Technical Report/TR-186-2-98-18 (*14 July 1998*)

Global Ray-bundle Tracing

László Szirmay-Kalos

Department of Control Engineering and Information Technology, Technical University of Budapest
Budapest, Műegyetem rkp. 11, H-1111, HUNGARY
szirmay@fsz.bme.hu

Abstract

The paper presents a single-pass, view-dependent method to solve the general rendering equation, using a combined finite element and random walk approach. Applying finite element techniques, the surfaces are decomposed into planar patches on which the radiance is assumed to be combined from finite number of unknown directional radiance functions by predefined positional basis functions. The directional radiance functions are then computed by random walk or by stochastic iteration using bundles of parallel rays. To compute the radiance transfer in a single direction, several global visibility methods are considered, including the global versions of the painter's, z-buffer, Weiler-Atherton's and planar graph based algorithms. The method requires no preprocessing except for handling point lightsources, for which a first-shot technique is proposed. The proposed method is particularly efficient for scenes including not very specular materials illuminated by large area lightsources or sky-light. In order to increase the speed for difficult lighting situations, walks can be selected according to their importance. The importance can be explored adaptively by the Metropolis and VEGAS sampling techniques.

Keywords: Rendering equation, global radiance, Monte-Carlo and quasi-Monte Carlo integration, Importance sampling, Metropolis method, z-buffer.

1. Introduction

The fundamental task of computer graphics is to solve a Fredholm type integral equation describing the light transport. This equation is called the *rendering equation* and has the following form:

$$L(\vec{x},\omega) = L^e(\vec{x},\omega) + \int_\Omega L(h(\vec{x},-\omega'),\omega') \cdot f_r(\omega',\vec{x},\omega) \cdot \cos^* \theta' \; d\omega' \tag{1}$$

where $L(\vec{x},\omega)$ and $L^e(\vec{x},\omega)$ are the radiance and emission of the surface in point $\vec{x}$ at direction ω, Ω is the directional sphere, $h(\vec{x},\omega')$ is the visibility function defining the point that is visible from point $\vec{x}$ at direction ω', $f_r(\omega',\vec{x},\omega)$ is the bi-directional reflection/refraction function, θ' is the angle between the surface normal and direction $-\omega'$, and $\cos^* \theta' = \cos\theta'$ if $\cos\theta' \geq 0$ and zero otherwise (figure 1).

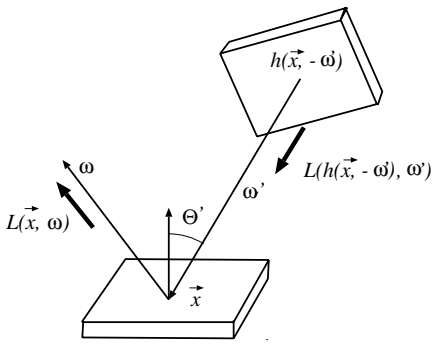


Figure 1: *Geometry of the rendering equation*

Since the rendering equation contains the unknown radiance function both inside and outside the integral, in order to express the solution, this coupling should be resolved. Generally, two methods can be applied for this: finite element methods or random walk methods.

Finite element methods project the problem into a finite function base and approximate the solution here. The pro-

© The Institute of Computer Graphics, Vienna University of Technology.

jection transforms the integral equation to a system of linear equations for which straightforward solution techniques are available. Finite element techniques that aim at the solution of the non-diffuse case can be traced back to the finite element approximation of the directional functions using *partitioned sphere*[10] or *spherical harmonics*[29], and to the application of *extended form factors*[28]. Since the radiance function is not smooth and is of 4-variate if non-diffuse reflection should also be considered, finite element methods require a great number of basis functions, and thus the system of linear equations will be very large. Although, hierarchical or multiresolution methods[4] and clustering[3] can help, the memory requirements are still prohibitive for complex scenes.

Random walk methods, on the other hand, resolve the coupling by expanding the integral equation into a Neumann series, and calculate the resulting high-dimensional integrals by numerical quadrature from discrete samples. A single discrete sample corresponds to a complete photon-path (called the *walk*) from a lightsource to the eye, which is usually built by d ray-shooting steps if the photon is reflected d times. Since classical quadrature rules are useless for the calculation of very high dimensional integrals, Monte-Carlo or quasi-Monte Carlo techniques must be applied.

In computer graphics the first Monte-Carlo random walk algorithm — called *distributed ray-tracing* — was proposed by Cook et al. [5], which spawned to a set of variations, including *path tracing*[11], *light-tracing*[9], *Monte-Carlo radiosity* [26, 18, 21], and two-pass methods which combine radiosity and ray-tracing [41].

The problem of naive generation of walks is that the probability that a shooting path finds the eye is zero for a pin-hole camera or very small if a non-zero aperture camera model is used, while the probability that a gathering random path ends in a lightsource may be very little if the lightsources are small, thus the majority of the paths do not contribute to the image at all, and their computation is simply waste of time.

Thus, on the one hand, random walk must be combined with a deterministic step that forces the walk to go to the eye and to find a lightsource. *Light tracing*[9] connects each bounce position to the eye deterministically. *Bi-directional path-tracing*[13, 39] methods start a walk from the eye and a walk from a lightsource and connect the bounce positions of the two walks.

On the other hand, importance sampling[30] should be incorporated to prefer useful paths along which significant radiance is transferred. Note that although the contribution on the image is a function of the complete path, computer graphics applications usually assign estimated importance to individual steps of this path, which might be quite inaccurate. In a single step the importance is usually selected according to the BRDF [9, 13], or according to the direction of the direct lightsources [27]. Combined methods that find the important directions using both the BRDF and the incident illumination have been proposed in [38, 14]. Just recently, Veach and Guibas[40] proposed the Metropolis method to be used in the solution of the rendering equation. Unlike other approaches, Metropolis sampling[17] can assign importance to a complete walk not just to the steps of this walk, and it explores important regions of the domain adaptively while running the algorithm. Thus no a-priori knowledge is required about the important rays to construct a probability density function in advance. Instead, the algorithm converges to this probability density automatically.

In order to reduce the noise of these methods, very many samples are required, especially when importance sampling cannot help significantly — that is when the lightsources are large and the surfaces are not very specular. One way of reducing the ray-object intersection calculation cost is storing this information in the form of *illumination networks*[1], but it has large memory requirements, and representing the light-transport of small number of predefined rays might introduce artifacts.

The proposed new method also combines the advantages of finite-element and random-walk approaches and can solve the general non-diffuse case. The method needs no preprocessing, the memory requirements are modest, and it is particularly efficient for scenes containing larger area lightsources and moderately specular surfaces — that is where other importance-sampling walk methods become inefficient.

2. Informal discussion of the algorithm

Walk methods proposed so far use individual ray-paths as samples of the integrand of the rendering equation.

However, ray-shooting may waste a lot of computation by ignoring all the intersection objects but the one closest to the eye. Thus it seems worth using a set of *global directions*[24, 18] for the complete scene instead of solving the visibility problem independently for different points $\vec{x}$. Moreover, ray-shooting is a simple but by no means the most effective visibility algorithm since it is unable to take advantage of image or object coherence. Other methods based on the exploitation of image coherence, such as the z-buffer, painter's, Warnock's, etc. algorithms can be considered as handling a bundle of parallel rays and solving the visibility problem for all of them simultaneously. Continuous (also called object-precision) methods can even determine the visibility problem independently of the resolution, which corresponds to tracing infinitely many parallel rays simultaneously.

The set of parallel global rays is called the *ray-bundle*.

These visibility algorithms assume that the surfaces are decomposed into planar patches, thus the proposed method also uses this assumption. On the other hand the patch decomposition also serves as the construction of the finite-element structure.

Using ray-bundles, we have to realize that even single-

© Institute of Computer Graphics 1998

ray techniques use recursive ray-tracing to simulate multiple interreflections. Thus ray-bundles should also be traced several times in different directions to model multiple interreflections. This tracing composes a walk using ray-bundles in each step.

2.1. Computation of global ray-bundle walks

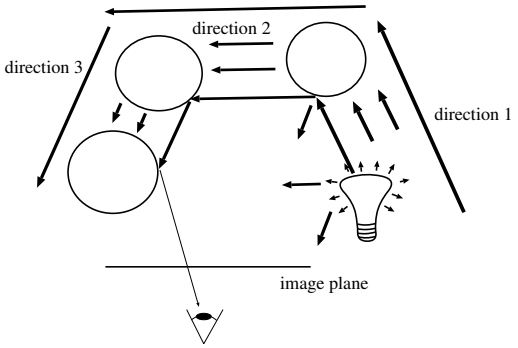


Figure 2: *A path of ray-bundles*

The algorithm takes samples of these global walks and uses them in the quadrature. A single walk starts by selecting a direction either randomly or quasi-randomly, and the emission transfer of all patches is calculated into this direction (figure 2). Then a new direction is found, and the emission is transferred and the irradiance generated by the previous transfer is reflected from all patches into this new direction. The algorithm keeps doing this for a few times depending on how many bounces should be considered, then the emission is sent and the irradiance caused by the last transfer is reflected towards the eye. Averaging these contributions results in the final image. When the radiance reflection is calculated from the previous direction to the current direction or to the direction of the eye, the radiance is attenuated by the BRDF of the corresponding surface element.

Concerning the memory requirements of the method, each patch holds the irradiance of the last step of the walk and the accumulated radiance towards the eye. Since the selected directions are the same for all surfaces, they must be stored only once. Consequently the memory requirement is comparable to that of the diffuse radiosity algorithms although it is also capable to handle specular reflections.

In order to make this method work two problems must be solved. The directions should be selected in a way that all the possible light-paths are covered and the integral quadrature should be accurately approximated. The application of random or low-discrepancy series on the directional sphere is proposed to solve this problem.

Secondly, efficient algorithms are needed that can compute the radiance transfer of all patches in a single direction, for which the generalization of discrete and continuous visibility algorithms are applied.

3. Reformulation of the rendering equation using finite-elements

Using finite element concepts, the radiance function is searched in the following form:

$$L(\vec{x},\omega) \approx L^{(n)}(\vec{x},\omega) = \sum_{j=1}^{n} L_j(\omega) \cdot b_j(\vec{x}) \qquad (2)$$

where $L^{(n)}(\vec{x},\omega)$ is the approximating radiance and $b_j(\vec{x})$ is a complete function system. In this function space, the

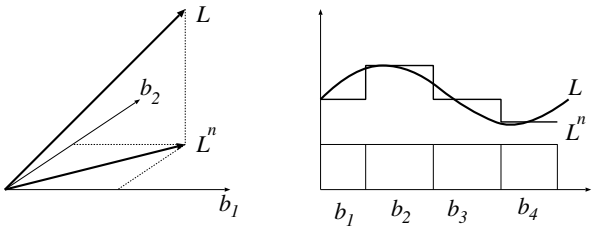


Figure 3: *Finite element approximation*

scalar product of two functions f, g is defined as the integral of their products on the total surface S:

$$\langle f, g \rangle = \int_S f(\vec{x}) \cdot g(\vec{x})\, d\vec{x}. \qquad (3)$$

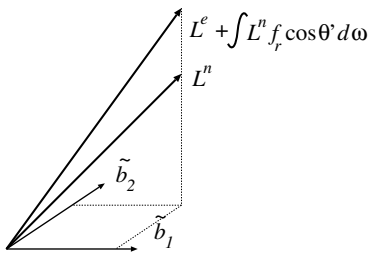


Figure 4: *Projection to the adjoint base*

Since the radiance is approximated in a subspace, we cannot expect the radiance approximation to satisfy the original rendering equation everywhere. Instead, equality is required in an appropriate subspace defined by *adjoint basis* functions $\tilde{b}_1(\vec{x}), \tilde{b}_2(\vec{x}), \ldots \tilde{b}_n(\vec{x})$ (figure 4). This set is called adjoint of $b_1(\vec{x}), b_2(\vec{x}), \ldots b_n(\vec{x})$ since we require that

$$\langle b_i(\vec{x}), \tilde{b}_j(\vec{x}) \rangle = \begin{cases} 1 \text{ if } i = j, \\ 0 \text{ otherwise,} \end{cases} \qquad (4)$$

Projecting the rendering equation into the subspace of $\tilde{b}_1(\vec{x}), \tilde{b}_2(\vec{x}), \ldots \tilde{b}_n(\vec{x})$ we obtain

$$\langle L^n(\vec{x},\omega), \tilde{b}_i(\vec{x}) \rangle = \langle L_e(\vec{x},\omega), \tilde{b}_i(\vec{x}) \rangle +$$

© Institute of Computer Graphics 1998

$$\langle \int_\Omega L^n(h(\vec{x},-\omega'),\omega')\cdot f_r(\omega',\vec{x},\omega)\cdot\cos^*\theta'\ d\omega', \tilde{b}_i(\vec{x})\rangle. \tag{5}$$

Using the orthogonal property stated by equation (4), we get

$$L_i(\omega) = L_i^e(\omega) +$$

$$\sum_{j=1}^{n}\int_\Omega L_j(\omega')\cdot\langle b_j(h(\vec{x},-\omega'))\cdot f_r(\omega',\vec{x},\omega)\cdot\cos^*\theta', \tilde{b}_i(\vec{x})\rangle\ d\omega'. \tag{6}$$

The same equation can also be presented in a matrix form:

$$\mathbf{L}(\omega) = \mathbf{L}^e(\omega) + \int_\Omega \mathbf{T}(\omega',\omega)\cdot\mathbf{L}(\omega')\ d\omega', \tag{7}$$

where $\mathbf{L}(\omega)|_i = L_i(\omega)$ is the vector of radiance values, and $\mathbf{T}(\omega',\omega)|_{ij} = \langle f_r(\omega',\vec{x},\omega)\cdot b_j(h(\vec{x},-\omega'))\cdot\cos^*\theta', \tilde{b}_i(\vec{x})\rangle$ is the *bi-directional transport matrix*.

Assume that the BRDF function $f_r(\omega',\vec{x},\omega)$ can be well approximated by $\tilde{f}_i(\omega',\omega)$ inside the support of $\tilde{b}_i$ (if the support of these basis functions is small, this is always possible). This allows for the separation of the transport matrix to a diagonal matrix $\mathbf{F}(\omega',\omega)$ of BRDF functions

$$\mathbf{F}(\omega',\omega)|_{ii} = \tilde{f}_i(\omega',\omega),$$

and to a *geometry matrix* $\mathbf{A}(\omega')$ that is independent of direction ω:

$$\mathbf{A}(\omega')|_{ij} = \langle b_j(h(\vec{x},-\omega'))\cdot\cos^*\theta', \tilde{b}_i(\vec{x})\rangle \tag{8}$$

The geometry matrix contains a scalar product of basis functions at points that are visible from each-other in direction ω'. Thus it expresses the strength of coupling as the degree of visibility.

Using the geometry matrix, equation (7) can also be written as

$$\mathbf{L}(\omega) = \mathbf{L}^e(\omega) + \int_\Omega \mathbf{F}(\omega',\omega)\cdot\mathbf{A}(\omega')\cdot\mathbf{L}(\omega')\ d\omega', \tag{9}$$

Note that equation (9) is highly intuitive as well. The radiance of a patch is the sum of the emission and the reflection of all irradiances. The role of the patch-direction-patch "form-factor matrix" is played by $\mathbf{A}(\omega')$.

