A Two-Pass Realistic Image Synthesis Method for Complex Scenes

Kurt Zimmerman and Peter Shirley

Department of Computer Science
Indiana University
Bloomington, IN 47405

Program of Computer Graphics
Cornell University
Ithaca, NY 14853

Abstract

This paper presents a survey of two-pass global illumination algorithms that use a local pass (gather) and shadow ray optimization techniques as well as a new two-pass algorithm which relies on spatial coherence to efficiently handle the problem of numerous luminaires. The algorithm presented operates on scenes with Lambertian reflectors and luminaires with arbitrary emission distributions. A radiosity prepass is performed on a simplified version of the scene. Bright reflecting surfaces are reclassified as light sources. A spatial data structure is then built that associates bright visible surfaces to regions in the environment. The visibility of these surfaces is probabilistically estimated. Finally, a view-dependent "gather" is performed at each pixel. This gather is made significantly more efficient than previous algorithms because of careful integration based on the contents of the spatial data structure.

CR Categories and Subject Descriptors: I.3.0 [Computer Graphics]: General; I.3.6 [Computer Graphics]: Methodology and Techniques.

Additional Key Words and Phrases: Radiosity, realistic image synthesis, global illumination, Monte Carlo Integration.

1 Introduction

In 1986 Kajiya introduced the rendering equation[8] and with it introduced an algorithm that, given sufficient computing time, can accurately solve all rendering problems that assume geometric optics. This has led to some speculation that global illumination is a solved problem[4]. However, a sufficient amount of computing time for Kajiya's algorithm is not practical for most rendering applications. Thus, since the introduction of Kajiya's algorithm, the goal of global illumination community has been a simple one: develop algorithms which can render realistic images to some accuracy goal in a practical amount of time. While there have been successful algorithms for simple scenes, most of these algorithms do not scale well when the environment becomes complex. So in spite of the volume of literature devoted to realistic rendering, a general and practical algorithm for producing physically accurate images of complex environments does not exist.

In scenes with specular surfaces, "two-pass" methods have been used to add $eye\, S^{\star} D S^{\star}\, light$ paths to illumination ray traced scenes[2], and to add $eye\, S^{\star} D^{\star}\, light$ paths when rendering precomputed radiosity solutions [23]. If the radiosity method has form-factors that account for specular transport, then all diffuse-specular transport chains are accounted for[12, 1, 19, 6]. Some authors note that the direct lighting can cause more detail in images, and use the precomputed solution only for indirect lighting [26, 16, 3].

A potential problem with many two-pass methods is that a texture map or mesh must be used to represent the irradiance on diffuse surfaces. This limits the utility of such methods for complex scenes. Rushmeier[1] observed that all lighting could be recomputed by "gathering" from a radiosity solution at each point[15]. This method has been used to generate images using piecewise-constant radiosity solutions[13, 20]. Lischinski et al. used a similar non-probabilistic gather to generate better meshes for hardware display[11]. More recently, Rushmeier et al.[14] observed that gathering from a piecewise constant radiosity solution could be done more efficiently if the radiosity solution were carried out for a geometrically simplified environment. They also observed that gathering from nearby objects gives noticeable artifacts. To get around this, they used Monte Carlo path tracing[8] where large errors where expected from the gather method.

In this paper, we introduce a two-pass algorithm: a radiosity prepass followed by a view-dependent "gather". The algorithm differs from previous two-pass methods; it is designed for environments with tens of thousand of primitives and hundreds of luminaires. This is the first attempt to combine geometric simplification[14] with Monte Carlo shadow ray optimization[17, 18]. More than a simple composition, the algorithm employs visibility and lighting coherence both to determine and to vary in space the regions upon which explicit gathering (shadow rays) and implicit gathering (reflection rays) are performed. This allows the algorithm to restrict the number of explicit transfer calculations down to one explicit gather and one implicit gather per viewing ray. In addition, visibility estimation and visibility coherence is used to reduce the probability of querying invisible or partially visible luminaires. As rays are not being sent to every luminaire, our algorithm will reach an acceptable image for complex scenes tens to thousands of times faster than previous two-pass solutions. The main limitation of this new method is that it is limited to predominantly diffuse scenes.

The term "two-pass method" can apply to any algorithm that performs two passes. For the purposes of this paper we restrict our discussion to those algorithms which use a view-independent pass followed by a view-dependent pass. A further restriction is that the values computed in the view-independent pass are not applied directly to the final image but are used to help calculate the displayed values. Because two-pass methods that use a "gather" for diffuse surfaces are relatively recent, we will begin with an overview of two-pass methods and gathering strategies in Section 2. Next, in Section 3, we present our new algorithm followed by the implementation, sample images, and results in Section 4. Finally we discuss the merits of our algorithm in Section 5.

[1] Although two-pass "gather" methods have recently become popular, it is often overlooked that they were introduced in Rushmeier's 1988 thesis. The idea of a gather is also closely related to Cohen et al.'s 1986 paper that used large patches to compute irradiance at each small element.

2 Background

In this section we review path tracing and two-pass methods that gather at diffuse surfaces. We begin in Section 2.1 by reviewing the rendering equation and the basis for two-pass methods. In Section 2.2 we review path tracing as it is used in practice. In Section 2.3 we review the *implicit gather* which gathers in directional space. In Section 2.4 we review the *explicit gather* which gathers in geometry space. In Section 2.5 we review Rushmeier et al.'s GSII algorithm. In Section 2.6 we review previous methods that can be directly applied to explicit gathers.

