Parallel hierarchical global illumination

Quinn O. Snell

Major Professor: John L. Gustafson
Iowa State University

Solving the global illumination problem is equivalent to determining the intensity of every wavelength of light in all directions at every point in a given scene. The complexity of the problem has led researchers to use approximation methods for solving the problem on serial computers. Rather than using an approximation method, such as backward ray tracing or radiosity, we have chosen to solve the Rendering Equation by direct simulation of light transport from the light sources. This paper presents an algorithm that solves the Rendering Equation to any desired accuracy, and can be run in parallel on distributed memory or shared memory computer systems with excellent scaling properties. It appears superior in both speed and physical correctness to recent published methods involving bidirectional ray tracing or hybrid treatments of diffuse and specular surfaces. Like "progressive radiosity" methods, it dynamically refines the geometry decomposition where required, but does so without the excessive storage requirements for "ray histories." The algorithm, called Photon, produces a scene which converges to the global illumination solution. This amounts to a huge task for a 1997-vintage serial computer, but using the power of a parallel supercomputer significantly reduces the time required to generate a solution. Currently, Photon can be run on most parallel environments from a shared memory multiprocessor to a parallel supercomputer, as well as on clusters of heterogeneous workstations.

Abstract

Solving the global illumination problem is equivalent to determining the intensity of every wavelength of light in all directions at every point in a given scene. The complexity of the problem has led researchers to use approximation methods for solving the problem on serial computers. Rather than using an approximation method, such as backward ray tracing or radiosity, we have chosen to solve the Rendering Equation by direct simulation of light transport from the light sources. This paper presents an algorithm that solves the Rendering Equation to any desired accuracy, and can be run in parallel on distributed memory or shared memory computer systems with excellent scaling properties. It appears superior in both speed and physical correctness to recent published methods involving bidirectional ray tracing or hybrid treatments of diffuse and specular surfaces. Like "progressive radiosity" methods, it dynamically refines the geometry decomposition where required, but does so without the excessive storage requirements for "ray histories." The algorithm, called Photon, produces a scene which converges to the global illumination solution. This amounts to a huge task for a 1997-vintage serial computer, but using the power of a parallel supercomputer significantly reduces the time required to generate a solution. Currently, Photon can be run on most parallel environments from a shared memory multiprocessor to a parallel supercomputer, as well as on clusters of heterogeneous workstations.

Parallel hierarchical global illumination

by

Quinn O. Snell

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Major Professor: John L. Gustafson

Iowa State University

Ames, Iowa

1997

Graduate College
Iowa State University

This is to certify that the Doctoral dissertation of

Quinn O. Snell

has met the dissertation requirements of Iowa State University

Committee Member

Committee Member

Committee Member

Committee Member

Major Professor

For the Major Program

For the Graduate College

DEDICATION

For my sweetheart. Kristy, it's finished.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I am grateful to all who helped and supported me towards the completion of the Ph.D. degree. Many of those people cannot be named, but special thanks must go to the following individuals:

John Gustafson, thank you for trusting in me and for your unselfish attitude in all our work together. I will always be thankful for the chance to work with you. You have made an indelible impact on my life.

Nan Ripley, thanks to you I made it through all the red tape and trips that came along with my research. You have been a great friend.

Kristy Snell, my wife and family are most important in my life and without you, I would have neither. You have held the family together during those long weeks at the end of each semester and especially during the last stages of writing. I will always love you.

Many others are too numerous to mention. A Ph.D. is not the work of just one individual. I have received help from many people through discussions and simple kindness. Thank you to all.

This work was performed at Ames Laboratory under Contract No. W-7405-Eng-82 with the U.S. Department of Energy. The United States government has assigned the DOE Report number IS-T 1816 to this thesis.

1 GLOBAL ILLUMINATION

The field of computer graphics has changed dramatically over the last two decades. One of the major goals has remained constant: Display a scene on the computer that is *photorealistic*. In other words, display a computer-generated scene that is indistinguishable from a photograph of the actual scene. Accomplishing this task amounts to correctly simulating all lighting effects within the scene.

The correct simulation of lighting effects involves determining how much light is received at each point and what happens to the light after it hits a given point. The reflection of light is governed by well-known laws such as Snell's law and the Fresnel equations. Determination of the amount of light at a given point is performed by an *illumination model*. Illumination models come in two varieties: *global* and *local*. In a local illumination model, illumination on a surface depends only on the surface properties and the light sources. Local illumination models do not account for illumination due to reflection off surfaces in the scene. Global illumination models consider all objects as sources of illumination.

The solution to the global illumination problem has many applications, architectural rendering and virtual reality being two of the more popular. In general terms, the global illumination problem is the determination of the color and intensity of light given off from every point in an environment in every direction. This contrasts with ray tracing, which attempts to estimate the light seen from one viewpoint. Global illumination is key to virtual reality efforts since correct views can be displayed quickly as the viewpoint moves.

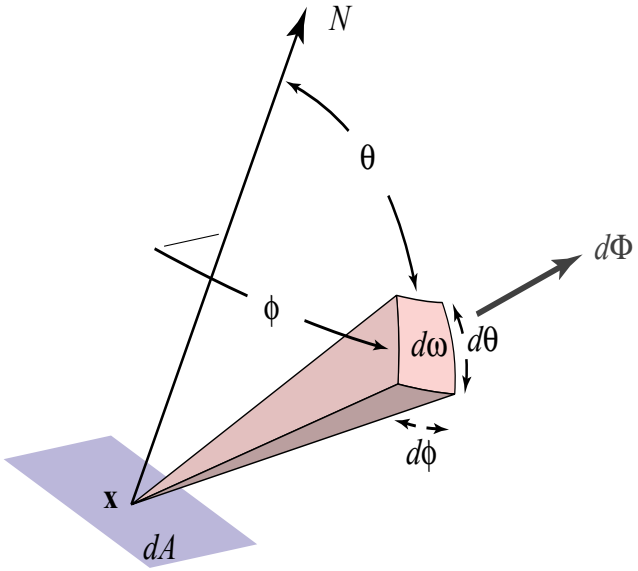


Figure 1.1 Definition of Radiance

The physical quantity desired for global illumination is the radiance, L, at point $\mathbf{x}$, in direction ψ (see Figure 1.1), described in the following equation [42, 45]:

$$L(\mathbf{x}, \psi) = \frac{dI}{dA\ cos\theta} \tag{1.1}$$

where ψ is a direction described by a vector or a (θ, ϕ) pair in spherical coordinates. The quantity I is the *radiant intensity* computed as:

$$I = \frac{d\Phi}{d\omega} \tag{1.2}$$

where ω is the solid angle originating at the point and Φ is the radiant flux.

Equation 1.1 shows that radiance is a function of position, $\mathbf{x}$, and viewing angle, (θ, ϕ). Therefore, any algorithm proposed to solve the global illumination problem must present a solution that is likewise dependent on position and viewing angle.

The Rendering Equation

The mathematics involved in a global illumination model have been summarized in a high-level continuous equation known as the Rendering Equation. The Rendering Equation was proposed by two authors in 1986: Kajiya [29] and Immel [27]. The representations are similar in that both are *Fredholm Equations of the Second Kind*. The differences will be discussed below.

The Rendering Equation as expressed by Kajiya is:

$$I(x,x') = g(x,x')\left[\epsilon(x,x') + \int_S \rho(x,x',x'')I(x',x'')dx''\right] \tag{1.3}$$

where:

$I(x,x')$	is the intensity of light passing from point x' to x
$g(x,x')$	is a "geometry term" explained below
$\epsilon(x,x')$	is the intensity of light emitted from x' to x
$\rho(x,x',x'')$	is the intensity of light scattered from x'' to x through x'
S	is the set of all surfaces in the scene

The geometry describing Kajiya's Rendering Equation is shown in Figure 1.2. The three-point transport from x'' through x' to x accounts for the directional reflectivity

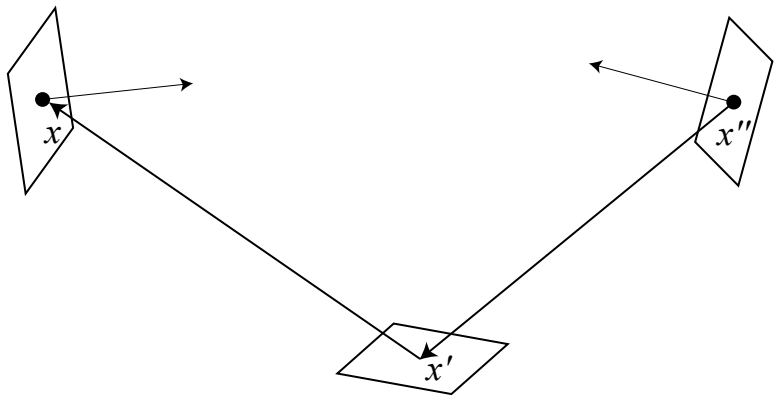


Figure 1.2 Geometry for Kajiya's Rendering Equation

of specular surfaces. The function $g(x, x')$ is a determination of the visibility of x from x', and takes a value of 1 if there is an unoccluded path between x from x', or 0 otherwise. It can also be used to simulate fog effects by assigning a percentage based on fog distribution. Note that, in this form, the equation integrates over all surfaces S.

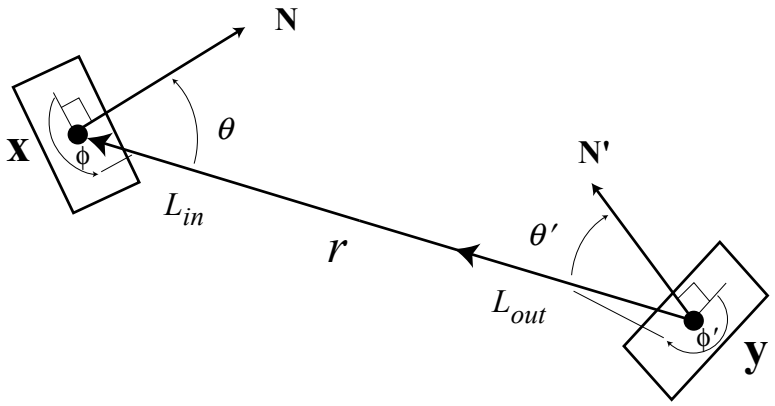


Figure 1.3 Geometry for Immel's Rendering Equation

Immel *et al.* formulate this equation in terms of *radiance* and thus integrate over all incoming angles (see Figure 1.3).

$$L_{out}(\mathbf{x}, \psi) = L_e(\mathbf{x}, \psi) + \int_{\psi'} \rho(\mathbf{x}, \psi, \psi') L_{in}(\mathbf{x}, \psi') \cos\theta' d\omega' \tag{1.4}$$

where:

$\mathbf{x}$	is a vector representing a point in the scene
ψ	is a direction represented as a (θ, ϕ) pair
$L_{out}(\mathbf{x}, \psi)$	is the radiance leaving point $\mathbf{x}$ in direction ψ
$L_{in}(\mathbf{x}, \psi')$	is the incoming radiance at point $\mathbf{x}$ in direction ψ'
L_e	is the emitted radiance
ρ	is the bidirectional reflectance distribution function (BRDF).

This formulation does not include provisions for occlusion at this point. It is accounted for during discretization using the function $HID(i, j, d)$ which evaluates to 1 if

patch i is visible from patch j along direction d, 0 otherwise. This is equivalent to the geometry term $g(i,j)$ used by Kajiya. The BRDF is simply the radiance of the surface seen from point $\mathbf{x}$ in direction ψ divided by the incident power per unit area. Determination of the BRDF for a given surface is a difficult task. It has become a related field of research and is being examined by many [54, 24, 11]. The graphics program at Cornell University under the direction of Donald Greenberg and Ken Torrance has developed methods for measuring the BRDF for a given surface [14].

