# Efficient Image-Based Methods for Rendering Soft Shadows


Maneesh Agrawala — Pixar Animation Studios
Ravi Ramamoorthi — Stanford University*
Alan Heirich — Compaq Computer Corporation
Laurent Moll — Compaq Computer Corporation

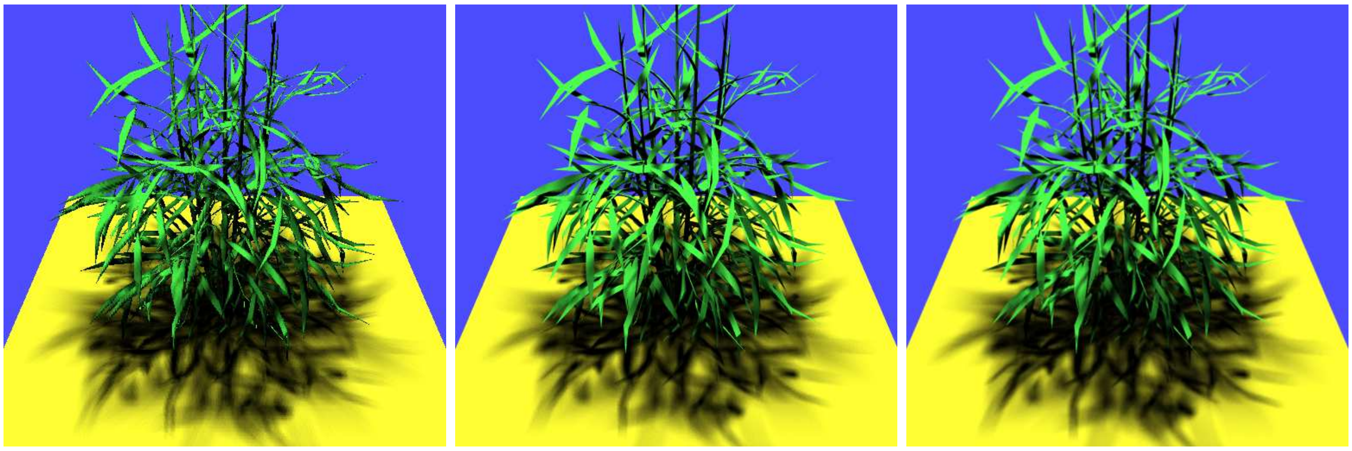



Figure 1: *A plant rendered using our interactive layered attenuation-map approach (left), rayshade (middle), and our efficient high-quality coherence-based raytracing approach (right). Note the soft shadows on the leaves. To emphasize the soft shadows, this image is rendered without cosine falloff of light intensity. Model courtesy of O. Deussen, P. Hanrahan, B. Lintermann, R. Mech, M. Pharr, and P. Prusinkiewicz.*


**Abstract**

We present two efficient image-based approaches for computation and display of high-quality soft shadows from area light sources. Our methods are related to shadow maps and provide the associated benefits. The computation time and memory requirements for adding soft shadows to an image depend on image size and the number of lights, not geometric scene complexity. We also show that because area light sources are localized in space, soft shadow computations are particularly well suited to image-based rendering techniques. Our first approach—*layered attenuation maps*—achieves interactive rendering rates, but limits sampling flexibility, while our second method—*coherence-based raytracing* of depth images—is not interactive, but removes the limitations on sampling and yields high quality images at a fraction of the cost of conventional raytracers. Combining the two algorithms allows for rapid previewing followed by efficient high-quality rendering.



**CR Categories:** I.3.7 [*Computer Graphics*]: Three-Dimensional Graphics and Realism—Shadowing, Raytracing

**Keywords:** Shadows, Raytracing, Image-Based Rendering


## 1 Introduction

Soft shadows from area light sources can greatly enhance the visual realism of computer-generated images. However, accurately computing penumbrae can be very expensive because it requires determining visibility between every surface point and every light.


*(maneesh,ravir)@graphics.stanford.edu (Alan.Herich,Laurent.Moll)@compaq.com
Address: Gates Wing 3B-386, Stanford University, Stanford, CA 94305.


The cost of many soft shadow algorithms grows with the geometric complexity of the scene. Algorithms such as ray tracing [5], and shadow volumes [6], perform visibility calculations in object-space, against a complete representation of scene geometry. Moreover, some interactive techniques [12, 27] precompute and display soft shadow textures for each object in the scene. Such approaches do not scale very well as scene complexity increases.

Williams [30] has shown that for computing hard shadows from point light sources, a complete scene representation is not necessary. He performs the visibility calculations in image space, against *shadow maps*—image-based representations of scene geometry. Although the shadows may suffer undersampling, bias, and aliasing artifacts, the cost of the algorithm is relatively independent of scene complexity. Further, it is possible to implement this method as a post-shading pass that modulates a shadowless rendering from the base renderer to include shadows. This makes it simple to add the method to any existing renderer, without modifying the base renderer, and does not limit the approach to particular geometric primitives. In this paper, we describe two efficient image-based techniques for rendering soft shadows that can be seen as logical extensions of Williams' approach.

In both methods, shadows are computed in image space. Therefore the time and memory requirements for adding soft shadows to an image are dependent only on image complexity and the number of lights, not geometric scene complexity. Neither algorithm computes per object textures, so texture mapping is not a bottleneck for us. This independence from geometric scene complexity allows us to efficiently compute soft shadows for large scenes, including those which have complex patterns of self-shadowing.

We will also show that soft shadows are a particularly good application for image-based rendering approaches. Since area light sources are localized in space, visibility changes relatively little across them. The depth complexity of the visible or partially visible scene as seen from a light (and stored in our shadow maps) is generally very low. Further, shadow maps rendered from the light source sparsely sample surfaces that are oblique to the light source. However, these surfaces are less important to sample well, because they are precisely the surfaces that are dimly lit.

The contributions of this paper are the two algorithms summarized below, which represent two ends of a spectrum.

**Layered Attenuation Maps:** Our first approach achieves interactive rendering rates but limits sampling flexibility, and can therefore generate undersampling and banding artifacts. We precompute a modified layered depth image (LDI) [25] by warping and combining depth maps rendered from a set of locations on the light source. The LDI stores both depth information and layer-based *attenuation maps* which can be thought of as projective soft shadow textures. During display, the proper attenuation is selected from the LDI in real time in software, and is used to modulate normal rendering without shadows. The precomputation is performed in a few seconds, and soft shadows are then displayed at several frames a second. Since the light source sample positions are chosen a priori, they are correlated for each surface location and this correlation can appear as banding in the final image.

**Coherence-Based Raytracing:** Our second approach removes limitations on sampling and yields high quality images, suitable for high resolution prerendered animations, but is not interactive. We precompute shadow maps from a few points on the light, often the boundary vertices. To shade a surface point, we trace shadow rays through the shadow maps rather than the scene geometry. The shadows rays are decorrelated since they are chosen independently for each surface point, and therefore banding is replaced by noise. While the general approach to ray tracing depth images is well-known[1] [15, 16, 18, 20], we develop several novel acceleration techniques for accelerating shadow ray computations.

The visible portion of a light source tends to change very little for surface points close to one another. We describe a new image-based technique for exploiting this coherence when sampling visibility along shadow rays. Our image-based raytracing approach with coherence-based sampling produces soft shadows at a fraction of the cost of conventional raytracers. While we combine both the image-based ray-tracing and sampling algorithms in a single renderer, they can be used independently (i.e. a standard geometric ray tracer might incorporate our coherence-based sampling method).

Our algorithms can be combined in an interactive lighting system; our fast layered attenuation map method can be used to interactively set the viewing transformation, and position the light source and geometry. Our coherence-based raytracing method can then be used to quickly generate final high-quality images. This is the approach we took to produce the results in this paper, and we believe this approach has many applications to lighting design.

The rest of this paper is organized as follows. Section 2 reviews previous work on soft shadows. Section 3 presents preliminaries, while Section 4 describes our interactive layered attenuation map algorithm. In section 5, we describe coherence-based raytracing of depth images. The results are presented in Section 6, and Section 7 discusses future work and conclusions.

## 2 Previous Work

There is a vast literature on shadow algorithms, which we touch on only briefly. Although a decade old, the survey by Woo et al. [32] is still an excellent reference.

### 2.1 Object-Based Methods

Soft shadows can be computed using object-space methods such as distributed ray tracing [5] and radiosity with discontinuity meshing [11, 17] or backprojection [7, 29]. Stark et al.[28] describe analytic methods for computing soft shadows. These approaches are computationally intensive and are not suitable for fast soft shadow generation for complex scenes.