2.1 Rendering Equation

We now examine the rendering equation for diffuse reflectors and general emitters. We review both the explicit (area-based) form presented by Kajiya [8], and the implicit (directional) form presented by Immel [7]. The implicit form is:

$$L_s(\mathbf{x},\hat{\omega}) = L_e(\mathbf{x},\hat{\omega}) + \frac{R(\mathbf{x})}{\pi}\int_{\mho} L_f(\mathbf{x},\hat{\omega}')(-\hat{\omega}'\cdot\hat{n})d\sigma(\hat{\omega}') \qquad (1)$$

where $L_s(\mathbf{x},\hat{\omega})$ is the surface radiance of $\mathbf{x}$ in direction $\hat{\omega}$, $L_e(\mathbf{x},\hat{\omega})$ is the emitted surface radiance in direction $\hat{\omega}$ at $\mathbf{x}$, $\mho$ is the unit hemisphere of incoming directions oriented about $\hat{n}$, $R(\mathbf{x})$ is the reflectivity at $\mathbf{x}$, $L_f(\mathbf{x},\hat{\omega}')$ is the field radiance from direction $\hat{\omega}'$ incident at $\mathbf{x}$, and σ is the solid angle measure. The explicit form is:

$$\begin{aligned} L_s(\mathbf{x},\hat{\omega}) &= L_e(\mathbf{x},\hat{\omega}) + \\ &\frac{R(\mathbf{x})}{\pi}\int_{\mathcal{X}} g(\mathbf{x},\mathbf{x}')L_s(\mathbf{x}',\hat{\omega}')(-\hat{\omega}'\cdot\hat{n})\frac{(\hat{\omega}'\cdot\hat{n}')}{\|\overrightarrow{\mathbf{x}-\mathbf{x}'}\|^2}\,dA(\mathbf{x}') \end{aligned} \qquad (2)$$

where $\mathcal{X}$ is the set of all points on surfaces and $g(\mathbf{x},\mathbf{x}')$ is the *geometry term*, which is zero if there is an obstruction between $\mathbf{x}$ and $\mathbf{x}'$ and one otherwise, $\|\overrightarrow{\mathbf{x}-\mathbf{x}'}\|$ is the distance between $\mathbf{x}$ and $\mathbf{x}'$, and A is the area measure, Figure 1.

Equations 1 and 2 are equivalent, but suggest different solution strategies. Equation 1 suggests sampling directions, and Equation 2 suggests sampling patches. Clearly we cannot sample every patch at every pixel for complex scenes, but we cannot naively sample directions because small bright surfaces will cause high errors. We want to explicitly sample a few very important surfaces, and directionally sample to estimate the contribution of all other surfaces.

2.2 Monte Carlo Path Tracing (MCPT)

A naive Monte Carlo[2] solution to an integral $I = \int_S f(\tau)\ d\tau$, where $\tau \in S$ can be expressed as

$$I \approx \sum_{i=1}^{n} \frac{f(\tau_i)}{p(\tau_i)}.$$

[2] Readers who are unfamiliar with Monte Carlo techniques should refer to a standard text such as Kalos and Whitlock[9].

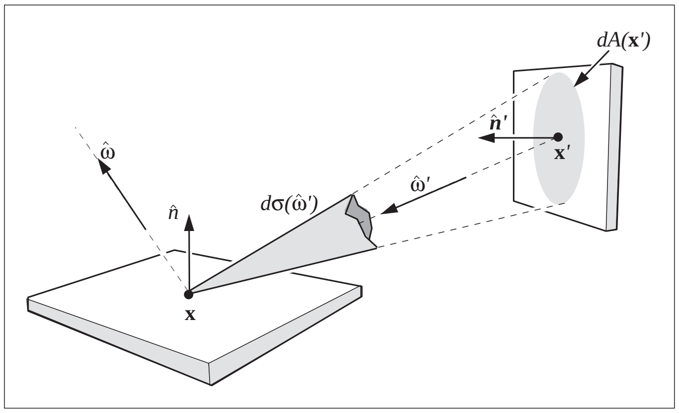

Fig. 1. Rendering equation

Here $\tau_1, \tau_2, \ldots, \tau_n$ are random variables distributed by the probability density function p, written $\tau_i \sim p$. If $n = 1$, $I \approx f(\tau_1)/p(\tau_1)$.

For an integral equation in the form of Equation 1 this suggests the approximation:

$$L_s(\mathbf{x}, \hat{\omega}) \approx L_e(\mathbf{x}, \hat{\omega}) + R(\mathbf{x}) L_f(\mathbf{x}, \hat{\omega}') \tag{3}$$

where $\hat{\omega}'$ is a random direction chosen with a cosine distribution.[3] Because $L_f(\mathbf{x}, \hat{\omega}')$ is not known, we trace a ray $\mathbf{x} + \hat{\omega}'$ to find $\mathbf{x}'$. If $L_f(\mathbf{x}, \hat{\omega}')$ is rewritten as $L_s(\mathbf{x}', \hat{\omega}')$ this suggests the following algorithm[4]:

```
color raycolor(ray, depth)
   color L = 0;
   if (ray hits at x)
      L += LE(x)
      if (depth < maxDepth)
         L += R(x)*raycolor(randomRay, depth+1)
      return L
   else
      return background
```

Because this is very inefficient for scenes with small bright sources, Kajiya[8] made the following modification (see Figure 2a):

```
color raycolor(ray, depth)
   color L = 0;
   if (ray hits at x)
```

[3] Note that because the directions are chosen with a cosine distribution, the probability density function $q(\hat{\omega}) = (-\hat{\omega}' \cdot \hat{n})/\pi$ which cancels the $(-\hat{\omega}' \cdot \hat{n})$ and $1/\pi$ terms in Equation 1.

[4] Here and in all subsequent pseudo code, `LE(x)` is the emitted color and `R(x)` is the reflected color at `x`.

```
      if (depth == 0)
         L += LE(x)
      L += R(x) * directLight(x, N(x))
      if (depth < maxDepth)
         L += R(x)*raycolor(randomRay, depth+1)
      return L
   else
      return background
```

where `directLight(x, N(x))` sends a shadow ray to a random point on each luminaire by evaluating Equation 2 defined over the set of emitters.