Dissertation Structure

This dissertation discusses the approaches to global illumination and in particular the search for a global illumination algorithm that converges to a solution of the Rendering Equation and is amenable to parallelization. The various algorithms are presented in chapter 2 along with an analysis of their respective prospects for parallelism.

Chapter 3 presents a general class of algorithms which use Monte Carlo simulation. Monte Carlo integration is introduced and its application to global illumination is discussed. The chapter also analyzes current approaches to global illumination that use Monte Carlo techniques.

A new algorithm based on Monte Carlo simulation of light transport, called Photon, is presented in chapter 4. Also included is a new approach to histogramming the statistical results of the Monte Carlo simulation. The chapter ends with a discussion of whether the method converges to a solution of the Rendering Equation.

The parallelization of Photon is explained in chapter 5. Sources of parallelism are examined and issues such as data decomposition, load balancing and performance are discussed. Chapter 6 presents conclusions and examines areas of future research, and a paper presenting these results that was published in the *Proceedings of the Sixth International Conference on High Performance Distributed Computing* is in the Appendix.

2 GLOBAL ILLUMINATION ALGORITHMS

With the background from chapter 1, we can now examine the various global illumination algorithms. The algorithms can be grouped together in two basic classes: those that solve for only a single viewpoint, and those that attempt the full global illumination solution and solve for every possible viewpoint. These algorithms either directly solve an equation which yields the intensity of light for every point in the scene or attempt to compute the intensity heuristically. The computation may use a deterministic or a Monte Carlo algorithm. The remainder of this chapter will consider *deterministic solution* methods. Algorithms that use Monte Carlo methods will be addressed in the next chapter.

Ray Tracing

Ray tracing from the viewpoint, as introduced by Whitted [55], was partly introduced to reduce the heavy computational requirements of forward ray tracing. The idea was to compute only what was being viewed. These traditional ray tracing methods are backward in that they start from the viewer and estimate the radiance by summing the effects of light sources on the point of closest intersection. Ray tracing calculates the radiance at a point by sending a ray from the viewpoint through a hole in the viewplane corresponding to a pixel and determining the point $\mathbf{p}$ of closest intersection (see Figure 2.1). If $\mathbf{p}$ lies on a non-specular surface, the radiance is calculated based on an extension to the Phong [37] model. The intensity I for point $\mathbf{p}$ with normal $\mathbf{N}$ in the

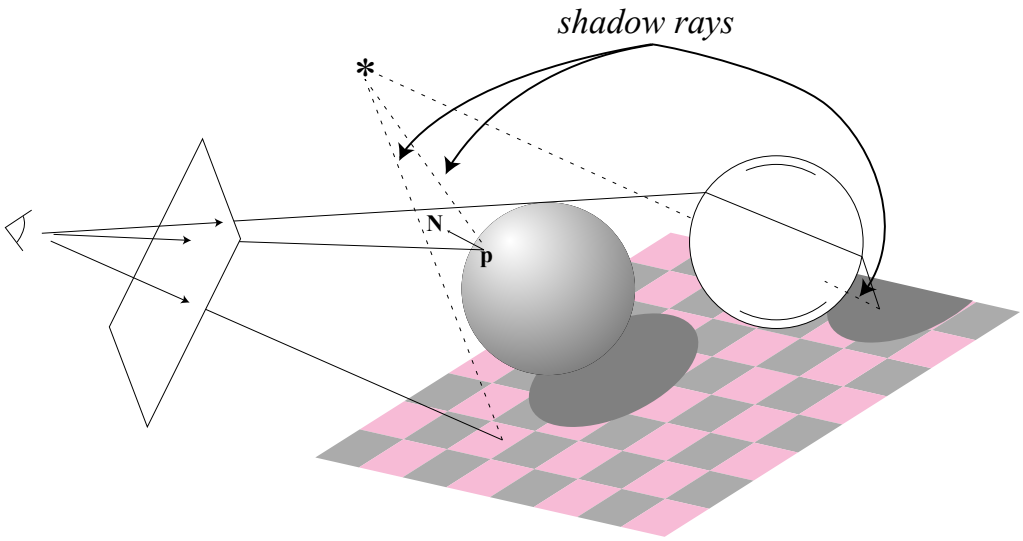


Figure 2.1 Ray Tracing Basics

Whitted model is:

$$I = I_a + k_d \sum_j (\mathbf{N} \cdot \mathbf{L}_j) I_j + k_s S + k_t T \tag{2.1}$$

where:

I_a	is the intensity of light due to ambient reflection
k_d	is the diffuse reflection constant
k_s	is the specular reflection constant
k_t	is the light transmission coefficient
$\mathbf{L}_j$	is the unit vector in the direction of light source j
S and T	are the intensity of specularly reflected and transmitted light.

The vector $\mathbf{L}_j$ is often termed a *shadow ray* as it also serves as a determination of whether light source j is visible from point $\mathbf{p}$. If a given light source is not visible, its effects do not contribute to the sum. The computational complexity of ray tracing increases with the number of light emitters whose effects must be added. Ray tracing is not physically correct because radiance depends on the light input from **all** surfaces,

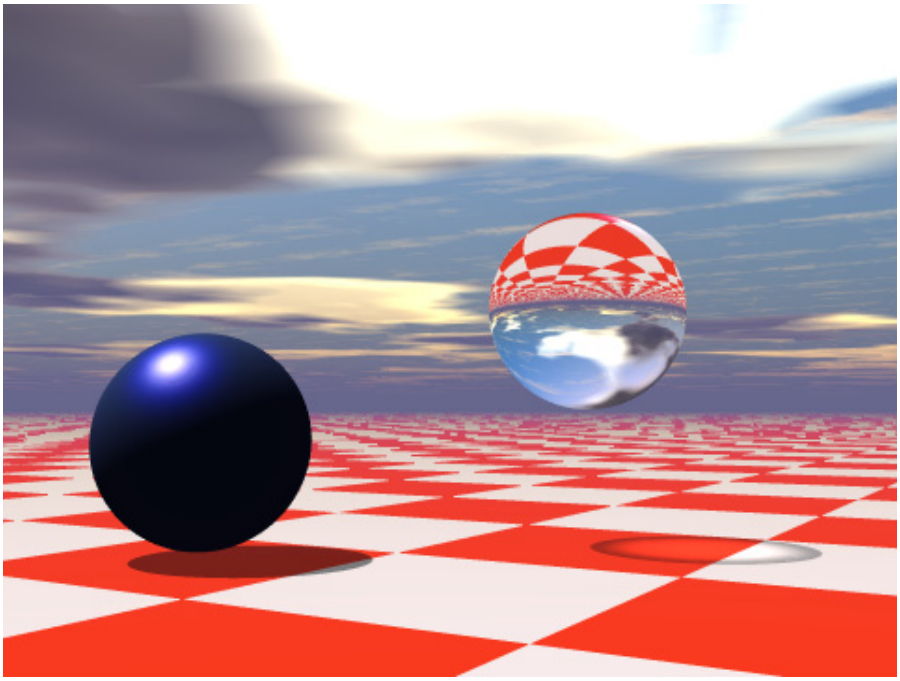


Figure 2.2 Ray Traced Scene

not just light emitters.

An example of a ray traced scene is shown in Figure 2.2. One of the disadvantages can be easily seen in this example. Since point light sources are used, the shadows from the spheres are very sharp even as the distance between the sphere and the ground plane grows. Also note that there is no color interaction between the blue sphere and the ground plane. In a natural environment, the colors of the ground plane would be affected by the blue of the sphere. The effect would be especially evident in the white blocks. More subtly, the refraction of the transparent sphere is too perfect. The sphere shows no reflective properties. A real crystal ball tends to reflect and refract. While this could be added to the geometry, it would greatly increase the complexity and run time of the program. An advantage of ray tracing is that it parallelizes with little effort if each processor can contain the entire geometry description. However, it never converges to the correct answer for realistic surfaces and thus the parallel efficiency is moot.

Radiosity

The class of radiosity algorithms came out of the study of radiant heat transfer. Instead of solving for radiant heat energy, the solution is found for visible light. Radiosity methods solve the Rendering Equation for the special case in which all surfaces are ideal diffuse reflectors. In other words, the radiance L is independent of the angle of emittance. Remember that Immel's formulation of the Rendering Equation 1.4 is

$$L_{out}(\mathbf{x}, \psi) = L_e(\mathbf{x}, \psi) + \int_{\psi'} \rho(\mathbf{x}, \psi, \psi') L_{in}(\mathbf{x}, \psi') \cos\theta' d\omega'.$$

Since radiance is preserved along a line of sight,

$$L_{in}(\mathbf{x}, \psi) = L_{out}(\mathbf{y}, \psi').$$

For ideal diffuse reflectors, the BRDF reduces to a constant and can thus be moved out of the integral. The Rendering Equation then becomes

$$L_{in}(\mathbf{x}) = L_e(\mathbf{x}) + \rho(\mathbf{x}) \int_{\psi} L_{out}(\mathbf{y}) \cos\theta' d\omega'.$$

We can then use the following relationship and convert the equation to a surface integral:

$$d\omega = \frac{\cos\theta}{r^2} dy.$$

This yields

$$L_{out}(\mathbf{x}) = L_e(\mathbf{x}) + \rho(\mathbf{x}) \int_S L_{out}(\mathbf{y}) \frac{\cos\theta \cos\theta'}{r^2} d\mathbf{y}. \tag{2.2}$$

This is the continuous form of the Radiosity Equation. The representative geometry is found in Figure 2.3. The quantity $\frac{\cos\theta \cos\theta'}{r^2}$ is referred to as the point to point *geometrical form factor* between $\mathbf{x}$ and $\mathbf{y}$. It is often represented by the function $F(\mathbf{x}, \mathbf{y})$.

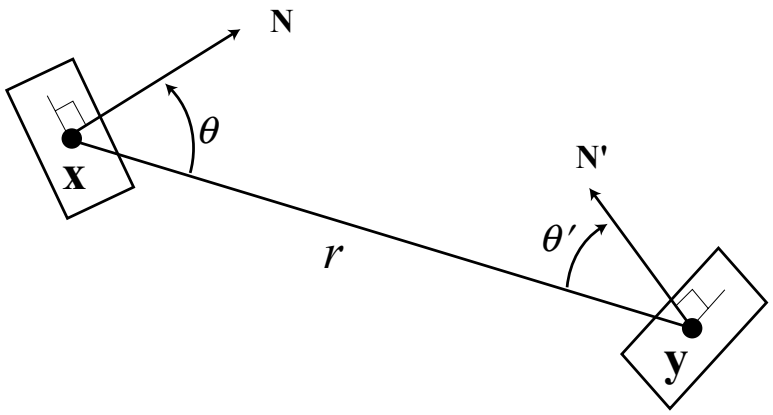


Figure 2.3 Geometry for Radiosity Equation

The continuous form of any equation is difficult to deal with in a computer environment. For numerical computation, Equation 2.2 can be written in discrete form. At this point, the geometry term $g(i,j)$ is also added to account for occlusion.

$$L_{out}(i) = g(i,j)\left[L_e(i) + \rho(i)\sum_{j=1}^{N} L_{out}(j)F(i,j)\right]. \tag{2.3}$$

While determination of the pointwise form factors is straightforward, the determination of the form factor between two arbitrary patches is not. It amounts to a surface integral across the patch at point $\mathbf{x}$ for all points on surface $\mathbf{y}$, resulting in

$$F(A_x, A_y) = \int_{A_x}\int_{A_y} \frac{\cos\theta\cos\theta'}{r^2}\, dA_x dA_y. \tag{2.4}$$

A detailed discussion and the closed form of the form factor for two arbitrary patches can be found in [39]. While a closed form exists, computation is a very arduous task. The complexity of form factor determination is perhaps the biggest motivation for Monte Carlo methods as there is no need to compute the form factor. Since the accuracy of the global illumination solution is only desired to a fixed precision, the form factor is often estimated using two circular patches for which closed form expressions of the form factor are simple.