Herf and Heckbert [12] combine a number of shadow images for each receiver using an accumulation buffer [9]. The method is object-based, and the precomputation time can grow quadratically with the number of objects being shadowed, making it impractical for large scenes. Furthermore, a separate (generally large) texture is created for each shadowed object. Our layered attenuation map approach is analogous in that we combine a similar number of depth images rendered from different points on the light. However, we improve on Herf and Heckbert's method by precomputing image-based textures simultaneously for the entire scene.

Soler and Sillion [27] use convolution on blocker images to compute fast approximate soft shadows. A primary advantage of their technique is that sampling artifacts that sometimes occur when averaging hard shadows are avoided. A disadvantage of their method is that they cluster geometry in object-space and the clusters cannot shadow themselves; to correct this for complex objects like plants or trees would require a very large number of clusters for the leaves. This increased number of clusters can greatly increase the computation time, obviating the benefits of the method. Separate textures are needed for each cluster being shadowed, which can strain the texture mapping hardware for complex objects. Furthermore, robust error control and automated clustering algorithms can be complicated to implement [26].

Hart et al. [10] develop a view dependent method to accelerate soft shadow computations for a standard ray tracer. They precompute a blocker list of geometry, stored in object space, for each image pixel by tracing a small number (often only one) of shadow rays to the light. When a blocker is found, they check if adjacent image pixels also "see" the same blocker using a recursive 4-connect flood-fill algorithm. The main stage of their algorithm first projects and clips each blocker to the light source and then computes the irradiance from the remaining portion of the light. While this method can greatly accelerate shadow computation, it is not well-suited for handling large amounts of tiny blocker geometry. As the size of geometric elements decreases, the probability that a blocker is missed in the blocker list precomputation phase increases, which can result in light leaks. Moreover, the storage of the blocker list and the projection and clipping of each blocker against the light source can become very expensive. While our coherence-based sampling bears some similarities to this approach, we remove two limitations. First, our approach is view independent. From a given set of precomputed shadows maps we can generate shadows for any view of the scene. Second, since our algorithm is image-based, its cost is independent of scene complexity. Small triangles are not a bottleneck for our coherence-based raytracing approach.

### 2.2 Image-Based Methods

Williams' shadow map algorithm [30] is an image-based alternative to object-space methods. Visibility along a shadow ray is determined by precomputing a shadow map from the light and then comparing the depth of each pixel in the final image to the corresponding depth in the shadow map. Percentage-closer filtering can be used for antialiasing [23] and projective textures [24] can be used for hardware implementation. Forward shadow mapping [33] is an alternative implementation when texture-mapping represents a bottleneck for normal rendering of the scene. All of these methods render hard shadows from point light sources.

Chen and Williams [2] describe a simple extension to shadow mapping for rendering soft shadows. They render a few key shadow maps at the vertices of the light source and then use view interpolation to compute shadow maps for each sample location on the interior. To render soft shadows, they simply perform the standard shadow map test on each interpolated map to compute average visibility. The view interpolation method suffers from two drawbacks. First, the final image must be projected into each interpolated shadow map independently. These projections can become expensive since they are required for each view in an interactive session. Second, like our layered attenuation map algorithm, banding artifacts can appear in the shadows, since the light source sam-

[1] McMillan [20] calls this inverse warping.

ple positions are chosen a priori.

Lischinski and Rappoport [16] use hierarchical raytracing of depth images as one of several image-based techniques for computing secondary rays in synthetic scenes. Keating and Max [14] point out that light leaks are a problem with this approach because each depth sample is treated independently as a 2D surface unconnected with adjacent samples. They extend Lischinski and Rappoport's method by aggregating adjacent depth samples into discrete depth buckets, forming relatively large flat surfaces. While this approach reduces light leaks, as the authors point out, it can also completely change the scene geometry. It is unclear how such changes affect the final image. While our raytracing algorithm is also based on that of Lischinski and Rappoport, we reduce the light leak problem by reconstructing more accurate surfaces from the depth samples. We also introduce several new acceleration techniques that improve the efficiency of the hierarchical algorithm, especially when using multiple reference shadow maps.

Guo [8] accelerates raytracing by using image-space coherence to reduce the number of primary rays traced without affecting the number of shadow rays traced per primary ray. Our coherence-based raytracing method exploits visibility coherence among shadow rays to reduce the number of shadow rays traced per primary ray. It may be possible to combine our approach with Guo's to exploit coherence for both shadow rays and primary rays.

## 3 Preliminaries

Irradiance from an area light source on a surface is given by

$$E = \int_{A_{light}} \left[ \frac{L \cos\theta_i \cos\theta_l}{\pi r^2} \right] V \, dA \qquad (1)$$

where $L$ is the radiance output from the light source, $\theta_i$ is the incident angle, and $\theta_l$ is the angle made with the light normal [3]. We are primarily concerned with the change in binary visibility $V$. In a post-shading approach, the lighting term is computed separately from visibility and is often approximated by treating the area light as a point light source. We can then independently compute an average visibility that attenuates the shadowless rendering.

$$ATT = \frac{1}{A} \int_A V \, dA \qquad (2)$$

It is also possible to implement both of our methods within the normal shading pass of the base renderer and compute equation 1 directly. For simplicity and efficiency, our layered attenuation map algorithm takes the former approach, separating visibility from lighting. For high quality results, our coherence-based raytracing algorithm takes the latter approach, directly computing equation 1. As in most soft shadow approaches, multiple lights are handled independently.

The integral in equation 2 is evaluated using quadrature by sampling a number of points on the light source. We assume there is a mapping from the unit square to the light source such that a uniform sampling of the square will uniformly sample the light. To choose $N^2$ sample locations on the light source, we stratify the unit square into $NxN$ cells and choose some jittered sample location within each cell. In our layered attenuation map approach, the same sample points on the light are used for shadowing each surface point. In contrast, our ray tracing algorithm chooses which points to sample on the light separately for each surface point, and thereby removes the banding artifacts that can appear in the former approach.

## 4 Layered Attenuation Maps

In this section, we describe our algorithm for precomputing and displaying layered attenuation maps. The reader will want to refer to the illustrations in figures 2 thru 6.

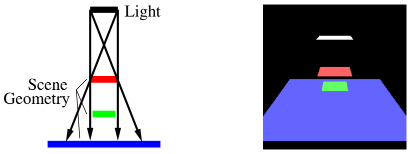



Figure 2: **Left:** *A schematic of the scene used to illustrate the layered attenuation map algorithm.* **Right:** *The scene without shadows,*

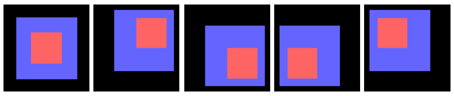


Figure 3: *Images taken from the light source center (leftmost) and the four corners (line 3 of the precomputation pseudocode).*

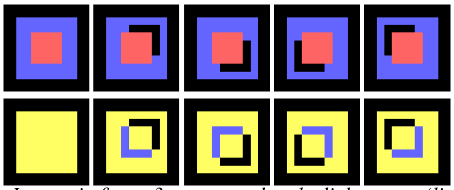


Figure 4: *Images in figure 3 are warped to the light center (line 5 precomputation pseudocode). On top, we show the first layer after warping, and below, the second layer. Yellow indicates absence of a second layer i.e. only one layer present. Regions not visible from a given light position show up as holes (black) when warped.*

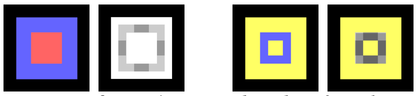


Figure 5: *Images in figure 4 are combined to form layered attenuation maps (lines 6 and 7 of the pseudocode). From left to right, layer 1, the texture on layer 1 (white indicates fully visible), layer 2 (yellow indicates absence of a second layer), and the texture on layer 2. Note that the completely occluded green square is not present at all.*

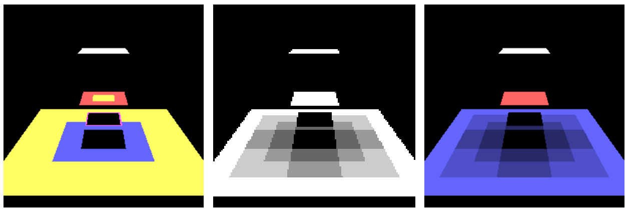


Figure 6: **Left:** *A visualization; white denotes the light, yellow: regions with only one layer, red: the first of two layers, blue: the second of two layers, black: umbral regions behind the last layer in the LDI, and magenta: when a point is between two layers (which happens for a very small region of the green square—at its edges—since it is culled from the LDI).* **Middle:** *Attenuation map. This modulates the basic image.* **Right:** *The final image.*

**Precomputation:** During the precomputation phase, we build the layered attenuation maps. This data structure consists of a separate list of layers for each pixel. Each layer stores depth, and the attenuation or fraction of the light that is visible from that point.