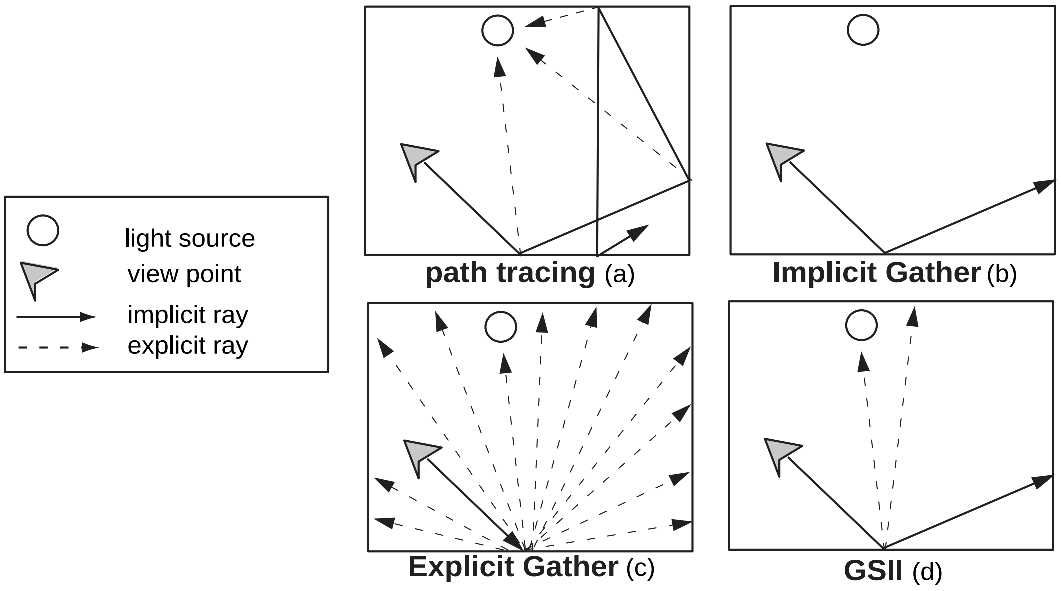

Fig. 2. Different view-dependent rendering methods.

2.3 Implicit Gather

Rushmeier[15] noted that we could terminate the recursion in the first algorithm above by running an approximate radiosity solution, resulting in the following integral (as opposed to an integral equation)

$$L_s(\mathbf{x}, \hat{\omega}) \approx L_e(\mathbf{x}, \hat{\omega}) + \int_{\bar{\mho}} \bar{L}(x, \hat{\omega}')(-\hat{\omega}' \cdot \hat{n}) \, d\bar{\sigma}(\hat{\omega}') \tag{4}$$

where $\bar{L}$ is the radiance from the radiosity solution. Solving the integral above is easier than solving the full rendering equation, and is the basic idea behind the two-pass methods discussed in this paper. Because solving Equation 4 can be thought of as calculating the light that **x** gets from the patches in the radiosity solution, it is often called a “gather”. Because it is a calculation for only a single point **x** it is also called a “local pass” (as opposed to a “global pass” which calculates radiances for all surfaces).

Rushmeier[15] modified the path tracing algorithm to solve Equation 4:

```
color raycolor(ray)
   color L = 0;
   if (ray hits at x)
      L += LE(x)
      L += R(x) * directLight(x, N(x))
      L += R(x)*radiositycolor(randomRay)
      return L
   else
      return background
```

Where `radiositycolor` returns the color of the radiosity patch seen in the direction of the random ray (note that the emitted portion is not there), Figure 2b.

2.4 Explicit Gather

Alternatively, Equation 4 can be solved by sending a shadow ray to every patch in the radiosity solution[13], Figure 2c:

```
color raycolor(ray)
   color L = 0;
   if (ray hits at x)
      L+= LE(x)
      for (i = 1 to numPatches)
         if (x + t(xi-x) hits at xi)
            L += LE(xi) + R(x)*L(xi)
      return L
   else
      return background
```

where `xi` is a point in the radiosity solution. This method is very expensive for large environments, but will be relatively noise free.

2.5 Geometric Simplification for Indirect Illumination (GSII)

Rushmeier et al.[14] observed that a two pass method could be more efficient if the implicit gather was from a geometrically simplified environment.

One drawback with this approach is that gathering from nearby objects will cause obvious artifacts. Even if only the illumination is simplified this proximity bug will occur (simplifying the geometry is just an extreme way of simplifying the field radiance function). To get around this problem a user defined parameter `r_thresh` is introduced that determines when to use the simplified radiosity solution or when to use MCPT. [5]

A combination of the code from Section 2.2 and Section 2.3 forms the algorithm:

```
depth = 0;
color raycolor(ray, depth)
   color L = 0;
   if (depth == 0)
      if (ray hits at x)
         L += LE(x);
         L += R(x) * directLight(x, N(x));
         if (depth < maxDepth)
            L += R(x)*raycolor(randomRay, depth+1);
         return L;
      else
         return background;
   else
      if ((ray hits at x) and
            (|x - ray_origin| < r_thresh))
         L += R(x) * directLight(x, N(x));
         if (depth < maxDepth)
            L += R(x)*raycolor(randomRay, depth+1);
         return L;
      else if (ray hits at xi)
         L += R(x)*radsimp_color(ray)
         return L
      else
         return background
```

where the value `|x - ray_origin|` represents the distance from the ray origin to the intersected point `x`, the point `xi` is in the simplified radiosity environment, and the function `radsimp_color` returns the color of the simplified radiosity patch seen in the direction of ray. Note that the function `directLight` sends shadow rays to every luminaire *and* to every bright reflecting zone (virtual luminaire) in the radiosity solution.

If it can be assumed that most renderings encounter the bulk of the computation in modeling the diffuse or nearly diffuse inter-reflection, then GSII is very effective. The simplified radiosity solution can be performed in a fraction of the time it may take for a full resolution solution and multiple levels of reflection rays required for MCPT are, in most cases, eliminated.

[5] This is a greatly simplified discussion. GSII also uses `r_thresh` to determine how much the radiosity environment is simplified. This allows GSII to produce a range of solutions from the Progressive Multi-pass Solution [3] if `r_thresh = 0` to MCPT if `r_thresh` is greater than the maximum distance between two points in the environment.

2.6 Efficient Explicit Gathers

The explicit gather as noted in Section 2.4 requires that rays be sent to every patch or, in the case of direct lighting, to every luminaire. Typically this is overkill because in many cases only a few light sources or patches are important to the illumination of the gather point. For direct lighting, Ward[24] suggested sorting the luminaires according to their contribution in descending order, sample the sources in order until the sum of the contributions is above some threshold, and then estimate the remaining contribution. This can be thought of as ranking the importance of each luminaire, then sampling the most important sources. Shirley and Wang[17] took this idea one step further by observing that one luminaire sample per viewing ray could be used if they could assure that important luminaires were more likely to be sampled than unimportant luminaires. This requires careful design of the probability density function p used in the single sample Monte Carlo estimate for the rendering equation in the form of Equation 2:

$$L_s(\mathbf{x},\hat{\omega}) \approx L_e(\mathbf{x},\hat{\omega}) + \frac{R(\mathbf{x})}{\pi} L_s(\mathbf{x}',\hat{\omega}') \frac{(-\hat{\omega}'\cdot\hat{n})(\hat{\omega}'\cdot\hat{n}')}{p(\mathbf{x}')\|\overrightarrow{\mathbf{x}-\mathbf{x}'}\|} \tag{5}$$