4. Numerical solution of the directional integrals

In order to simplify the notations, the integral operator of the rendering equation is denoted by $\mathcal{T}$:

$$\int_\Omega \mathbf{T}(\omega',\omega)\cdot\mathbf{L}(\omega')\ d\omega' = \mathcal{T}\mathbf{L}(\omega). \tag{10}$$

Thus the short form of the rendering equation is:

$$\mathbf{L}(\omega) = \mathbf{L}^e(\omega) + \mathcal{T}\mathbf{L}(\omega). \tag{11}$$

In equation (11) the unknown radiance function $\mathbf{L}(\omega)$ appears on both sides. Substituting the right side's $\mathbf{L}(\omega)$ by the complete right side, which is obviously $\mathbf{L}(\omega)$ according to the equation, we get:

$$\mathbf{L}(\omega) = \mathbf{L}^e(\omega) + \mathcal{T}(\mathbf{L}^e(\omega) + \mathcal{T}\mathbf{L}(\omega)) =$$

$$\mathbf{L}^e(\omega) + \mathcal{T}\mathbf{L}^e(\omega) + \mathcal{T}^2\mathbf{L}(\omega). \tag{12}$$

Repeating this step m times, the original equation can be expanded into a Neumann series:

$$\mathbf{L}(\omega) = \sum_{i=0}^{m}\mathcal{T}^i\mathbf{L}^e(\omega) + \mathcal{T}^{m+1}\mathbf{L}(\omega). \tag{13}$$

If integral operator $\mathcal{T}$ is a contraction, that is if

$$||\mathcal{T}\mathbf{L}(\omega)|| \le \lambda\cdot||\mathbf{L}(\omega)||, \qquad \lambda < 1, \tag{14}$$

then $\lim_{m\to\infty}\mathcal{T}^{m+1}\mathbf{L} = 0$, thus

$$\mathbf{L}(\omega) = \sum_{i=0}^{\infty}\mathcal{T}^i\mathbf{L}^e(\omega). \tag{15}$$

The contractive property of $\mathcal{T}$ comes from the fact that a reflection or refraction always decreases the energy. Using, for example, the infinite norm, we obtain

$$||\mathcal{T}\mathbf{L}(\omega)||_\infty \le \max_{\vec{x}}\int_\Omega f_r(\omega',\vec{x},\omega)\cdot\cos^*\theta'\ d\omega'\cdot||\mathbf{L}(\omega)||_\infty$$

$$= \max_{\vec{x}} a_{\vec{x}}(\omega)\cdot||\mathbf{L}(\omega)||_\infty,$$

where $a_{\vec{x}}(\omega)$ is the *albedo*[16] of the material at point $\vec{x}$. For physically plausible material models the albedo must be less than 1.

The terms of this infinite Neumann series have intuitive meaning as well: $\mathcal{T}^0\mathbf{L}^e(\omega) = \mathbf{L}^e(\omega)$ comes from the emission, $\mathcal{T}^1\mathbf{L}^e(\omega)$ comes from a single reflection (called 1-bounce), $\mathcal{T}^2\mathbf{L}^e(\omega)$ from two reflections (called 2-bounces), etc.

Using the definition of integral operator $\mathcal{T}$, the full form of the Neumann series is:

$$\begin{aligned}\mathbf{L}(\omega) \ &= \ \mathbf{L}^e(\omega) + \int_\Omega \mathbf{T}(\omega_1',\omega)\cdot\mathbf{L}^e(\omega_1')\ d\omega_1' \\ &+ \int_\Omega\int_\Omega \mathbf{T}(\omega_1',\omega)\cdot\mathbf{T}(\omega_2',\omega_1')\cdot\mathbf{L}^e(\omega_2')\ d\omega_2' d\omega_1' \\ &+ \ \ldots\end{aligned} \tag{16}$$

In practice the infinite sum of the Neumann series is always approximated by a finite sum. The number of required

© Institute of Computer Graphics 1998

terms is determined by the contractivity λ of operator $\mathcal{T}$ — that is the overall reflectivity of the scene. Let us denote the maximum number of calculated bounces by D. The truncation of the Neumann series introduces a bias in the estimation, which can be tolerated if D is high enough.

In order to simplify the notations, we introduce the *d-bounce irradiance* $\mathbf{J}_d$ for $d = 1, 2, \ldots$ as follows:

$$\begin{aligned}\mathbf{J}_0 &= \mathbf{A}(\omega'_D) \cdot \mathbf{L}^e(\omega'_D),\\ \mathbf{J}_d &= 4\pi \cdot \mathbf{A}(\omega'_{D-d}) \cdot \mathbf{F}(\omega'_{D-d+1}, \omega'_{D-d}) \cdot \mathbf{J}_{d-1}\end{aligned}$$

where $\mathbf{J}_d$ is a $d+1$ dimensional function of directions $(\omega'_{D-d}, \omega'_{D-d+1} \ldots, \omega'_D)$.

The d-bounce irradiance represents the irradiance arriving at each patch, that is emitted from a patch and is bounced exactly d times.

Similarly, we can define the *max d-bounce irradiance* $\mathbf{I}_d$ for $d = 1, 2, \ldots$ as follows:

$$\begin{aligned}\mathbf{I}_0 &= \mathbf{A}(\omega'_D) \cdot \mathbf{L}^e(\omega'_D),\\ \mathbf{I}_d &= \mathbf{A}(\omega'_{D-d}) \cdot\\ &\quad \cdot \left(\mathbf{L}^e(\omega'_{D-d}) + 4\pi \cdot \mathbf{F}(\omega'_{D-d+1}, \omega'_{D-d}) \cdot \mathbf{I}_{d-1}\right)\end{aligned}$$

where $\mathbf{I}_d$ is a $d+1$ dimensional function of directions $(\omega'_{D-d}, \omega'_{D-d+1} \ldots, \omega'_D)$.

The max d-bounce irradiance represents the irradiance arriving at each patch after completing a path of length d following the given directions, and gathering and then reflecting the emission of the patches visited during the path.

Limiting the analysis to at most $D+1$ bounces, the solution of the rendering equation can be obtained as a $2D$-dimensional integral:

$$\mathbf{L}(\omega) = \left(\frac{1}{4\pi}\right)^D \cdot \int_\Omega \ldots \int_\Omega \left[\mathbf{L}^e(\omega) + 4\pi \cdot \mathbf{F}(\omega'_1, \omega) \cdot \mathbf{I}_D\right] \, d\omega'_D \ldots d\omega'_1. \tag{17}$$

This high-dimensional integral ($2D$ is 10 to 20 in practical cases) can be evaluated by numerical quadrature. Since classical quadrature rules, such as the trapezoidal rule or Gaussian quadrature, are not appropriate for the evaluation of high-dimensional integrals due to their dimensional explosion, Monte-Carlo techniques are proposed.

A single walk can be characterized by the vector of the transillumination directions of individual steps, that is by $(\omega'_1, \omega'_2, \ldots, \omega'_D)$. Monte Carlo methods generate the samples randomly using an appropriate probability distribution and setting the weight function as the inverse of the probability density. The concept of importance sampling suggest us to select a probability distribution that concentrates on points that are responsible for great contribution to the final image and neglect those walks that have no or negligible contributions. However, usually no a-priory information is available about the important walks, thus the required probability density cannot be constructed. Note that when tracing individual rays, we can approximate the contribution as the product of the BRDFs (as usually done in Monte-Carlo ray-tracing algorithms) or as the contribution of 1-bounces (direct-lighting computation). However, when a bundle of rays is traced simultaneously, different patches would prefer different continuation directions since their normals and BRDFs can be different.

Thus either we use uniformly distributed random or quasi-random samples, or the importance information is built up during the simulation in an adaptive manner.

In the next sections, the application of uniformly distributed random and quasi-random sequences is investigated, then the incorporation of the importance using the Metropolis[17] and VEGAS[15] methods are discussed.

4.1. Simple Monte-Carlo, or quasi-Monte Carlo integration

In order to evaluate formula (17), M random or quasi-random[12] walks should be generated (the difference is that in Monte-Carlo walks the directions are sampled randomly while in quasi-random walks they are sampled from a $2D$-dimensional low-discrepancy sequence, such as the $2D$-dimensional *Halton* or *Hammersley* sequence[20]). When the D-bounce irradiance is available, it is multiplied by the BRDF defined by the last direction ω_1 and the viewing direction ω to find a Monte-Carlo estimate of the radiance that is visible from the eye position. Note that this step makes the algorithm view-dependent.

There are basically two different methods to calculate the image estimate. On the one hand, evaluating the BRDF once for each patch, a radiance value is assigned to them, then in order to avoid "blocky" appearance, bi-linear smoothing can be applied.

Using Phong interpolation, on the other hand, the radiance is evaluated at each point visible through a given pixel using the irradiance field, the surface normal and the BRDF of the found point. In order to speed up this procedure, the surface visible at each pixel, the visibility direction and the surface normal can be determined in a preprocessing phase and stored in a map. Phong interpolation is more time consuming but the generated image is not only numerically precise, but is also visually pleasing.

The final image is the average of these estimates. The complete algorithm — which requires just one variable for each patch i, the max d-bounce irradiance $\mathbf{I}[i]$ — is summarized in the following:

© Institute of Computer Graphics 1998

for $m = 1$ **to** M **do** *// samples of global walks*
  Generate $(\omega_1^{(m)}, \omega_2^{(m)}, \ldots, \omega_D^{(m)})$
  $\mathbf{I} = 0$
  for $d = 0$ **to** $D-1$ **do** *// a walk*
    $\mathbf{I} = \mathbf{A}(\omega'_{D-d}) \cdot \left(\mathbf{L}^e(\omega'_{D-d}) + 4\pi \cdot \mathbf{F}(\omega'_{D-d+1}, \omega'_{D-d}) \cdot \mathbf{I}\right)$
  endfor
  Calculate the image estimate from the irradiance $\mathbf{I}$
  Divide the estimate by M and add to the Image
endfor
Display Image

4.2. Combined and bi-directional walking techniques

The algorithm that has been derived directly from the quadrature formulae uses direction ω_1 to evaluate the contribution of 1-bounces, directions (ω_1, ω_2) for the 2-bounces, $(\omega_1, \omega_2, \omega_3)$ for the 3-bounces, etc. This is just a little fraction of the information that can be gathered during the complete walk. We could also use the samples of ω_1, ω_2, ω_3, etc. to calculate the 1-bounce contribution, (ω_1, ω_2), (ω_1, ω_3), ..., (ω_2, ω_3), etc. combinations of directions for 2-bounces, etc. This is obviously possible, since if the samples of $(\omega_1, \omega_2, \ldots \omega_D)$ are taken from a uniform sequence, then all combinations of its elements also form uniform sequences in lower dimensional spaces.

If all possible combinations are used, then each random walk generates $\binom{D}{d}$ samples for the d-bounces, which can be used to increase the accuracy of the method. Note that the increased accuracy of this "*combined*" method is for free in terms of additional visibility computation.

However, due to the dependence of the BRDF functions on two directions and due to the fact that different bounces will be estimated by different numbers of samples, the required storage per patch is increased to $D(D+1)/2$ variables. Since D is 5 to 8 in practical cases, this storage overhead is affordable.

Now each patch is represented by a triangle matrix $\mathcal{I}$, where the (i, j) element stores the sum of those i-bounce irradiances where the last direction is ω_j. Table 1 shows an example for $D = 3$.

The complete combined algorithm is shown below:

for $m = 1$ **to** M **do**
  Generate $(\omega_1^{(m)}, \omega_2^{(m)}, \ldots, \omega_D^{(m)})$
  for $d = 0$ **to** $D-1$ **do** *// quasi-random walk*
    $\mathcal{I}[1][D-d] = \mathbf{A}(\omega'_{D-d}) \cdot \mathbf{L}^e(\omega'_{D-d})$
    for $b = 2$ **to** $d+1$ **do**
      $\mathcal{I}[b][D-d] = 0$
      for $pd = b-1$ **to** $d-1$ **do**
        $\mathcal{I}[b][D-d]$ += $4\pi \cdot \mathbf{A}(\omega_{D-d}^{(m)}) \cdot \mathbf{F}(\omega_{D-pd}^{(m)}, \omega_{D-d}^{(m)}) \cdot \mathcal{I}[b-1][D-pd]$
      endfor
    endfor
  endfor
  Divide the b-bounce estimates ($\mathcal{I}[b][d]$) by $\binom{D}{b}$
  Calculate the image estimate from the $\mathcal{I}[b][d]$ estimates
  Divide the estimate by M and add to the Image
endfor
Display Image

Furthermore, when the radiance is transferred to a direction, the required information to transfer the radiance to the opposite direction is also available, that is when computing geometry matrix $\mathbf{A}(\omega')$ for some direction, the matrix for the reverse direction $\mathbf{A}(-\omega')$ is usually also known paying very little or no additional effort.

The improvement that takes advantage of this is called the *bi-directional* algorithm.

Due to the $\cos^*$ function only one of the elements $\mathbf{A}(\omega')_{ij}$ and $\mathbf{A}(-\omega')_{ij}$ can be non zero. It means that bi-directional techniques do not even require additional storage and a single geometry matrix can be used to store the values for both ω' and $-\omega'$. It can be decided whether a matrix element is valid for ω' or $-\omega'$ by inspecting the angle between its normal vector and the given directions.

Formally, in the bi-directional technique ω_1, $-\omega_1$, ω_2, $-\omega_2$, etc. are used to calculate the 1-bounce contribution, (ω_1, ω_2), $(\omega_1, -\omega_2)$, $(-\omega_1, \omega_2)$, $(-\omega_1, -\omega_2)$, (ω_1, ω_3), $(\omega_1, -\omega_3)$, etc. combinations of directions for 2-bounces, etc. This multiplies the number of samples used for the computation of 1-bounces by 2, for the 2-bounces by 4 and generally for the d bounces by 2^d, which can be quite significant.

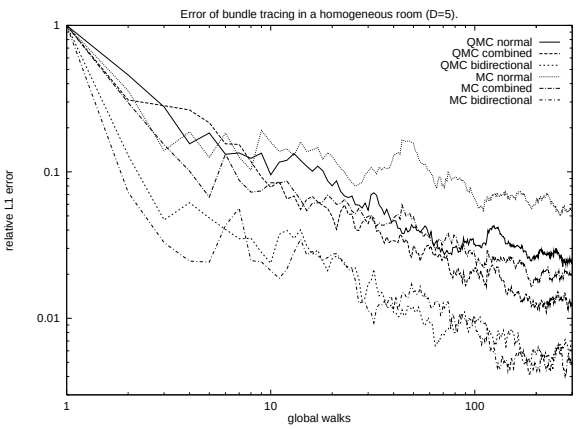


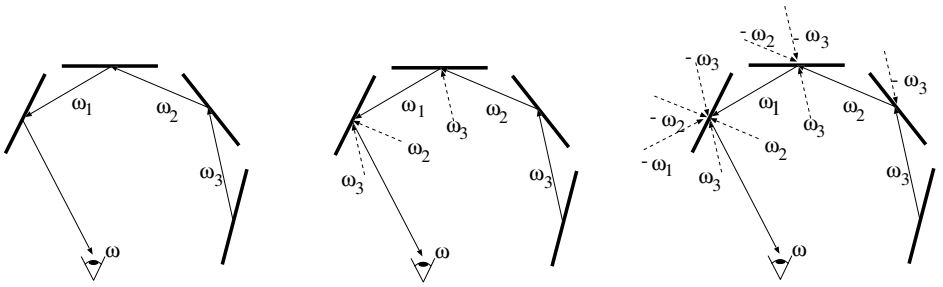
Figure 6: *Combined and bi-directional walking techniques versus normal walk*

The additional samples of the "combined" and particularly the "bi-directional" walking techniques increase the accuracy as shown by figure 6. The test scene was the homogeneous Cornell-box where all surfaces have constant 0.5 diffuse reflectance and emission, which allowed to solve the

© Institute of Computer Graphics 1998

last direction	1-bounce	2-bounce	3-bounce
3	$\mathbf{J}_0(\omega_3)$		
2	$\mathbf{J}_0(\omega_2)$	$\mathbf{J}_1(\omega_3,\omega_2)$	
1	$\mathbf{J}_0(\omega_1)$	$\mathbf{J}_1(\omega_3,\omega_1)+\mathbf{J}_1(\omega_2,\omega_1)$	$\mathbf{J}_2(\omega_3,\omega_2,\omega_1)$

Table 1: *Irradiance matrix of a patch for $D = 3$*



normal combined bi-directional

Figure 5: *Normal, combined and bi-directional walking technique*

rendering equation analytically[12] (the solution is $L = 1 - 2^{-(D+1)}$) to find a reference for the error analysis. Note that although quasi-Monte Carlo sampling is generally better, the improvement provided by the combined and bi-directional methods is less for the quasi-Monte Carlo walk than for the Monte-Carlo walk. This can be explained by the fact that the low-discrepancy points are so "well-designed" that mixing different sets of them does not improve the quadrature much further.

