direction and oriented finite elements. By combining object-space data with discontinuity information extracted from evaluated pixel radiance, the DCM achieves both time and storage efficiency. Thus it is possible to treat discontinuities in a complex scene that could barely be loaded into the main memory of our computer. The DCM is also shown capable of effectively treating a variety of discontinuities, including several types that cannot be handled by existing discontinuity algorithms. For a global illumination scene consisting of smooth surfaces, the DCM generates high-quality images much faster than progressive ray tracing systems based on adaptive sampling. When the Monte Carlo components of a lighting calculation system are challenged by very difficult lighting configurations, our algorithm can still produce high-quality images efficiently by relocating the computational resources to the small percentage of pixels needed by the DCM.

Several related research topics remain to be explored. The DCM gains its power from the directional coherence in the image plane. Other forms of coherence in the radiance function should also be investigated. An example is Teller's radiance interpolant, which makes use of the coherence between images from nearby viewpoints [31]. Another area of research is the use of sophisticated color vision models to improve our refinement strategy so that less computation is distributed to areas of little perceptual importance. Bolin's work has shown promising results in this direction [4]. Finally, we expect to see growing interest in image-space discontinuities in the near future. Our experiments not only indicate the importance of properly treating image-space discontinuities, but also demonstrate the power of image-space discontinuity information. In general, we believe techniques for manipulating image data will become more important as the average size of the polygons passing through the graphics pipeline approach the size of individual pix-

els, and we hope that our work stimulates future research in this increasingly exciting area.

Acknowledgments

I would like to thank Demetri Terzopoulos for his help at several critical moments. Without his help this work would not be possible. Thanks also to Greg W. Larson for his insightful comments on some of the ideas presented and for numerous consultations on RADIANCE, to Bede Liu and Wenjun Zeng for helpful discussions, to John Funge for proofreading part of this paper, and to the anonymous reviewers for their constructive critique.

References

[1] J. Arvo. The Irradiance Jacobian for Partially Occluded Polyhedral Sources. In A. Glassner, editor, *Computer Graphics Proceedings, Annual Conference Series*, pages 75–84, July 1994.

[2] L. D. Bergman, H. Fuchs, E. Grant, and S. Spach. Image Rendering by Adaptive Refinement. In D. C. Evans and R. J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 29–37, August 1986.

[3] J. Bloomenthal. Edge Inference with Applications to Antialiasing. In *Computer Graphics (SIGGRAPH '83 Proceedings)*, volume 17, pages 157–162, July 1983.

[4] M. R. Bolin and G. W. Meyer. A Frequency Based Ray Tracer. In R. Cook, editor, *Computer Graphics Proceedings, Annual Conference Series*, pages 409–418, August 1995.

[5] A. T. Campbell III and D. S. Fussell. Adaptive Mesh Generation for Global Diffuse Illumination. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 155–164, August 1990.

[6] S. E. Chen, H. E. Rushmeier, G. Miller, and D. Turner. A Progressive Multi-Pass Method for Global Illumination. In T. W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 165–174, July 1991.

[7] N. Chin and S. Feiner. Near Real-Time Shadow Generation Using BSP Trees. In J. Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 99–106, July 1989.

[8] M. F. Cohen, S. E. Chen, J. R. Wallace, and D. P. Greenberg. A Progressive Refinement Approach to Fast Radiosity Image Generation. In J. Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 75–84, August 1988.

[9] R. L. Cook, T. Porter, and L. Carpenter. Distributed Ray Tracing. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 137–145, July 1984.

[10] F. Crow. Shadow Algorithms for Computer Graphics. In *Computer Graphics (SIGGRAPH '77 Proceedings)*, volume 11, pages 242–248, July 1977.

[11] G. Dretakkis and E. Fiume. A Fast Shadow Algorithm for Area Light Sources Using Backprojection. In A. Glassner, editor, *Computer Graphics Proceedings, Annual Conference Series*, pages 223–230, July 1994.

[12] A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: Accelerated Ray Tracing System. *IEEE Computer Graphics and Applications*, 6(4):16–26, July 1986.

[13] A. Glassner. *Principles of Digital Image Synthesis*, volume 1. Morgan Kaufmann, 1995.

[14] P. Heckbert. Discontinuity Meshing for Radiosity. *Third Eurographics Workshop on Rendering*, pages 203–226, May 1992.