**procedure** Precompute
1 **foreach** light sample $l_i$
2 $\quad Viewpoint \leftarrow l_i$
3 $\quad$ Render(SCENE)
4 $\quad$ **foreach** pixel $(x, y)$
5 $\qquad (x', y') \leftarrow$ WarpCenter$(x, y, z(x, y))$
6 $\qquad$ Insert($(x', y')$,$z$,$\epsilon$)
7 Process Attenuation Maps

For each of a number of samples (typically 64) on the light, we render an image looking into the scene along the normal to the light at the sample location. In line 5 of the pseudocode, we transform the pixel into a central reference frame—the view from the light's center. For planar light sources, this warp is especially simple, being given by

$$\mathbf{dp} = -\frac{\mathbf{dv}}{z} \tag{3}$$

where $\mathbf{dp}$ is the vector disparity (change) in pixels, $\mathbf{dv}$ is the (known) vector difference in viewing positions. $z$ is measured from the viewpoint into the scene, and is the same for both views. In general, a projective transformation is required.

In line 6, we insert the transformed pixel into the *layered-depth image* (LDI). In our algorithm, each layer contains a depth value and an integer count. If the transformed depth value is already in the LDI (to tolerance $\epsilon$), we simply increment the count of the appropriate layer by one. The count corresponds to the number of light samples visible to a point in a given layer at a given pixel. If the depth does not exist in the layer list, we add a new layer to the list, setting its count to one.

Holes, or gaps, can occur when warping image-based representations of large objects in line 5. Splatting is often used to combat this problem. Since our viewpoints on the light are all close to each other (assuming the light is a relatively small finite object), we adopt a simple strategy. For each transformed (fractional) point, the four neighboring (integer) pixels are considered in line 6 of the pseudocode. To avoid double-counting, we increment the count for a given layer at a given pixel at most once for each viewpoint. Note that this splatting can slightly overestimate object size, making the shadows appear somewhat lighter.

Finally, line 7 computes an attenuation map by dividing the count in a layer by the total number of samples used in the outer loop. This corresponds to the fraction of the light that is visible.

**Display:** As shown in the pseudocode below, in the display phase of the algorithm, the scene is first rendered normally with lighting. Note that we do not interfere with the texture path, so the normal rendering can include textures.

**procedure** Display
1 RenderWithLightingAndTextures(SCENE)
2 **foreach** pixel $(x, y)$
3 $\quad (x', y', z') \leftarrow$ WarpLDI($(x, y, z(x, y))$)
4 $\quad layer \leftarrow$ Layer($(x', y')$,$z'$,$\epsilon$)
5 $\quad color \leftarrow color$ * AttMap($(x', y')$,$layer$)

In line 3, each pixel is then projected back to the viewpoint at the center of the light source, and is associated with the nearest pixel in the precomputed LDI. The appropriate projection matrix is continuously updated during an interactive session.

In line 4, the list of layers at the corresponding pixel in the LDI is traversed and depths are compared with the transformed depth from the input pixel using a tolerance (shadow bias) $\epsilon$. If no depth matches, the rendered point is not visible from anywhere on the light, and the attenuation applied in the next step is 0. In line 5, the base color of the image pixel is modulated by the attenuation map corresponding to the layered attenuation map for $layer$ computed in the previous step.

**Discussion:** The time for precomputation is proportional to the number of light source samples used, while the time for display is proportional to the average depth complexity of the LDI. The precomputation can be performed quickly because we use only fast image warping operations instead of slower raytracing or backprojection. The display phase can be carried out at interactive rates because the depth complexity of the LDI is very low. Since the light samples are close together, and the LDI only stores points visible from somewhere on the light, the average number of layers in our representation is significantly less than for an LDI that represents the entire scene. As we will see in section 6.1, LDI depth complexity increases slowly after the first few light samples i.e. very few new layers are created in line 6 of the precomputation pseudocode.

We need to render the scene from many light sample locations to precompute accurate attenuation maps. If rendering the scene separately for each sample is expensive, we may instead warp depth images from key locations such as the corners of the light. In our implementation, this is not an issue because we use standard graphics hardware as the base renderer—for line 3 precomputation, and line 1 display. However, all other parts of our precomputation and display routines are implemented entirely in software.

To implement the precomputation and display routines in hardware would require hardware support for LDI creation and lookup. This is somewhat complicated because the depth complexity of LDIs is not fixed a priori. For soft shadows, however, the final depth complexity of the LDIs tends to be very low. Therefore, it may be possible to limit the number of layers and implement the display phase in hardware by combining shadows separately computed for each layer—either using the accumulation buffer and the SGI shadow map extensions, or using a programmable image-compositing framework [13].

## 5 Coherence-Based Raytracing

The layered attenuation map method is suitable for rapid previewing because of its fast precomputation phase and its interactive display phase—whose time complexity is independent of the number of light source samples. However, final images for applications such as prerendered animation, require high quality antialiased artifact-free shadows. To render such images efficiently, we have developed a coherence-based raytracing algorithm.

The algorithm combines two independent image-based methods: a hierarchical raytracing technique and a coherence-based sampling technique. We begin by precomputing shadow maps from several locations in the scene. Our raytracing algorithm places no restrictions on the position and orientation of the reference views; we typically use views from the exterior vertices of the light. To shade a surface point, we compute visibility along each shadow ray by tracing it through each shadow map, until either an intersection is found or we pass through all the shadow maps. Our coherence-based sampling algorithm reduces the number of shadow rays cast to a light source by sampling light source visibility only where changes in visibility are most likely.

For the layered attenuation map approach, the light source sampling is done during precomputation. On the other hand, it is done during display in the coherence-based ray tracing method, making the precomputation phase independent of the number of light source samples, and the display time proportional to the number of shadow rays traced.

### 5.1 Raytracing Depth Images

Raytracing depth images is a well known technique [15, 20] . After a quick summary of the algorithm, we describe several new modifications to it, which improve both its accuracy and efficiency.

To trace a shadow ray against a single reference image (shadow map), we can first project it onto the reference image plane, and then step along this *epipolar ray* from pixel to pixel, checking for intersections with the scene geometry—represented as depths in the shadow map. The intersection calculation is performed in two phases. The first phase is a quick *overlap test* to determine if intersection is possible. As we step along the ray, we maintain the epipolar depth interval $[Z_{enter}, Z_{exit}]$ of the shadow ray that spans the current reference pixel. If the corresponding reference image depth $Z_{ref}$ is inside the epipolar depth interval, the second phase of the intersection test is performed to determine the exact point of intersection.

Recently, several papers [1, 16, 18] have described a hierarchical version of this raytracing algorithm that is similar to earlier work on raytracing height fields [22]. As a pre-process, two quadtrees are constructed from the reference image, one storing maximum depth values and one storing minimum depth values. The hierarchical algorithm performs the overlap test in a coarse-to-fine manner using these quadtrees. Thus, the raytracer can efficiently skip over large sections of the ray that cannot contain intersections. At the leaf-node level the exact intersection test is applied as before. Our pseudocode is adapted from [16]:

```
procedure Trace(QTreesNode,Ray,Z_enter,Z_exit)
if(Leaf(QTreesNode))
   check for exact intersection
else
   RefIntrvl ← [QTreesNode(MIN),QTreesNode(MAX)]
   EpiIntrvl ← [Z_enter,Z_exit]
   if(OverLap(EpiIntrvl,RefIntrvl))
      foreach non-empty Child of QTreesNode
         Update(Z_enter,Z_exit)
         Trace(Child,Ray,Z_enter,Z_exit)
```

The exact intersection test requires reconstructing a surface from the depth values stored in the reference image. Two common reconstruction techniques are triangulation[19] and bilinear interpolation[18, 21]. However, both methods impose costly exact intersection checks. A much simpler approach is to assume each reference image depth pixel represents a plane, which we call a *floor*, that runs parallel to the reference image plane. The floors are connected at their edges with vertical planes we call *walls*[2]. Although the floors-and-walls approach may yield a blocky surface reconstruction compared to triangulation or bilinear interpolation, in practice we have found that such artifacts are easy to control by generating higher resolution reference images.