Here $\mathbf{x}'$ is a random point on a selected surface and $\mathbf{x}' \sim p$. As an example, assume two luminaires l_1 and l_2 are sampled according to the probability density functions $p_1(\mathbf{x}')$ and $p_2(\mathbf{x}')$. These functions can be combined into one *mixture density* by applying a weighted average:

$$p(\mathbf{x}') = \alpha p_1(\mathbf{x}') + (1-\alpha)p_2(\mathbf{x}')$$

where $\alpha \in [0,1)$. This function can then be applied in Equation 5. The function p is indeed a probability density because its integral over the two luminaires is one and it is strictly positive over all points on the luminaires. The coefficients α and $(1-\alpha)$ are called mixing weights.

This idea can be extended to N luminaires if one can determine the mixing weights $\alpha_1, \alpha_2, \ldots, \alpha_N$. If each luminaire's estimated contribution to the gather point can be determined then we can use these estimates to determine the mixing weights. If L_i' is the estimated contribution for light l_i then we can define

$$\alpha_i = \frac{L_i'}{L_1' + L_2' + \cdots + L_N'}$$

The mixture density is then

$$p(\mathbf{x}') = \alpha_1 p_1(\mathbf{x}') + \alpha_2 p(\mathbf{x}') + \cdots + \alpha_N p_N(\mathbf{x}')$$

where $\sum_{i=1}^{N} \alpha_i = 1$. Determining which light to sample can be done by generating a uniform random number $\xi \in [0,1)$ and selecting l_i such that

$$\sum_{j=1}^{i-1} \alpha_j \leq \xi < \sum_{j=1}^{i} \alpha_j$$

The estimates for $L_1', L_2', \ldots, L_n'$ can be obtained quickly by assuming that each luminaire is visible to the gather point.

The important point to notice about this process is that the values $\alpha_1, \alpha_2, \ldots, \alpha_N$ must be generated for each direct lighting calculation. While generating these estimates is much faster than generating an equivalent number of shadow rays, even this amount of computation can be crippling when the number of luminaires becomes hundreds or thousands. For this reason, recent work[18] suggests the use of spatial subdivision of light sources. The reasoning for this is that seldom are all lights important to every region in the environment. Therefore a list of important lights should vary in space. This can be done by attaching an important light list to the cells of a conventional subdivision structure such as a regular grid. To determine which light sources are entered into which cells an influence region is defined for each source. An influence box is then intersected with the subdivision structure and the light is inserted into the important list for each cell in the intersection, Figure 3. All sources not in the important list for a particular cell are considered unimportant to that cell. Probabilistically unimportant lights will be sampled but this should be infrequent. If it can be assumed that the unimportant luminaires contribute an equal amount of radiance to a scene, then a new probability density function can be formed with the mixing weights determined by

$$\alpha_i = \frac{L'_i}{L'_1 + L'_2 + \cdots + ML'_u}$$

where M is the number of unimportant luminaires and L'_u is the estimated average unimportant contribution.

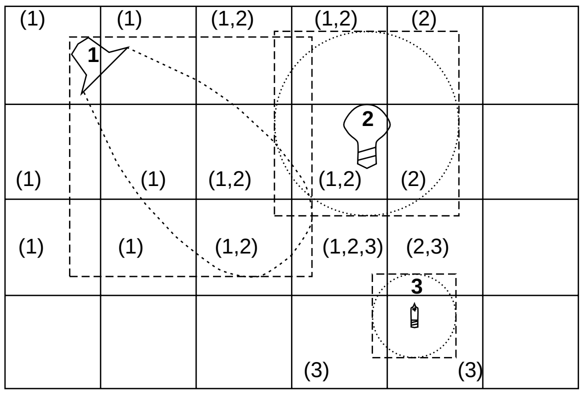

Fig. 3. Three lights with influence boxes intersecting a spatial grid. The important light list for each cell is indicated by numbers in parentheses.

3 The New Gather Algorithm

In this section we discuss the development of our algorithm. In Section 3.1 we explain why a simple combination of methods described in Section 2 is problematic. In Section 3.2 we propose a solution to this problem and in Section 3.3 we present the pseudocode for our algorithm.

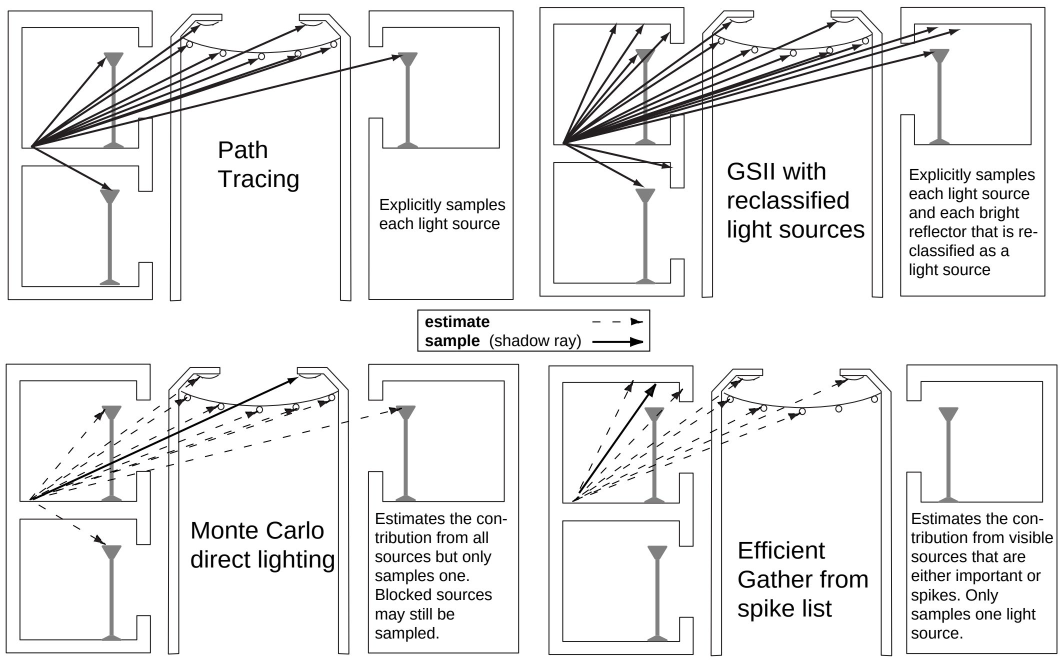
Path Tracing
Explicitly samples each light source
GSII with reclassified light sources
Explicitly samples each light source and each bright reflector that is re-classified as a light source
estimate
sample (shadow ray)
Monte Carlo direct lighting
Estimates the contribution from all sources but only samples one. Blocked sources may still be sampled.
Efficient Gather from spike list
Estimates the contribution from visible sources that are either important or spikes. Only samples one light source.