[15] A. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, 1989.

[16] J. Kajiya. The Rendering Equation. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 143–150, August 1986.

[17] C. Kolb. Rayshade User's Guide and Reference Manual. *Rayshade home page at graphics.stanford.edu*, January 1992.

[18] M. Kunt, A. Ikonomopoulos, and M. Kocher. Second-Generation Image Coding Techniques. *Proc. of IEEE*, 73(4):549–574, 1985.

[19] M. E. Lee, R. A. Redner, and S. P. Uselton. Statistically Optimized Sampling for Distributed Ray Tracing. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 61–67, July 1985.

[20] D. Lischinski, F. Tampieri, and D. P. Greenberg. Combining Hierarchical Radiosity and Discontinuity Meshing. In *Computer Graphics Proceedings, Annual Conference Series*, pages 199–208, 1993.

[21] S. Mallat and S. Zhong. Characterization of Signals from Multiscale Edges. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(7):710–732, 1992.

[22] D. P. Mitchell. Generating Antialiased Images at Low Sampling Densities. In M. C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 65–72, July 1987.

[23] J. Painter and K. Sloan. Antialiased Ray Tracing by Adaptive Progressive Refinement. In J. Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 281–288, July 1989.

[24] F. Pighin, D. Lischinski, and D. Salesin. Progressive Previewing of Ray-Traced Images Using Image-Plane Discontinuity Meshing. *Eurographics Workshop on Rendering 1997*, May 1997.

[25] H. Rushmeier and G. Ward. Energy Preserving Non-Linear Filters. In A. Glassner, editor, *Computer Graphics Proceedings, Annual Conference Series*, pages 131–138, July 1994.

[26] P. Schroeder and P. Hanrahan. On the Form Factor between Two Polygons. Technical Report CS-404-93, Princeton University, Computer Science Department, 1993.

[27] P. Shirley. Hybrid Radiosity/Monte Carlo Methods. In *Siggraph 94 Course on Advanced Radiosity*, August 1994.

[28] A. J. Stewart and S. Ghali. Fast Computation of Shadow Boundaries Using Spatial Coherence and Backprojections. In A. Glassner, editor, *Computer Graphics Proceedings, Annual Conference Series*, pages 231–238, July 1994.

[29] I. Sutherland, R. Sproull, and R. Schumacker. A Characterization of Ten Hidden-Surface Algorithms. *ACM Computing Surveys*, 6(1):387–441, March 1974.

[30] S. Teller. Computing the Antipenumbra of an Area Light Source. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 139–148, July 1992.

[31] S. Teller, K. Bala, and J. Dorsey. Conservative Radiance Interpolants for Ray Tracing. *Eurographics Workshop on Rendering 1996*, May 1996.

[32] C. Vedel. Computing Illumination from Area Light Sources by Approximate Contour Integration. In *Proceedings of Graphics Interface '93*, Toronto, Canada, May 1993.

[33] G. J. Ward. The RADIANCE Lighting Simulation and Rendering System. In A. Glassner, editor, *Computer Graphics Proceedings, Annual Conference Series*, pages 459–472, July 1994.

[34] G. J. Ward, F. M. Rubinstein, and R. D. Clear. A Ray Tracing Solution for Diffuse Interreflection. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, pages 85–92, August 1988.

[35] T. Whitted. An Improved Illumination Model for Shaded Display. In *Computer Graphics (SIGGRAPH '79 Proceedings)*, volume 13, pages 1–14, August 1979.

[36] M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide*. Addison Wesley Developers Press, 1996.

[37] G. Wyvill, C. Jay, D. McRobie, and C. McNaughton. Pixel Independent Ray Tracing. In *Computer Graphics: Developments in Virtual Environments (Proc. CG International '95)*, pages 43–55. Springer-Verlag, 1995.

[38] G. Wyvill and P. Sharp. Fast Antialiasing of Ray Traced Images. In *New Advances in Computer Graphics (Proc. CG International '95)*, pages 579–588. Springer-Verlag, 1989.

[39] W. Zeng and B. Liu. Geometric-Structure-Based Error Concealment with Novel Applications in Block-Based Low Bit Rate Coding. *IEEE Trans. Cir. and Sys. for Video Tech.*, to appear, 1998.