Assuming every pair of adjacent reference pixels is connected with a wall yields incorrect intersections at silhouette edges of unconnected objects. We mitigate the problem by assuming adjacent reference image pixels are connected only if they differ in depth by less than a user specified *gap bias*. We check for intersections with walls only when adjacent reference pixels are connected.

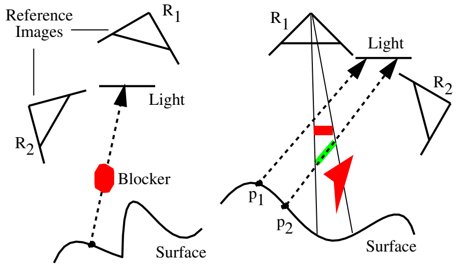
In the next two subsections, we describe new methods for accelerating the hierarchical traversal and for efficiently combining information from multiple reference images.

### 5.1.1 Accelerating the Hierarchical Traversal

**Hierarchical Connectedness:** By traversing the epipolar ray in a coarse-to-fine manner, the hierarchical algorithm can quickly eliminate sections of the epipolar ray that cannot possibly intersect scene geometry. However, the only way to find an intersection is to recurse through the min/max quadtrees all the way to the finest level and then perform the exact intersection test. If it is not necessary to determine the exact point of intersection, we can modify the algorithm to determine whether or not the ray is blocked before descending to the finest level.

When building the min/max quadtrees, we also build a connectedness quadtree. At the leaf level, each pixel is connected to itself. At the next level, we determine whether each group of four adjacent pixels form a single surface by checking if their depths fall within the gap bias. We continue computing connectedness in this manner all the way to the root of the tree. At any stage in the hierarchical traversal, if the epipolar depth interval contains the corresponding reference image min/max interval and the connectedness quadtree reports that all four children of the current node are connected, then the ray must intersect some geometry within the node. Thus, it is possible to report that the ray is blocked without recursing to the finest level of the quadtree.

[2] To compute the exact intersection, we first check if reference image depth $Z_{ref}$, lies within the epipolar depth interval for the leaf node pixel. If so, the ray intersects the floor. At the exiting edge of the pixel we look up the reference depth for the adjacent pixel $Z_{ref2}$ and if $Z_{exit}$ lies within $[Z_{ref}, Z_{ref2}]$, the ray intersects a wall. Linear interpolation can be used to find the exact point of intersection along the ray if necessary.




(a) Ranking Reference Images (b) Occlusion Intervals

Figure 7: *Handling multiple reference images. In (a) reference image $R_1$'s image plane is more perpendicular to the ray than $R_2$'s image plane and therefore yields a shorter epipolar ray. We rank $R_1$ higher than $R_2$ since it is faster to traverse and more likely to yield an intersection. In (b) the shadow ray starting at $p_1$ is a* clear miss *in $R_1$ since it never passes behind geometry. Since the ray is entirely visible in $R_1$ we do not need to trace it in any other reference image and can immediately declare the ray unblocked. For the shadow ray starting at $p_2$ only the green occlusion interval found in $R_1$ is traced in $R_2$.*

**Min/Max Clipping:** The epipolar ray is initially clipped to the viewing frustum of the reference image. We can aggressively reduce the length of the epipolar ray by clipping it against the min/max bounds of the current node in the min/max quadtrees. By clipping the epipolar ray upon entering the Trace procedure, at every level of the recursion we ensure that the overlap test is always performed against the smallest possible piece of the epipolar ray[3].

### 5.1.2 Efficiently Combining Multiple Reference Images

Using a single reference image, it is possible to miss intersections. If the desired ray intersects geometry that is not visible in the reference view due to occlusion, the intersection can not be found. Using multiple reference images that capture different portions of the scene geometry can mitigate this problem. However, multiple views generally contain redundant information about the scene as well[4]. Our layered attenuation map algorithm handles this redundancy by creating an LDI that only retains a single sample at each depth. A drawback of the LDI approach is that it requires resampling the reference images which can reduce numerical accuracy. We take a two pronged solution that avoids this resampling. First we rank the reference images so the that images most likely to generate an intersection are traversed first. We adapt Chang's[1] approach of ranking the reference images by the length of the clipped epipolar ray in each image (see figure 7(a)). As soon as an intersection is found we move on to the next shadow ray. Second, for each reference image after the first, we only trace portions of the ray that were not visible in the previous images.

**Clear Miss:** Shadow ray computation requires traversing the epipolar ray in every reference image, until some intersection is found. For blocked shadow rays, the loop over reference images can be exited early, as soon as the intersection is found. This type of early exit is sometimes possible for unblocked rays as well. If we traverse an epipolar ray in one reference image and find no intervals in which it passes behind occluding geometry, the entire ray was visible in the reference view. We can safely report that the shadow ray is not blocked. This type of *clear miss* allows us to exit the function early, without traversing the epipolar ray in any other reference image.

**Occlusion Intervals:** As we traverse the epipolar ray in some reference image, the overlap test forces the recursion all the way

[3] Marcato[18] performs this clipping only once at the coarsest level of the recursion.

[4] Choosing the optimal set of reference views to cover the space of a given scene while minimizing redundant information is a difficult problem which we do not address in this paper.

down to the leaf level each time the ray enters and exits a region behind occluding geometry, We determine that an intersection did not occur at such silhouette boundaries only after we perform the exact intersection test which includes a check for connectedness. As we traverse the ray, we store a list of these *occlusion intervals* in which the ray passes behind occluding geometry. These are the only intervals that need to be checked in the other reference images[5] as shown in figure 7(b).

### 5.2 Sampling the Light Source

Stochastic raytracers typically distribute shadow rays independently over the entire light source for each surface point in the scene. Yet, visibility as a function of the position of the surface point tends to change slowly. Moreover, object-space rendering algorithms such as z-buffered scan conversion or REYES [4] shade a single surface at a time, moving from surface point to surface point in some continuous order. Thus, the surface points are generally processed in groups that lie close to one another. In this section we develop a technique for exploiting this coherence to reduce the region of the light source for which we cast shadow rays.

As described in section 3, sampling the light source requires a mapping from the unit square to the surface of the light source. We consider the set of cells stratifying the unit square as a 2D image array covering the light source. The *visibility image* is a binary image storing whether or not each light source cell is blocked. The key idea of our algorithm is to predict the visibility image for surface point $p_i$ based on the light source visibility for the previous surface points. The algorithm is described in the following pseudocode:

**procedure SoftShad**

```
1   BlockerPtsList ← ∅
2   foreach surface point p_i
3     Predict(p_i,BlockerPtsList,VisImg,cellsTodo,cellsUsePrev)
4     cellsDone ← ∅
5     while not empty(cellsTodo)
6       l_j ← cellsTodo.dequeue
7       blockPt = Trace(ray(p_i,l_j))
8       Update(BlockerPtsList,blockPt,l_j)
9       if(isBlocked(blockPt) != VisImg[l_j])
10        VisImg[l_j] ← isBlocked(blockPt)
11        foreach cell l_k adjacent to l_j
12          if(l_k in cellsUsePrev)
13            cellsTodo.enqueue(l_k)
14      cellsDone.enqueue(l_j)
15    color_i ← Shade(VisImg)
```

The BlockerPtsList stores the intersection point for each occluded shadow ray we have traced. Initially it is the empty set (line 1) and we insert new blocker points every time we trace a shadow ray (line 7). Blocker points that are no longer relevant are removed in the Predict procedure (line 3) as described in section 5.2.2.