4.3. Stochastic iteration

The methods introduced so far fall into the category of random walk techniques which calculate only the first D terms of the infinite Neumann series and simply ignore the rest. Consequently, the estimate will be biased. This section, however, presents an assymptotically unbiased solution that is based on the concept of iteration.

Let us recall again the short form of the projected rendering equation (formula (11)):

$$\mathbf{L}(\omega) = \mathbf{L}^e(\omega) + \mathcal{T}\mathbf{L}(\omega).$$

Iterational techniques realize that the solution of this integral equation is the *fixed point* of the following iterational scheme

$$\mathbf{L}_n = \mathbf{L}^e + \mathcal{T}\mathbf{L}_{n-1} \tag{18}$$

thus if operator $\mathcal{T}$ is a contraction, then this scheme will converge to the solution from any initial function.

A possible way of storing the approximating functions $\mathbf{L}_n$, is the application of finite-element techniques also in the directional domain. However, this suffers from two critical problems. On the one hand, an accurate finite-element approximation usually requires very many basis functions, which in turn need a lot of storage space.

On the other hand, when finite element techniques are applied, operator $\mathcal{T}$ is only approximated, which introduces some non-negligible error in each step. If the contraction ratio of the operator is λ, then the total accumulated error will be approximately $1/(1-\lambda)$ times the error of a single step, which can be unacceptable for highly reflective scenes.

Both problems, but especially the evaluation of the integral operator can be successfully attacked by *stochastic iteration*.

The basic idea of stochastic iteration is that instead of approximating operator $\mathcal{T}$ in a deterministic way, a much simpler random operator is used during the iteration which "behaves" as the real operator just in the "average" case.

Suppose that we have a random operator $\mathcal{T}^*$ so that

$$E[\mathcal{T}^*\mathbf{L}] = \mathcal{T}\mathbf{L} \tag{19}$$

for any function $\mathbf{L}$.

During stochastic iteration a random sequence of operators $\mathcal{T}_1^*, \mathcal{T}_2^*, \ldots \mathcal{T}_i^* \ldots$ is generated, which consists of sample realizations of $\mathcal{T}^*$.

Using this sequence of random transport operators, the it-

© Institute of Computer Graphics 1998

erational scheme will not converge, but it will generate samples that fluctuate around the real solution. Thus the solution can be found by averaging the estimates of the subsequent iterational steps.

Formally the sequence of the iteration is the following:

$$\begin{aligned}
\mathbf{L}_1 &= \mathbf{L}^e + \mathcal{T}_1^*\mathbf{L}^e \\
\mathbf{L}_2 &= \mathbf{L}^e + \mathcal{T}_2^*\mathbf{L}^e + \mathcal{T}_2^*\mathcal{T}_1^*\mathbf{L}^e \\
&\dots \\
\mathbf{L}_M &= \mathbf{L}^e + \mathcal{T}_M^*\mathbf{L}^e + \mathcal{T}_M^*\mathcal{T}_{M-1}^*\mathbf{L}^e + \dots
\end{aligned} \tag{20}$$

Averaging the M steps, we obtain:

$$\tilde{\mathbf{L}} = \frac{1}{M}\sum_{i=1}^{M}\mathbf{L}_i =$$

$$\mathbf{L}^e + \frac{1}{M}\sum_{i=1}^{M}\mathcal{T}_i^*\mathbf{L}^e + \frac{1}{M}\sum_{i=1}^{M-1}\mathcal{T}_{i+1}^*\mathcal{T}_i^*\mathbf{L}^e + \dots$$

$$\mathbf{L}^e + \frac{1}{M}\sum_{i=1}^{M}\mathcal{T}_i^*\mathbf{L}^e + \frac{M-1}{M}\cdot\frac{1}{M-1}\sum_{i=1}^{M-1}\mathcal{T}_{i+1}^*\mathcal{T}_i^*\mathbf{L}^e + \dots \tag{21}$$

In order to prove that $\tilde{\mathbf{L}}$ really converges — in the sense of stochastic convergence — to the solution of the integral equation, first it is shown that the expectation value of

$$\mathcal{T}_{i+k}^*\mathcal{T}_{i+k-1}^*\dots\mathcal{T}_{i+1}^*\mathcal{T}_i^*\mathbf{L}^e$$

is $\mathcal{T}^{k+1}\mathbf{L}^e$. For $k=0$, it comes directly from the requirement of equation (19). For $k=1$, the total expectation value theorem can be applied:

$$E[\mathcal{T}_{i+1}^*\mathcal{T}_i^*\mathbf{L}^e] =$$

$$\int E[\mathcal{T}_{i+1}^*\mathcal{T}_i^*\mathbf{L}^e|\mathcal{T}_i^*\mathbf{L}^e = \mathbf{L}]\cdot d\Pr[\mathcal{T}_i^*\mathbf{L}^e = \mathbf{L}] =$$

$$\int E[\mathcal{T}_{i+1}^*\mathbf{L}]\; d\Pr[\mathcal{T}_i^*\mathbf{L}^e = \mathbf{L}]. \tag{22}$$

Using requirement (19) for the expected value inside the integral, then taking advantage of the fact that the order of the integral operator and the expected value calculation can be exchanged, we obtain:

$$\int E[\mathcal{T}_{i+1}^*\mathbf{L}]\; d\Pr[\mathcal{T}_i^*\mathbf{L}^e = \mathbf{L}] = \int \mathcal{T}\mathbf{L}\; d\Pr[\mathcal{T}_i^*\mathbf{L}^e = \mathbf{L}] =$$

$$\mathcal{T}\int \mathbf{L}\; d\Pr[\mathcal{T}_i^*\mathbf{L}^e = \mathbf{L}] = \mathcal{T}E[\mathcal{T}_i^*\mathbf{L}^e] = \mathcal{T}^2\mathbf{L}^e. \tag{23}$$

which concludes our proof for the $k=1$ case. The very same idea can be used recursively for more than two terms ($k \geq 2$ case).

Returning to the averaged solution $\tilde{\mathbf{L}}$, its expected value is then

$$E[\tilde{\mathbf{L}}] = \mathbf{L}^e + \mathcal{T}\mathbf{L}^e + \frac{M-1}{M}\mathcal{T}^2\mathbf{L}^e + \frac{M-2}{M}\mathcal{T}^3\mathbf{L}^e + \dots \tag{24}$$

which converges to the real solution if M goes to infinity. Note also that there is some energy "defect" for higher order terms for finite M values. This can be neglected for high number of iterations, or can even be reduced by ignoring the first few iterations in the averaged result [18].

Finally, it must be explained why random variable $\tilde{\mathbf{L}}$ stochastically converges to its expected value. Looking at formula (21) we can realize that it consists of sums of the following form:

$$\frac{1}{M-k}\cdot\sum_{i=1}^{M-k}\mathcal{T}_{i+k}^*\mathcal{T}_{i+k-1}^*\dots\mathcal{T}_{i+1}^*\mathcal{T}_i^*\mathbf{L}^e.$$

According to the theorems of large numbers, and particularly to the Bernstein[23] theorem, these averages really converge to the expected value if the terms in the average are not highly correlated. It means that random variables $\mathcal{T}_{i+k}^*\mathcal{T}_{i+k-1}^*\dots\mathcal{T}_i^*\mathbf{L}^e$ and $\mathcal{T}_{j+k}^*\mathcal{T}_{j+k-1}^*\dots\mathcal{T}_j^*\mathbf{L}^e$ should not have strong correlation if $i \neq j$ (for the precise definition what strong correlation means here, refer to[23]) This is always true if the sequence of operators are generated from independent random variables, which will be the case in the proposed algorithm.

4.3.1. Definition of the random transport operator

In order to use this general stochastic iterational scheme in practice, the key problem is the definition of the random transport operator. This operator should meet the requirement of equation (19), should be easy to compute and it should allow the compact representation of the $\mathcal{T}_i^*\mathbf{L}$ functions.

Generally the domain of $\mathbf{L}$ is a 2-dimensional continuous space, so is the domain of $\mathcal{T}\mathbf{L}$. From the point of view of compact representation, what we have to avoid is the complete representation of these functions over the complete domain. Thus those random transport operators are preferred, which require the value of $\mathbf{L}$ just in a single direction (or to be more general, just in a few directions).

In itself, this is not enough, since even a single direction can result in a continuous $\mathcal{T}_i\mathbf{L}$ function, which must be stored and re-sampled for the subsequent iteration. The solution of this problem is the postponing of the complete calculation of $\mathcal{T}_i\mathbf{L}$ until it is known where its value is needed in the next iteration step. It means that the random transport operator is decomposed into two phases, where the first phase depends on the current and the second on both the current and the next directions. An appropriate point for the decomposition is when the irradiance is already generated, but its effect is not yet computed on the surfaces.

© Institute of Computer Graphics 1998

A straightforward selection of the random transport operator is the bi-directional transport matrix $\mathbf{T}(\omega', \omega)$ multiplied by 4π. The required phases are established by decomposing the bi-directional transport matrix into the geometry matrix $\mathbf{A}(\omega')$ and the BRDF matrix $\mathbf{F}(\omega', \omega)$.

If the global directions are sampled from a uniform distribution, this selection satisfies equation (19) since

$$E[4\pi \cdot \mathbf{T}(\omega', \omega) \cdot \mathbf{L}(\omega')] =$$

$$\int_{\Omega} \mathbf{F}(\omega', \omega) \cdot \mathbf{A}(\omega') \cdot \mathbf{L}(\omega') \, d\omega' = \mathcal{T}\mathbf{L}(\omega).$$

The complete algorithm — which requires just one variable for each patch i, the irradiance $\mathbf{I}[i]$ — is summarized in the following:

$\mathbf{I} = 0$
for $m = 1$ **to** M **do** *// iterational cycles*
  Generate random global direction $\omega^{(m)}$
  $\mathbf{I} = \mathbf{A}(\omega'_{D-d}) \cdot \left(\mathbf{L}^e(\omega'_{D-d}) + 4\pi \cdot \mathbf{F}(\omega'_{D-d+1}, \omega'_{D-d}) \cdot \mathbf{I}\right)$
  Calculate the image estimate from the irradiance $\mathbf{I}$
  Divide the estimate by M and add to the Image
endfor
Display Image

Note that this algorithm is quite similar to the global walk algorithm, but it does not reinitialize the irradiance vector after each Dth step. In fact, it generates a single infinite walk, and adds the effect of the lightsources to the reflected light field and computes the image accumulation after each step.

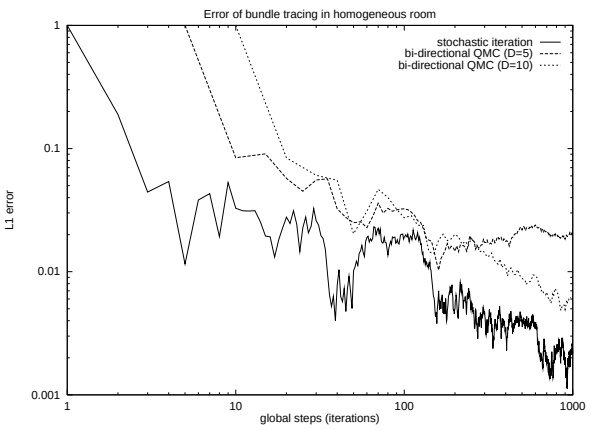


Figure 7: *Stochastic iteration versus bi-directional walking techniques of length 5 and of length 10*

Figure 7 compares the convergence of the stochastic iteration to that of the bi-directional global walks for the homogeneous Cornell box scene. Note that unlike in figure 6 here the axis shows the number of steps instead of the number of walks (a walk consists of 5 or 10 steps respectively). Since global walks provide estimates after a complete walk, their error curves has 5 or 10 times lower horizontal resolution. In contrast to how figure 6 has been generated, in these measurements the bias errors (2^{-6} for $D = 5$ and 2^{-11} for $D = 10$) have not been compensated to demonstrate that the global walks are biased while the stochastic iteration is not.

We can conclude that stochastic iteration is significantly better than the best of the global walk techniques.

5. Calculation of the radiance transport in a single direction

The key of the calculation of the radiance transport is the determination of geometry matrix $\mathbf{A}$.

Examining the elements of the geometry matrix

$$\mathbf{A}(\omega')|_{ij} = \langle b_j(h(\vec{x}, -\omega')) \cdot \cos^* \theta', \tilde{b}_i(\vec{x}) \rangle =$$

$$\int_S b_j(h(\vec{x}, -\omega')) \cdot \tilde{b}_i(\vec{x}) \cdot \cos^* \theta' \, d\vec{x}$$

we can realize that its computation requires the determination of whether a point of the support of basis function i is visible from a point of the support of basis function j in a given direction $-\omega'$ (note that $b_j(h(\vec{x}, -\omega')) \cdot \tilde{b}_i(\vec{x})$ is non zero only if $\vec{x}$ is in the support of $\tilde{b}_i$ and the point $h(\vec{x}, -\omega')$ is in the support of b_j). For the algorithms to be introduced, the support of a basis function is a planar triangle, while the support of an adjoint basis function is either a planar triangle or a single vertex.

Because of the apparent analogies to the transillumination radiosity method[18, 32], the direction ω' is called the *transillumination direction*.

Note that an element of the geometry matrix is non zero only if $\cos^* \theta' \geq 0$, that is if patch i is facing towards the transillumination direction. Faces meeting this requirement are called *front faces*, while those faces which cannot meet this are called *back faces*. Obviously, only front faces can get radiance contribution from a transillumination direction (this can be lifted easily to allow transparent materials).

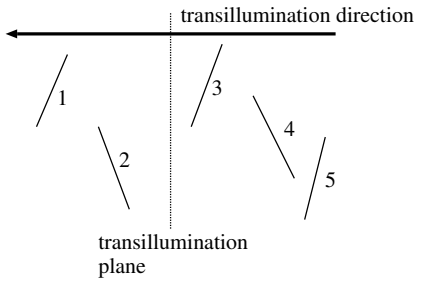


Figure 8: *Global visibility algorithms*

Note that the determination of the geometry matrix is a

© Institute of Computer Graphics 1998

global visibility problem, since only the viewing direction is fixed but the eye position is not. In fact, the eye position should visit all surface points or all vertices depending on the selected adjoint base.