Fig. 4. How different methods send shadow rays.

3.1 Overestimates: Spikes

Since the explicit gather from Section 2.4 is relatively noise free, and since the spatial subdivision method of Section 2.6 makes explicit gathering more efficient, a logical progression would be to combine these with geometric simplification to create an algorithm which performs only an explicit gather. All surfaces in the simplified environment would be considered light sources and, as in GSII, the algorithm could resort to MCPT if a selected surface is less than `r_thresh` from the gather point. However, this strategy would produce very noisy images. The problem is that we cannot assume that all unimportant lights contribute equally to the illuminated point. In practice many unimportant luminaires will not contribute at all. Another problem is that it is possible for a luminaire with high radiance, a star for example, to be unimportant. Such a source would be selected infrequently, but when it is selected the combination of high power and low probability will produce a large estimate. Such an over estimate will require many extra samples to overcome.

In quantitative terms, this problem can be seen in the following way. If α_u is the probability of having to select one of M unimportant lights then

$$\alpha_u = \frac{ML'_u}{L'_1 + L'_2 + \cdots + ML'_u}$$

If the random value ξ is greater than $\sum_{i=1}^{N} \alpha_i$, where N is the number of important luminaires, then it is necessary to select from the unimportant set with probability α_u.

Assuming that unimportant lights contribute equally, a uniform selection from the unimportant list is made with probability $1/M$. If the selected luminaire is then visible a sample point on the luminaire is then selected with $p_u(\mathbf{x}')$. The value for the probability density $p(\mathbf{x}')$ will then be

$$p(\mathbf{x}') = \frac{\alpha_u p_u(\mathbf{x}')}{M}$$

If $p(\mathbf{x}')$ is inserted into Equation 5, it is easy to see that a large M or small α_u and/or $p_u(\mathbf{x}')$ may give a gross overestimate or spike.

3.2 Eliminating Spikes

To eliminate this problem of large overestimates we propose that the offending high luminance but low contribution sources be sampled explicitly. This amounts to having a "spike" list (high luminance patch list) rather than an importance list at each grid or octree cell. A spike list differs from an importance list because it includes any source l_i which exceeds some user defined threshold regardless of whether or not the influence region for l_i intersects the cell. This means that many sources which would be considered unimportant by the method described at the end of Section 2.6 will be included in a spike list. In practice all bright sources are automatically included in the spike list whereas dim sources and reclassified sources (bright reflectors) are tested against their influence regions to determine membership.

A problem with maintaining any list of spikes is that a scene may contain many spikes (e.g. an office building). However, in most scenes only a few spikes are visible (non-zero geometry term) at any given point. These "invisible" spikes should not be in the list.

To accomplish this exclusion of invisible patches from a spike list, we perform a visibility preprocess in each cell. Thus sources which are not visible to a cell will not be included in the spike list and can be ignored. Of course, if many bright sources are visible, as would be true in the middle of a football stadium, then the spike list will be large. However, for most architectural scenes, including night time city scenes, the visibility preprocess will provide significant savings. In the event that the spike list is large, we will still only send one shadow ray for an explicit gather from all spikes, so even several hundred spikes should be allowable without too much degradation in performance. Because visibility calculation in complex scenes is very difficult[21], we will use a point sample approach to visibility estimation, so we will falsely exclude some spikes. This amounts to truncation, but will never occur for spikes that are completely visible.

3.3 The Algorithm

The algorithm performs the following steps, the first two are taken directly from GSII:

1. Simplify surfaces and create a coarse mesh for the radiosity solution according to user defined parameters. [6]

[6] Automatically simplifying a scene is a difficult problem worthy of a PhD thesis by itself. For our algorithm the user must provide this simplified version of the scene. This is not as great a hardship for the user as one might guess. The generation of a simplified environment, a process called *massing*, is commonly employed by designers and architects[5].

2. Perform radiosity solution with the coarse mesh over the geometrically simplified environment.
3. Create the lighting grid by treating all surfaces in the simplified radiosity solution as luminaires.
4. Perform the ray tracing final pass according to the following algorithm:

```
color raycolor(ray, depth, list)
  color L = 0;
  if (depth == 0)
     if (ray hits at x)
        list = getSpikeList(x);
        L += LE(x);
        L += R(x) * spikeColor(x, N(x), list);
        if (depth < maxDepth)
           L += R(x) *
             raycolor(randomRay, depth+1, list);
        return L;
     else
        return background;
  else
     if ((ray hits at x) and
         (|x - ray_origin| < r_thresh))
        list = getSpikeList(x);
        L += R(x) * spikeColor(x, N(x), list);
        if (depth < maxDepth)
           L += R(x) *
             raycolor(randomRay, depth+1, list);
        return L;
      else if (ray hits at xi)
              if (xi is not in list)
                 L += R(x)*radsimp_color(ray);
              return L;
      else
        return background;
```

where the function `getSpikeList(x)` returns the spike list corresponding to the cell in which `x` resides. The function `spikeColor` uses the Monte Carlo direct lighting method shown in Section 2.6. Note that a check must be made to determine if the point `xi` resides on a surface in the spike list. If it does reside on such a surface then the radiance at `xi` is accounted for in the explicit gather on the spike list.

