1 Distribution Ray Tracing: Theory and Practice

Peter Shirley
Changyaw Wang
Computer Science Department
Indiana University

Proceedings of the Third Eurographics Workshop on Rendering

1.1 Introduction

Distribution ray tracing uses Monte Carlo integration to solve the rendering equation. This technique was introduced by Cook et. al, and was notable because of its simplicity and its ability to simulate areal luminaires, camera lens effects, motion blur, and imperfect specular reflection[5]. Distribution ray tracing has been extended and modified by many researchers, most notably by Kajiya who added true indirect illumination[12]. Distribution ray tracing has also been used as the viewing component of radiosity systems (e.g [37, 3]). In this paper we examine some of the central issues of distribution ray tracing that have been overlooked in the literature but are still of importance.

Our interest in distribution ray tracing arises for three basic reasons. The first reason is that we believe any rendering system that has imperfect specular reflection (e.g. brushed steel) will use distribution ray tracing as a viewing method. Second, distribution ray tracing is a natural way to render outdoor scenes, where the effects of single and double reflection dominate color, and where the geometry is extremely complex. Third, it is possible to run ray tracing on large distributed memory multiprocessors[25].

Although the implementation of a simple distribution ray tracer is usually implied to be straightforward, there are several important implementational issues that have not been discussed in the literature, and the relationship between distribution ray tracing code and distribution ray tracing theory has never been completely stated. In this paper, we present some implementational and mathematical details that we have found to be important in the design of our software. In Section 1.2, we discuss the basics of Monte Carlo Integration and Quasi-Monte Carlo integration. In Section 1.3, we apply Monte Carlo integration to pixel filtering and use this as an example of how the mathematics influence the design of distribution ray tracing code. Section 1.4, we discuss the illumination calculations in a distribution ray tracer. This section argues that illumination calculation is not well understood and discusses some problems that need further research. Section 1.5 discusses the need for perception based display of the output of a distribution ray tracer. Section 1.6 discusses several implementational considerations we feel are non-obvious and have recently forced us to redesign our software. Finally, Section 1.7 summarizes our results and current plan of research.

1.2 Monte Carlo Integration and Quasi-Monte Carlo Integration

In Monte Carlo integration, random points with some distribution are used to find an approximate value for the integral. For an integral I of a function f over some space S, we can generate an estimate of I using some set of random points ξ_1 through ξ_N, where each ξ_i is distributed according to a probability density function p:

Although distribution ray tracing is usually phrased as an application of Equation 1.1, many researchers replace the ξ_i with more evenly distributed (quasi-random) samples (e.g. [4, 19]). This approach can be shown to be sound by analyzing decreasing error in terms of some discrepancy measure[44, 43, 19, 28] rather than in terms of variance. However, it is often convenient to develop a sampling strategy using variance analysis on random samples, and to instead use non-random, but equidistributed samples in an implementation. This approach is almost certainly correct, but its justification and implications have yet to be explained.

We often have integrals that take the form of a strictly positive weighted average of a function:

$$I = \int_{x \in S} w(x) f(x) d\mu_x$$

where w is a weighting function with unit volume. To solve this by Equation 1.1, the optimal choice for the probability function is $p(x) = Cw(x)f(x)$, but as is often pointed out, this choice requires us to already know the value of I. Instead, people often either choose uniform p, or set $p(x) = w(x)$[4, 24, 16].

In graphics we usually repeatedly perform an integral using many different f chosen from some set of functions F. To decide which p to use, we could try to minimize the average variance of our estimate across all $f \in F$. We cannot do this with such a vague definition of F, but we can approximate our situation in graphics by assuming that we are no more likely to have large values of f in a particular point in S than any other. This leads to the conclusion that the average value of $f^2(x)$ at any given x is some constant $< f^2 >$. The average variance of the estimator $I' = w(\xi)f(\xi)/p(\xi)$ is then just:

$$var_{ave}(I') = \int_{x \in S} \frac{w^2(x) < f^2 >}{p(x)} d\mu_x - I^2$$

Which can be shown by calculus of variations to be minimal when $p = w$. This implies that the intuitively appealing choice of $p = w$ has some theoretical justification as well.

1.3 Pixel Filtering

The color of a pixel $I(i, j)$ can be expressed as an integral:

$$I(i,j) = \int_{\mathbf{p} \in S} w(\mathbf{p}) L(\mathbf{p}) dA_{\mathbf{p}} \tag{1.2}$$

where $\mathbf{p}$ is a point on the viewport (or filmplane if a camera model is used), $L(\mathbf{p})$ is the radiance seen through the viewport at $\mathbf{p}$, and S is the non-zero region of the filter function w.