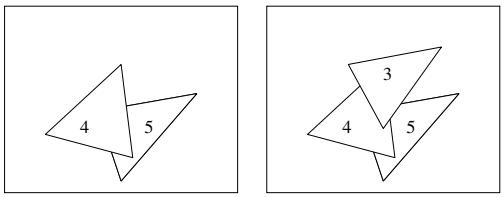


Image seen from patch 3 Image seen from patch 2

Figure 9: *Scene as seen from two subsequent patches*

Looking at figure 8, it is easy to see that the global visibility problem can be solved in an incremental way if the patches are visited in the order of their position in the transillumination direction. In fact what is visible from a patch differs just in a single patch from what is visible from the next patch. This single patch may appear as a new and may hide other patches (figure 9). The required sorting is not obvious if the patches overlap in the transillumination direction, but this can be solved in a way as proposed in the painter's algorithm[19]. On the other hand, in our case the patches are usually small, thus simply sorting them by their center introduces just a negligible error.

Although sorting seems worthwhile, it is not the only alternative. Thus the proposed visibility algorithms will be classified according to whether or not an initial sorting is required.

At a given point of all global visibility algorithms the objects visible from the points of a patch must be known. This information is stored in a data structure called the *visibility map*. The visibility map can also be regarded as an image on the plane perpendicular to the transillumination direction. This plane is called the *transillumination plane* (figure 8).

The algorithms that generate the visibility map can be either discrete or continuous.

For *discrete algorithms* that decompose the transillumination plane to small pixels of size δP, the visibility map is simply a rasterized image where each pixel can store either the index of the visible patch or the radiance of the visible point.

For *continuous algorithms*, the visibility map identifies those regions in which the visibility information is homogeneous (either the same patch is seen or no patch is seen).

Discrete algorithms are faster and the rendering hardware and the z-buffer of workstations can also be exploited [37, 35] but in order to handle all patches simultaneously, the "window" of the algorithm should include all patches. The large window, however, should be decomposed into sufficiently small pixels to provide the required precision, which might result in high resolution requirements for sparse scenes. Continuous algorithms are free from these resolution problems, but are usually more difficult to implement and are much slower.

The computation of the geometry matrix also depends on the selected basis functions and the finite-element algorithm (adjoint basis). We consider two different sets of basis functions and two finite element approaches. In the first case the Galjerkin method is applied for piece-wise constant basis functions. Secondly, piece-wise linear basis functions are used in a point-collocation algorithm.

5.1. Galerkin's method with piece-wise constant basis functions

Let us use the following basis functions

$$b_j(\vec{x}) = \begin{cases} 1 \text{ if } \vec{x} \in A_j, \\ \\ 0 \text{ otherwise.} \end{cases} \tag{25}$$

In Galerkin's method, the unknown directional functions $L_i(\omega)$ are found to ensure that approximation (2) is the real solution of the radiance equation in the subspace induced by the basis functions b_i. To satisfy normalization criteria, the adjoint base is selected as follows:

$$\tilde{b}_j(\vec{x}) = \begin{cases} 1/A_j \text{ if } \vec{x} \in A_j, \\ \\ 0 \text{ otherwise.} \end{cases} \tag{26}$$

Since

$$\langle b_j(\vec{x}), \tilde{b}_i(\vec{x}) \rangle = \int_S b_j(\vec{x}) \cdot \tilde{b}_i(\vec{x}) \, d\vec{x} = \begin{cases} 1 \text{ if } i = j, \\ \\ 0 \text{ otherwise.} \end{cases} \tag{27}$$

the element i, j of the geometry matrix is

$$\mathbf{A}(\omega')|_{ij} = \langle b_j(h(\vec{x}, -\omega')) \cdot \cos^* \theta', \tilde{b}_i(\vec{x}) \rangle =$$

$$\frac{1}{A_i} \cdot \int_{A_i} b_j(h(\vec{x}, -\omega')) \cdot \cos^* \theta' \, d\vec{x}. \tag{28}$$

Since the integrand of this equation is piece-wise continuous, the integral can also be evaluated analytically:

$$\frac{1}{A_i} \cdot \int_{A_i} b_j(h(\vec{x}, -\omega')) \cdot \cos^* \theta' \, d\vec{x} = \frac{A(i, j, \omega')}{A_i}, \tag{29}$$

where $A(i, j, \omega')$ expresses the projected area of patch j that is visible from patch i at direction $-\omega'$. In the unoccluded case this is the intersection of the projections of patch i and patch j onto the transillumination plane. If occlusion occurs, the projected areas of other patches that are in between patch i and patch j should be subtracted as shown in figure 10.

Having the visibility map of patches visible from A_i, the

© Institute of Computer Graphics 1998

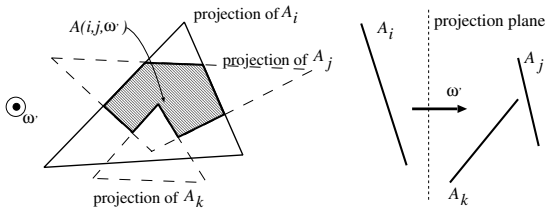


Figure 10: *Interpretation of* $A(i, j, \omega')$

computation of $A(i, j, \omega')$ requires to determine which regions are inside the projection of A_i and to sum the areas.

In the following sections different continuous and discrete visibility algorithms are presented to determine the necessary visibility information.

5.1.1. Continuous algorithm with initial sorting: Local visibility map

This algorithm process the patches in the order defined by the transillumination direction and maintain the visibility graph dynamically [32].

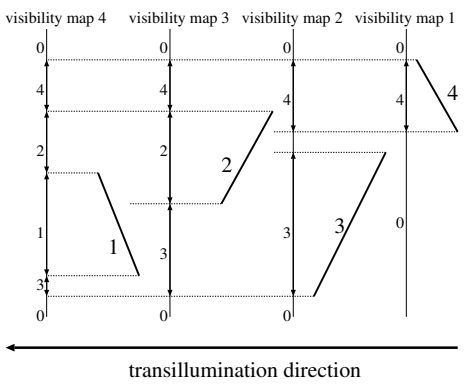


Figure 11: *Local visibility maps*

When the processing of patch i is started, the visibility map shows which patches are visible from patch i. To calculate the $A(i, j, \omega')$ values, those patches which have projections either entirely or partly in the projection of patch i are selected from the visibility map, and are clipped onto patch i to clearly separate inner regions. The process of clipping of the patches onto each other is quite similar to the Weiler-Atherthon visibility algorithm[42]. Then projected areas of those patch parts which are inside the projection of patch i are summed to find the $A(i, j, \omega')$ values.

When we step onto the patch next to patch i, a new visibility map is created by replacing those region parts that are inside the projection of A_i by the projection of A_i. Then, if patch i can reflect energy onto the next patch (it is a back facing patch with respect to the transillumination direction), then patch i should be added to the visibility map, otherwise, the place of the projection of patch i will be empty.

The algorithm, that maintains a list L for the sorted patches, V for the projected patches that are currently in the visibility map, O for those clipped, not-hidden patches whose projections are outside patch i and I for those clipped, not-hidden patches whose projections are inside patch i, is as follows:

list L = Sort patches in direction ω'
visibility map V = { }
for each patch i in L **do**
  Clip patches in V onto patch i and
  generate: O = outside list, I = inside list
  if patch i is front facing **then**
    $A(i, j, \omega') = \sum_{j \in I} A(j)$
    $V = O$
  else
    $V = O$ + patch i
  endif
endfor

If the number of patches is n, then the size of the visibility map is $O(n)$ in the average case but $O(n^2)$ in the worst-case. Thus the resulting algorithm that compares each patch with the actual visibility map will have $O(n^2)$ average case and $O(n^3)$ worst case time-complexity. This is not acceptable when complex scenes are processed. In order to reduce the complexity, we can use the wide selection of computational geometry methods that usually apply spatial decomposition on the plane to reduce the number of unnecessary comparisons[34].

5.1.2. Continuous algorithm without initial sorting: Global visibility map

This algorithm first projects all the polygon vertices and edges onto the transillumination plane, then determine all the intersection points between the projected edges, and form a planar graph that is a superset of the set of projected edges of the polygons [34, 32].

In the resulting planar graph, each territory represents a list of patches that can be projected onto the territory. Furthermore, if the patches do not intersect, the order of patches is also unique in each territory.

Thus, to compute $A(i, j, \omega')$ for some i, the lists of the territories should be visited to check whether i is included. If patch i is found, then the patch next to it on the list should be retained to find index j, and the area of the territory should be added to $A(i, j, \omega')$.

The draft of the algorithm to generate the data structure is the following:

© Institute of Computer Graphics 1998

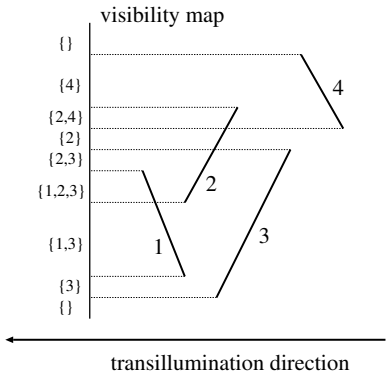


Figure 12: *Global visibility map*

project vertices and edges onto the transillumination plane
calculate all intersection points between projected edges
compute the graph of the induced planar subdivision
for each region of this graph **do**
    Sort patches visible in this region
endfor

The speed of the algorithm is considerably affected by how well its steps are implemented. A simplistic implementation of the intersection calculation, for example, would test each pair of edges for possible intersection. If the total number of edges is n, then the time complexity of this calculation would be $O(n^2)$. Having calculated the intersection points, the structure of the subdivision graph has to be built, that is, incident nodes and arcs have to be assigned to each other somehow. The number of intersection points is $O(n^2)$, hence both the number of nodes and the number of arcs fall into this order. A simplistic implementation of the graph computation would search for the possible incident arcs for each node, giving a time complexity of $O(n^4)$. This itself is inadmissible in practice, not to mention the possible time complexity of the further steps.

However, applying the results of computational geometry, we can do it much better. Algorithms are available that can do it in $O((n+i)\log n)$ [8, 37] time where n is the number of patches (or edges) and i is the number of edge intersections, or even in $O(n^{1+\varepsilon}\sqrt{k})$ time[6] where k is the number of edges in the visibility map. The number of intersections i and the number of edges k are in $O(n^2)$ in the worst-case, but are in $O(n)$ in practical scenes.

5.1.3. Discrete algorithm with initial sorting: Global painter's algorithm

Discrete algorithms determine the visible patches for each front facing patch through a discretized window. This is a visibility problem, and the result is an "image" of the patches, assuming the eye to be on patch i, the window to be on the transillumination plane and the color of patch j to be j if the patch is facing to patch i and to be 0 otherwise.

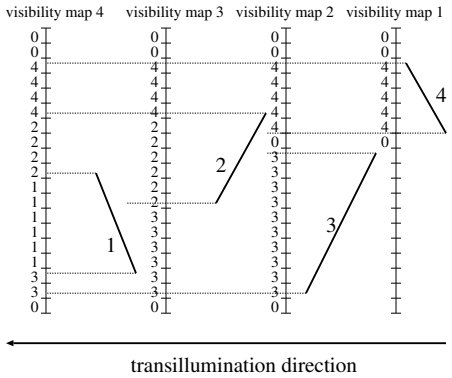


Figure 13: *Application of painter's algorithm*

If the patches are sorted in the transillumination direction and processed in this order, the computation of $A(i,j,\omega')$ requires the determination of the pixel values inside the projection of patch i. Then, to proceed with the next patch in the given order, the pixels covered by patch i are filled with i if patch i is not front facing and 0 otherwise. The two steps can be done simultaneously by a modified scan-conversion algorithm that reads the value of the image buffer before modifying it.

This is summarized in the following algorithm [31]:

Sort patches in direction ω' (painter's algorithm)
for each patch i classify patch to *front* and *back*
Clear image
for each patch i in the sorted order **do**
    if patch i is front facing **then**
        for each pixel of patch i
            j = Read pixel
            $A(i,j,\omega')$ += δP
            Write 0 to the pixel
        endfor
    else Render patch i with color i
endfor

Sorting a data set is known to have $O(n \log n)$ time complexity, so does the painter's algorithm in the average case. A single cycle of the second **for** loop contains only instructions that work with a single patch and an "image", thus the time required for a single cycle is independent of the number of patches. Since the **for** loop executed n number of times, the time complexity of the **for** loop is $O(n)$. Consequently the algorithm requires $O(n \log n)$ time.

© Institute of Computer Graphics 1998

5.1.4. Discrete algorithm without initial sorting: exploitation of the hardware z-buffer

In this section another method is proposed that traces back the visibility problem to a series of z-buffer steps to allow the utilization of the z-buffer hardware of workstations [36].

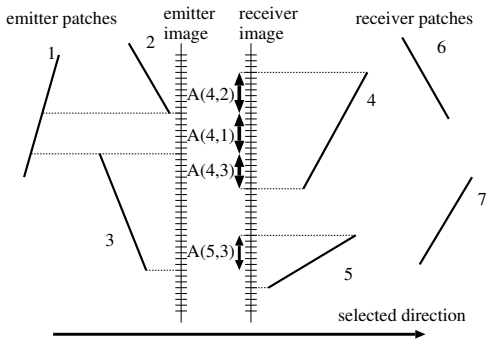


Figure 14: *Calculating the power transfer*

The radiance is transferred by dynamically maintaining two groups of patches, an *emitter group* and a *receiver group*, in a way that no patch in the receiver group is allowed to hide a patch in the emitter group looking from the given transillumination direction.

Let the two classes of patches be rendered into two image buffers — called the emitter and receiver images, respectively — setting the color of patch j to j and letting the selected direction be the viewing direction for the receiver set and its inverse for the emitter set.

Looking at figure 14, it is obvious that a pair of such images can be used to calculate the radiance transfer of all those patches which are fully visible in the receiver image. The two images must be scanned parallely and when i (that is the index of patch i) is found in the receiver image, the corresponding pixel in the emitter image is read and its value (j) is used to decide which $A(i, j, \omega')$ should be increased by the area of the pixel.

In order to find out which patches are fully visible in the receiver image, the number of pixels they cover is also computed during scanning and then compared to the size of their projected area. For those patches whose projected area is approximately equal to the total size of the covered pixels, we can assume that they are not hidden and their accumulated irradiances are valid, thus these patches can be removed from the receiver set and rendered into the emitter image to calculate the radiance transfer for other patches (this is the strategy to maintain the emitter and receiver sets automatically).

This leads to an incremental algorithm that initially places all patches in the receiver set. Having calculated the receiver image by the z-buffer algorithm, the radiance transfer for the fully visible patches are evaluated, and then they are moved from the receiver set to the emitter set. The algorithm keeps doing this until no patch remains in the receiver set (cyclic overlapping would not allow the algorithm to stop, but this can be handled by a clipping as in the painter's algorithm[19]). The number of z-buffer steps required by the algorithm is quite small even for complex practical scenes [24]. Exploiting the built-in z-buffer hardware of advanced workstations, the computation can be fast.