To produce a solution of equation 2.3, the surfaces of a scene are discretized into small *patches* $p(i), i = 1$ to N, which are assumed to be of constant radiance $L_{out}(i)$ and constant reflectivity $\rho(i)$. The equation is applied for each patch in the scene. This reduces (2.3) to a system of linear equations solvable by traditional Gaussian elimination [6, 7] expressible by

$$(I - \mathbf{p}F)\mathbf{b} = \mathbf{e}, \tag{2.5}$$

where:

- I is the identity matrix,
- $\mathbf{p}$ is the reflectivity vector,
- F is the form factor matrix,
- $\mathbf{b}$ is the radiosity vector, and
- $\mathbf{e}$ is the emittance vector.

All reflectivity values are positive and less than one, and the form factor matrix row sums total one with the diagonal element equal to zero. The resulting matrix is one in which the diagonal elements are ones and the sum of the absolute values of the off-diagonal elements is less than 1. The resulting Gerschgorin circle [28] is centered at 1 with radius less than 1. Therefore, the eigenvalues are all positive and thus the matrix that is formed is diagonally dominant, making the system solvable using iterative methods such as Jacobi and Gauss-Seidel iteration [8]. If the reflectivity range is bound, the condition number of the matrix is known. For a known answer precision and condition number, the number of iterations is constant thus reducing the complexity of the problem from $O(N^3)$ to $O(N^2)$. A detailed discussion of this can be found in [6].

In 1991, Hanrahan *et al.*[23] applied hierarchical methods similar to those used by Appel's N-body algorithm [2] to radiosity. The method relies on the fact that, like N-body calculations, the interaction between patches decreases as the square of the distance

between them. This means that patches that are farther away can be grouped and summarized by a single geometrical form factor. The algorithm adaptively subdivides a surface into smaller patches to improve the accuracy of the geometrical form factor. The adaptive subdivision introduced by Hanrahan improved answer quality and reduced storage compared to full discretization, where all patches are of the same size regardless of importance. However, the adaptive nature depended not on the overall error in the answer, but on the error in a single form factor. Reducing the error in a form factor by half does not necessarily reduce the overall error similarly. Consider a corner in the shadow underneath a desk: Refining the geometry by splitting patches in this area does not improve overall answer quality. It is dark and thus the error associated with the patches will be small. Hanrahan's method incorrectly assumes that answer quality is proportional to the number of patches. What results is a plethora of patches that may be unnecessary.

Hanrahan uses an idealized one-dimensional proof to state that hierarchical radiosity is $O(N)$. That is, the work associated with and N-patch problem is proportional to N. It makes no statement about closeness to a physically correct answer. This is true in a one-dimensional world, however, it does not consider the added complexity of two and three dimensions. In one dimension, each patch interacts with a constant number of patches making it $O(N)$. In two and three dimensions, this is not the case.

It is clear that pure radiosity methods cannot yield a true global illumination solution for general surfaces. The answer produced is view-independent, but it does not account for glare effects, mirrors, or glossy surfaces. Surfaces and effects such as these have different radiance depending on the view angle. Also, due to the tightly coupled nature of these progressive radiosity methods, parallelization has met with little success [7, 51]. But perhaps most importantly, methods for estimating form factors are fraught with difficulties and are nonconvergent or very slowly convergent to the correct values.

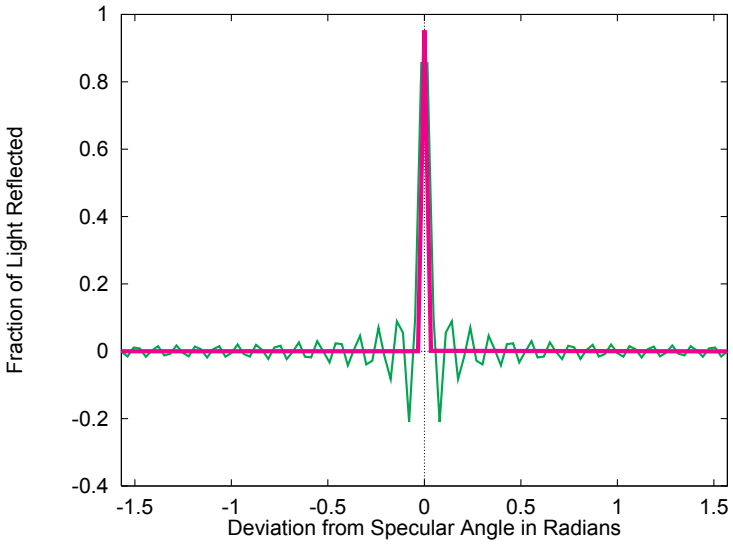


Figure 2.4 Spherical Harmonic Approximation to Specular Reflection Using 30 Terms

Two-Pass Approaches

Radiosity correctly accounts for illumination from diffuse surfaces. The formulation above, however, cannot model any other surface. In light of this fact, many [46, 53] have added a secondary ray tracing pass to account for viewing angle-dependent illumination. In the first pass, the system is solved using radiosity methods for the diffuse surfaces and lighting effects. Then a ray tracing pass is performed which follows specular transfers. While the method produces pictures that are appealing to the eye, they are not physically correct and do not accurately account for the semi-diffuse reflections that occur in nature.

Extended Radiosity

Radiosity does not account for specular reflection, and two-pass approaches are not physically correct. Radiosity is an attractive approach in that it directly solves the Rendering Equation. Sillion [44] uses spherical harmonics to summarize the directional light intensity at each vertex. This method has problems accurately representing specular

reflections. To understand this, consider a specular reflection function such that a spike of intensity occurs at some point. The use of spherical harmonics to represent the spike is shown Figure 2.4. Even at 30 terms the accuracy leaves much to be desired, and moreover, there will always be *ringing* near the spike. Also, requiring possibly hundreds of terms for each specular reflective spike is an excessive demand on memory.

Recently, Aupperle [4] has extended hierarchical radiosity to account for specular effects using three-point light transport. His algorithm is based on directly solving the Rendering Equation using a method similar to Hanrahan's approach to radiosity. The geometrical form factors become 3-point transfer percentages rather than the 2-point transfer percentages used in radiosity. In other words, the method must determine and record the percentage of light from patch x'' that bounces off patch x' and hits patch x, thus accounting for specular reflections.

Aupperle's algorithm is appealing in that it directly solves the Rendering Equation, but computation amounts to a hierarchical tensor product. This is computationally intensive and the prospects for parallelism are small due to its tightly coupled nature. Another drawback of this method is that it only computes the light seen from each *surface* in the scene. This means that to see the scene from any other point in space, a physical viewplane must be inserted into the scene. This severely restricts the views to those of a "fly on the wall." Perhaps most importantly, it inherits the error of Hanrahan's method of equating patch count to answer quality, producing a proliferation of patches that have little contribution to error reduction.

Summary

This chapter has considered various global illumination algorithms. Ray tracing is based on tracing the supposed path of light from the viewpoint backwards into the scene. While this reduces the computational effort and is highly parallel, it does not give the

physically correct solution. Radiosity is based on solving for the equilibrium light radiation in a room. The solution method considers the global effects of light transport, but does not solve for view-angle dependent effects such as specular reflections. Since radiosity methods solve for global effects and ray tracing works well for specular effects, two-pass methods have been developed using a hybrid radiosity ray tracer. However, this approach is also not physically correct. It does not correctly account for semi-diffuse reflections. Lastly, we considered an extension to radiosity in which the global illumination equation is directly solved based on three-point light transport. This effort is appealing, but due to the tightly-coupled hierarchical three-dimensional matrix, parallelization efforts for it appear doomed.

3 MONTE CARLO TECHNIQUES IN COMPUTER GRAPHICS

The term "Monte Carlo method" refers to any algorithm which uses random numbers to generate a statistically convergent solution. Monte Carlo methods are employed in many areas such as computational physics, computational chemistry, and operations research, as well as computer graphics. Most of these algorithms fall into one of two categories: Monte Carlo integration, and Monte Carlo simulation.

Monte Carlo integration is the term applied to algorithms that use random numbers to approximate integrals. If a physical process is modeled using random numbers, the algorithm is referred to as a Monte Carlo simulation. Both techniques have been used in the field of computer graphics for calculations such as radiosity and in distribution ray tracing [10, 9] for soft shadows and motion blur effects. The distinction will be clarified below.

In this chapter, the basics of Monte Carlo integration and simulation are presented. The discussion will cover the general techniques and present relevant global illumination algorithms. For a more general treatment of Monte Carlo techniques, the reader is referred to one of the classic Monte Carlo texts [22, 43, 56] or more recent publications [49, 31].

Background

Monte Carlo algorithms are based on the use of random variables. To understand and appreciate this class of algorithm, one must understand some basic properties of random variables from probability theory.

Random variables come in two varieties: continuous and discrete. Because problems in computer graphics involve integrals of continuous functions, we will focus on continuous random variables. A random variable is considered to be *continuous* if it can take on any value in the interval (a, b) and its density can be represented by an integral. The variable ξ is defined by specifying the interval containing all its possible values, and a function, $p(x)$, called the *probability density function*. Given an interval (a', b') such that $a \leq a' \leq b, a' \leq b' \leq b$, the probability that ξ falls in the interval (a', b') is equal to

$$P(a' < \xi < b') = \int_{a'}^{b'} p(x)dx.$$

Figure 3.1 shows a graphical view of this. The area of the shaded region is the probability that ξ falls in the interval. The density function must satisfy two conditions:

1. The density is strictly positive in the interval (a, b):

$$p(x) > 0. \tag{3.1}$$

2. The integral of $p(x)$ over the interval is equal to one:

$$\int_a^b p(x)dx = 1. \tag{3.2}$$

The expected value, $E(\xi)$, of the random variable ξ is

$$E(\xi) = \int_a^b xp(x)dx. \tag{3.3}$$

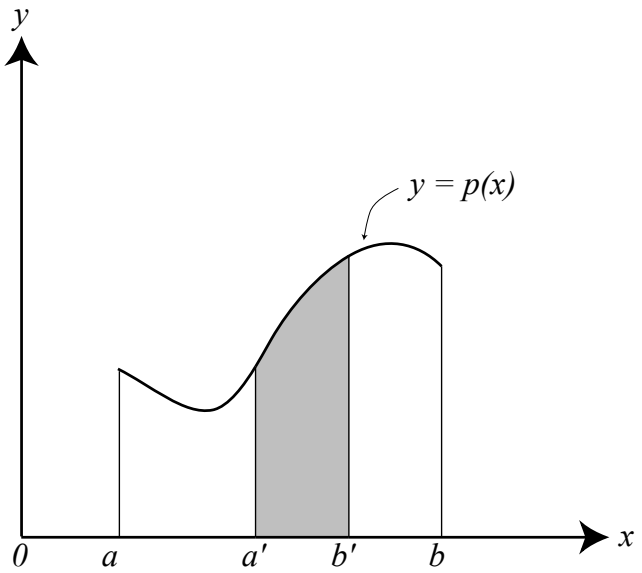


Figure 3.1 Probability Density

and the variance of ξ is

$$var(\xi) = E(\xi^2) - (E(\xi))^2. \tag{3.4}$$

Monte Carlo Integration

It can be shown [22, 43] for an arbitrary continuous function $f(x)$ and random variable $\eta = f(\xi)$ that

$$E(\eta) = \int_a^b f(x)p(x)dx,$$

which can be approximated as

$$E(\eta) \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i). \tag{3.5}$$

This leads to the idea of Monte Carlo estimation of integrals. If we simply substitute $g(x)$ for $f(x)p(x)$ we arrive at

$$\int_a^b g(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} \frac{g(x_i)}{p(x_i)}. \tag{3.6}$$

Note that, due to (3.1), there will never be a division by zero. However, in situations where the probability $p(x_i)$ is very small, the division amplifies roundoff error and results in a poor numerical approximation. Using this method, researchers have approximated the value of integrals found in computer graphics such as the BRDF.