Like other parts of the full rendering equation, Equation 1.2 can be solved by Monte Carlo Integration. Rewriting with the assumption that the same origin-centered weighting function is used for every pixel yields the estimator :

$$I(i,j) \approx \frac{1}{N} \sum_{k=1}^{N} \frac{w(x_k, y_k) L(i + 0.5 + x_k, j + 0.5 + y_k)}{p(x_k, y_k)} \tag{1.3}$$

This assumes a coordinate system where a pixel (i, j) has unit area and is centered at $(i + 0.5, j + 0.5)$ as suggested by Heckbert[9].

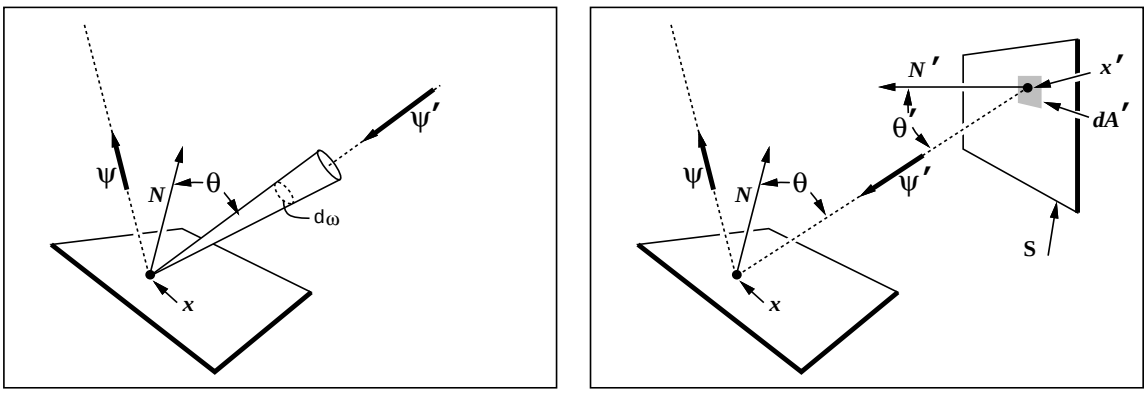

FIGURE 1.1. Definitions for the rendering equations.

L, and saying that there is an average $< L^2 >$ over a large number of pixels and images is not unreasonable because the distribution of intensities is fairly uniform if taken over many viewing frames. In other words, the bright spots will not favor particular places on the film.

Once a w is chosen for filtering, implementation is straightforward provided that w is strictly positive (as it must be if negative pixel colors are disallowed). Points can be chosen uniformly from $[0,1]^2$ and then a warping transformation can be applied to distribute the points according to w[32, 28, 16].

For several practical and theoretical reasons[27] we have chosen the width 2 weighting function that is non-zero on $(x,y) \in [-1,1]^2$:

$$w(x,y) = (1-|x|)(1-|y|) \tag{1.4}$$

We generate random points with density equal to w by applying a transformation to a uniform random pair $(r_1, r_2) \in [0,1]^2$. The transformed sample point is just $(t(r_1), t(r_2)$, where the transformation function t is:

$$t(u) = \begin{cases} -1+\sqrt{2u} & \text{if } u < 0.5 \\ 1-\sqrt{2(1-u)} & \text{if } u \geq 0.5 \end{cases}$$

1.4 The Two Forms of the Rendering Equation

In this section we describe how a distribution ray tracer uses Monte Carlo integration to solve the rendering equation. This amounts to designing a probability density function p. This is a solved, but tricky, problem for diffuse and ideal specular surfaces. However, it is not only an unsolved problem for imperfect specular surfaces, we show that the simple diffuse and specular strategy does not easily generalize to the imperfect specular case for certain BRDF.

The rendering equation can be written down in two basic ways, inspired by the two diagrams in Figure 1.1. It can be written down in terms of all directions visible to $\mathbf{x}$ (as in [10]):

$$L(\mathbf{x},\psi) = \int_{\text{incoming } \psi'} \rho(\mathbf{x},\psi,\psi') L(\mathbf{x},\psi') \cos\theta \, d\omega' \tag{1.5}$$

or it can written down as an integral over all surfaces (as in [12]):

Equations 1.5 and 1.6 suggest two different Monte Carlo solutions. If Equation 1.5 is used, then a scattered ray is sent from $\mathbf{x}$ in a random direction. If Equation 1.6 is used, a shadow ray is sent to each luminaire to evaluate g. Interestingly, a standard distribution ray tracer does both of these things: Equation 1.5 is used for perfect mirrors, and Equation 1.6 is used for direct lighting on diffuse surfaces.

This combination of solution methods makes the implementation of this part of a distribution ray tracer very confusing. We show that the implementation can be directly mapped to the mathematics, and discuss how this is intimately related to the BRDF ρ at $\mathbf{x}$.