In the following algorithm R denotes the collection of the receiver patches.

```
R = all patches
Clear emitter_image
while R is not empty
    Clear receiver_image
    Render patches in R into receiver_image via z-buffer
    for each pixel P
        r = receiver_image[P]
        e = emitter_image[P]
        patch[r].visible_size += δP
        A(r, e, ω') += δP
    endfor
                                    // Move patches to the emitter set
    for each patch p
        if patch[p].visible_size ≈ projected size of patch p
          Remove patch p from R
                and render it to emitter_image
        endif
    endfor
endwhile
```

When checking whether or not the visible size is approximately equal to the projected patch size, the allowed tolerance is the total area of the pixels belonging to the edge of the patch, which in turn equals to the sum of the horizontal and vertical sizes of a triangular patch.

This algorithm "peels" the scene by removing the layers one by one. The sequences of evolving receiver and emitter images are shown in figure 15 and in figure 16, respectively. Note that the first receiver image contains all patches, thus its pair is an empty image that is not included. The pair of the second receiver image is the first shown emitter image, etc. The last emitter image includes all patches thus its emitter pair is empty, and therefore is not shown.

5.1.5. Discrete algorithm without initial sorting: software z-buffer

For the sake of completeness, we mention that the global z-buffer algorithm [18] can also be used for our purposes. This method stores not just the closest patch index and its z value in the buffer, but the whole list of those patches which can be projected onto this pixel. The patches are scan-converted by the z-buffer algorithm, and are inserted into the lists associated with the covered pixels. The lists of pixels can be used to compute $A(i, j, \omega')$ similarly to the continuous global visibility map algorithm.

© Institute of Computer Graphics 1998

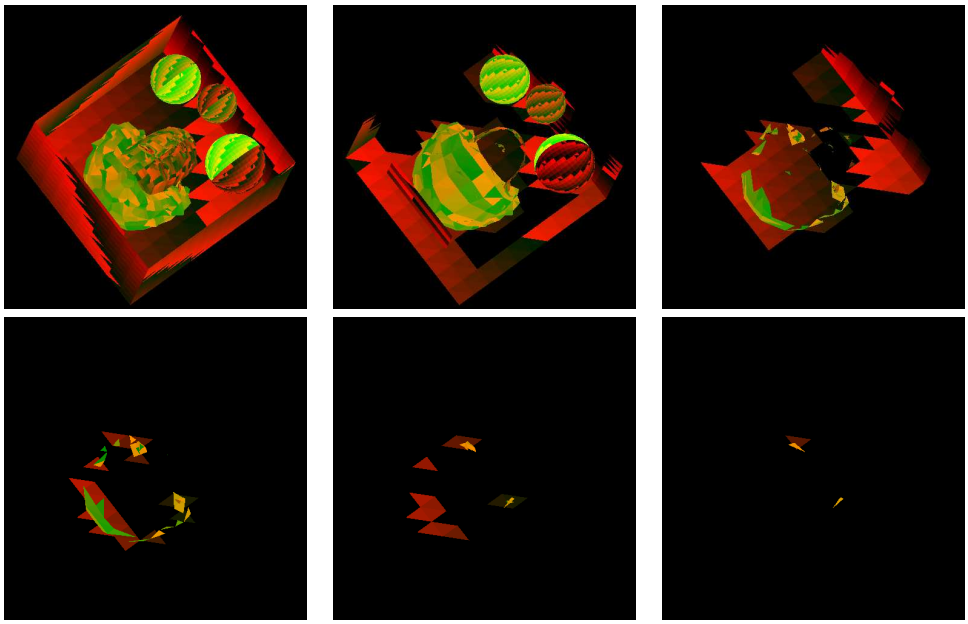


Figure 15: *Steps of the evolution of receiver images*

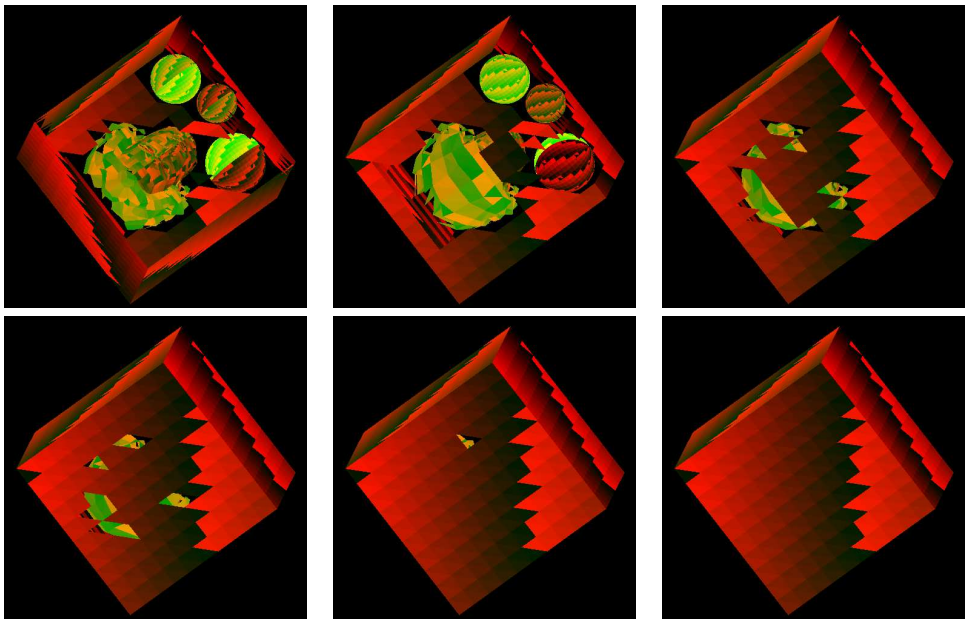


Figure 16: *Steps of the evolution of emitter images*

© Institute of Computer Graphics 1998

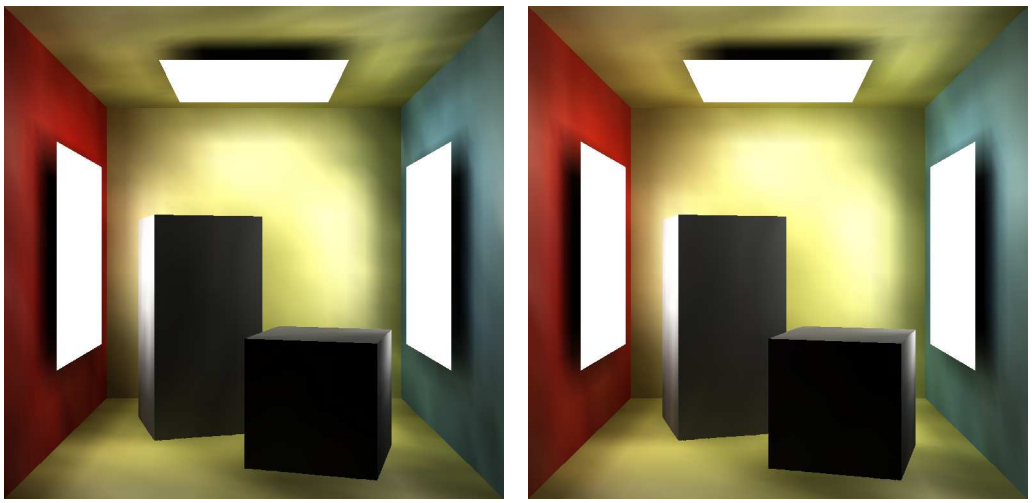


Figure 17: *A Cornell box as rendered using* 100×100 *(left) and* 1000×1000 *(right) pixels for the visibility map*

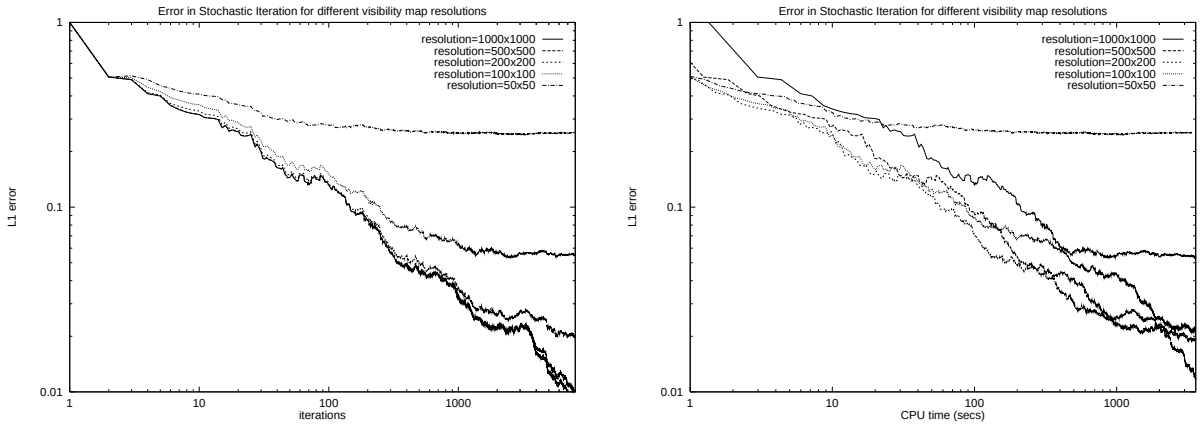


Figure 18: *Comparison of the error curves using visibility maps of different resolutions as a function of iterations (left) and of computation time (right)*

5.2. Analysis of the finite resolution problem of discrete methods

In order to find out how important the resolution of the visibility map, a Cornell box scene (figure 17) consisting of 3705 triangular patches has been rendered with the global painter's algorithm having set the resolution to different values. Since the resolution can only be interpreted when compared to the size of the patches, table 2 summarizes the average projected patch sizes in pixels and also the residual errors of the iteration.

Note that the resolution plays negligible role until the average patch per pixel ratio is significantly greater than one (left of figure 18). For instance, the error curves of resolu-

resolution	pixel-per-patch	residual error
50×50	0.5	0.25
100×100	2	0.05
200×200	8	0.02
500×500	55	≤ 0.01
1000×1000	220	≤ 0.01

Table 2: *Average pixel per patch and the discretization errors*

© Institute of Computer Graphics 1998

tions 1000×1000 and 500×500 can hardly be distiquished in the left side of figure 18. This can be explained by the stochastic nature of the algorithm. Each radiance transfer uses a different direction, thus a different discrete approximation of the size of the patch. Although this approximation can be quite inaccurate in a single step, the expected value of these approximations will still be correct. As the algorithm generates the result as the average of the estimates, these approximation errors will be eliminated. The effect of the low resolution is just an "additional noise".

However, when the pixel size becomes comparable to the projected size of the patches, then some of the patches may be omitted in each radiance transfer, which generates energy defect. The iteration will deteriorate from the real solution as we can clearly see it in figure 18 when the resolution is lower than 200×200.

The computation time is roughly proportional to the number of pixels in the visibility map thus it is desireable to keep the resolution low (right of figure 18). For example, the 500 stochastic iterations that generated the left and the right of figure 17 needed 3 and 8 minutes respectively. The optimal selection of the resolution is the minimal number which guarantees that even the smallest patches are projected onto a few pixels.

5.3. Point collocation method with piece-wise linear basis functions

In this method [33] the radiance variation is assumed to be linear on the triangles. Thus, each vertex i of the triangle mesh will correspond to a "tent shaped" basis function b_i that is 1 at this vertex and linearly decreases to 0 on the triangles incident to this vertex (figure 19). Assume that the shading normals are available at the vertices.

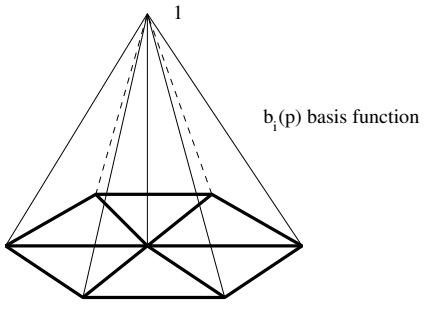


Figure 19: *Linear basis function in 3-dimension*

In the point-collocation method, the unknown directional functions $L_j(\omega)$ are determined to ensure that the residual of the approximation is zero at the vertices of the triangle mesh. This corresponds to Dirac-delta type adjoint basis functions, where $\tilde{b}_j(\vec{x})$ is non-zero at vertex i only.

Thus the geometry matrix is

$$\mathbf{A}(\omega')|_{ij} = \langle b_j(h(\vec{x}, -\omega')) \cdot \cos^* \theta', \delta(\vec{x} - \vec{x}_i) \rangle =$$

$$b_j(h(\vec{x}_i, -\omega')) \cdot \cos^* \theta'(\vec{x}_i). \tag{30}$$

5.3.1. Calculation of the irradiance at vertices

Since now the irradiance is not piece-wise constant but piece-wise linear, it is better to evaluate $\mathbf{A}(\omega') \cdot \mathbf{I}$ directly than evaluating the geometry matrix and the irradiance separately. Thus we have to find

$$(\mathbf{A}(\omega') \cdot \mathbf{I})[i] = \sum_{j=1}^{n} b_j(h(\vec{x}_i, -\omega')) \cdot \cos^* \theta'(\vec{x}_i) \cdot \mathbf{I}[j]. \tag{31}$$

In this formula the $\tilde{b}_j(h(\vec{x}_i, -\omega'))$ factor is non-zero for those j indices which represent a vertex of the patch visible from $\vec{x}_i$ at direction ω'. The exact value can be derived from the calculation of the height of the linear "tent" function at point $h(\vec{x}_i, -\omega')$. This means that having identified the patch visible from $\vec{x}_i$ at direction ω', the required value is calculated as a linear interpolation of the irrandiances of the vertices of this patch.

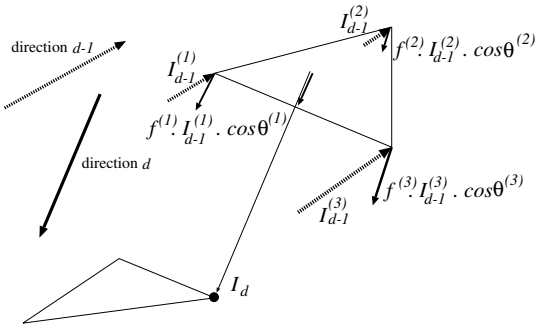


Figure 20: *Global visibility algorithm for the vertices*

To solve it for all patches, the triangular patches are sorted in direction ω', then painted one after the other into an image buffer. For vertex i of each triangle, the "color" is set to

$$\mathbf{L}_i^e(\omega'_{D-d}) + 4\pi \cdot \mathbf{F}_{ii}(\omega'_{D-d+1}, \omega_{D-d}) \cdot \mathbf{I}[i]$$

at step d and the linear interpolation hardware (Gouraud shading) is used to generate the color (or irradiance) inside the triangle. For back-facing patches this step clears the place of the triangle in the "image".

If the triangles that are in front of the given triangle in direction ω' are rendered into the image buffer, then the radiance illuminating the vertices of the given triangle is readily available in the current image buffer. Assuming that patches are processed in the order of the transillumination direction, every patch should be rendered only once into the image buffer.

© Institute of Computer Graphics 1998

Thus the calculation of the irradiances at a given transillumination direction is:

```
Sort patches in direction ω' (painter's algorithm)
for each patch i classify patch to front and back
Clear image-buffer
for each patch i in sorted order do
  if patch i is front facing then
    for each vertex v[i] of the patch i
      color[v[i]] = L^e_{v[i]}(ω'_{D-d}) + 4π·
                    F_{v[i],v[i]}(ω'_{D-d+1}, ω'_{D-d}) · I[v[i]]
    endfor
    Render patch i into the image-buffer
  else
    for each vertex v[i] of the patch i
      I[v[i]] = (image buffer at projection of v[i])·
                cos* θ'[v[i]]
    endfor
    Render patch i with color 0
  endif
endfor
```

The processing of a single direction for all patches require a sorting step and the rendering of each triangle into a temporary buffer. This can be done in $O(n \log n)$ time.