Modified Ray Tracing

Veach and Guibas [52] have extended ray tracing by using Monte Carlo methods to evaluate the BRDF. At each ray-surface intersection, the BRDF is sampled by emitting an appropriate distribution of rays and summing their contribution. For a diffuse surface, the distribution examined would be widely scattered, but for a specular surface the distribution would be much more narrow. This method seems to correctly solve for the radiance given a *single* viewpoint; repositioning requires recomputation. This method is suitable for generating still images, but it is patently inefficient for use in touring a scene. The recomputation of global illumination for each viewpoint would not allow for the redraw speeds needed.

Monte Carlo Simulation

All Monte Carlo methods are simulations of some physical process. Consider the evaluation of a definite integral in 2-D space. The process that is being simulated is a *hit-or-miss* process. Random points are repeatedly picked and determined to be below or above the curve. In the end, the ratio of the number of points below the curve to the total points is used to estimate the area and thus approximate the integral.

In some cases, it is convenient to focus on the *process* to be simulated rather than the definite integral being evaluated. Many times, a statistical model of behavior is known but an analytical model is not. These processes are candidates for Monte Carlo

Simulation.

The defining difference between Monte Carlo Integration and Monte Carlo Simulation is the effect of random numbers on the algorithm. Monte Carlo Integration uses random numbers and their respective probability to approximate an integral, but the random numbers do not affect the decisions that are made in the algorithm. A simulation based on Monte Carlo methods relies on the random numbers to control the flow of the simulation. Decisions are made based on random probabilities.

In particular, consider the process by which light is radiated throughout a scene. Rather than try to solve the integral equation, the behavior of light can be numerically simulated. The simulation involves *emitting* photons from the light source and tracing them throughout the scene. This seems like a simple method, and it is simple in many ways. The problem that must be faced is that for each photon, all its interactions must be accounted for and recorded. Since a numerical simulation of light in a scene must emit large numbers of photons, the memory space required for the interactions is also very large. The process is also computationally intensive. Emitting a photon amounts to tracing the photon through the scene and determining its intersection points. This method was disregarded in the early 1980s due to its heavy resource demands. It was at this time that Whitted introduced his ray-tracing method which only considered the subset of rays that were actually seen and a subset of the contributions to the color of each ray.

If the intensity of reflected light from a surface is expressed as a function, determining the intensity of light for all points is much like determining an unknown function for which Monte Carlo methods also serve well. If we can determine whether a given point is above or below the curve, we can determine the area under the curve. Further, if we discretize the domain, the area for each subinterval can be determined. As we choose random points in the interval, the containing subinterval is determined and a tally is maintained. In the end, the relative subinterval *hit* counts will reveal the shape

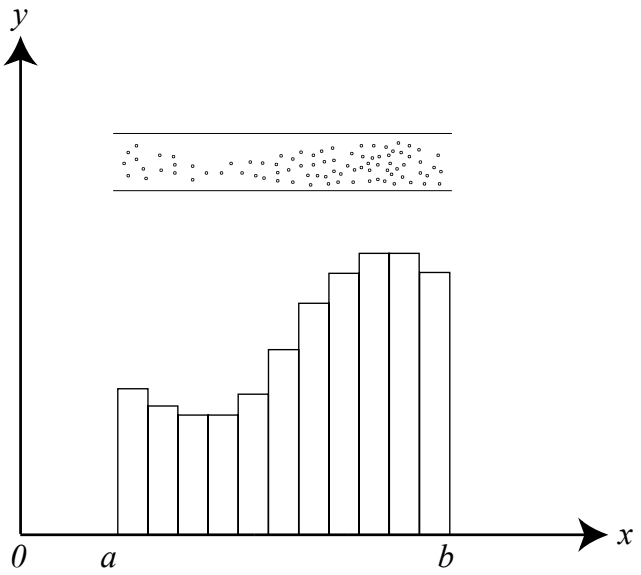


Figure 3.2 Histogramming

of the curve (see Figure 3.2). Obviously, the accuracy of this method depends on the discretization. A finer discretization will produce a higher resolution curve but will require more storage.

In like manner, we can determine the view-independent radiance function along a surface; simply replace the area histograms with volume histograms. Imagine the intensity of light given off by a polygon as a functional surface where the height of the function indicates intensity (see Figure 3.3). Now we can follow a similar algorithm to determine this function. This works well for calculating a view-independent solution but does not account for view-dependent effects such as glare. We will address this in the following chapter. Although the method has high absolute computational and storage demands, it is very economical among methods that compute views from any spatial viewpoint. Since the process of radiation is being simulated, it has the possibility of modeling all lighting effects including polarization and fluorescence.

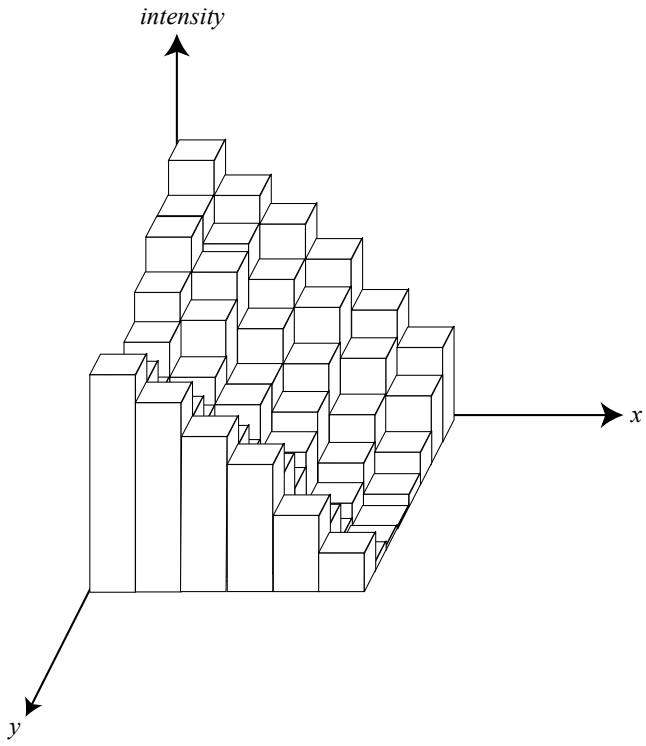


Figure 3.3 3-Dimensional Histogramming

Light Transport Simulation

With the ever-increasing power and speed of today's computers coupled with the diminishing cost of memory, some researchers [40, 36, 20] have returned to tracing light **from** the light source through the scene. Methods such as these use Monte Carlo techniques to simulate the transport of light through a scene. As the light passes through the scene, each interaction with a surface is recorded. Viewing is accomplished by summarizing the interactions and computing a color for each visible point in the scene.

The Density Estimation algorithm proposed by Shirley *et al.* [40] goes through three phases: particle tracing, density estimation, and meshing. In the particle tracing phase, photons are generated and the light transport is simulated. Each interaction is recorded in a "hit point" file in mass storage, thus saving the ray history of each photon. The density estimation phase processes the hit point file and generates an

approximate irradiance function H for each surface, and the meshing phase generates a set of Gouraud-shaded polygons which can then be used for viewing.

Since H is only a function of position, the Density Estimation algorithm produces a view-independent solution to the global illumination problem that does not include specular effects. To account for the viewing angle dependence, a separate ray-tracing pass is performed for each viewpoint. While the pictures produced are appealing to the eye, the ray-tracing pass cannot correctly account for partially specular transmissions, because the light could come from more than one point. Also, the hit point file size is $O(n)$ with respect to the number of photons simulated. This generates huge files that must be distilled to find H.

The Density Estimation algorithm has been parallelized [57] to increase the speed of the simulation. Parallelization is accomplished by two algorithms: The first handles the particle tracing phase while the second performs the density estimation and mesh generation. Due to the nature of the problem, the speedup obtained in the parallel particle tracing is quite impressive, approximately 15 on 16 processors for one geometry. However, for the same geometry the parallel density estimation and meshing phase only reaches a speedup of approximately 8.5 for the same 16 processors. The authors admit that the density estimation and meshing phase speedup is limited by the time needed to process the surface with the largest number of "hit points." In some cases, the speedup in this phase is a mere 4.5 for 16 processors. Hence, parallel approaches to date have not been scalable. The approach we present in Chapter 4 has excellent scaling properties overall.

Adaptive Histogramming

One way to reduce the amount of storage for histogram bins is to adapt the histogram to the unknown curve. This cannot be done *a priori* as the curve is unknown. However, by keeping track of a few extra values, we can *adaptively* create the histogram according

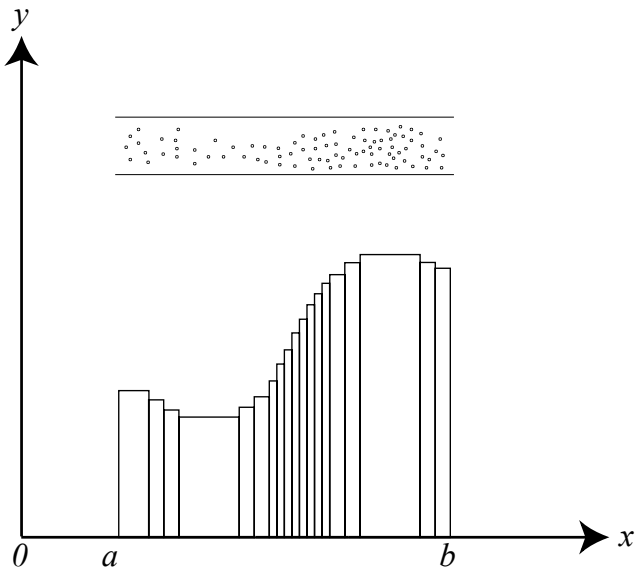


Figure 3.4 Adaptive Histogramming

to the distribution. This method, referred to as *splitting* in the literature [49], is not widely discussed or used. John Gustafson, at Ames Laboratory, independently developed the adaptive histogramming algorithm for Monte Carlo particle transport.

Initially, start with a single subinterval corresponding to the desired interval. As random points are chosen, one keeps track of the number of points landing in each half of the interval. When the halves are statistically different, split the interval and repeat the process on the resulting discretization. Over time the discretization will adapt to the shape of the curve. In those places of steep gradient, a finer discretization will be produced thus increasing the accuracy and limiting the storage requirements to those areas where it is needed (see Figure 3.4).