When Equation 1.5 is used, we can view $\rho(\mathbf{x}, \psi, \psi') \cos\theta$ as a weighting function and sample according to it. Because there is some energy absorbed by a surface, this gives us the estimator:

$$L(\mathbf{x}, \psi) \approx R(\mathbf{x}, \psi) L(\mathbf{x}, \xi) \tag{1.7}$$

where ξ is a random direction with density proportional to $\rho(\mathbf{x}, \psi, \psi') \cos\theta$. The reflectivity term is simply:

$$R(\mathbf{x}, \psi) = \int_{\text{incoming } \psi'} \rho(\mathbf{x}, \psi, \psi') \cos\theta d\omega$$

For an ideal specular surface, the ξ will always be the ideal reflection direction. For a dielectric, ξ can be chosen randomly between reflected and transmitted directions[1], or it can be split into two integrals as is done in a Whitted-style ray tracer[42]. For a diffuse surface, ξ will be follow cosine distribution: $p(\psi') = \cos\theta/\pi$.

When Equation 1.6 is used, the sampling takes place over all surfaces in the environment. In practice, only the direct lighting is calculated, so the integration space becomes all luminaire surfaces. This can be split into one integral for each surface[4], or can be viewed as a single sampling space[16, 30]. To simplify this discussion, we will assume only one luminaire, so the sampling space is just a single surface. Looking at Equation 1.6, an ideal estimator for diffuse luminaires would result if we sampled according to the density:

$$p(x') = C g(\mathbf{x}, \mathbf{x}') \rho(\mathbf{x}, \psi, \psi') \cos\theta \frac{\cos\theta'}{\|\mathbf{x}' - \mathbf{x}\|^2}$$

where C is a normalization constant. In practice, this isn't practical because the geometry term g and the BRDF ρ can be very difficult to characterize. Instead, many researchers[5, 12] sample uniformly within the solid angle subtended by the luminaire, which yields:

$$p(x') = C' \frac{\cos\theta'}{\|\mathbf{x}' - \mathbf{x}\|^2} \tag{1.8}$$

Even this must only be approximated[1] for polygonal luminaires[38], but can be exactly applied for spherical luminaires[14, 38]. If Equation 1.8 is used to choose points on the luminaire, then radiance can be estimated to be:

$$L(\mathbf{x}, \psi) \approx g(\mathbf{x}, \mathbf{x}') \rho(\mathbf{x}, \psi, \psi') L(\mathbf{x}', \psi') \cos\theta \omega \tag{1.9}$$

where ω is the total solid angle subtended by the luminaire as seen by $\mathbf{x}$.

We call the use of Equation 1.5 an *implicit* direct lighting calculation because any scattered ray that hits a luminaire will account for light from that luminaire. The use of Equation 1.6 is an *explicit* direct lighting calculation because each luminaire is explicitly

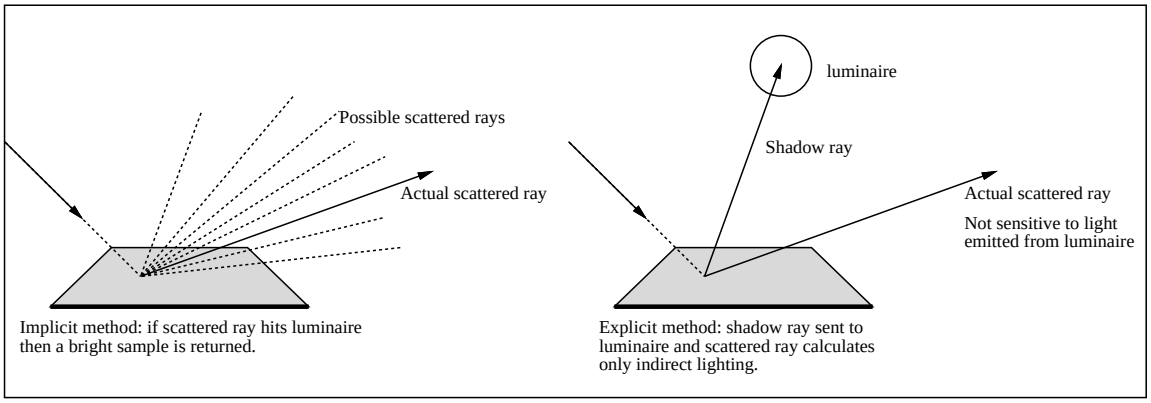

FIGURE 1.2. Implicit and explicit lighting calculation.