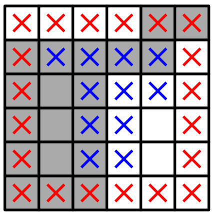
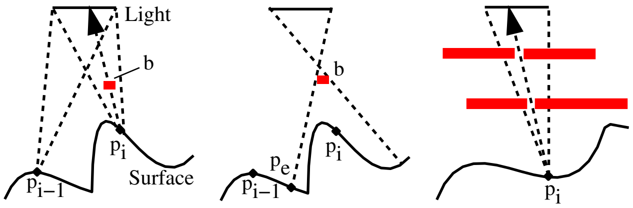


Figure 8: *Predicted Visibility Image. Gray boxes represent occluded cells, and white boxes represent unoccluded cells. Each cell marked with an X is initially placed in the cellsTodo list by the Predict procedure. Blues X's represent cells at edges between occluded and unoccluded regions, while red X's represent cells at the exterior edges of the light source. Cells that do not contain an X are initially placed in the cellsUsePrev list.*

We generate a predicted visibility image by projecting each point in the current BlockerPtsList onto the light source. Assuming a planar light source, the appropriate projection matrix is formed using $p_i$ as the center of projection and the light source as the image. In the predict procedure, we also build two lists of light source cells based on our confidence of the predicted visibility. We assume that changes in visibility are most likely to occur in two places—at the boundaries between blocked and unblocked regions on the interior of the light source and at the exterior edges of the light source. If the predicted visibility for each interior cell is the same as the predicted visibility value of all of its neighboring cells, we are confident of the prediction and we add the cell to the cellsUsePrev list. Otherwise, the cell is at an edge between blocked and unblocked regions, confidence is low, so we add the cell to the cellsTodo list. Since we cannot examine a complete neighborhood for the cells on the exterior edges of the light source, we add all of these cells to the cellsTodo list. This is illustrated in figure 8. If the BlockerPtsList is empty, we put all of the light source cells in the cellsTodo list.




(a) Missed Blocker (b) Shadow Volume of b (c) Missed Hole

Figure 9: *Prediction errors. In (a) the blocker point b is not seen from surface point $p_{i-1}$. We predict the light is completely unblocked at $p_i$ and never trace the ray through b. By increasing the surface sampling density so that $p_i$ falls on $p_e$, the edge of the shadow volume due to b (shown in(b)), the blocker is found. In (c), a ray starting at $p_i$ passes through the aligned holes in the blockers. To ensure that this ray is traced, we must increase the light source sampling density.*

The main loop of the algorithm (lines 5 – 14) traces shadow rays for each light source cell in the cellsTodo list. When the value of a traced ray differs from the prediction held in the corresponding visibility image cell (line 9) we move any adjacent cell (considering all 8 neighbors) in the cellsUsePrev list to the cellsTodo list. As we find prediction errors, we spread out the lower confidence values to the neighboring cells using an 8-connect flood-fill algorithm. The main loop ends when we have processed all cells in the cellsTodo list, and we then shade the surface point using the current version of the visibility image.

#### 5.2.1 Prediction Errors

As shown in figure 9(a),(c) there are two types of errors that may occur with our prediction technique; 1) *missed blocker:* a ray inside a region of the frustum predicted as unblocked is actually blocked and 2) *missed hole:* a ray inside a region of the frustum predicted as blocked is unblocked. Both types of prediction errors can lead to visible artifacts if the ray containing the blocker or the hole is predicted with high confidence and is not traced.

There is a fundamental difference between these two errors. The missed blocker error can be diminished by increasing surface sampling density. Reducing the missed hole error requires increased light sampling. To better understand this difference, we simplify the situation. Assume the light source is entirely visible from $p_{i-1}$ and there is some small object $b$ blocking a ray between $p_i$ and some point on the interior of the light. The missed blocker error occurs because $b$ is completely inside the frustum of $p_i$ but outside the frustum of $p_{i-1}$, as shown in figure 9(a). Thus, our prediction for $p_i$ requires that we only trace rays to the edges of the light source and since each of these traced rays agrees with our predicted visibility (unblocked), we never trace the ray through $b$.

Consider however, the shadow volume due to $b$ and its intersection with the surface $P$. There must be some point $p_e$ on the edge of the intersection that lies between $p_i$ and $p_{i-1}$. Since $p_e$ is on the surface of the shadow volume, the ray with origin at $p_e$ and passing though $b$ must intersect the edge of the light source (figure 9(b)). Since we always trace rays for points on the edge of the light

[5] Chang[1] also computes occlusion intervals, but uses them to invalidate intersections along the epipolar ray, rather than as an acceleration technique.

source we correctly find the blocker $b$. As long as the surface points are sufficiently close to one another, this property holds regardless of the direction (on the surface) in which we approach $p_i$. Therefore, if we increase the surface sampling density, we reduce the missed blocker errors. A similar argument applies when regions of the light source are empty rather than the entire light source. Note that, if the distance between the current surface point and the previous surface point is larger than a given tolerance, we disregard the predicted visibility and trace all the shadow rays.

While it may seem a similar argument would apply to missed holes, there is a special case when multiple holes align to allow shadow rays to reach the interior of the light unoccluded. Suppose as in figure 9(c), at surface point $p_i$ the holes in the two blockers align so that a single shadow ray to the interior of the light source is unblocked. For every surface point in any neighborhood of $p_i$, every shadow ray to the light is blocked. There is no surface point for which a shadow ray to an exterior edge of the light "sees" the hole. To ensure this ray is traced, we must increase light source sampling density. This in turn increases the precision of our predicted visibility since our blocker points list samples the surfaces more finely and we project them onto the light more accurately.

### 5.2.2 Updating the Blocker Points List

A drawback of our coherence-based sampling approach is that we must store and projectively warp each point in the blocker points list. Storing the list requires that the raytracer explicitly compute intersection points. Therefore, we cannot use the hierarchical connectedness optimization described in section 5.1.1. While warping a single point is relatively fast, if the blocker points list is large, the time to warp every point in the list can be significant.

We have designed several optimizations that limit the size of the blocker points list. We remove any blocker point that projects to a point outside the light source. This maintains some locality within the list and ensures that it stays relatively small. Often, multiple blocker points will warp to the same cell of the light. This is especially true for cells at edges between blocked and unblocked regions, since these are exactly the cells that we trace for every surface point. Such blocker points essentially provide redundant information and generally lie very close to each other in object space. One option is to keep only one of the blocker points that warp to the same light source cell. We generalize this approach. In each cell, we maintain a higher resolution grid (typically 3x3) and within each high resolution cell we keep only a single blocker point. With these optimizations, the size of the blocker points list is at most the light sampling resolution times the high resolution grid resolution. In contrast, Hart et al. [10] precompute and store blocker lists of geometry for each pixel in the final image. Their lists can contain redundancies and the size of each list is limited only by image resolution and the total number of shadow rays traced in their precomputation phase.

As a side benefit, we can use our high resolution grid to directly insert cells into the cellsDone list. Before prediction, we compute a jittered sample location for each cell of the light source. During the prediction phase, if a blocker point warps to a high-resolution cell containing a sample location, we place the corresponding light source cell in the cellsDone list. In this case, the blocker point lies close enough to the ray we would trace that we can assume the ray is blocked.

## 6 Results

Our results are presented in figures 1, 12, 13, and 14. Each image was originally rendered at 512x512 pixels and uses the equivalent of 256 light source samples. Some images have been cropped to preserve space, but the timings in figure 10 are for the entire uncropped image. The light is rectangular for each of these scenes. It lies directly above the center of the geometry for the plant and the flower, and is displaced toward the head of the dragon. We have deliberately shown extremely complex examples. The light position is chosen so that almost all the geometry either casts shadows or is itself shadowed. Also, our viewpoints have been picked so that almost the entire geometry and shadow region is visible. Soft shadows for these scenes would be expensive to generate by object-space methods, including the interactive approaches of Herf and Heckbert [12] or Soler and Sillion [27], since the scenes contain a large number of triangles, and complex self-shadowing interactions. Further, the triangles are typically very small, especially for the dragon and the flower, making these scenes difficult to handle by an approach that stores geometric information per pixel such as that of Hart et al. [10]. For quality comparisons, we have included images generated by rayshade.

**Performance: Layered Attenuation Maps** Our layered attenuation map approach is implemented using standard graphics hardware under OpenGL as the base renderer. No special features, such as antialiasing, or shadow mapping extensions, are used. The hardware is used only for rendering shadowless images; the other parts of our precomputation and display phases work entirely in software. The running times for the precomputation phase on an SGI Onyx 2 Infinite Reality, and with an LDI size of 512x512, are shown in the left of figure 10. The main operations are software warping of rendered images and insertion into the LDI (lines 5 and 6 of the precomputation pseudocode), so the running time is largely independent of scene complexity. We see that coarse shadows (64 light samples), that may be suitable for most previewing work, can be precomputed in between 5 and 10 seconds, while higher-quality versions (256 samples) can be precomputed in about half a minute. Regardless of light sampling density, images with soft shadows can be displayed interactively at 5-10 frames per second, with the appropriate attenuation being chosen in real time in software.