4 Implementation and Results

For our implementation, we use an axis aligned regular grid to store the spike lists. This grid is of the same size and dimension as the geometry grid used to store the full

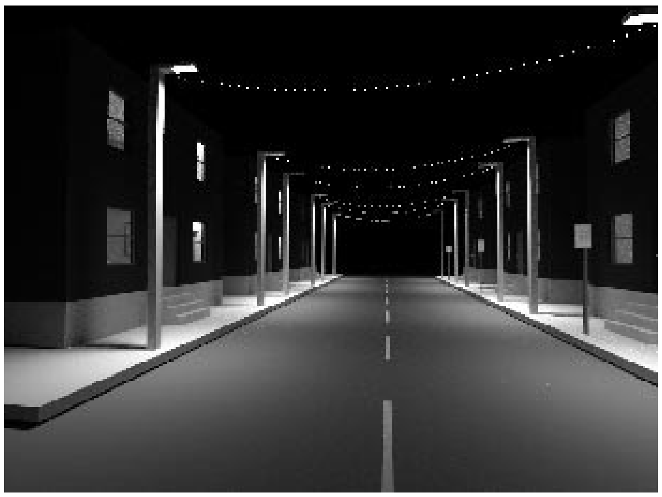

Fig. 5. New Gather with 64 samples per pixel

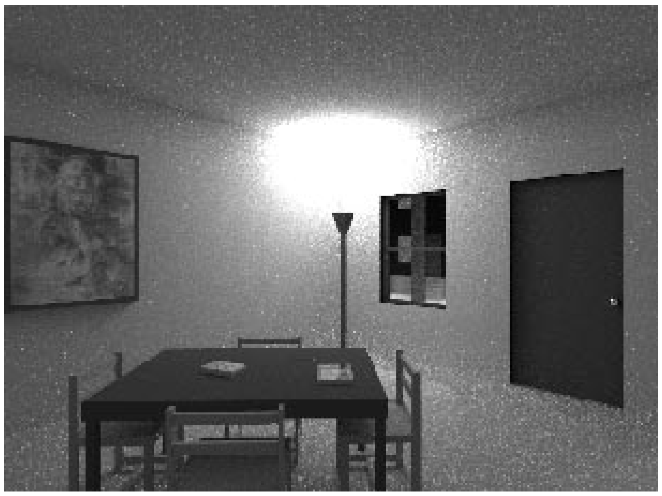

Fig. 6. New Gather with 64 samples per pixel

resolution environment. By making the lighting grid the same size as the geometry grid we can eliminate the storage of spike lists in cells which do not contain geometry. We introduce user defined values $\underline{E}$ and $\overline{E}$ as spike thresholds. We assume that any light source l_i with emittance $E_i > \overline{E}$ can produce a spike in the image. Therefore we insert l_i into each grid cell's spike list provided that l_i is visible from the cell. If $E_i < \underline{E}$ then it is assumed that l_i cannot produce a spike and is left to be sampled implicitly. If $\underline{E} < E_i < \overline{E}$ then the influence region as in[17] is used to determine the cells in which l_i may cause a spike. Again, l_i is inserted into a cell only if it is visible from the cell.

The visibility tests are performed by sending rays through the environment from random points in a grid cell to random points on the light source. A visibility variable g_i is maintained which represents the percentage of rays which are not blocked. This is similar to the visibility estimate of Smits et al.[20]. This technique is not as accurate as the techniques presented by Teller and Hanrahan[21] but is applicable in “less well-behaved” environments. In our implementation we use a self-refining method which initially sends 10 rays, if each ray reaches the light source then g_i is set to 1 and l_i is inserted into the spike list. If more than one but less than 10 rays reach the light source, 10 more samples are generated and $newg_i$ is stored. If $\|newg_i - g_i\| \leq 0.1$ then $g_i = (newg_i + g_i)/2$ and l_i is inserted into the spike list. If l_i is not inserted or if one or fewer rays reach the light source, this process continues until a maximum number of rays are sent. In our implementation the maximum number is 100. If $g_i = 0$ then the light source is fully occluded and can be left out of the cell's spike list.

The spike lists are sampled during the ray tracing pass according to the linear method from Shirley and Wang[17] with the geometry term included in the computation. The mixing weights are then defined by:

$$\alpha_i = \frac{g_i L_i}{g_1 L_1 + g_2 L_2 + \cdots + g_N \mathrm{L}_N}$$

If we choose l_i from the spike list with corresponding probability density function p_i then the probability density function for the estimate is

$$p(\mathbf{x}') = \alpha_i p_i(\mathbf{x}')$$

Note that the visibility term g which in most Monte Carlo methods is either 1 or 0 is now worked into the density function.

There is one other implementation detail which must be noted: the spike list chosen is not necessarily the list corresponding to the region or cell which contains the illuminated point. To prevent large errors from occuring at cell boundaries the choosing of a spike list is weighted based on the location of the illuminated point in the cell. Thus illuminated points near to the edge of a cell may cause a neighboring cell's spike list to be chosen.

The images in Figures 5 and 6 were produced with our algorithm with 64 samples per pixel. [7] These images represent two different view points in the same test environment which contains 1666 primitives including 318 light sources. The radiosity prepass was performed on simplified version of this environment that was tessellated into 32294 triangles. A view of the radiosity solution is shown in Figure 7. For comparison, a MCPT solution with 64 samples per pixel is shown in Figure 8. For this scene, the extra storage required for the spatial data structure containing the spike lists was of the same order as the grid structure which stored the actual scene description. These two structures and their associated geometry accounted for about 20% of the total storage. The remaining 80% was due to the simplified radiosity solution and its corresponding spatial structure. [8]

[7] The original images can be viewed in 24bit color on the web at http://www.cs.indiana.edu/hyplan/kuzimmer/EGRW95.html

[8] This is due in part to an inefficient mesh structure for the simplified geometry. The authors are rectifying this now. We expect that the storage required for the radiosity solution and its spatial structure to be about 50% of the total requirement after this improvement.