The difficult decision to make for adaptive histogramming is when to split a histogram bin. A bin is hypothesized to have a uniform distribution such that the left and right halves have the same number of points. Each point that lands in the bin has some probability p of being in the left half and probability $q = 1 - p$ of being in the right half. The resulting distribution is binomial. If we wait until we have a significant number

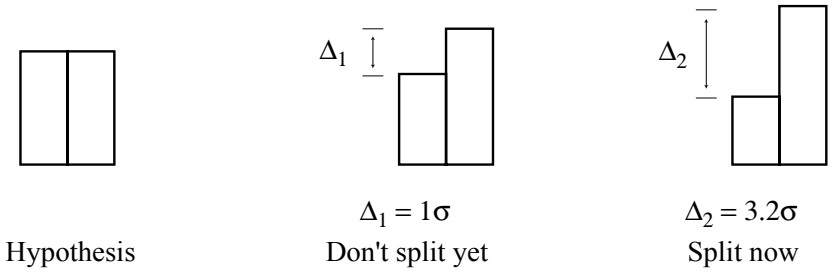


Figure 3.5 Adaptive Histogram Algorithm

of points in a bin before we decide to split, the distribution can be approximated as a normal distribution with mean $\mu = np$ and standard deviation $\sigma = \sqrt{npq}$ [35, 13].

The decision to split a bin is made by deciding that the proposed left and right halves have different distributions. When a point is determined to be in a bin, we also decide which half-bin it belongs in. As the algorithm proceeds, the totals in the two halves may begin to differ (see Figure 3.5). As the difference grows, we can decide whether to reject the hypothesis of an even distribution. In our program, when the two halves differ by more than 3σ we reject the hypothesis. Using 3σ means that with probability 0.9974, we will reject correctly. Although there is a probability that the rejection is incorrect, it will not significantly affect the accuracy of the algorithm. What will result is a bin that was not needed. However, the likelihood of this is very small and thus will not significantly affect the storage required for the algorithm. The result of this algorithm is that refinement is performed only where it is needed to improve accuracy.

The choice of 3σ as a splitting criterion is based on a storage economy versus discretization error argument. Values less than three tend to split histogram bins more often, thus decreasing discretization error but increasing storage demands. Increasing the splitting criterion beyond 3σ reduces splitting, thus reducing storage demands, but also increasing discretization error.

Summary

This chapter introduced Monte Carlo methods for numerical integration and simulation. Monte Carlo integration uses random variables in estimating the value of a definite integral. Numerical simulations that use random variables and distributions are termed Monte Carlo simulations. This method simulates the physical process to numerically solve problems. To reduce the storage demands of Monte Carlo methods, histogramming can be used.

The method is used by graphics researchers to evaluate the BRDF or to simulate the process of radiation. Veach used Monte Carlo integration to augment the ray-tracing method and evaluate the BRDF at each point of intersection. This process only generates a single view. Any repositioning requires a total recomputation for the new viewpoint.

The Density Estimation algorithm involves simulation of the radiation process. For each interaction point between a photon and a surface, the *vital statistics* are recorded. This algorithm results in excessively large files that must be processed to determine the resulting scene.

Adaptive histogramming is a technique that can be used to reduce storage demands with increased quality. Histogram bins are split when there is a statistically significant difference between the distributions of the left and right half.

Parallel processing has been successful on low-fidelity approaches, but unsuccessful as fidelity has increased. However, the Monte Carlo approach appears to be highly parallelizable and capable of producing high quality results.

4 PHOTON

Like Zareski *et al.* [57] the tightly coupled nature of radiosity and its poor prospects for parallelism turned us to Monte Carlo light transport simulation. Its inherent parallelism and ability to account for all non-wavelike lighting effects promise a scalable solution to Kajiya's Rendering Equation. It also solves the problem of accurate form factor computation.

Algorithm Description

We have created an algorithm which we call *Photon* that simulates light transport through a scene. Each emitted photon is traced through the scene until it is probabilistically absorbed. Each time the photon is reflected, a count is maintained which records the number of reflected photons. The count is recorded in a data structure known as a *bin*. A bin is a description of a geometry subset along with the number of photons that have been reflected within that subset.

A high-level description of the algorithm used by Photon is presented in Figure 4.1. We have used the convention that output parameters are indicated using an ampersand to reflect that a reference parameter is being passed. At the heart of a light transport simulator such as Photon are four routines: GeneratePhoton, DetermineIntersection, DetermineBin, and Reflect. These form the basis of the simulation, and also account for most of the simulation time. Each will be discussed below.

```
for iphot = 1 to nphot do
    GeneratePhoton(&photon, &bin);
    UpdateBinCount(&bin);
    absorbed = FALSE;
    while(not absorbed)
        DetermineIntersection(photon, &poly);
        DetermineBin(photon, &bin, poly);
        if (Reflect(&photon, bin) == TRUE)
            UpdateBinCount(&bin);
            if (NeedsSplit(bin) == TRUE)
                Split(&bin);
        else
            absorbed = TRUE;
    endwhile
endfor
```

Figure 4.1 Algorithm used by Photon.

Photon Generation

Computationally emitting photons from a geometrical primitive amounts to the simulation of a *luminaire* (a light emitting object). A diffuse luminaire emits photons in all directions. A random point on the unit hemisphere is used to calculate that direction from any point on the luminiare (see Figure 4.2). Methods such as those by Shirley [40] and Sillion [46] calculate a random direction using the following formula:

$$(x, y, z) = \left(\cos(2\pi\xi_1)\sqrt{\xi_2},\ \sin(2\pi\xi_1)\sqrt{\xi_2},\ \sqrt{1-\xi_2}\right)$$

Using temporary variables to avoid recalculation results in the following algorithm:

$$tmp1 = 2\pi * random()$$

$$tmp2 = random()$$

$$tmp3 = \sqrt{tmp2}$$

$$x = \cos(tmp1) * tmp3$$

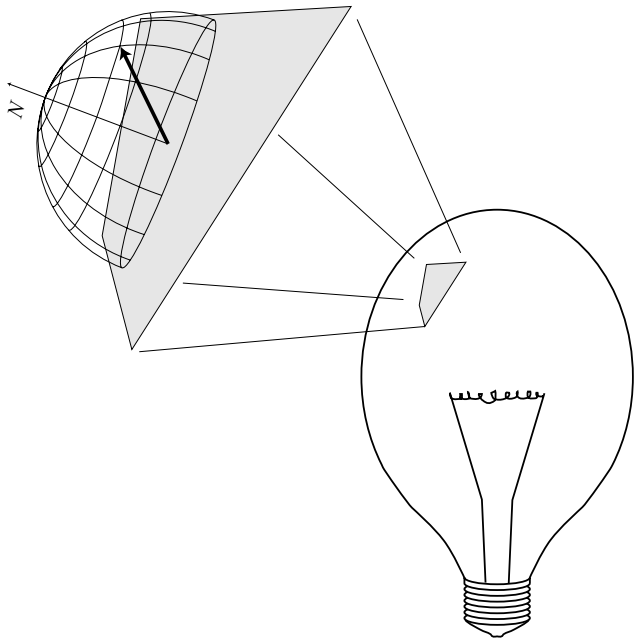


Figure 4.2 Photon Emission

$$y = \sin(tmp1) * tmp3$$
$$z = \sqrt{1 - tmp2}$$

If each random number generation involves 3 floating point operations, this algorithm generates 34 floating point operations (we use the Lawrence Livermore National Laboratory convention that *sin* and *cos* count as 8 operations, and square root as 4). The method used by Photon takes a different approach. Rather than directly calculating the photon direction using only two random number generations, random planar coordinate pairs are generated until they fall within a unit circle, afterwhich the final coordinate is calculated based on the pair.

Again, allowing for 3 floating-point operations in random number generation, one iteration of the loop of the algorithm in Figure 4.3 takes 13 floating-point operations. The loop must be executed at least once, and with a probability of $1 - \pi/4$ the generated coordinate pair will lie outside of the unit circle and the process must be started again. Calculation of the average number of instructions executed by the loop results in an

```
do
    x = random() * 2.0 - 1.0;
    y = random() * 2.0 - 1.0;
    tmp = x * x + y * y;
while(tmp ≥ 1)
z = sqrt(1 - tmp);
```

Figure 4.3 Photon Generation Algorithm.

infinite geometric series:

$$inst = 13 + 13q + 13q^2 + 13q^3 + 13q^4 + \ldots$$

$$inst = 13(\frac{1}{1-q})$$

$$inst = 16.55$$

where q is the probability that the coordinate pair is outside the unit circle. After the coordinate pair is determined, the calculation of z adds 5 floating-point operations. The resulting operation count is 22, which is 12 fewer operations than the method of Shirley. It can easily be shown that both methods generate a uniform distribution of emitted photons. In an environment where possibly billions of photons must be emitted, the method used by Photon presents a substantial savings. This algorithm was developed by John Gustafson at Ames Laboratory in conjunction with this work. Experiments show that our photon generation kernel is about twice as fast as the formula presented by Shirley and Sillion.

The above method accounts well for diffuse lighting. In nature, not all lighting is diffuse. For example, the light we receive from the sun is very directional in nature. Photon simulates this phenomenon by "limiting" or focusing the emitted light. This is easily performed by scaling the unit circle in the generation method above. This limits the angle of emittance, thus providing a directional quality (see Figure 4.4).

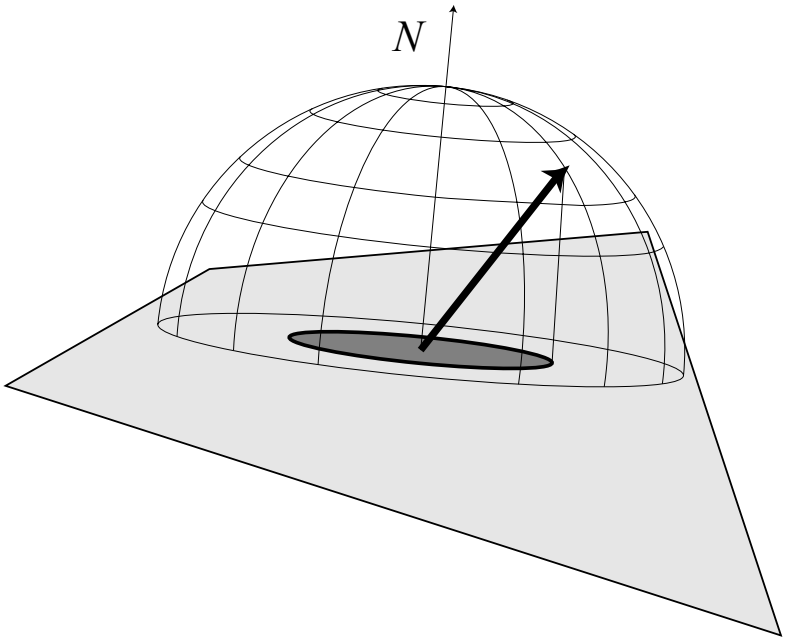


Figure 4.4 Scaling for Directional Lighting

To accurately simulate sunlight, consider that the sun, from our viewpoint, is a disc of approximately one half degree. So, in reference to Figure 4.4, the unit circle must be scaled such that θ is one quarter degree. Therefore, scaling the unit circle by 0.005 and generating restricted-angle photons from a plane yields a light source that simulates sunlight, and correctly blurs shadows as the distance from the occluding object increases. Most rendering programs and graphics packages incorrectly consider the sun or any other light as a point light source, thereby producing unrealistically sharp shadows (see Figure 2.2).

Intersection Determination

Once a photon is generated, the problem breaks down to determining the point of closest intersection. This subject is covered in many ray tracing texts and papers [55, 16, 33]. There are many approaches to speeding up intersection testing that can be applied in this setting, such as bounding volumes [16] and tree structures [38, 17].

However, techniques such as ray coherence [25] are not applicable.

Increasing the speed of intersection determination holds the most promise for decreasing solution time. It is also this area that offers the most prospects for parallelization. Each photon is an independent entity and thus its path can be computed without respect to others. The only dependency is accounting for each photon interaction, which will be discussed in the next chapter.