**Performance: Coherence-Based Raytracing** Our coherence based sampling technique is designed to work with object-space rendering algorithms that shade and render the scene surface by surface. We have implemented our algorithm within the shading pass of Pixar's PhotoRealistic Renderman which is based on the REYES [4] object-space rendering algorithm. In the REYES algorithm, geometric primitives are split into micropolygons that are shaded and then scan converted to create the final image. Micropolygons are shaded in a coherent manner, in groups that lie close to one another both in screen space and on the geometric primitive. Thus, lighting tends to change slowly from micropolygon to micropolygon. Since the micropolygons are shaded before computing their visibility, we compute soft shadows for every surface in the viewing frustum regardless of whether it is visible or not[6]. In contrast, standard raytracing renderers and the post-shading approach used with layered attenuation maps only perform the soft shadow computation on visible points. Therefore, direct comparisons between the running times of our coherence-based approach and other methods is difficult.

Running times and speedups for our coherence-based raytracing algorithm are presented in figure 10. The precomputation consists of rendering shadow maps at a resolution of 1024x1024 from the four corners of the light and constitutes a small fraction of the total running time. Note that the no-acceleration column refers to standard hierarchical image-based raytracing without any of our new acceleration techniques. Adding our raytracing acceleration techniques alone, without coherence-based sampling, provides a fairly consistent speedup of around 2.20x across the three scenes, regardless of light source sampling density. Much of this performance increase is due to the clear miss optimization which allows

[6] Rendering the plant, flower and dragon in Renderman at 512x512 image resolution requires shading 765134, 1344886 and 772930 surface samples respectively.

| Layered Attenuation Maps | | | |
|---|---|---|---|
| Scene | Triangles | # Light Samp. | Precom. Time (s) |
| Plant | 5247 | 64 | 6.0 |
| | | 256 | 22.4 |
| Flower | 35109 | 64 | 7.4 |
| | | 256 | 28.1 |
| Dragon | 77890 | 64 | 7.7 |
| | | 256 | 29.4 |

| Coherence-Based Raytracing | | | | | | |
|---|---|---|---|---|---|---|
| Scene | Precom.. Time (s) | # Light Samp. | No Accel t(1000 s) | RT accel speedup | Accel + Coherence speedup | rays |
| Plant | 236 | 64 | 8.36 | 2.14x | **7.74x** | 28.80 |
| | | 256 | 33.30 | 2.16x | **8.52x** | 88.74 |
| | | 1024 | 131.96 | 2.16x | **10.12x** | 287.59 |
| Flower | 332 | 64 | 4.59 | 2.15x | **3.64x** | 34.86 |
| | | 256 | 18.20 | 2.18x | **3.82x** | 120.29 |
| | | 1024 | 71.42 | 2.14x | **3.95x** | 424.24 |
| Dragon | 140 | 64 | 3.92 | 2.28x | **10.59x** | 27.57 |
| | | 256 | 15.42 | 2.27x | **12.96x** | 79.86 |
| | | 1024 | 61.93 | 2.27x | **15.18x** | 249.45 |

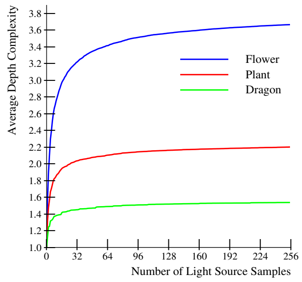



Figure 10: **Left:** *Precomputation times (in seconds) for our layered attenuation map approach on an SGI Onyx2 Infinite Reality. The major operations are image-based—warping rendered images and inserting into the LDI—so running time grows slowly with increased geometric scene complexity. The scenes can be displayed at 5-10 frames per second after precomputation is completed.* **Middle:** *Performance of Coherence-Based Raytracing on a 300 Mhz processor. The no acceleration column refers to hierarchical raytracing without any of our acceleration techniques and provides a baseline set of running times. The next two columns show the speedups achieved by including our raytracing accelerations and then both the accelerations and the coherence-based sampling. The final column shows the average number of rays actually traced with coherence-based sampling.* **Right:** *Increase in average depth complexity over non-empty LDI pixels in our layered attenuation map as a function of number of light samples. After the first few samples, the complexity increases very slowly. The flower has a lot more very small geometry, and so the final average visible depth complexity is higher, though even in this case, it is fairly low. This graph is a more meaningful measure of scene complexity than simple polygon count; the size of the LDI is proportional to this complexity.*

the algorithm to efficiently process many of the unoccluded shadow rays without visiting all four shadow maps.

As we increase the number of light samples, sizes of the fully blocked and unblocked areas on the light source grow faster than the lengths of the edges between them. We exploit this perimeter versus area growth ratio with our coherence-based sampling algorithm since we initially trace shadow rays only at the edges between blocked and unblocked regions. For all three scenes, we see that the speedups increase as we increase the light source sampling rate. Similarly, the ratio of average rays traced to the total number of light source samples decreases. If we increase the light source sampling rate by 4x, the running time of the original hierarchical raytracer and even that of the version with accelerations, but without coherence-based sampling, increase by roughly 4x. When we add coherence-based sampling however, the increase in running time is significantly smaller (i.e. the speedup increases as we increase the light source sampling density, especially for the plant and dragon). We have observed that adding coherence-based sampling causes no noticeable difference in image quality compared to hierarchical raytracing without coherence-based sampling (figure 13).

The speedup due to coherence-based sampling is relatively low for the flower scene. We believe this is largely due to the thin (typically much smaller than a pixel) geometry of the branches in the head of the flower. For points in the penumbra region on the ground plane, there are few large regions of the light source that are either fully occluded or unoccluded. Since most of the light source is at an edge between occluded and unoccluded regions, the coherence-based sampling approach provides little benefit. In contrast, the coherence-based approach achieves the largest speedups for the dragon scene. Although the triangles are still very small, this scene tends to contain larger blocked and unblocked regions on the light source.

Asymptotically, coherence-based sampling will make the number of rays traced proportional to the square-root of the number of light samples, rather than the number of samples (because, in the limit, the number of rays traced depends on the length of the perimeter between blocked and unblocked regions, not area). In the limit, we therefore expect a 4x increase in light source sampling to cause the number of rays actually traced to increase by only 2x, not 4x. At the light source sampling densities we've tested, we see an increase of a little more than 3x.

### 6.1 Discussion

In some respects, soft shadow representation is a model application for the use of image-based representations.

**Depth Complexity:** Since the light is localized in space, samples on it are close together, and visibility does not change significantly between the samples. This means the depth complexity of the completely and partially visible portions of the scene, as seen from the light, is very low, as seen in the graph in figure 10. Note that geometry that is occluded from everywhere on the light is completely excluded from our image-based representations. Therefore, as compared to an LDI that represents the entire scene, the LDIs in our layered attenuation map approach require significantly fewer layers. Furthermore, the complexity of the representation increases very slightly after the first few samples on the light. In the context of coherence-based raytracing, this low complexity means a sparse set of shadow maps suffices to produce high-quality results.

**Sampling:** Since our shadow maps are rendered from points on the light, surfaces whose normals make large angles to that of the light are sampled poorly. Since only a single LDI is used, this is more of an issue for layered attenuation maps than for coherence-based raytracing. However, these surfaces will also usually be very dimly lit—because of cosine falloff in light intensity—diminishing the visibility of sampling artifacts, as seen in the left of figure 11.

**Artifacts:** The images produced by both of our algorithms look very plausible when seen by themselves, and are also very close to those produced by rayshade. However, our layered attenuation map method produces some artifacts:

- **Insufficient Depth Sampling:** If the LDI samples depths of some surfaces insufficiently, we will not be able to tell whether a point in the final image occurs on, above, or below the surface, as seen in the left of figure 11. This is less of a problem with coherence-based raytracing since we do not resample into a single LDI. Therefore, those shadow maps that better sample the surfaces in question are used to generate the shadows. Note that since both of our algorithms require the use of error tolerances, we cannot use mid-point shadows [31]. Therefore, both methods require the user to specify a value for shadow bias.