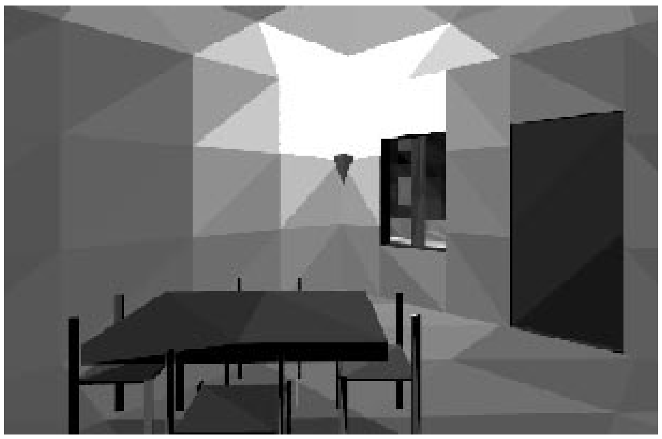

Fig. 7. Solution for radiosity prepass

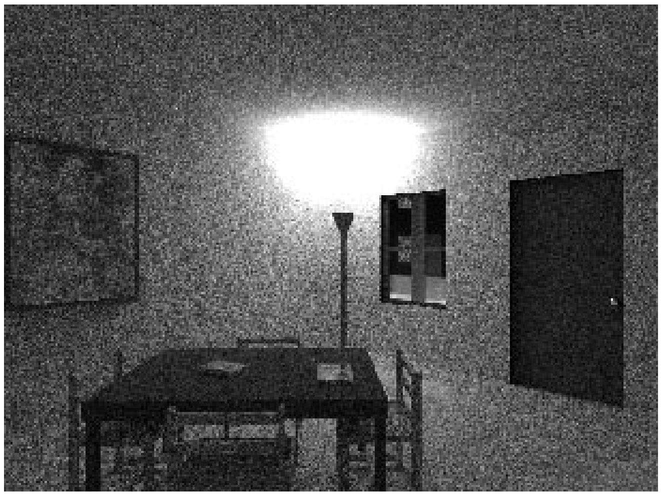

Fig. 8. MCPT with 64 samples per pixel

While it is quite possible to construct scenes where the spike lists will require substantial storage (ie. the stadium example), we expect requirements similar to those of our test scene on average. Table 1 compares the performance of our algorithm with MCPT and GSII. [9] These results are for 300 by 200 image sampled at one sample per pixel on the above mentioned environment. The small image size and sampling rate were necessary

[9] Our implementation of GSII is somewhat deficient because we do not reclassify bright reflectors as light sources. Because of this, we do not provide an example of an image using this method, but we expect the image quality to be similar for the view depicted in Figure 5 and slightly less noisy for the view depicted in Figure 6. It should be noted that reclassification will only increase the number of explicit samples necessary for this method.

MCPT	
Viewing Rays	60,000
Implicit Rays	422,640
Explicit (shadow) Rays	147,429,560
Preprocessing Time	none
Rendering Time	20 hours

GSII	
Viewing Rays	60,000
Implicit Rays	68,652
Explicit (shadow) Rays	21,973,800
Preprocessing Time	10 minutes
Rendering Time	1.6 hours

Our Algorithm	
Viewing Rays	60,000
Implicit Rays	68,992
Explicit (shadow) Rays	69,476
Preprocessing Time	1.2 hours
Rendering Time	4 minutes

Table 1. Three algorithms run on the test environment, 300x200 images at one sample per pixel.

because of the computational expense of MCPT. The times are for a single processor Silicon Graphics workstation with an R4400 processor.

The artificially small image size and sampling rate used to generate the values in Table 1 does not display the true power of our method. For a larger image of the same scene with a higher sampling rate per pixel, the preprocessing times for our algorithm remain unchanged and the implicit and explicit ray numbers will continue to be of the same order as the number of viewing rays. Using the above table as a guide, approximate times for a 1000x1000 image of the same scene with 64 samples per pixel would be on the order of 126 weeks for MCPT, 53 days for GSII, and 3 days for our algorithm.

5 Discussion

The algorithm presented is the first two-pass method for complex diffuse scenes with many luminaires. However there are many important issues that must be addressed.

5.1 Generality

We have presented results for only one model. This model is somewhat unusual in having so many lights that are only decorative. However, there are twelve street lights in the scene as well as twelve floor lamps as well as several reclassified sources like the ceiling and walls above the floor lamps. Because of the use of visibility estimates and visibility coherence, the sampling of these sources within rooms is relatively efficient. In the street where most of the street lamps are visible, the predictive weighting will cause the nearby lamps to be preferentially sampled. Our method works well for scenes which contain many bright reflectors, many luminaires and much occluding geometry (i.e., an office building). However, other methods may be preferred for scenes which contain few primary sources or scenes which contain numerous primary sources which are visible everywhere.

5.2 Error

Our algorithm is no longer a pure Monte Carlo method (unlike [8, 22]), so our solution will have both noise and deterministic error. Note that almost all current methods are such hybrids; e.g., most visibility checks are Monte Carlo[20]. We have not made any analytical statements about the magnitude of this error in our results. Bounding our error will be difficult[10], but must be investigated. However, the lack of an error bound should not dismiss the significance of the results.

In practice, the global illumination community is working on two ends of a spectrum. On one end, error bounds are derived and algorithms are developed based on these bounds. On the other end, algorithms are developed for practical use. Both of these efforts are worthwhile. Hopefully, the two ends will one day meet. In the mean time, those working on practical algorithms for complex scenes should be comparing their results with real environments as suggested by Ward[25]. We intend to do this.

5.3 Truncation

The current algorithm performs point sampling to determine visibility when generating the spike lists. If all of the samples from a region (or cell) to a light source are blocked then the source is not included in the spike list and it is not sampled explicitly with a shadow ray. Furthermore, as shown in the algorithm of Section 3.3, reflection rays only gather the reflected components. Thus the possibility exists that some contribution may be truncated. Truncation may also occur when the emission for a source falls between the user defined values $\underline{E}$ and $\overline{E}$. In this case the influence region for the luminaire determines its inclusion into the spike list. In our test scene, the Christmas lights strung across the street fall into this range. These lights have small influence regions which extend to the street but probably do not extend to the rooms in the buildings. Removing one of these lights introduces very little error, but removing all of them produces a significant error.

Clusters. We are investigating the possibility of hierarchically clustering sources and then sampling the cluster as if it were a single source in areas beyond the importance regions of these luminaires. This approach could also be applied to the unimportant lights which must be included in the spike lists because of an emission value greater than $\overline{E}$. For example, if the test image included thousands of stars in the night sky, the current implementation would include most of these stars in the spike list for a region in the middle of the street. Clustering could combine these into one giant luminaire. In fact, spreading the emissions from all of these sources across a cluster may move the emission below the spike

[10] Surprisingly, almost no global illumination algorithms have known error bounds on their solutions; Lischinski et al.[10] have provided us with error bounds on piecewise-constant radiosity solutions. We are aware of no other such results.

threshold. The night sky could then be left to be sampled by reflection rays. Clustering would also reduce the probability of incorrectly removing sources which are mostly but not totally occluded. An approach similar to that presented by Smits et al.[20] should work well for scenes containing diffuse emitters. However, properly combining directional sources into clusters will require further work.