Reflection

The reflection model used in Photon is based on the work of Xiao He *et. al* [24]. The intent is to make the algorithm as accurate as possible by incorporating the physics of surface-light interaction. This model incorporates all the aspects of the BRDF and includes polarization and masking/self-shadowing effects.

Using this model, Photon has the potential to model polarized light which has not been a well-explored area in computer graphics. Currently, we are working on determining the impact of incorporating polarization in computer graphics. This work is being done by Sairam Sankaranarayanan who has incorporated the reflection model into Photon.

Four-Dimensional Histograms

Photon uses adaptive subdivision of histogram bins, as previously described, to discover the radiance function for a given surface. To allow for the adaptive subdivision, each bin is is parameterized. When the bin is split, two daughter bins are created to keep a more accurate accounting of the light interactions with the surface. For each parameter, a little extra work is performed to determine which daughter bin would have tallied the reflected photon. We only split a bin when the two daughter bins probably have very different photon counts.

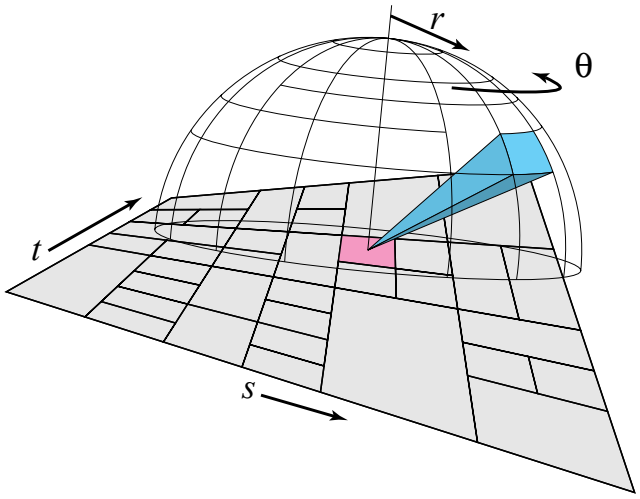


Figure 4.5 Dimensions used in Histogram Binning

As discussed in Chapter 3, the distribution of photons in a given bin is binomial. This allows us to calculate the sample standard deviation as $\sigma = \sqrt{npq}$ where n is the number of photons in the bin and p and q represent the probabilities of a photon being in the left or right daughter bin. These probabilities can be approximated as $p = l/n$ and $q = 1-p$, where l is the number of photons that would be in the left daughter bin. To improve accuracy, p is calculated based on the daughter bin with the most photons. When the two proposed daughter bins differ by more than 3 times the standard deviation, the bin is split. Using the normal approximation to the binomial distribution, 3σ gives us 99.7% confidence that the two bins have different distributions. In this way, Photon adapts the histogram to best reflect the intensity gradient.

As shown in Equation 1.1, radiance is a function of location and viewing angle. It is for this reason that Photon maintains four dimensions for each bin. We use bilinear parameters s and t to represent position on the surface; to record the angle of reflection, cylindrical coordinates r and θ are used (see Figure 4.5). The use of cylindrical coordinates as opposed to spherical coordinates ϕ and θ make the computation of diffuse reflection probability densities simpler. Color is actually a fifth dimension, but one not

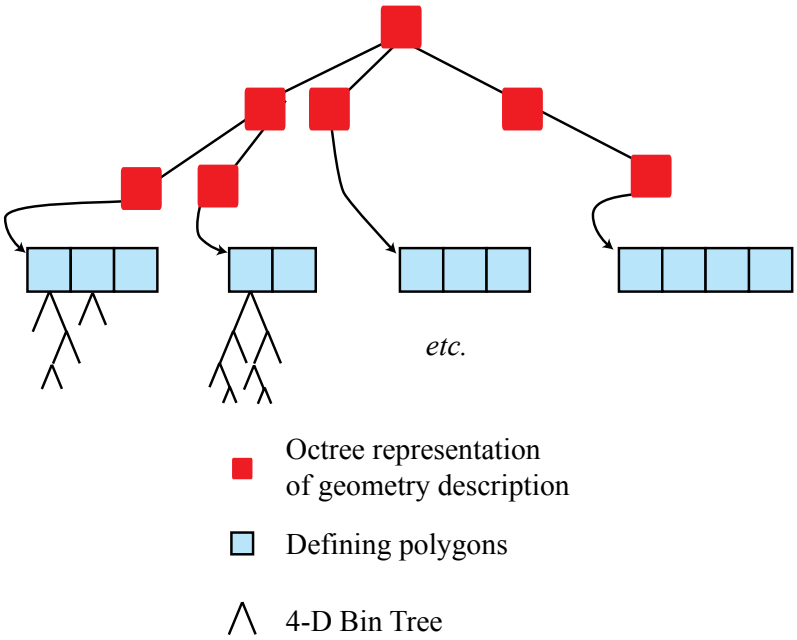


Figure 4.6 Photon Data Structure

subject to hierarchical subdivision in this model. The cylindrical r coordinate is not the usual one. It is the projected radial distance within the unit circle of the direction vector.

When splitting a bin, the goal is to also split the photon distribution in half. Given that the distribution is uniform, it is clear that splitting along the bilinear parameters s and t splits the distribution in half for a nontrapezoidal patch. Trapezoidal patches are not split optimally, but will still converge correctly. Likewise, splitting θ also splits a diffuse surface distribution in half. The last parameter used in splitting the bin is the squared radius r of the projected direction angle. This choice was made because splitting the elevation angle of spherical coordinates does not split a Lambertian distribution in half, nor does splitting the spherical radius. However, splitting the squared radius does indeed result in half the area and thus split the distribution in half, for Lambertian (diffuse) surfaces.

For each geometrical primitive, a bin tree is maintained to record photon counts. The

Figure 4.7 Harpsichord Practice Room

result is a forest of bin trees which is depicted in Figure 4.6. Above the forest is an octree decomposition of the geometry. This data structure is capable of recording the answer of a global illumination model with the color of every patch as a function of the position on the patch and the viewing direction. In this way, we have a discrete representation of the radiance L for all points in a scene. A purely diffuse surface requires only planar bin subdivisions while a specular surface requires more angular bin subdivisions.

Our method can correctly solve a global illumination model that contain mirrors. This can be seen in the Harpsichord Practice Room (Figure 4.7). While the defining geometry is not complex, the patch geometry has 150,000 view-dependent polygons, and it demonstrates some very important concepts. The scene depicts a harpsichord in a room with skylights and a mirrored music shelf. Note the reflection in the mirror. We must stress that this is not the result of a single view-dependent solution. This mirror

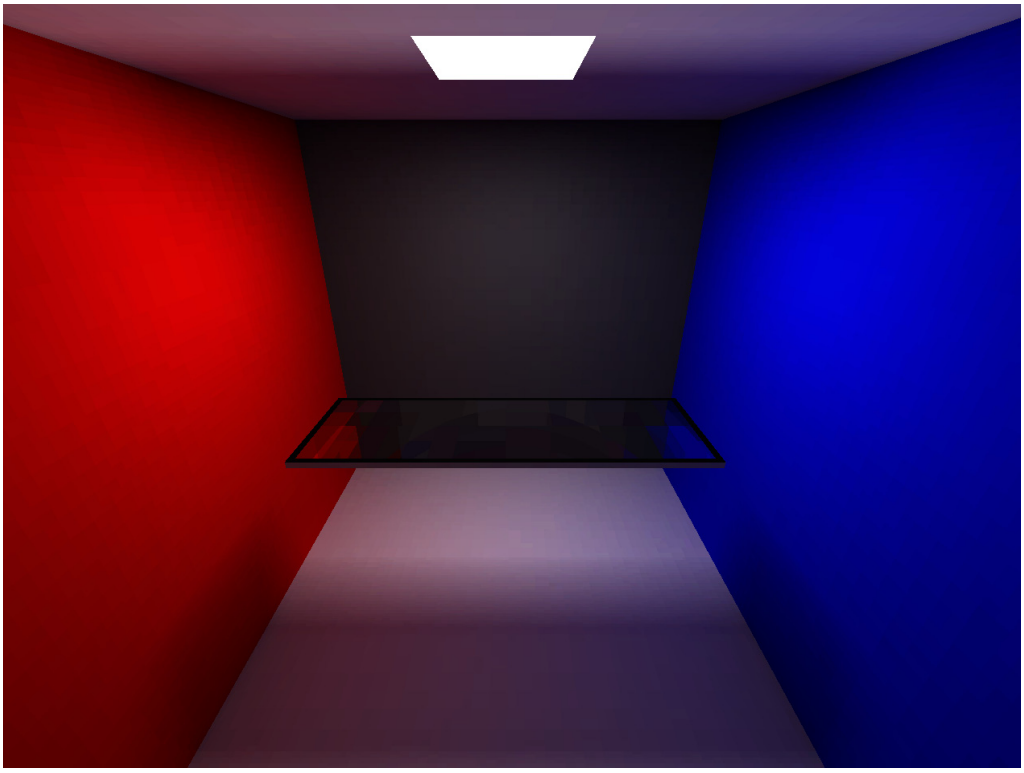


Figure 4.8 Cornell Box with Mirror

can be viewed from all angles correctly as the radiance for all angles is stored in the bin tree for the mirror. Note also that we have *not* resorted to the usual trick of constructing a duplicate room on the other side of the mirror. The mirror is like any other surface, but with a richer set of histogram information. Also notice that the shadowing produced by the skylights is slightly blurred in contrast to the protruding shadow of the harpsichord which is sharper.

Figure 4.8 shows the Cornell Box, named after the seminal research in radiosity done at Cornell University. Floating in the center of the room is a mirror, added for purposes of testing Photon. As with all specular surfaces rendered by Photon, the mirror can be viewed from all angles without recomputation or a full ray tracing pass. Note that in all figures, there is no Gouraud shading performed on the individual patches. This was purposely done to show the adaptive nature of Photon as well as to preserve integrity.

*

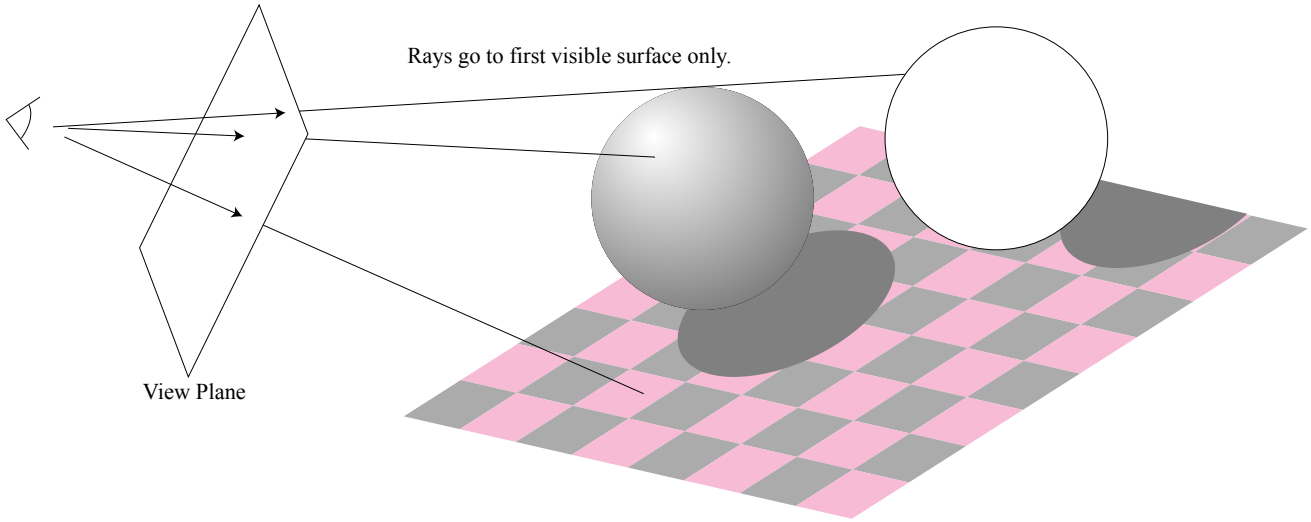


Figure 4.9 Viewing Stage

Viewing Simulation Results

The approach to computer graphics taken by Photon is to break the rendering process into two pieces: *simulation* and *viewing*. Approaches such as ray tracing combine and confuse the pieces. Ray tracers compute the light intensity value of a pixel on the view plane and do not consider all the light interactions. Radiosity, conversely, computes the light interactions but does not consider the view point and thus the specular interactions. Photon determines all the light interactions and stores them in a database. Once the simulation is finished, all that remains is to determine what is displayed. It is much like turning on the lights in a room and then walking in. When the lights are turned on, all the interactions take place. Walking into the room places the viewer in the environment where the light can enter the eye. Thus, all that is needed is to determine what is seen. This can be reduced to a single-step ray trace (see Figure 4.9).