- **Insufficient Attenuation Map Sampling:** Even if a surface has constant depth, insufficient sampling can cause blockiness when the attenuation map is magnified and reprojected. As shown in the right of figure 11, simple bilinear filtering of four neighboring attenuation map values—analogous to percentage-closer filtering [23]—for each image pixel can diminish the visibility of the artifacts. However, the results may still be inaccurate because a limited number of samples

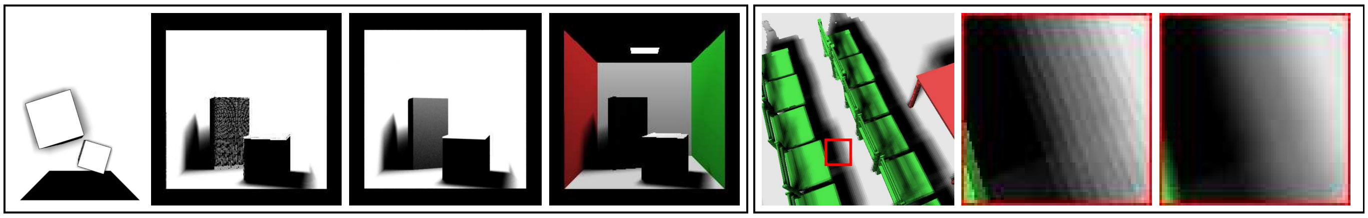


Figure 11: *Artifacts.* **Left:** *Insufficient depth sampling. Leftmost is the precomputed texture for layer 1 using layered attenuation maps. The sides of the large box are very poorly sampled. Next is the attenuation map, which has artifacts on the poorly sampled side. Similar artifacts are produced by the SGI (hard) shadow map hardware. However, coherence-based raytracing (third image) is able to do a much better job. Including cosine-falloff of light intensity, the final image produced by our layered attenuation map algorithm is shown rightmost, and the artifacts are considerably diminished.* **Right:** *Insufficient attenuation map sampling. Left is a thumbnail of the scene. The middle shows an extreme closeup of the ground which indicates blocky textures from magnification and reprojection. In the rightmost image, simple bilinear filtering reduces the perceptibility of the artifacts. Models courtesy of Peter Shirley.*

are available for reconstruction. Since coherence-based raytracing does not precompute textures, and thereby predetermine their resolution and sampling pattern, this is not an issue for that method.

- **Banding:** Since the same light samples are used for all surfaces, banding may occur as seen in figure 15. Note that banding is present in the attenuation map, and can therefore not be removed simply by post-filtering on the attenuation map, similar to that discussed above. In the coherence-based raytracing method, banding is replaced by noise since light samples are decorrelated for all surface points.

These artifacts are somewhat more apparent in high resolution images than at the size of the images in the printed version of this paper. To clearly show the artifacts here, we have zoomed very close. Similarly, to show banding, we have reduced the number of light source samples in figure 15 only.

As can be seen from the results, sampling artifacts are generally not a problem with coherence-based raytracing, so this technique is suitable for producing final high-quality images.

## 7 Conclusions and Future Work

We have described two efficient image-based methods for computing soft shadows. These methods can be seen as extensions of an extremely popular technique for hard shadows—shadow maps—and produce results significantly faster than traditional approaches. The algorithms can be combined for rapid previewing followed by efficient high-quality rendering. We have also demonstrated how soft shadows are an ideal application for image-based approaches. As future work, we would like to investigate better sampling strategies, the use of adaptive biases, and hardware implementation of the display phase for layered attenuation maps.

**Acknowledgements:** We are indebted to Tom Lokovic for developing the hierarchical connectedness acceleration for the raytracing approach and for implementing an initial version of coherence-based raytracing in Pixar's Photorealistic Renderman. Special thanks to Reid Gershbein, Tony Apodaca, Mark VandeWettering and Craig Kolb for initial discussions, and to Bill Mark, James Davis, and Pat Hanrahan for reviewing early drafts. The Siggraph reviewers provided many helpful comments.

## References

[1] L. W. Chang. Combining multiple reference images in an inverse warper. M.eng. thesis, MIT, 1998.

[2] S. E. Chen and L. Williams. View interpolation for image synthesis. In *SIGGRAPH 93 proceedings*, pages 279–288, 1993.

[3] M. F. Cohen and J. R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press, 1993.

[4] R. L. Cook, L. Carpenter, and E. Catmull. The Reyes image rendering architecture. In *SIGGRAPH 87 proceedings*, pages 95–102, 1987.

[5] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *SIGGRAPH 84 proceedings*, pages 137–145, 1984.

[6] F. C. Crow. Shadow algorithms for computer graphics. In *SIGGRAPH 77 proceedings*, pages 242–248, 1977.

[7] G. Drettakis and E. Fiume. A fast shadow algorithm for area light sources using backprojection. In *SIGGRAPH 94 proceedings*, pages 223–230, 1994.

[8] B. Guo. Progressive radiance evaluation using directional coherence maps. In *SIGGRAPH 98 Proceedings*, pages 255–266, 1998.

[9] P. E. Haeberli and K. Akeley. The accumulation buffer: Hardware support for high-quality rendering. In *SIGGRAPH 90 proceedings*, pages 309–318, 1990.

[10] D. Hart, P. Dutre, and D. P. Greenberg. Direct illumination with lazy visibility evaluation. In *SIGGRAPH 99 proceedings*, pages 147–154, 1999.

[11] P. Heckbert. Discontinuity meshing for radiosity. In *Eurographics Rendering Workshop 92 proceedings*, pages 203–226, May 1992.

[12] P. Heckbert and M. Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, Carnegie Mellon University, 1997.

[13] A. Heirich and L. Moll. Scalable distributed visualization using off-the-shelf components. In *Symposium on Parallel Visualization and Graphics*, pages 55–60, 1999.

[14] B. Keating and N. Max. Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. In *Eurographics Rendering Workshop 99 proceedings*, 1999.

[15] S. Laveau and O. Faugeras. 3D scene representation as a collection of images and fundamental matrices. Technical Report 2205, INRIA, February 1994.

[16] D. Lischinski and A. Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Eurographics Rendering Workshop 98 proceedings*, pages 301–314, 1998.

[17] D. Lischinski, F. Tampieri, and D. P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications*, 12(6):25–39, November 1992.

[18] R.W. Marcato, Jr. Optimizing an inverse warper. M.eng. thesis, MIT, 1998.

[19] W. R. Mark, L. McMillan, and G. Bishop. Post-rendering 3D warping. In Michael Cohen and David Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 7–16, April 1997.

[20] L. McMillan. *An Image–Based Approach to Three–Dimensional Computer Graphics*. Phd thesis, Department of Computer Science, University of North Carolina, 1997.

[21] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *SIGGRAPH 95 Proceedings*, pages 39–46, 1995.

[22] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH 89 proceedings*, pages 41–50, 1989.

[23] W. T. Reeves, D. H. Salesin, and R. L. Cook. Rendering antialiased shadows with depth maps. In *SIGGRAPH 87 proceedings*, pages 283–291, 1987.

[24] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. E. Haeberli. Fast shadows and lighting effects using texture mapping. In *SIGGRAPH 92 proceedings*, pages 249–252, 1992.

[25] J. W. Shade, S. J. Gortler, L. He, and R. Szeliski. Layered depth images. In *SIGGRAPH 98 proceedings*, pages 231–242, 1998.

[26] C. Soler and F. X. Sillion. Automatic calculation of soft shadow textures for fast, high-quality radiosity. In *Eurographics Rendering Workshop 98 proceedings*, pages 199–210, 1998.

[27] C. Soler and F. X. Sillion. Fast calculation of soft shadow textures using convolution. In *SIGGRAPH 98 proceedings*, Computer Graphics Proceedings, Annual Conference Series, pages 321–332, 1998.

[28] M. Stark, E. Cohen, T. Lyche, and R. F. Riesenfeld. Computing exact shadow irradiance using splines. In *SIGGRAPH 99 proceedings*, 1999.

[29] A. J. Stewart and S. Ghali. Fast computation of shadow boundaries using spatial coherence and backprojection. In *SIGGRAPH 94 proceedings*, pages 231–238, 1994.

[30] L. Williams. Casting curved shadows on curved surfaces. In *SIGGRAPH 78 proceedings*, pages 270–274, 1978.

[31] A. Woo. *Graphics Gems III*, chapter The Shadow Depth Map Revisited, pages 338–342. Academic Press, 1992.

[32] A. Woo, P. Poulin, and A. Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, November 1990.

[33] H. Zhang. Forward shadow mapping. In *Eurographics Rendering Workshop 98 proceedings*, pages 131–138, 1998.