6. Handling sky-light illumination

The visibility methods introduced so far can easily be extended for sky-light illumination by initializing the image on the transillumination plane by a special value if the direction points downwards (sky is usually above the horizon). When the radiance is transferred and this value is found in a given pixel, then the irradiance of the receiver is updated according to the intensity of the sky-light.

7. Application of importance sampling

The global illumination method proposed so far is particularly efficient if the lighting distribution in the scene does not exhibit high variations. For difficult lighting conditions importance sampling can help, which prefers those sequences of directions that transport significant radiance towards the eye.

Importance sampling is a general variance reduction method that can improve Monte-Carlo and quasi-Monte Carlo quadrature [30].

Suppose that integral $I = \int_V f(\mathbf{z})\ d\mathbf{z}$ needs to be evaluated. For our case, I represents the image and f is responsible for determining the contribution of a global path denoted by $\mathbf{z}$, thus both I and f are vectors. In order to rank the domain points according to the size of $f(\mathbf{z})$, a scalar importance function $\mathcal{I}(\mathbf{z})$ must be defined that can show where the elements in vector f are large.

A straightforward definition of this importance function is letting $\mathcal{I}$ be the sum of luminances of all pixels of the image. This importance function really concentrates on those walks that have a significant influence on the image. Using the luminance information is justified by the fact that the human eye is more sensitive to luminance variations than to color variations.

Importance sampling requires the generation of samples $\{\mathbf{z}_1, \mathbf{z}_2, \ldots \mathbf{z}_M\}$ according to a probability density $p(\mathbf{z})$ — which is at least approximately proportional to $\mathcal{I}(\mathbf{z})$ — and using the following formula:

$$I = \int_V \frac{f(\mathbf{z})}{p(\mathbf{z})} \cdot p(\mathbf{z})\ d\mathbf{z} = E\left[\frac{f(\mathbf{z})}{p(\mathbf{z})}\right] \approx \frac{1}{M} \cdot \sum_{i=1}^{M} \frac{f(\mathbf{z}_i)}{p(\mathbf{z}_i)}. \tag{32}$$

Since no a-priori information is available which these important directions are, some kind of adaptive technique must be used. In this section the application of the VEGAS and Metropolis sampling methods are considered.

7.1. VEGAS sampling

The VEGAS algorithm[15] is an adaptive Monte-Carlo method that generates a probability density automatically and in a separable form. The reason of the requirement of separability is that D number of k-dimensional tables need much less space than a single $D \cdot k$-dimensional table. Formally, let us assume that the probability density can be defined in the following product form:

$$p(\omega_1, \omega_2, \ldots, \omega_D) \propto g_1(\omega_1) \cdot g_2(\omega_2) \ldots \cdot g_D(\omega_D). \tag{33}$$

It can be shown [15] that the optimal selection of g_1 is

$$g_1(\omega_1) = \sqrt{\int \frac{\mathcal{I}^2(\omega_1, \ldots, \omega_D)}{g_2(\omega_2) \ldots g_D(\omega_D)}\ d\omega_2 \ldots d\omega_D}, \tag{34}$$

and similar formulae apply to $g_2, \ldots, g_D$.

These $g_1, \ldots g_D$ functions can be tabulated as 2-dimensional arrays (note that a single direction is defined by 2 scalars ϕ and θ). The (i, j) element of this matrix represents the importance of the directional region where

$$\phi \in [(\frac{2i\pi}{N}, \frac{2(i+1)\pi}{N}], \quad \theta \in [\frac{j\pi}{N}, \frac{(j+1)\pi}{N})].$$

This immediately presents a recursive importance sampling strategy. The algorithm is decomposed into phases consisting of a number of samples. At the end of each phase weights $g_1, \ldots g_D$ are refined, to provide a better probability density for the subsequent phase. Assuming that $g_1, \ldots g_D$ are initially constant, a standard Monte-Carlo method is initiated, but in addition to accumulating to compute the integral, $g_1, \ldots g_D$ are also estimated using equation (34). The

© Institute of Computer Graphics 1998

Monte-Carlo estimate of the new $g_1(\omega_1)$ is

$$g_1^{(\mathrm{new})}(\omega_1) = \sqrt{\sum_{i=1}^{M} \frac{\mathcal{I}^2(\omega_1,\ldots,\omega_D)}{g_2^2(\omega_2)\ldots g_D^2(\omega_D)}}. \qquad (35)$$

Then for the following phase, the samples are selected according to the g functions. In order to calculate a sample for ω_i, for instance, a single random value is generated in the range of 0 and the sum of all elements in the array defining g_i. Then the elements of the array is retained one by one and summed to a running variable. When this running variable exceeds the random sample, then the searched directional region is found. The direction in this region is then found by uniformly selecting a single point from the region.

VEGAS method is not optimal in the sense that the probability density can only be approximately proportional to the importance even in the limiting case since only product form densities are considered.

We have to mention that the original VEGAS method used 1-dimensional g functions, but in our case, the 2 scalars defining a single direction are so strongly correlated, thus it is better not to separate them.

In theory, higher dimensional tables could also be used, but this would pose unacceptable memory requirements.

8. Metropolis sampling

The Metropolis algorithm[17] is a Monte-Carlo quadrature method that incorporates adaptive importance sampling by exploring the properties of the integrand automatically. Unlike the VEGAS method, it converges to the optimal probability density that is proportional to the importance, that is in the limiting case:

$$\mathcal{I}(\mathbf{z}) = b \cdot p(\mathbf{z}).$$

However, this probability density cannot be stored, thus in the Monte-Carlo formula the importance should be used instead, in the following way:

$$I = \int_V \frac{f(\mathbf{z})}{\mathcal{I}(\mathbf{z})} \cdot \mathcal{I}(\mathbf{z})\, d\mathbf{z} = b \cdot \int_V \frac{f(\mathbf{z})}{\mathcal{I}(\mathbf{z})} \cdot p(\mathbf{z})\, d\mathbf{z} =$$

$$b \cdot E\left[\frac{f(\mathbf{z})}{\mathcal{I}(\mathbf{z})}\right] \approx \frac{b}{M} \cdot \sum_{i=1}^{M} \frac{f(\mathbf{z}_i)}{\mathcal{I}(\mathbf{z}_i)} \qquad (36)$$

In order to generate samples according to $p(\mathbf{z}) = 1/b \cdot \mathcal{I}(\mathbf{z})$ a Markovian process is constructed whose stationary distribution is just $p(\mathbf{z})$. The definition of this Markovian process $\{\mathbf{z}_1, \mathbf{z}_2, \ldots \mathbf{z}_i \ldots\}$ is as follows:

for $i = 1$ **to** M **do**
  Based on the actual state $\mathbf{z}_i$,
    choose another random, tentative point $\mathbf{z}_t$
  if $(\mathcal{I}(\mathbf{z}_t) \geq \mathcal{I}(\mathbf{z}_i))$ **then** accept $(\mathbf{z}_{i+1} = \mathbf{z}_t)$
  else *// accept with the importance degradation*
    Generate random number r in $[0, 1]$.
    if $r < \mathcal{I}(\mathbf{z}_t)/\mathcal{I}(\mathbf{z}_i)$ **then** $\mathbf{z}_{i+1} = \mathbf{z}_t$
    else $\mathbf{z}_{i+1} = \mathbf{z}_i$
  endif
endfor

The generation of the next tentative sample is governed by a *tentative transition function* $T(\mathbf{x} \to \mathbf{y})$. In the algorithm we use symmetric a tentative transition function, that is $T(\mathbf{x} \to \mathbf{y}) = T(\mathbf{y} \to \mathbf{x})$. The transition probability of this Markovian process is:

$$P(\mathbf{x} \to \mathbf{y}) = \begin{cases} T(\mathbf{x} \to \mathbf{y}) & \text{if } \mathcal{I}(\mathbf{y}) > \mathcal{I}(\mathbf{x}), \\ T(\mathbf{x} \to \mathbf{y}) \cdot \mathcal{I}(\mathbf{y})/\mathcal{I}(\mathbf{x}) & \text{otherwise.} \end{cases} \qquad (37)$$

In equilibrium state, the transitions between two states $\mathbf{x}$ and $\mathbf{y}$ are balanced, that is

$$p(\mathbf{x}) \cdot P(\mathbf{x} \to \mathbf{y}) = p(\mathbf{y}) \cdot P(\mathbf{y} \to \mathbf{x}).$$

Using this and equation (37), and then taking into account that the tentative transition function is symmetric, we can prove that the stationary probability distribution is really proportional to the importance:

$$\frac{p(\mathbf{x})}{p(\mathbf{y})} = \frac{P(\mathbf{y} \to \mathbf{x})}{P(\mathbf{x} \to \mathbf{y})} = \frac{T(\mathbf{y} \to \mathbf{x})}{T(\mathbf{x} \to \mathbf{y})} \cdot \frac{\mathcal{I}(\mathbf{x})}{\mathcal{I}(\mathbf{y})} = \frac{\mathcal{I}(\mathbf{x})}{\mathcal{I}(\mathbf{y})}. \qquad (38)$$

If we select initial points according to the stationary distribution — that is proportionally to the importance — then the points visited in the walks originated at these starting points can be readily used in equation (36).

8.1. Metropolis solution of the directional integrals

The Metropolis approximation of the radiance vector is:

$$\mathbf{L}(\omega) =$$

$$(\frac{1}{4\pi})^D \int_\Omega \ldots \int_\Omega \left[\mathbf{L}^e(\omega) + 4\pi \cdot \mathbf{F}(\omega_1', \omega) \cdot \mathbf{I}_D\right] d\omega_D' \ldots d\omega_1' \approx$$

$$(\frac{b}{M}) \sum_{m=1}^{M} \frac{\mathbf{L}^e(\omega) + 4\pi \cdot \mathbf{F}(\omega_1', \omega) \cdot \mathbf{I}_D(m)}{\mathcal{I}(\omega_1^{(m)}, \omega_2^{(m)}, \ldots, \omega_D^{(m)})}. \qquad (39)$$

where M is the number of samples (also called *mutations*) and b is the integral of the importance function over the whole space.

© Institute of Computer Graphics 1998

8.1.1. Definition of the tentative transition function

The statespace of the Markovian process consists of D-dimensional vectors of directions that define the sequence of directions in the global walks. Thus the tentative transition function is allowed to modify one or more directions in these sequences.

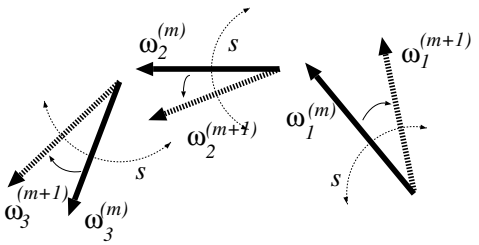


Figure 21: *Mutation strategy*

The set of possible sequences of directions can be represented by a $2D$-dimensional unit cube (each direction is defined by two angles).

In the actual implementation random mutations that are uniformly distributed in a $2D$ dimensional cube of edge-size s are used. In order to find the extent of the random perturbation, several, contradicting requirements must be taken into consideration.

First of all, in order to cover the whole statespace of unit size, the number of mutations should be much greater than s^{-2D}. From a different point of view this states that the mutations cannot be very small. Small mutations also emphasize the start-up bias problem which is a consequence of the fact that the Markovian process only converges to the desired probability density (this phenomenon will be examined in detail later).

On the other hand, if the mutations are large, then the Markovian process "forgets" which regions are important, thus the quality of importance sampling will decrease.

Finally, another argument against small mutations is that it makes the subsequent samples strongly correlated. Note that Monte-Carlo quadrature rules usually assume that the random samples are statistically independent, which guarantees that if the variance of random variable $f(\mathbf{z})$ is σ, then the variance of the Monte-Carlo quadrature will be $\sigma/\sqrt{M}$ after evaluating M samples. Since Metropolis method uses statistically correlated samples, the variance of the quadrature can be determined using the Bernstein theorem[23], which states that the variance of the quadrature is

$$\sigma \cdot \sqrt{\frac{1 + 2\sum_{k=1}^{M} R(k)}{M}} \tag{40}$$

where $R(k)$ is an upperbund of the correlation between $f(\mathbf{z}_i)$ and $f(\mathbf{z}_{i+k})$. It means that strong correlation also increases the variance of the integral estimate.

8.1.2. Generating an initial distribution

The Metropolis method promises to generate samples with probabilities proportional to their importance in the stationary state.

Although the process converges to this probability from any initial distribution as shown in figure 22, the samples generated until the process is in the stationary state should be ignored.

To ensure that the process is already in the stationary case from the beginning, initial samples are also selected according to the stationary distribution, i.e. proportionally to the importance function. Selecting samples with probabilities proportional to the importance can be approximated in the following way. A given number of seed points are found in the set of sequences of global directions. The importances of these seed points are evaluated, then, to simulate the distribution following this importance, the given number of initial points are selected randomly from these seed points using the discrete distribution determined by their importance.

8.1.3. Automatic exposure

Equation (39) also contains an unknown b constant that expresses the luminance of the whole image. The initial seed generation can also be used to determine this constant. Then at a given point of the algorithm the total luminance of the current image — that is the sum of the importances of the previous samples — is calculated and an effective scaling factor is found that maps this luminance to the expected one.

8.2. Variance reduction

Metropolis method may ignore calculated function values if their importance is low. However, these values can be used to reduce variance. Suppose that the importance degrades at step i. Thus the the process is in $\mathbf{z}_t$ with probability $(1 - \mathcal{I}(\mathbf{z}_t))/\mathcal{I}(\mathbf{z}_i)$ and in $\mathbf{z}_{i+1}$ with probability $\mathcal{I}(\mathbf{z}_t)/\mathcal{I}(\mathbf{z}_i)$. In order to compute the integral quadrature, random variable $f(\mathbf{z}_i)/\mathcal{I}(\mathbf{z}_i)$ is needed. A common variance reduction technique is to replace a random variable by its mean, thus we can use

$$\left(1 - \frac{\mathcal{I}(\mathbf{z}_t)}{\mathcal{I}(\mathbf{z}_i)}\right) \cdot \frac{f(\mathbf{z}_i)}{\mathcal{I}(\mathbf{z}_i)} + \frac{\mathcal{I}(\mathbf{z}_t)}{\mathcal{I}(\mathbf{z}_i)} \cdot \frac{f(\mathbf{z}_t)}{\mathcal{I}(\mathbf{z}_t)}.$$

instead of $f(\mathbf{z}_i)/\mathcal{I}(\mathbf{z}_i)$.

8.3. Evaluation of the performance of the Metropolis method

To evaluate the new algorithm, and particularly the efficiency of the Metropolis sampling the scene of figure 23 has been selected. The surfaces have both diffuse and specular reflection and the lightsource is well hidden from the camera (figure 23).