Once the point of closest intersection is determined, the color seen must be calculated. This is done by determining the bin parameters of a photon that would have

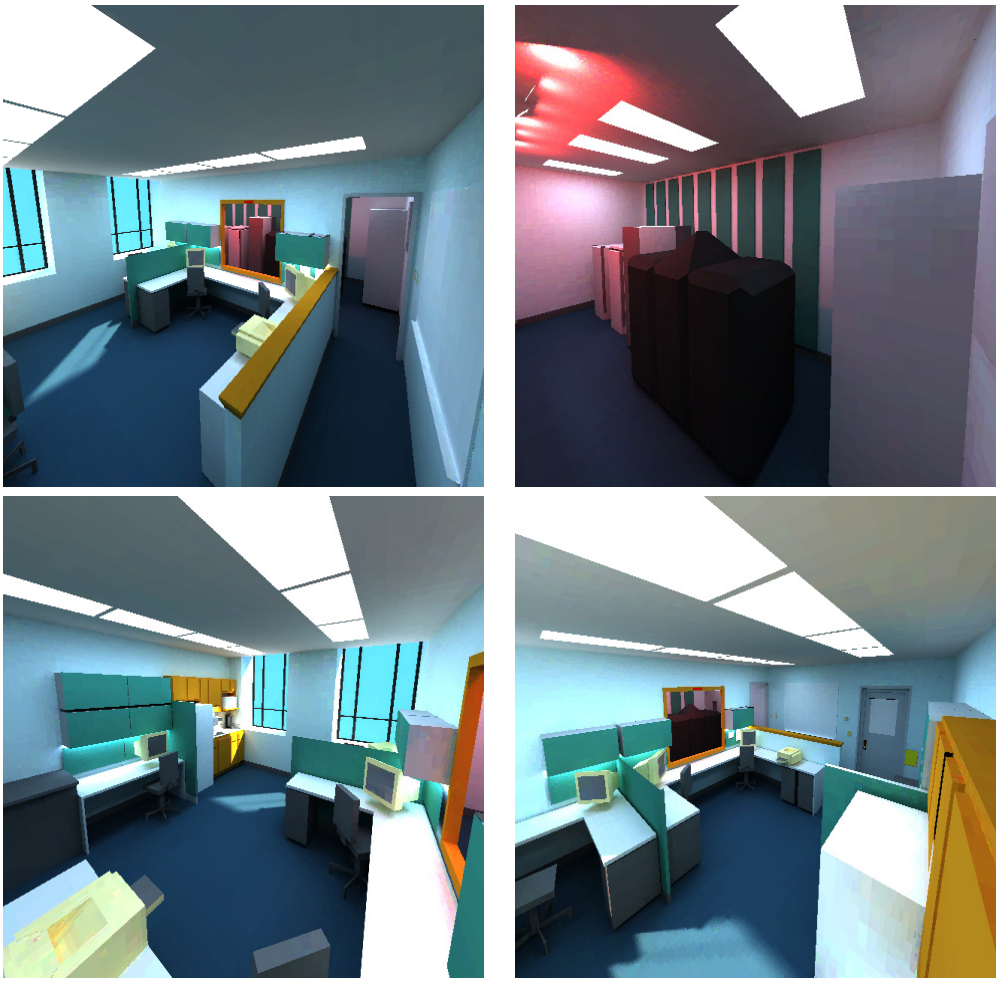


Figure 4.10 Different Viewpoints Using the Same Answer File

traveled along the path from the object to the viewpoint. Since s and t cannot be easily determined from an arbitrary point on a patch, we recursively determine which half of the current bin the point is in and traverse the tree. When a leaf node is reached, the displayed color is calculated based on the ratio of the number of reflected photons in the bin and the total number of photons. The tree traversal and intersection determination is performed by the routines `DetermineIntersection` and `DetermineBin` which are shared between the simulation and viewing programs.

Figure 4.10 demonstrates one of the advantages of Photon. This figure depicts a series

of viewpoints from the same scene. Although the viewpoint is changing, no recalculation of the global illumination is needed. All scenes were generated from the same solution file.

Summary

We have created a new algorithm for global illumination. The algorithm, called *Photon*, is based on Monte Carlo simulation of light transport in a scene. Photon generation, intersection testing, and reflection calculation combine to make up the bulk of the time in a Photon simulation. A new method for photon generation is used which takes fewer operations. Intersection testing is a well-studied topic addressed in ray tracing; it presents the greatest possibility for speedup via optimization and parallelism. The reflection model employed by Photon is based on the work of Xia He *et al.* This model incorporates physical optics and describes all the surface-light interactions. It includes polarization of light which is currently being studied as to its effects on computer-generated images. The photon data structure used here appears to be unique in its representation of global light properties of a scene. The structure accurately represents the discrete form of the radiance function for every surface. Finally, a scene can be viewed by simply determining what is seen given a viewpoint and perspective.

5 PARALLELIZATION OF PHOTON

The parallelization of a program must begin at algorithm design time. Parallelization of an inherently sequential algorithm will yield little parallelism. For example, parallel hierarchical radiosity efforts have met with little success. This is a major reason that we chose Monte Carlo methods for Photon. The Monte Carlo algorithm is inherently parallel corresponding to the parallel nature of global illumination. This chapter will focus on the relevant issues pertaining to the parallelization of Photon.

Two parallel programming models will be considered here: shared memory and distributed memory. Photon has been parallelized for both the shared memory model and the distributed memory model. Both models have their advantages and disadvantages which will be discussed with respect to Photon.

Parallelization Issues

Performance

When discussing parallel programming, one must have a measure of performance for comparing approaches and for demonstrating their respective advantages. For our purposes, we will use speedup as a comparator. Speedup can be defined in many ways. Roughly speaking, speedup is the ratio of the speed of a parallel version and the speed of a sequential version of an algorithm. The problems surrounding "speedup" include how to define speed and when to measure the speed. One can consider a time-based measure of speed by measuring how long it takes to complete a fixed task. We will

term this *fixed-size speedup*. Another approach is to consider a work based approach, i.e. how much work can be done in a given amount of time. We will term this *fixed-time speedup*[19, 21]. The problem with these two approaches and others like them is how to determine the size or duration in fixing the given metric. Examination of a program at different execution durations can, and often does, yield different speedup results.

We have chosen to present the full speedup picture as a function of execution time. A single simulation is broken up into batches of photons. After each batch, Photon calculates the processing rate in *number of simulated photons per second*. The particular batch sizes will be discussed below. By plotting this speed versus time, a trace of the program speed can be displayed. Placing execution traces of different number of processors on the same graph reveals speedup. One can interpolate fixed-time speedup by examining the graph values at a set time. For ease of reading, we have added a speedup scale on the right side of each graph. Speedup is defined as 1.0 for the best serial version of the program (not the parallel version run on one processor).

Another related metric for parallel programs is scalability. A successful parallelization is scalable in time and memory. In other words, a truly scalable program runs twice as fast on twice as many processors and uses, at most, twice as much memory. This is not always a realizable goal. To the extent that it is realizable is a measure of scalability.

Random Number Generation

Monte Carlo techniques are based on the use of random numbers. There has been much research done in this area to insure that pseudo-random number sequences emulate a true random sequence (see [30]). When parallelizing a program that uses random numbers, one must insure that two processors are not using the same random number sequence. Otherwise there is a duplication of effort. Again much research has been done in this area (see [1, 5, 12]). The basic idea is to split the pseudo random sequence into subsequences. Using knowledge about the random number generator, a processor

Table 5.1 Test Geometry Sizes

Geometry	Defining Polygons	View-Dependent Polygons
Cornell Box	30	397,000
Harpsichord Practice Room	100	150,000
Computer Laboratory	2000	350,000

divides the sequence into P equal parts, where P is equal to the number of processors. It then calculates the beginning point in the appropriate subsequence. This is termed the *leapfrog* method. The particular random number generator used by Photon has a period of 2^{47}, which is divided into subsequences for the number of processors, thus yielding individual periods of $2^{47}/P$.

Test Configurations

When testing a program and generating speedup results, it is important to test under a variety of conditions. Photon has been designed to run on a wide variety of supercomputing platforms. It can be run on a traditional supercomputer, a shared memory multiprocessor, and a cluster of workstations, and has been tested on each of the platforms. For purposes of determining scalability beyond 16 processors, Photon was run on an ensemble of up to 64 processors of an IBM SP2. The other two platforms were an SGI Power Onyx and a cluster of SGI Indy workstations. The latter two platforms demonstrate Photon's ability to adapt to the communication medium.

All speedup graphs in this chapter are based on three input files to Photon: the Cornell Box, the Harpsichord Practice Room, and the Computer Laboratory. The geometry files range in complexity from 30 polygons for the Cornell Box, to 100 polygons for the Harpsichord Practice Room, to approximately 2000 polygons for the Computer Laboratory. Table 5.1 shows the resulting number of mesh polygons after subdivision. It is important to note here that while the polygon counts for the geometries seem small, these are the defining polygons and not the resulting mesh polygons after subdivision.