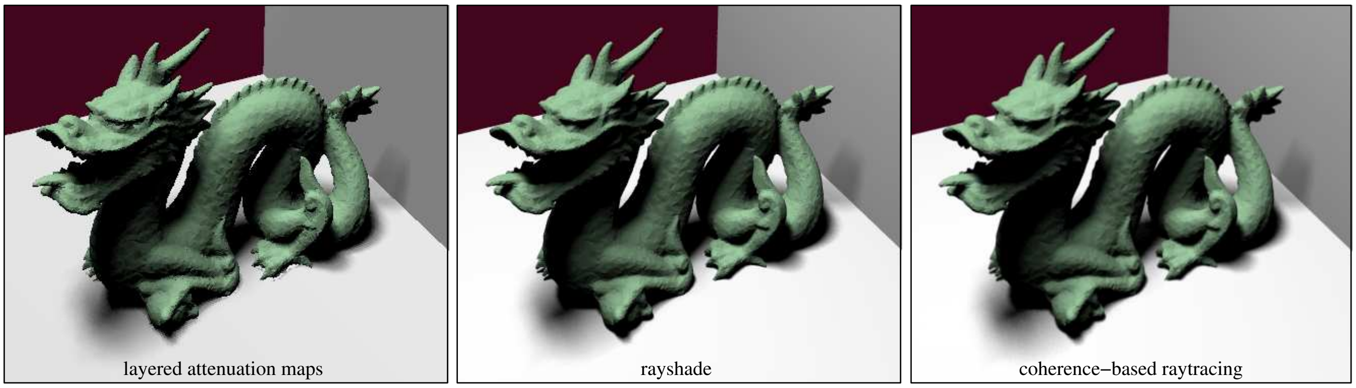
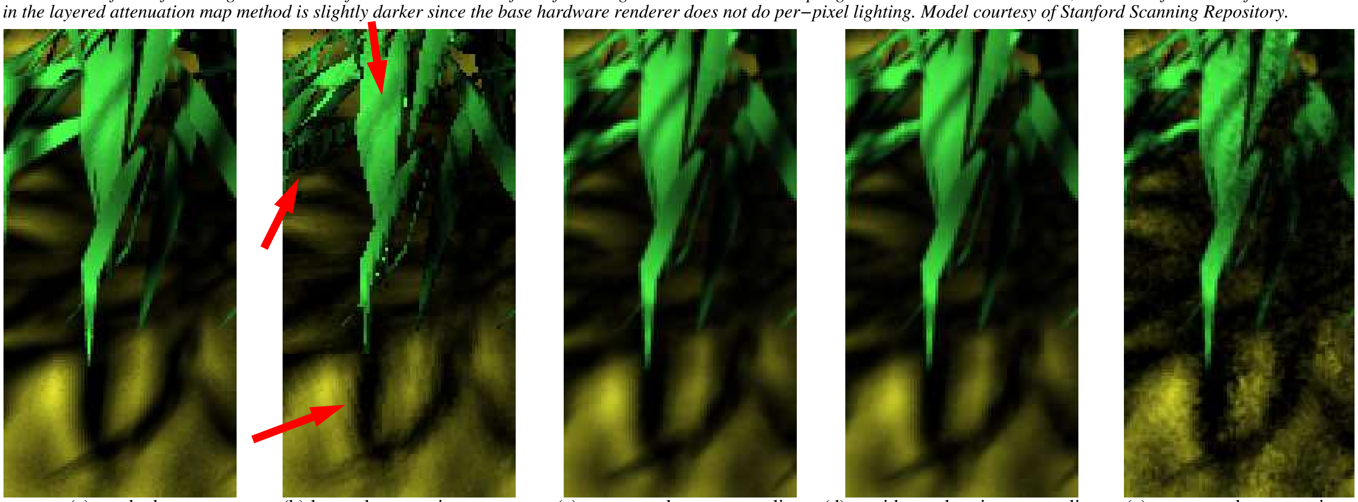
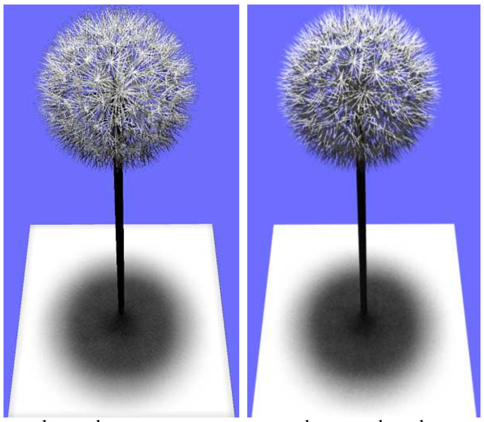



Figure 12: *Soft shadows for a dragon model shaded using a Lambertian model. The rayshade and layered attenuation map images were rendered with 256 light source samples. The coherence−based raytracing image used an average of only 79.86 light source samples. Splatting in the layered attenuation map method slightly increases the brightness of the shadow at the front of the dragon and the artifacts around the rear foot of the dragon are due to undersampling and shadow bias errors. Also, the color of the white floor and wall in the layered attenuation map method is slightly darker since the base hardware renderer does not do per−pixel lighting. Model courtesy of Stanford Scanning Repository.*

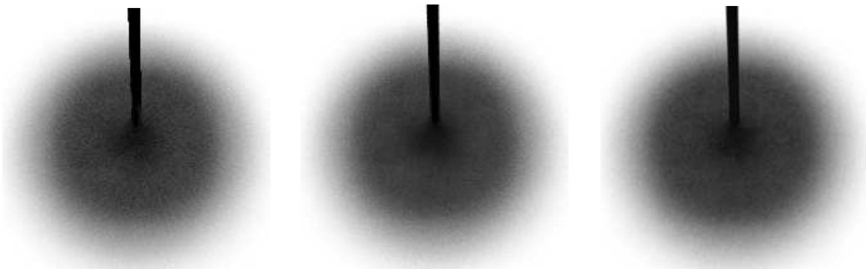


(a) rayshade | (b) layered attenuation maps | (c) rt no accel. − same quality 33300 seconds, 256 rays | (d) rt with acceleration + sampling 3908 seconds, 88.74 rays on avg. | (e) rt no accel. − same time 3980, seconds, 20 rays

Figure 13: *Closeups for the plant in Figure 1. The coherence−based raytracing image (d) is almost indistinguishable from that without acceleration (c), and both are very close to the image produced by rayshade (a). Our coherence−based method is 8.52 times faster than the unaccelerated hierarchical raytracer (c). An equal time comparison is provided in (e). Note that the times listed are for the entire image, not just the closeups. At the scale of the closeup, there are some artifacts for our layered attenuation map approach (b), as indicated by the red arrows. However, at normal scales as in Figure 1 these artifacts are less prominent, and are usually tolerable for interactive applications.*

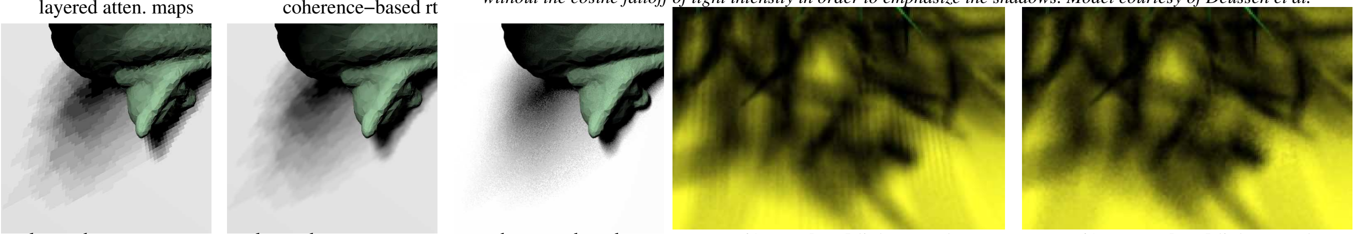


layered atten. maps | coherence−based rt



layered attenuation maps | rayshade | coherence−based raytracing

Figure 14: *The flower is an extreme test case for our algorithms since there are a large number of long thin triangles significantly smaller than a pixel. This causes the base hardware renderer for our layered attenuation map approach to exhibit serious aliasing artifacts in the "head" of the flower. Nevertheless, both methods capture the complex self shadowing effects that cause the bottom of the "head" to be darker than the top. Our ground shadows closely match the shadow produced by rayshade. Many other soft shadow methods would have significant difficulty in rendering this shadow efficiently and correctly. Note that as for the plant we render these images without the cosine falloff of light intensity in order to emphasize the shadows. Model courtesy of Deussen et al.*



layered atten. maps 16 light samples − unfiltered | layered atten. maps 16 light samples − filtered | coherence−based rt 16 light samples − noise | rt − 64 **correlated** light samples | rt − 64 **uncorrelated** light samples

Figure 15: *Banding Artifacts.* ***Left:*** *Banding artifacts at the front of the dragon are due to the fixed light source sampling of our layered attenuation map approach. Filtering cannot eliminate the banding artifacts. The banding is replaced by noise when the light source samples are chosen independently for each surface point as with our coherence−based raytracing approach.*