The error measurements of the Metropolis method with

© Institute of Computer Graphics 1998

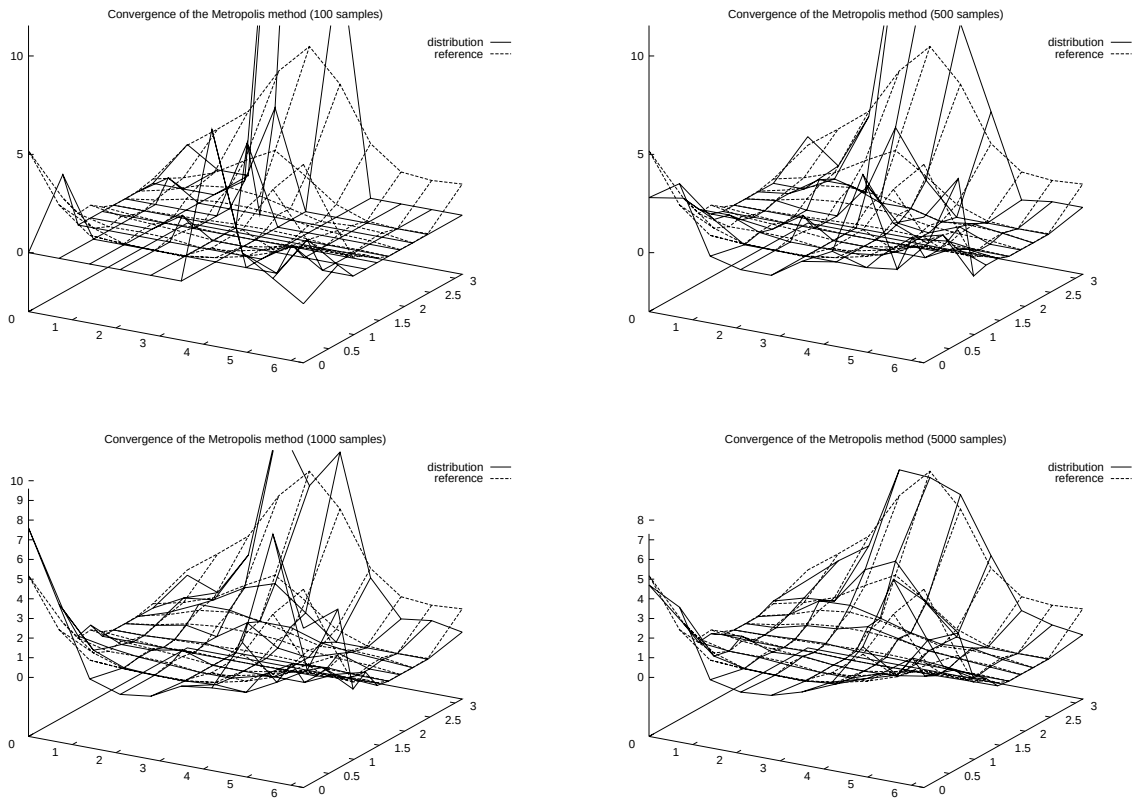


Figure 22: *Convergence of the first-bounce as computed by 100, 500, 1000 and 5000 Metropolis samples*

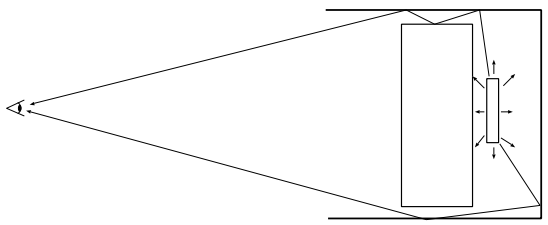


Figure 23: *A test scene*

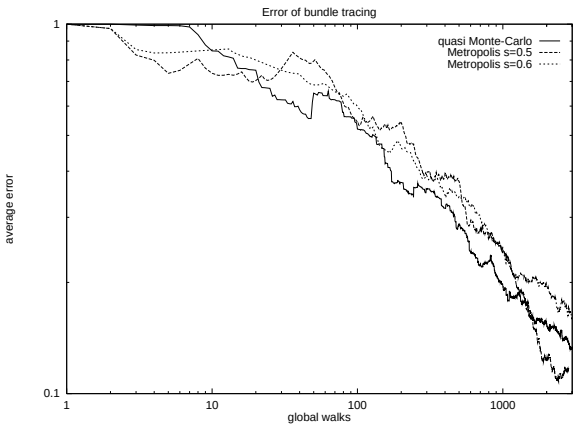


Figure 24: *Error measurements for the "difficult scene"*

different perturbation size, and for quasi-Monte Carlo samples are shown by figure 24.

The image generated using Metropolis walks are shown infigure 25.

Considering the performance of the Metropolis method for our algorithms, we have to conclude that for homogeneous scenes, it cannot provide significant noise reduction compared to quasi-Monte Carlo walks. This is due to the fact that the integrand of equation (7) is continuous

© Institute of Computer Graphics 1998

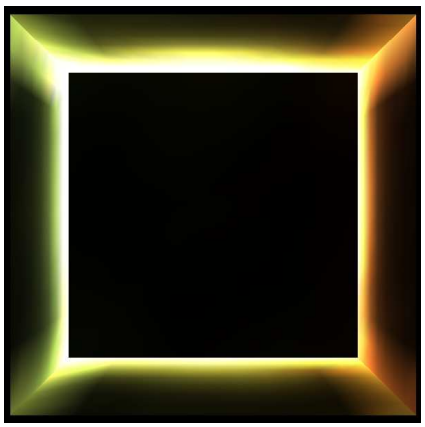


Figure 25: *The image of the scene with difficult lighting rendered Metropolis walks*

and is of finite variation unlike the integrand of the original rendering equation, thus if its variation is modest then quasi-Monte quadrature is almost unbeatable. The combined and bi-directional walking techniques cause even further smoothing which is good for the quasi-Monte Carlo but bad for the Metropolis sampling.

On the other hand the number of samples was quite low (we used a few thousand samples, while in [40] the number of samples was 50 million). For so few samples the Metropolis method suffers from the problems of initial bias and correlated samples. Due to the smooth integrand, the drawbacks are not compensated by the importance sampling.

8.4. Evaluation of the start-up bias

In order to theoretically evaluate the start-up bias, let us examine a simplified, 1-dimensional case when the importance is constant, thus the transition proposed by the tentative transition function is always accepted.

In this case, the probability density in the equilibrium is constant. The question is how quickly the Metropolis method approaches to this constant density (figure 26).

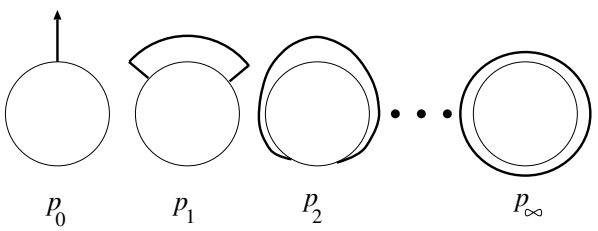


Figure 26: *Evaluation of the uniform distribution*

Metropolis method can generate samples following a given probability density in a closed interval. Since random mutations may result in points that are outside the closed interval, the boundaries should be handled in a special way.

If the variable of an integrand denotes "angle of direction", then the interval can be assumed to be "circular", that is, the external points close to one boundary are equivalent to the internal points of the other boundary. Using this assumption, let us suppose that the domain of the integration is $[-\pi, \pi]$ and the integrand is periodic with 2π.

Let the probability distribution at step n be p_n. The Metropolis method is initiated from a single seed at 0, thus $p_0 = \delta(x)$. Assume that transition probability $P(y \to x)$, which is equal to the tentative transition probability for constant importance, is homogeneous, that is $P(y \to x) = P(x - y)$. Using the total probability theorem, the following recursion can be established for the sequence of p_n:

$$p_{n+1}(x) = \int_{-\infty}^{\infty} p_n(y) \cdot P(x \to y)\, dy =$$

$$\int_{-\infty}^{\infty} p_n(y) \cdot P(x - y)\, dy = p_n * P, \tag{41}$$

where $*$ denotes the convolution operation.

Applying Fourier transformation to this iteration formula, we can obtain:

$$p^*_{n+1} = p^*_n \cdot P^*, \tag{42}$$

where $p^*_{n+1} = \mathcal{F} p_{n+1}$, $p^*_n = \mathcal{F} p_n$ and $P^* = \mathcal{F} P$.

Since the domain is "circular", i.e. x denotes the sample point as $x + 2k\pi$ for any integer k, the probability density is periodic, thus it can be obtained as a Fourier series:

$$p_n(x) = \sum_{k=-\infty}^{\infty} a_k^{(n)} e^{\jmath k x}, \tag{43}$$

where $\jmath = \sqrt{-1}$. The Fourier transform is thus a discrete spectrum:

$$p^*_n(f) = \sum_{k=-\infty}^{\infty} a_k^{(n)} \cdot \delta(f - k) \tag{44}$$

Substituting this into equation (42), we get

$$p^*_{n+1}(f) = \left(\sum_{k=-\infty}^{\infty} a_k^{(n)} \cdot \delta(f - k) \right) \cdot P^*(f) =$$

$$\sum_{k=-\infty}^{\infty} a_k^{(n)} \cdot P^*(k) \cdot \delta(f - k), \tag{45}$$

thus $a_k^{(n+1)} = a_k^{(n)} \cdot P^*(k)$.

© Institute of Computer Graphics 1998

Using the same concept n times, and taking into account that the initial distribution is $\delta(x)$, we can obtain:

$$p_n^*(f) = \sum_{k=-\infty}^{\infty} \left(P^*(k)\right)^n \cdot \delta(f-k) \tag{46}$$

thus in the original domain

$$p_n(x) = \sum_{k=-\infty}^{k=\infty} \left(P^*(k)\right)^n \cdot e^{jkx} \tag{47}$$

The L_2 error between p_n and the stationary distribution is then

$$||p_n - p_\infty||_2 = \sqrt{\int_0^1 |p_n(x) - a_0^{(\infty)}|^2 \, dx} \tag{48}$$

Note that according to the definition of the Fourier series

$$a_0^{(n)} = \frac{1}{2\pi} \cdot \int_{-\pi}^{\pi} p_n(x) \, dx = 1 \tag{49}$$

independently of n, thus $a_0^{(\infty)}$ is also 1. Using this and substituting equation (47) in equation (48), we get the following error for the distribution:

$$||p_n - p_\infty||_2 = \sqrt{\sum_{k=-\infty, k\neq 0}^{k=\infty} |P^*(k)|^{2n}} \tag{50}$$

8.4.1. Starting from multiple seeds

So far we have assumed that the integrand is estimated from a single random walk governed by the Markovian process. One way of reducing the startup bias is to use several walks initiated from different starting points, called seeds, and combine their results.

If the initial point is generated from seedpoints $x_1, x_2, \ldots, x_N$ randomly selecting x_i with probability α_i ($\sum_{i=1}^{N} \alpha_i = 1$), then the initial probability distribution is the following

$$p_0(x) = \sum_{i=1}^{N} \alpha_i \cdot \delta(x - x_i) \tag{51}$$

Using the same concept as before, the probability density after n steps can be obtained in the following form

$$p_n(x) = \sum_{k=-\infty}^{k=\infty} \sum_{i=1}^{N} \alpha_i \cdot \left(P^*(k)\right)^n \cdot e^{jk(x-x_i)} \tag{52}$$

The error of the probability distribution after step n is then

$$||p_n - p_\infty||_2 = \sqrt{\sum_{k=-\infty, k\neq 0}^{k=\infty} \left| P^*(k) \cdot \sum_{i=1}^{N} \alpha_i \cdot e^{jk(x-x_i)} \right|^{2n}}. \tag{53}$$

8.4.2. Analysis of uniform random perturbations

Let the perturbation be the selection of a point following uniform distribution from an interval of size Δ centered by the current point. Formally the transition probability is

$$P(x \to y) = \begin{cases} 1/\Delta & \text{if } |x-y| < \Delta, \\ 0 & \text{otherwise.} \end{cases} \tag{54}$$

The Fourier transform of this function is

$$P^*(k) = \frac{\sin k\pi\Delta}{k\pi\Delta} \tag{55}$$

which can be rather big even for large k values. This formula, together with equation (50) allows to generate the graph of the startup errors for different sample numbers and for different perturbation size (figure 27).

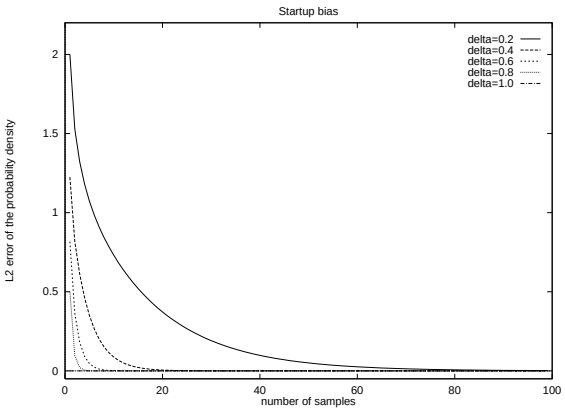


Figure 27: *Startup error for different perturbation size* Δ

Note that the probability density is not accurate for many iterations if the perturbation size is small compared to the size of the domain. This situation gets just worse for higher dimensions.

9. Preprocessing the point lightsources

As other global radiosity methods, this method is efficient for large area lightsources but loses its advantages if the lightsources are small [25]. This problem can be solved by a "*first-shot*" that shoots the power of the point lightsources onto other surfaces, then removes them from the scene [2]. Since the surfaces can also be non-diffuse, the irradiance received by the patches from each point lightsource should be stored (this requires l additional variables per patch, where l is the number of point lightsources). The secondary, non-diffuse emission to a direction is computed from these irradiances.

Formally, the unknown radiance L is decomposed into two terms:

$$L = L^{ep} + L^{np} \tag{56}$$

© Institute of Computer Graphics 1998

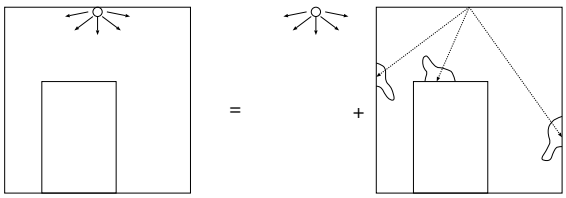


Figure 28: *First shot technique*

where L^{ep} is the emission of the small, point-like lightsources, L^{np} is the emission of the area lightsources and the reflected radiance. Substituting this into the rendering equation we have:

$$L^{ep} + L^{np} = L^e + \mathcal{T}(L^{ep} + L^{np}) \tag{57}$$

Expressing L^{np} we obtain:

$$L^{np} = (L^e - L^{ep} + \mathcal{T}L^{ep}) + \mathcal{T}L^{np}. \tag{58}$$

Introducing the new lightsource term

$$L^{e*} = L^e - L^{ep} + \mathcal{T}L^{ep} \tag{59}$$

which just replaces the point lightsources (L^{ep}) by their effect ($\mathcal{T}L^{ep}$), the equation for L^{np} is similar to the original rendering equation:

$$L^{np} = L^{e*} + \mathcal{T}L^{np}. \tag{60}$$

It means that first the direct illumination caused by the point lightsources must be computed, then they can be removed from the scene and added again at the end of the computation.

9.1. Diffuse shot

Getting rid of the point lightsources may reduce the variation of integrand, but medium sized lightsources still pose problems. One way of handling these is decomposing them into finite number of small lightsources and preprocess the scene in a way proposed by the previous section. However, due to the fact that the first-shot of l point lightsources requires l additional variables per patch, this approach becomes very memory demanding. Thus a different approach is needed, which decomposes the transport operator instead of subdividing the lightsources.