queried using shadow rays (see Figure 1.2). Which should be used, an implicit or explicit direct lighting calculation? Clearly, the implicit method must be used for perfect mirrors, because that method implicitly evaluates the delta function BRDF. For a diffuse surface, the explicit method is usually used for direct lighting, and the implicit method is used only for indirect lighting[12, 41, 16]. To decide which method to use we analyze the variance of each method in a simplified case: a diffuse luminaire S subtending solid angle ω with emitted radiance E, and no absorption[2]. We also assume that there are no other objects other than the one being illuminated, so the background radiance is zero (the luminaire is the only radiance source). The estimator for the implicit method becomes:

$$L(\mathbf{x}, \psi) \approx L(\mathbf{x}, \xi)$$

where ξ is distributed according to probability density function $\rho(\mathbf{x}, \psi, \psi') \cos\theta$. The term $L(\mathbf{x}, \xi)$ will be E if ξ goes toward the luminaire, and zero otherwise. We can write down the variance of this estimator, var_1 as an integral over the luminaire S by eliminating the portion of the integrated function:

$$var_1 = \int_{\psi' \text{ toward } S} \rho(\psi, \psi_{in}) \cos\theta E^2 d\omega - L^2(\mathbf{x}, \psi) \tag{1.10}$$

The variance, var_2, of the explicit estimator is:

$$var_2 = \int_{\psi' \text{ toward } S} \rho^2(\psi, \psi') \cos^2\theta E^2 \omega d\omega - L^2(\mathbf{x}, \psi)$$

If var_1 is larger, we should use explicit direct lighting. Otherwise we should use an implicit (scattered ray) calculation. We can evaluate the quantity $var_1 - var_2$ and test its sign to see which method is to be preferred. This quantity is:

$$var_1 - var_2 = E^2 \int_{\psi' \text{ toward } S} \rho(\psi, \psi') \cos\theta [1 - \omega\rho(\psi, \psi') \cos\theta] d\omega \tag{1.11}$$

For the case of a diffuse surface ($\rho = 1/\pi$), and a spherical luminaire directly above $\mathbf{x}$, the equality case for variance difference occurs when $\omega = 1.404\pi$ sr., so using the second method is best unless the luminaire is very large (such as a cloudy sky).

A more interesting case is imperfect mirrors. We use a Phong-like BRDF similar to the one used in [10]. The probability scattering function for a direction is:

$$p(\alpha, \beta) = \frac{N+2}{8\pi} \cos^N \frac{\alpha}{2}$$

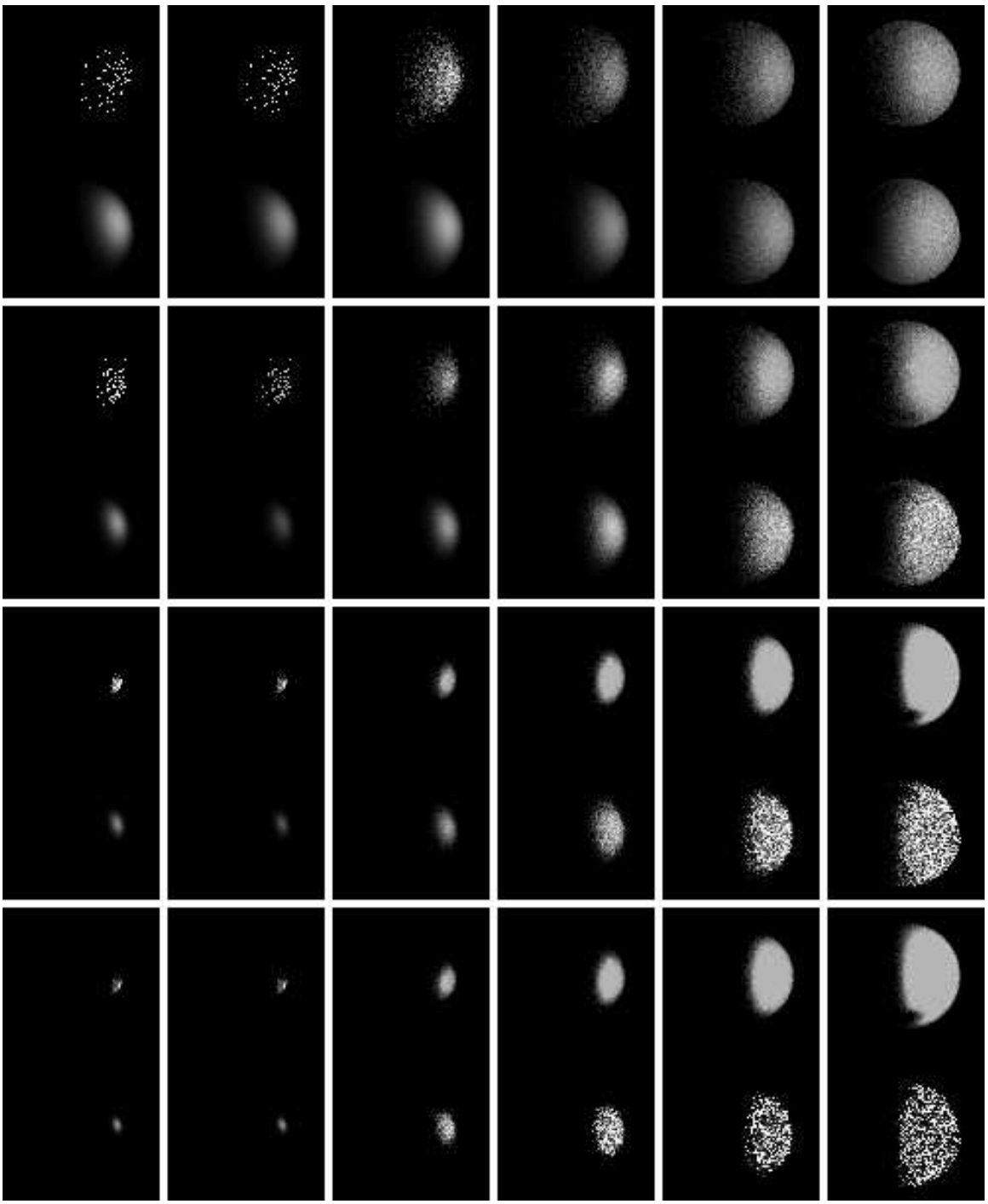

FIGURE 1.3. Phong spheres with luminaire to the right. Phong exponent is constant in each row. To to bottom: N = 10, 30, 90, 270. Luminaires increase in solid angle from left to right, with $\omega = 0.0083\pi, 0.0124\pi, 0.0496\pi, 0.106\pi, 0.288\pi, 2\pi$ sr. In each frame, scattered rays are sent on the top, and explicit direct lighting is calculated on the bottom.

where α is the angle relative to the ideal reflection direction, β is an angle around the reflection direction, and N is a Phong-like exponent. As N becomes large, the behavior becomes that of a mirror. The energy conservation properties of this function become increasingly unphysical as N gets smaller, but it is approximately correct and is easy to implement. A scattered direction can be generated with the warping function

$$(\alpha, \beta) = (2 \arccos[(1 - r_1)^{\frac{1}{N+2}}], 2\pi r_2)$$

where r_1 and r_2 are random numbers on $[0, 1]$. Applying Equation 1.11 to the Phong function, yields an expression with the same sign (positive constants eliminated) as:

$$4(N+1)(1 - \cos^{N+2} \frac{\alpha}{2}) - (N+2)^2(1 - \cos^2 \frac{\alpha}{2})(1 - \cos^{2N+2} \frac{\alpha}{2}) \qquad (1.12)$$

When this expression is zero, it is a borderline decision whether to calculate using the direct lighting implicitly or explicitly. We have used Methematica's root finding feature for several values of N on Equation 1.12 to find what solid angle ω (in this case $\omega = 2\pi[1 - \cos \alpha]$) the luminaire S should cover for a borderline decision. Our results are that for $N = 10, 30, 90, 270$, we get $\omega = 0.0124\pi, 0.0496\pi, 0.106\pi, 0.288\pi$ respectively. In Figure 1.3, we test the expression in Equation 1.12 by testing both the implicit and explicit method for $N = 10, 30, 90, 270$. If our analysis is correct, we expect near equal accuracy for the fifth column from the left in the top row, and one step to the left for each successive row, thus ending with equality in the second from the left in the bottom row. This agrees fairly well with the actual pictures.

The unfortunate implication of this test case is that deciding between the implicit and explicit method is *not* a local problem. The decision cannot be made soley on material properties of an object. In fact, the correct method may vary for different points on a particular object. This issue clearly needs further investigation, and perhaps more complex probability density functions are the ultimate solution.

1.5 Representation of Output

Most rendering programs use a simple mapping to take floating-point radiance values to limited precision (e.g. 8-bit per channel) pixel values. A common technique is to set an arbitrary radiance value to be white (e.g. 1.0) and scale other radiance values in kind. If this is done, then the luminaires in the picture will usually have radiance values far above the white point. These high radiance values also map to white. This brings up a problem on luminaire edges: the samples on the pixel that are within the luminaire boundary have high radiance values (e.g over 100), and the samples outside the luminaire boundaries have a small radiance value (e.g. 0.1). When we average these samples and then map them to pixel color, even one luminaire sample will send the pixel to white. This produces jagged luminaire boundaries. If we map to color and then average, the luminaires will be nicely antialiased, but other pixels will have incorrect estimates (e.g. a white pixel with half the samples at 0.5 and half at 1.5 will map to 0.75 which is too dark).