5.4 Realism

Diffuse-only solutions are limited in their ability to depict realistic environments. Ward[25] found that much realism can be added if he used non-diffuse transport only for the last non-specular bounce before the eye (ie. he replaces $eye\, S^\star GG^\star\, light$ paths with $eye\, S^\star GD^\star\, light$ paths, where G is a general BRDF). This could be implemented in a two-pass method by gathering from a diffuse solution, but by using a non-diffuse BRDF for the local pass. An example of this can be seen in Figure 6 where we can see the reflection of the floor lamp in the book cover. An alternative would be to gather from a non-diffuse radiosity solution, which would account for the specular-diffuse-eye transport not handled well by path tracing, etc. Current non-diffuse radiosity solutions will not fare well on large environments. Determining how to combine a simplified environment with non-diffuse transport may prove to be fruitful research.

6 Conclusion

We have presented the first two-pass global illumination method that has a gather that is practical for complex diffuse scenes with many luminaires. This algorithm borrows the ideas of simplified geometry and light source reclassification from GSII. It also performs visibility estimates to associate visible light sources to regions in the environment. An implementation of this method has been run on a scenes with tens of thousands of surfaces and hundreds of luminaires. On this scene our implementation was about 20 times faster than the most efficient two-pass methods.

The crux of this paper is on explicitly sampling a few very important surfaces, and directionally sampling to estimate the contribution of the remaining surfaces. In addition, the use of visibility coherence was used to lower variance. This amounts to dividing the integral into two components, and applying different quadrature methods for each component. The techniques in this paper provide a mechanism that efficiently makes this division.

References

1. John M. Airey and Ming Ouh-young. Two adaptive techniques let progressive radiosity outperform the traditional radiosity algorithm. Technical Report TR89-20, University of North Carolina at Chapel Hill, August 1989.
2. James Arvo. Backward ray tracing. *Developments in Ray Tracing*, pages 259–263, 1986. ACM Siggraph '86 Course Notes.
3. Shenchang Eric Chen, Holly Rushmeier, Gavin Miller, and Douglass Turner. A progressive multi-pass method for global illumination. *Computer Graphics*, 25(4):165–174, July 1991. ACM Siggraph '91 Conference Proceedings.

4. Michael F. Cohen. Is image synthesis a solved problem? In *Proceedings of the Third Eurographics Workshop on Rendering*, pages 161–167, 1992.
5. Donald P. Greenberg. Computers and architecture: advanced modeling and rendering algorithms allow designers and clients to walk through buildings long before construction. *Scientific American*, 264:104–109, February 1991.
6. Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics*, 24(3):145–154, August 1990. ACM Siggraph '90 Conference Proceedings.
7. David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A radiosity method for non-diffuse environments. *Computer Graphics*, 20(4):133–142, August 1986. ACM Siggraph '86 Conference Proceedings.
8. James T. Kajiya. The rendering equation. *Computer Graphics*, 20(4):143–150, August 1986. ACM Siggraph '86 Conference Proceedings.
9. Malvin H. Kalos and Paula A. Whitlock. *Monte Carlo Methods*. John Wiley and Sons, New York, N.Y., 1986.
10. Dani Lischinski, Brian Smits, and Donald P. Greenberg. Bounds and error estimates for radiosity. *Computer Graphics*, 28(3):67–74, July 1994. ACM Siggraph '94 Conference Proceedings.
11. Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. *Computer Graphics*, pages 199–208, August 1993. ACM Siggraph '93 Conference Proceedings.
12. Thomas J. V. Malley. A shading method for computer generated images. Master's thesis, University of Utah, June 1988.
13. Mark C. Reichert. A two-pass radiosity method driven by lights and viewer position. Master's thesis, Cornell Program of Computer Graphics, January 1992.
14. Holly Rushmeier, Charles Patterson, and Aravindan Veerasamy. Geometric simplification for indirect illumination calculations. In *Graphics Interface '93*, pages 227–236, May 1993.
15. Holly E. Rushmeier. *Realistic Image Synthesis for Scenes with Radiatively Participating Media*. PhD thesis, Cornell University, May 1988.
16. Peter Shirley. A ray tracing algorithm for global illumination. In *Graphics Interface '90*, pages 205–212, May 1990.
17. Peter Shirley and Changyaw Wang. Distribution ray tracing: Theory and practice. In *Proceedings of the Third Eurographics Workshop on Rendering*, pages 200–209, 1992.
18. Peter Shirley, Changyaw Wang, and Kurt Zimmerman. Monte carlo techniques for direct lighting calculations. *ACM Transactions on Graphics (TOG)*, 1995. accepted for publication.
19. François X. Sillion and Claude Puech. A general two-pass method integrating specular and diffuse reflection. *Computer Graphics*, 23(3):335–344, July 1989. ACM Siggraph '89 Conference Proceedings.
20. Brian E. Smits, James R. Arvo, and David H. Salesin. A clustering algorithm for radiosity in complex environments. *Computer Graphics*, 28(3):435–442, July 1994. ACM Siggraph '94 Conference Proceedings.
21. Seth Teller and Pat Hanrahan. Global visibility algorithms for illumination computations. *Computer Graphics*, 27:239–246, August 1993. ACM Siggraph '94 Conference Proceedings.
22. Eric Veach and Leonidas Guibas. Bidirectional estimators for light transport. In *Proceedings of the Fifth Eurographics Workshop on Rendering*, pages 147–162, June 1994.
23. John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. A two-pass solution to the rendering equation: a synthesis of ray tracing and radiosity methods. *Computer Graphics*, 21(4):311–320, July 1987. ACM Siggraph '87 Conference Proceedings.
24. Greg Ward. Adaptive shadow testing for ray tracing. In *Proceedings of the Second Eurographics Workshop on Rendering*, 1991.

25. Gregory J. Ward. The radiance lighting simulation and rendering system. *Computer Graphics*, 28(2), July 1994. ACM Siggraph '94 Conference Proceedings.
26. Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. *Computer Graphics*, 22(4):85–92, August 1988. ACM Siggraph '88 Conference Proceedings.