Let us express the BRDF of the surfaces as a sum of the diffuse and non-diffuse (specular) terms (note that the available BRDF representations do exactly this),

$$f_r(\omega', \vec{x}, \omega) = f_d(\vec{x}) + f_{nd}(\omega', \vec{x}, \omega)$$

and let us express the transport operator as the sum of diffuse and non-diffuse reflections:

$$\mathcal{T} = \mathcal{T}_d + \mathcal{T}_{nd},$$

$$\mathcal{T}_d L = \int_\Omega L(h(\vec{x}, -\omega'), \omega') \cdot \cos^* \theta' \cdot f_r(\vec{x})\ d\omega',$$

$$\mathcal{T}_{nd} L = \int_\Omega L(h(\vec{x}, -\omega'), \omega') \cdot \cos^* \theta' \cdot f_{nd}(\omega', \vec{x}, \omega)\ d\omega'.$$

The basic idea of the "diffuse shot" technique is that $\mathcal{T}_d L^e$ can be calculated in a preprocessing phase by known techniques, for example, by a gathering-type radiosity algorithm. The storage of the found $\mathcal{T}_d L^e$ requires just one variable per patch (this is why we handled the diffuse reflectance separately).

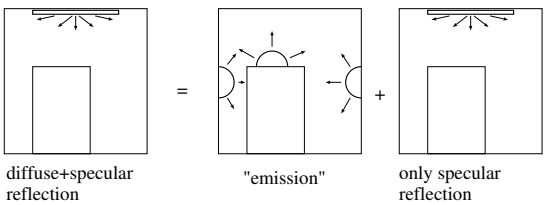


Figure 29: *First step of the diffuse shot technique*

Then, during the global walks, the first step should only be responsible for the non-diffuse reflection. The diffuse part is added to the result of this first step. Note that this method handles the first step in a special way, thus it requires the different bounces to be stored separately, as it is done by the combined and bi-directional methods.

In order to formally present the idea, let us denote the result of the diffuse shot by $\mathbf{L}_{\rm dif}$. The calculation of the *d-bounce irradiance* $\mathbf{J}_d$ for $d = 1, 2, \ldots$ is modified as follows:

$$\begin{aligned}
\mathbf{J}_0 &= \mathbf{A}(\omega'_D) \cdot \mathbf{L}^e(\omega'_D), \\
\mathbf{J}_1 &= 4\pi \cdot \mathbf{A}(\omega'_{D-d}) \cdot \left(\mathbf{F}_{nd}(\omega'_{D-d+1}, \omega'_{D-d}) \cdot \mathbf{J}_1 + \mathbf{L}_{\rm dif},\right) \\
\mathbf{J}_d &= 4\pi \cdot \mathbf{A}(\omega'_{D-d}) \cdot \mathbf{F}(\omega'_{D-d+1}, \omega'_{D-d}) \cdot \mathbf{J}_{d-1},
\end{aligned}$$

where $\mathbf{F}_{nd}$ is the non-diffuse reflectance function.

The method can also be explained as a restructuring of the Neumann-series expansion of the solution of the rendering equation in the following way:

$$L = L^e + \mathcal{T}L^e + \mathcal{T}^2 L^e + \mathcal{T}^3 L^e + \ldots =$$

$$(L^e + \mathcal{T}_d L^e) + \mathcal{T}_{nd} L^e + \mathcal{T}(\mathcal{T}_d L^e + \mathcal{T}_{nd} L^e) + \ldots \tag{61}$$

where $\mathcal{T}_d L^e$ is known after the preprocessing phase.

10. Simulation results

Figure 30 shows a scene as rendered after the first shot and after 500 walks of length 5. The scene contains specular,

© Institute of Computer Graphics 1998

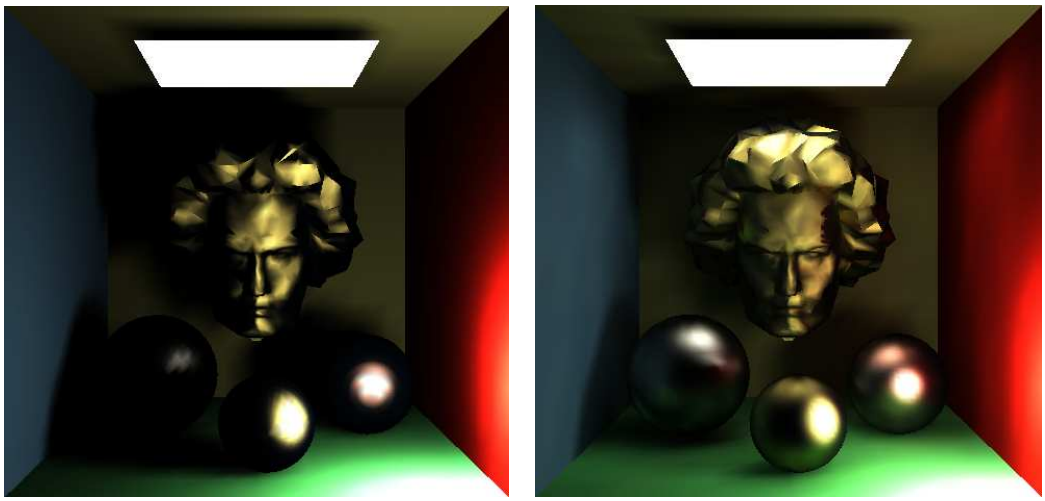


Figure 30: *A scene after the "first-shot"(left) and after 500 global walks (right)*

Figure 31: *A scene of a Beethoven and a teapot rendered by stochastic iteration after 500 iterations (left) and when fully converged (right)*

© Institute of Computer Graphics 1998

Figure 32: *A smooth mountain with a nearby "moon" and a flat lake*

Figure 33: *A rocky mountain with a nearby "moon" and a "wavy" lake*

© Institute of Computer Graphics 1998

metallic objects tessellated to 9605 patches, and is illuminated by both area (ceiling) and point (right-bottom corner) lightsources. A global radiance transfer took about 0.7 second on a Silicon Graphics O2 computer. Since the radiance information of a single patch is stored in 18 float variables (1 for the emission, 1 for the irradiance generated by the point lightsource, $D(D+1)/2 = 15$ for the irradiances and 1 for the accumulating radiance perceived from the eye), the extra memory used in addition to storing the scene is only 0.7 Mbytes.

A similar scene consisting of 9519 patches has been rendered by stochastic iteration (figure 31). The left image has been calculated by 500 steps which took 9 minutes.

Figure 32 shows a fractal terrain containing 14712 patches after 500 global walks which provide an accuracy within 2 percents. The illumination comes from both a sperical lightsource placed close to the mountain and from the homogenous sky-light. A global radiance transfer took approximately 1.1 seconds and the radiance information required 1 Mbytes.

In figure 33 the fractal surface has been tessellated further and waves are added to the water. This scene contains 59614 patches and has been rendered by the stochastic iteration (45 minutes computation time).

11. Conclusions

This paper presented a combined finite-element and random-walk algorithm to solve the rendering problem of complex scenes including also glossy surfaces. The basic idea of the method is to form bundles of parallel rays that can be traced efficiently, taking advantage of the z-buffer hardware. Unlike other random walk methods using importance sampling [9, 13, 40], this approach cannot emphasize the locally important directions, but handles a large number (1 million) parallel rays simultaneously instead, thus it is more efficient than those methods when the surfaces are not very specular.

The time complexity of the algorithm depends on the used global visibility algorithm. For example, the global painter's algorithm has $O(n \log n)$ complexity (n is the number of patches), which is superior to the $O(n^2)$ complexity of classical, non-hierarchical radiosity algorithms.

The memory requirement is comparable to that of the diffuse radiosity algorithms although the new algorithm is also capable to handle non-diffuse reflections or refractions. Since global ray-bundle walks are computed independently, the algorithm is very well suited for parallelization.

In order to incorporate importance sampling, the Metropolis and VEGAS methods have been considered, but only the Metropolis method was examined in details. For homogeneous scenes, Metropolis sampling could not provide significant noise reduction compared to quasi-Monte Carlo walks. This is due to the fact that the integrand of equation (7) is continuous and is of finite variation unlike the integrand of the original rendering equation, thus if its variation is modest then quasi-Monte quadrature is almost unbeatable. If the radiance distribution has high variation (difficult lighting conditions), then the Metropolis method becomes more and more superior. On the other hand, the Metropolis method is sensitive to its parameters such as the extent of perturbation. Future research should concentrate on the automatic and "optimal" determination of these control parameters.

The paper also presented an unbiased algorithm that was based on stochastic iteration. This algorithm seems to be significantly better than the finite-length approaches, in terms of both speed and storage space.

12. Acknowledgments

This work has been supported by the National Scientific Research Fund (OTKA), ref.No.: F 015884 and the Austrian-Hungarian Action Fund, ref.No.: 29p4, 32öu9 and 34öu28. Special thanks go to Werner Purgathofer for the valuable discussions and for making the resources of the Institute of Computer Graphics available for this research. Thanks also go to mathematicians Sarolta Inczédy, László Neumann, Attila Neumann and István Deák for drawing my attention to the mathematical and computer graphics techniques on which the presented methods are built, including, for example, the transillumination method, and for urging me to pursue this research.

References

1. C. Buckalew and D. Fussell. Illumination networks: Fast realistic rendering with general reflectance functions. *Computer Graphics (SIGGRAPH '89 Proceedings)*, 23(3):89–98, July 1989.

2. F. Castro, R. Martinez, and M. Sbert. Quasi Monte-Carlo and extended first-shot improvements to the multi-path method. In *Spring Conference on Computer Graphics '98*, pages 91–102, 1998.

3. P. H. Christensen, D. Lischinski, E. J. Stollnitz, and D. H. Salesin. Clustering for glossy global illumination. *ACM Transactions on Graphics*, 16(1):3–33, 1997.

4. P. H. Christensen, E. J. Stollnitz, D. H. Salesin, and T. D. DeRose. Global illumination of glossy environments using wavelets and importance. *ACM Transactions on Graphics*, 15(1):37–71, 1996.

5. R. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, pages 137–145, 1984.

6. M. de Berg. *Efficient Algorithms for Ray Shooting and Hidden Surface Removal*. PhD thesis, Rijksuniversiteit te Utrecht, The Nederlands, 1992.

7. I. Deák. *Random Number Generators and Simulation*. Akadémia Kiadó, Budapest, 1989.

© Institute of Computer Graphics 1998

8. F. Dévai. *Computational Geometry and Image Synthesis*. PhD thesis, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, Hungary, 1993.

9. P. Dutre, E. Lafortune, and Y. D. Willems. Monte Carlo light tracing with direct computation of pixel intensities. In *Compugraphics '93*, pages 128–137, Alvor, 1993.

10. D. S. Immel, M. F. Cohen, and D. P. Greenberg. A radiosity method for non-diffuse environments. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, pages 133–142, 1986.

11. J. T. Kajiya. The rendering equation. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, pages 143–150, 1986.

12. A. Keller. A quasi-Monte Carlo algorithm for the global illumination in the radiosity setting. In H. Niederreiter and P. Shiue, editors, *Monte-Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, pages 239–251. Springer, 1995.

13. E. Lafortune and Y. D. Willems. Bi-directional path-tracing. In *Compugraphics '93*, pages 145–153, Alvor, 1993.

14. E. Lafortune and Y. D. Willems. A 5D tree to reduce the variance of Monte Carlo ray tracing. In *Rendering Techniques '96*, pages 11–19, 1996.

15. G. P. Lepage. An adaptive multidimensional integration program. Technical Report CLNS-80/447, Cornell University, 1980.

16. R. Lewis. Making shaders more physically plausible. In *Rendering Techniques '93*, pages 47–62, 1993.

17. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

18. L. Neumann. Monte Carlo radiosity. *Computing*, 55:23–42, 1995.

19. M. E. Newell, R. G. Newell, and T. L. Sancha. A new approach to the shaded picture problem. In *Proceedings of the ACM National Conference*, pages 443–450, 1972.

20. H. Niederreiter. *Random number generation and quasi-Monte Carlo methods*. SIAM, Pennsilvania, 1992.

21. S. N. Pattanik and S. P. Mudur. Adjoint equations and random walks for illumination computation. *ACM Transactions on Graphics*, 14(1):77–102, 1995.

22. M. Pellegrini. Monte Carlo approximation of form factors with error bounded a priori. *Discrete and Computational Geometry*, 1997. (to appear).

23. Alfréd Rényi. *Wahrscheinlichkeitsrechnung*. VEB Deutscher Verlag der Wissenschaften, Berlin, 1962.

24. M. Sbert. *The Use of Global Directions to Compute Radiosity*. PhD thesis, Catalan Technical University, Barcelona, 1996.

25. M. Sbert, X. Pueyo, L. Neumann, and W. Purgathofer. Global multipath Monte Carlo algorithms for radiosity. *Visual Computer*, pages 47–61, 1996.

26. P. Shirley. Discrepancy as a quality measure for sampling distributions. In *Eurographics '91*, pages 183–194. Elsevier Science Publishers, 1991.

27. P. Shirley, C. Wang, and K. Zimmerman. Monte Carlo techniques for direct lighting calculations. *ACM Transactions on Graphics*, 15(1):1–36, 1996.

28. F. Sillion and C. Puech. A general two-pass method integrating specular and diffuse reflection. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, pages 335–344, 1989.

29. F. X. Sillion, J. R. Arvo, S. H. Westin, and D. P. Greenberg. A global illumination solution for general reflectance distributions. *Computer Graphics (SIGGRAPH '89 Proceedings)*, 25(4):187–198, 1991.

30. I. Sobol. *Die Monte-Carlo Methode*. Deutscher Verlag der Wissenschaften, 1991.

31. L. Szirmay-Kalos and T. Fóris. Sub-quadratic radiosity algorithms. In *Winter School of Computer Graphics '97*, pages 562–571, Plzen, Czech Republic, 1997.

32. L. Szirmay-Kalos, T. Fóris, L. Neumann, and B. Csébfalvi. An analysis to quasi-Monte Carlo integration applied to the transillumination radiosity method. *Computer Graphics Forum (Eurographics'97)*, 16(3):271–281, 1997.

33. L. Szirmay-Kalos, T. Fóris, and W. Purgathofer. Non-diffuse, random-walk radiosity algorithm with linear basis functions. *Machine Graphics and Vision*, 7(1):475–484, 1998.

34. L. Szirmay-Kalos, T. Fóris, and W. Purgathofer. Quasi-Monte Carlo global ray-bundle tracing with infinite number of rays. In *Winter School of Computer Graphics '98*, pages 386–393, Plzen, Czech Republic, 1998.

35. L. Szirmay-Kalos and G. Márton. On hardware implementation of scan-conversion algorithms. In *8th Symp. on Microcomputer Appl.*, Budapest, Hungary, 1994.

36. L. Szirmay-Kalos and W. Purgathofer. Global ray-bundle tracing with hardware acceleration. In *Rendering Techniques '98*, pages 47–62, 1998.

37. L. Szirmay-Kalos (editor). *Theory of Three Dimensional Computer Graphics*. Akadémia Kiadó, Budapest, 1995.

38. E. Veach and L. Guibas. Optimally combining sampling techniques for Monte Carlo rendering. In *Rendering Techniques '94*, pages 147–162, 1994.

39. E. Veach and L. Guibas. Bidirectional estimators for light transport. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 419–428, 1995.

40. E. Veach and L. Guibas. Metropolis light transport. *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 65–76, 1997.

41. J. R. Wallace, M. F. Cohen, and D. P. Greenberg. A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, pages 311–324, 1987.

42. Kevin Weiler and Peter Atherton. Hidden surface removal using polygon area sorting. In *Computer Graphics (SIGGRAPH '77 Proceedings)*, pages 214–222, 1977.

© Institute of Computer Graphics 1998