This problem stems from the finite dynamic range of computer graphics display medium. This problem is very serious in any system that has visible luminaires. One way around this problem is to use a floating point output format such as that used in Ward's *Radiance* program[40], and to convert to a displayed image using a perceptually based spatially varying map such as the one used by Tumblin and Rushmeier[34]. This will map the

the problem of images with great ranges in radiance. Further modifications can include corrections for dark adaptation and chromatic adaptation of the viewer. For example, in a virtual reality system, a user entering a house lit by incandescent light from outside at night should first see a very bright yellowish scene, and should gradually adapt to see a neutral normally illuminated room. Especially dark scenes should lose their chromatic qualities. For especially bright spots in a scene, effects such as those used used by Nakamae et al.[21] have proven effective.

1.6 Implementational Issues

Although usually talked about is if trivial, writing a full-blown ray tracer is difficult, and object-oriented design techniques have proven useful[13, 8, 29]. We outline some of our more recent design decisions in this section.

In implementing a camera model[4], we have assumed a thin-lens camera model based on the fundamental rules of a thin-lens. We have not modeled the $\cos^{-4}\theta$ falloff in intensity at the film-plane, because real cameras include corrections for this effect[20]. We have also allowed moving a camera with changing field-of-view. When doing this, it is important to generate the rays in the original object space, rather than transforming the objects into a canonical viewing space, because the latter would force the objects to be transformed for every ray.

Both [13] and [29] suggest that objects have a simple hit routine and a query function to later determine material properties for the nearest object. We have found that this is an unnatural approach if procedural objects are used, because the objects will have to be regenerated to get orientation and material information. Instead we can have the hit routine calculate all information. This makes the implementation procedural objects cleaner at the expense of a simple hit routine. We decided against caching any information because we want to port our implementation to a distributed system, where it helps to have a read-only scene database[3]. Currently, we have three separate hit routines, one for viewing rays, another for shadow rays, and a third for power (radiosity) rays. This is because the power rays do write to the database, so splitting of the routines was essential.

We organize our objects by maintaining a linked list, where each unbounded or infinite primitive (e.g. infinite plane, field of multiple instance grass) occupy one node in the list, and all bounded primitives reside in a BSP tree[11, 33]. The BSP tree also occupies one node in the list. Our code has been designed in the spirit of [13], where every object or object collection can be treated with the same interface, so the management of a list containing both simple primitives and a BSP tree is straightforward to manage. To support multiple instancing, we allow subdivision structures to be nested arbitrarily as in [13].

We have adopted a fairly simple adaptive sampling strategy in the spirit of [17, 24]. More effective strategies have been demonstrated for textures[36], but we have not pursued these because our system is targeted at environments where most complexity comes from geometry rather than texture. Schlick has implemented an effective antialiasing scheme for distribution ray tracing[26], but his initial sampling phase could miss detail in a highly complex environment, so we have not implemented his method.

For a material model, we allow five types of surfaces[29], luminaires, lambertian, conductors, dielectrics, and polished. The last three allow a Phong style exponent to control

[3] Andrew Glassner has suggested that this caching could be done while maintaining the read-only condition

specularity. The chief difference between our model and more traditional ones is the inclusion of Fresnel Equations for the reflectivity of conductors, dielectrics, and polished materials. We have not implemented a more physically based model for BRDF functions, such as the one recently presented in [7], because we have not found an analytical attack on the mathematics of scattered ray distribution. We also allow solid textures[23] to modify material parameters such as the amount of polish on a surface. We have tried to allow effects such as those in [35] while still disallowing physically invalid materials. This is the part of the code where we are the most unsure of proper design.

We use a spectral color model in our code with twenty one evenly spaced spectral samples from 400nm to 700nm. We originally implemented the four unevenly spaced spectral samples suggested by Meyer[18], but we found that metallic colors generated using the Fresnel Equations lacked richness with only four samples. In all other ways, the four samples were satisfactory. We have had difficulty finding spectral reflectance data for many materials. Two of our most valuable sources have been [22] for conductor parameters, and [2] for the reflectance data for common artist pigments. We still support the input of RGB data for texture maps and diffuse reflectances, and use the Glassner's method[6] to choose a plausible spectrum associated with the RGB triple. The most important consideration here is the selection of basis functions that disallow negative spectral values at any wavelength. This can be accomplished by using the RGB primaries associated with the RGB input as basis functions.

For sample point selection, we choose N sampling points from $[0,1]^N$ in the spirit of [19], and then use appropriate warping transforms for our sample points to be distributed properly (e.g. points on the disk of the camera lens, reflected ray directions). Whether better distributions could be obtained by sampling directly (without warping) in the integration domain is an open question.

One of the most difficult sections of the code deals with the interaction of viewing rays and objects. As mentioned in Section 1.4, a specular surface may calculate direct lighting implicitly by sending a scattered ray. A diffuse surface will also send a scattered ray, but only to calculate the indirect lighting. This means that different scattered rays may be evaluating different integrals (one the light from all surfaces, the other the *reflected* light from all surfaces). This bookkeeping is handled by associating a parameter with each ray specifying which integral is being solved. A viewing ray also carries along two sampling pairs to choose shadow rays and reflection rays in a stratified manner. If one of these pairs are used (e.g. a shadow ray is generated by using the pair to choose a point on a luminaire), then the reflected ray will carry a newly generated random pair. If it is not used (e.g. a perfect mirror is hit by the ray), then the old pair is carried by the reflected ray. This will prevent reflections of shadows being noisier than directly viewed shadows.

For scenes with many luminaires, it is important to optimize the generation and use of shadow rays. Instead of pruning methods, which have been shown to work on scenes with many luminaires[39, 15], we use probabilistic methods. We send only one shadow ray to all luminaires[30]. For extremely complex scenes with thousands of luminaires, we construct a spatial subdivision structure with each spatial cell containing a list of important luminaires[31].

1.7 Conclusion

In this paper we have discussed some theoretical and practical issues involving distribution ray tracing. The most important result in this paper was dealt with in Section 1.4: choosing

on BRDF characteristics; the relation of an illuminated point and the luminaire must also be considered. This eliminates any possibility of a simple low-noise implementation of ray-object interaction in a distribution ray tracer, unless much better sampling methods of complex probability density functions are developed.

We believe that many of the implementational problems we have encountered over the years have also been encountered by other researchers. Unfortunately, the literature communicates very little of this information. As evidence of this, one should note that SIGGRAPH course notes are often referenced more than official journals. We think that researchers should publish more implementational details of their systems, because the translation of mathematical and physical principals is often very interesting science itself.

In the future of distribution ray tracing research, we need to develop better adaptive sampling techniques, support for procedural objects such as displacement maps, procedural material properties, and examine different ways to incorporate radiosity calculations into subenvironments of our scene.

1.8 References

[1] James Arvo and David Kirk. Particle transport and image synthesis. *Computer Graphics*, 24(3):63–66, August 1990. ACM Siggraph '90 Conference Proceedings.

[2] Norman F. Barnes. Color characteristics of artists' pigments. *Journal of the Optical Society of America*, May 1939.

[3] Shenchang Eric Chen, Holly Rushmeier, Gavin Miller, and Douglass Turner. A progressive multi-pass method for global illumination. *Computer Graphics*, 25(4):165–174, July 1991. ACM Siggraph '91 Conference Proceedings.

[4] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, January 1986.

[5] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics*, 18(4):165–174, July 1984. ACM Siggraph '84 Conference Proceedings.

[6] Andrew S. Glassner. How to derive a spectrum from an rgb triplet. *IEEE Computer Graphics and Applications*, 9(7):95–99, 1989.

[7] Xiao D. He, Kenneth E. Torrence, Francois X. Sillion, and Donald P. Greenberg. A comprehensive physical model for light reflection. *Computer Graphics*, 25(4):175–186, July 1991. ACM Siggraph '91 Conference Proceedings.

[8] Paul S. Heckbert. Writing a ray tracer. In Andrew S. Glassner, editor, *An Introduction to Ray Tracing*. Academic Press, San Diego, CA, 1989.

[9] Paul S. Heckbert. What are the coordinates of a pixel? In Andrew Glassner, editor, *Graphics Gems*. Academic Press, New York, NY, 1990.

[10] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A radiosity method for non-diffuse environments. *Computer Graphics*, 20(4):133–142, August 1986. ACM Siggraph '86 Conference Proceedings.

[11] Frederik W. Jansen. Data structures for ray tracing. In L. R. A. Kessener, F. J. Peters, and M. L. P. van Lierop, editors, *Data Sturctures for Raster Graphics*, pages 57–373. Springer-Verlag, Netherlands, 1986.

[12] James T. Kajiya. The rendering equation. *Computer Graphics*, 20(4):143–150, August 1986. ACM Siggraph '86 Conference Proceedings.

[13] David Kirk and James Arvo. The ray tracing kernel. In *Proceedings of Ausgraph*, pages 75–82, July 1988.

[14] David Kirk and James Arvo. Unbiased variance reduction for global illumination. In *Proceedings of the Second Eurographics Workshop on Rendering*, 1991.

[15] A. Kok and F. Jansen. Source selection for the direct lighting calculation in global illumination. In *Proceedings of the Second Eurographics Workshop on Rendering*, 1991.

[16] Brigitta Lange. The simulation of radiant light transfer with stochastic ray-tracing. In *Proceedings of the Second Eurographics Workshop on Rendering*, 1991.

[17] Mark E. Lee, Richard A. Redner, and Samuel P. Uselton. Statistically optimized sampling for distributed ray tracing. *Computer Graphics*, 19(3):61–68, July 1985. ACM Siggraph '85 Conference Proceedings.

[18] Gary W. Meyer. Wavelength selection for synthetic image generation. *Computer Vision, Graphics, and Image Processing*, 41:57–79, 1988.

[19] Don P. Mitchell. Spectrally optimal sampling for distribution ray tracing. *Computer Graphics*, 25(4), July 1991. To appear in ACM Siggraph '88 Conference Proceedings.

[20] Earl N. Mitchell. *Photographic Science.* John Wiley and Sons, New York, N.Y., 1984.

[21] Eihachiro Nakamae, Kazufumi Kaneda, Takashi Okamoto, and Tomoyuki Nishita. A lighting model aiming at drive simulators. *Computer Graphics*, 24(3):395–404, August 1990. ACM Siggraph '90 Conference Proceedings.

[22] Edward D. Palik. *Handbook of Optical Constants of Solids.* Academic Press, New York, N.Y., 1985.

[23] Ken Perlin and Eric M. Hoffert. Hypertexture. *Computer Graphics*, 23(3):253–262, July 1989. ACM Siggraph '89 Conference Proceedings.

[24] Werner Purgathofer. A statistical method for adaptive stochastic sampling. *Computers & Graphics*, 11(2):157–162, 1987.

[25] John Salmon and Jeff Goldsmith. A hypercube ray-tracer. In *The 3rd Conference on Hypercube Concurrent Computers and Applications Vol. II*, pages 1194–1206, 1988. ACM Press.

[26] Christophe Schlick. An adaptive sampling technique for multidimensional integration by ray-tracing. In *Proceedings of the Second Eurographics Workshop on Rendering*, 1991.

[28] Peter Shirley. Discrepancy as a quality measure for sampling distributions. In *Eurographics '91*, pages 183–193, September 1991.

[29] Peter Shirley, Kelvin Sung, and William Brown. A ray tracing framework for global illumination. In *Graphics Interface '91*, pages 117–128, June 1991.

[30] Peter Shirley and Changyaw Wang. Direct lighting by monte carlo integration. In *Proceedings of the Second Eurographics Workshop on Rendering*, 1991.

[31] Peter Shirley and Changyaw Wang. Luminaire sampling in distribution ray tracing. Technical Report 340, Department of Computer Science,Indiana University, January 1992.

[32] Y. A. Shreider. *The Monte Carlo Method.* Pergamon Press, New York, N.Y., 1966.

[33] Kelvin Sung. Ray tracing with the bsp tree. In David Kirk, editor, *Graphics Gems 3.* Academic Press, New York, NY, 1992. (to appear).

[34] Jack Tumblin and Holly Rushmeier. Tone reproduction for realistic computer generated images. Technical Report GIT-GVU-91-13, Graphics, Visualization, and Usuability Center, Georgia Institute of Technology, 1991.

[35] Steve Upstill. *The Renderman Companion.* Addison-Wesley, Reading, MA, 1990.

[36] Theo van Walsum, Peter R. van Nieuwenhuizen, and Frederik Jansen. Refinement criteria for adaptive stochastic ray tracing of textures. In *Eurographics '91*, pages 155–166, September 1991.

[37] John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. A two-pass solution to the rendering equation: a synthesis of ray tracing and radiosity methods. *Computer Graphics*, 21(4):311–320, July 1987. ACM Siggraph '87 Conference Proceedings.

[38] Changyaw Wang. Physically correct direct lighting for distribution ray tracing. In David Kirk, editor, *Graphics Gems 3.* Academic Press, New York, NY, 1992. (to appear).

[39] Greg Ward. Adaptive shadow testing for ray tracing. In *Proceedings of the Second Eurographics Workshop on Rendering*, 1991.

[40] Greg Ward. Real pixels. In James Arvo, editor, *Graphics Gems 2.* Academic Press, New York, NY, 1991.

[41] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. *Computer Graphics*, 22(4):85–92, August 1988. ACM Siggraph '88 Conference Proceedings.

[42] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[43] H. Wozniakowski. Average case complexity of multivariate integration. *Bulliten (New Series) of the American Mathematical Society*, 24(1):185–193, January 1991.

[44] S. K. Zeremba. The mathematical basis of monte carlo and quasi-monte carlo meth-