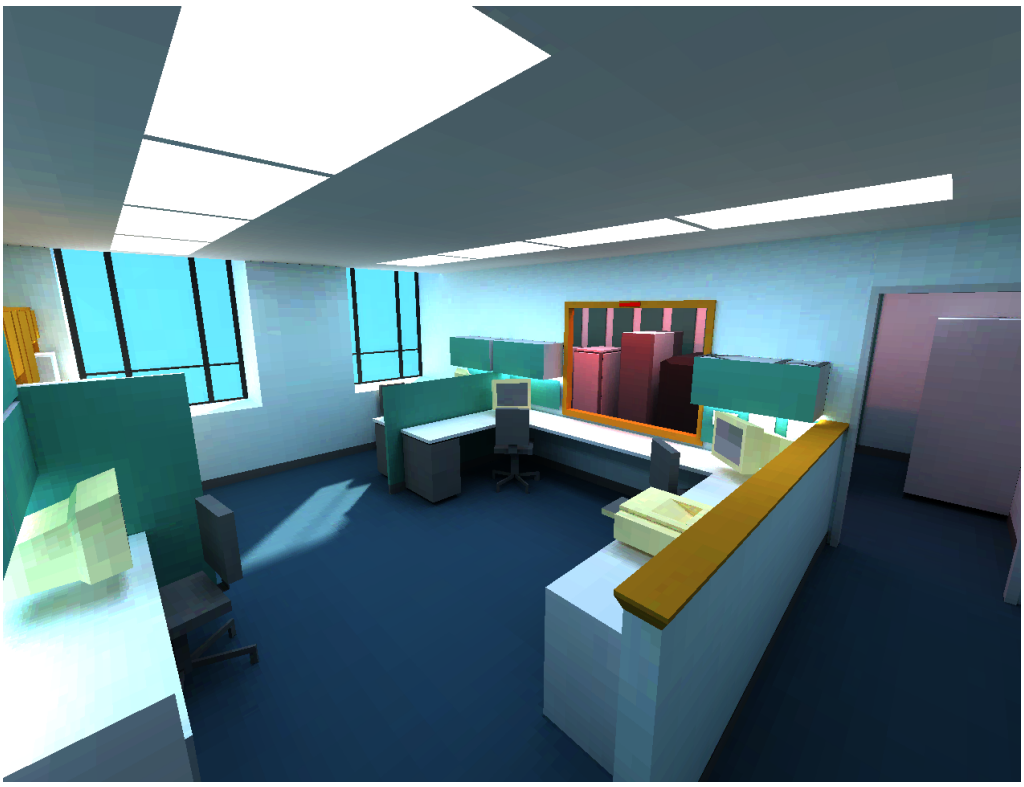


Figure 5.1 The Computer Laboratory

Most rendering programs count the mesh polygons as the total number of geometry polygons. Also note that these are *view-dependent polygons*. Each polygon possibly contains multiple angle-dependent bins. The number of view-dependent polygons is disproportionately high for the Cornell Box due to the large mirror in the center of the box and the simulation has been run much longer to generate a higher level of detail.

The rooms were chosen for their varying degree of content as well as surface types. The Cornell Box (Figure 4.8) and the Harpsichord Practice Room (Figure 4.7) have previously been shown. Figure 5.1 shows the rendered geometry of the Computer Lab.

Shared Memory Parallelization

Shared memory parallelization of Photon is relatively simple. The geometry data structure becomes a shared database with multiple processors accessing and modifying

```
forall iprocessor = 1 to nprocessors do
    for iphot = 1 to nphot/nprocessors do
        GeneratePhoton(&photon, &bin);
        UpdateBinCount(&bin);
        absorbed = FALSE;
        while(not absorbed)
            DetermineIntersection(photon, &poly);
            DetermineBin(photon, &bin, poly);
            if (Reflect(&photon, bin) == TRUE)
                UpdateBinCount(&bin);
                if (NeedsSplit(bin) == TRUE)
                    Lock(bin);
                    Split(&bin);
                    UnLock(bin);
            else
                absorbed = TRUE;
        endwhile
    endfor
endforall
```

Figure 5.2 Shared Memory Algorithm.

it. As in all shared memory programming, one must minimize memory conflicts. All updates to the database must be mutually exclusive. This is an area of extensive research in database and operating system theory [32, 47]. Mutually exclusive access is insured through the use of semaphores to lock access to nodes in the bin forest (as described previously in Figure 4.6), and follows a multiple reader, single writer protocol.

When Photon starts a simulation in shared memory, multiple processes start up and share access to the geometry and bin forest. When a bin needs to be split, it locks access to the bin, splits it, and then releases the lock as can be seen in Figure 5.2. The `forall` loop is a parallel construct that executes the loop statements on all processors. During the splitting phase, all other processes may read any other part of the bin forest except the bin that is being modified. This is to promote as much parallelism as possible.

Distributed Memory Parallelization

Programming Photon for distributed memory systems introduces two inter related issues: data distribution and load balancing. Unlike the parallel Density Estimation algorithm [57] which uses two parallel programs, Photon is parallelizable in both storage and work requirements. Each processor performs the same algorithm generating and tracing photons through the scene. Since the data structure is distributed, the processors must communicate to update the bin forest and synchronize. This algorithm is presented in Figure 5.3. The statements in bold type are changes from the original serial algorithm that will be discussed below.

The Message Passing Interface standard (MPI) [34] was chosen for all interprocessor communications. This choice gave us the greatest flexibility and portability. Many supercomputers now have a native implementation of MPI. It also allows us to run Photon on our network of workstations.

Data Distribution

The data structure used by Photon for storing the histogram bins is made up of two distinct sections: the geometry description and the histogram bin forest. While the memory requirements to store the geometry information for a given scene remain constant throughout the run, the memory requirements for the bin forest tend to be nearly $O(n)$ in the number of photons. Figure 5.4 clearly shows that after an initial buildup of memory, the size of the bin forest tends to increase sub-linearly. The bin forest is the largest data structure in memory and thus the target for distribution. For this implementation, the bin forest was chosen for distribution among the processors and the geometry was replicated across all processors.

Each processor is assigned a section of the bin forest (see Figure 5.5). That processor is responsible for maintaining the photon tallies and splitting bins when and where

```
for iphot = 1 to nphot/batchsize do
    for jphot = 1 to batchsize do
        GeneratePhoton(&photon, &bin);
        UpdateBinCount(&bin);
        absorbed = FALSE;
        while(not absorbed)
            DetermineIntersection(photon, &poly);
            DetermineBin(photon, &bin, poly);
            if (Reflect(&photon, bin) == TRUE)
                if(bin.ProcessorID == MyID)
                    UpdateBinCount(&bin);
                    if (NeedsSplit(bin) == TRUE)
                        Split(&bin);
                else
                    EnQueue(photon, &Q[bin.ProcessorID]);
            else
                absorbed = TRUE;
        endwhile
    endfor
    for iprocessor = 1 to nprocessors do
        if (iprocessor != MyID)
            SendQ(iprocessor, Q[i], sizeof(Q[i]));
    endfor
    for i = 1 to nprocessors - 1 do
        ReceiveQ(&TempQ, ANYPROCESSOR, size)
        for j = 1 to size do
            DetermineBin(TempQ[i], &bin);
            UpdateBinCount(&bin);
            if (NeedsSplit(bin) == TRUE)
                Split(&bin);
        endfor
    endfor
endfor
```

Figure 5.3 Distributed Memory Algorithm.

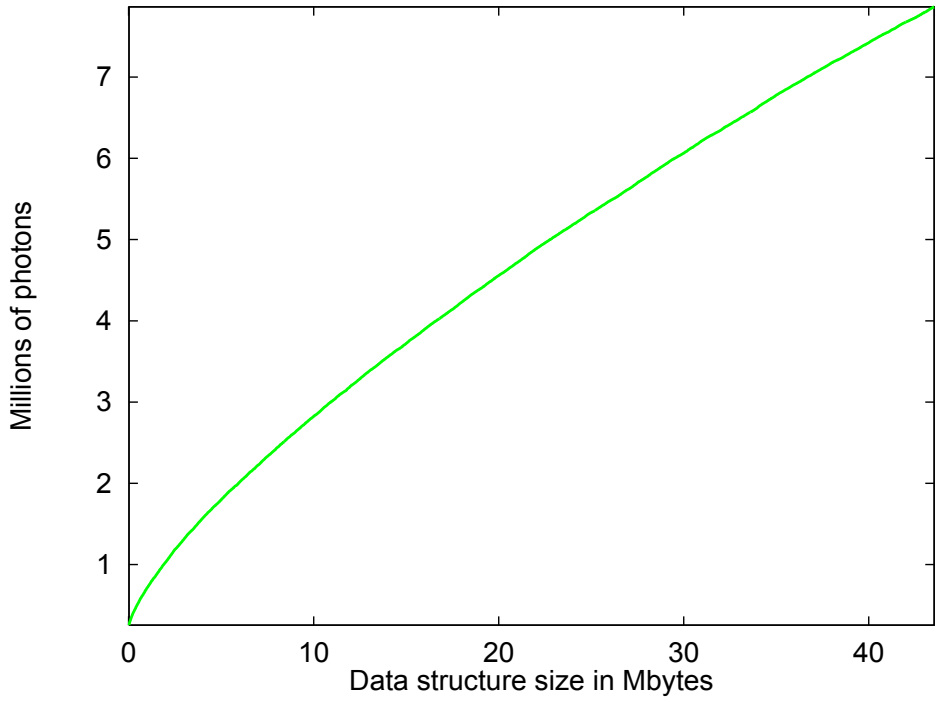


Figure 5.4 Memory Requirements for the Harpsichord Practice Room

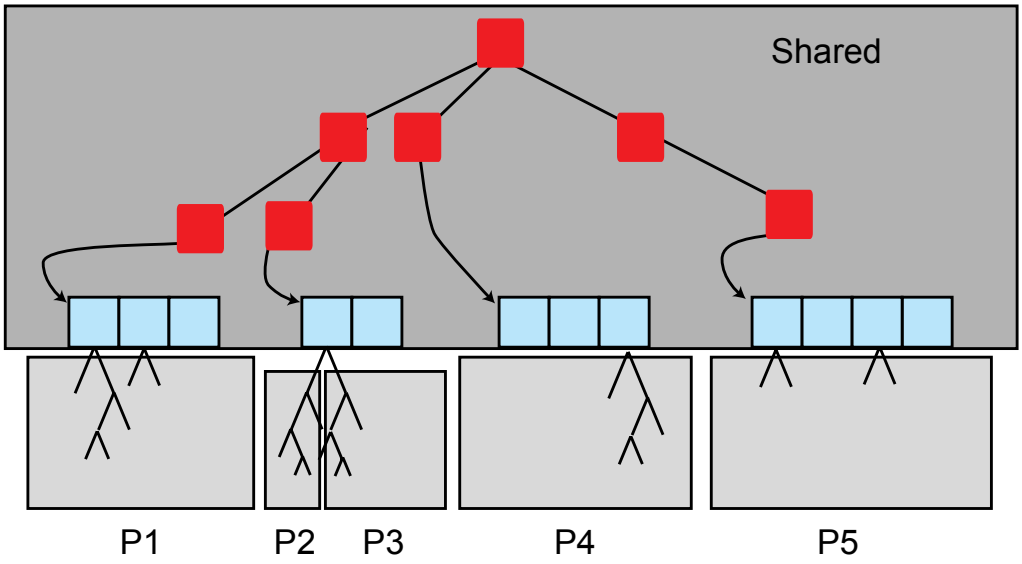


Figure 5.5 Data Distribution for Parallel Photon

necessary. To do this, every reflected photon must be forwarded to the processor which owns the bin. To save on message overhead and increase performance, photons are queued and batched for transmission. This results in an all-to-all communication period following each particle tracing phase. When a set of photons are received, all that is required is to determine the bin of interaction and update the appropriate parameter tallies, splitting bins where necessary.

Load Balancing

Due to the nature of the algorithm, naive load balancing for Photon can lead to disastrous results. Consider a darkly painted room with only a spotlight that is focused on the floor. If the floor is assigned solely to processor 0, that processor will have to do all the work as all the photons generated must be passed to processor 0 to update tally counts. Clearly this limits parallelism. The same thing can happen in a more general environment. If a major percentage of the light receiving polygons are assigned to one processor, parallelism will be restricted.

Avoiding the above situation is important to promote scalability. In the current implementation, initially all processors are assigned ownership of the entire geometry. During this load balancing phase, k photons are generated and traced through the scene. The parameter k has been chosen to best generate a *good* (not optimal) balance. Due to the random nature of Photon, it appears that k does not depend on the size of geometry. However, more research needs to be done in this area. No tallying is performed until the photons have been traced. Then each processor goes through the photons in the same order, thus producing the same bin forest. At this point, we are able to use the photon counts for each bin to determine an appropriate load balance. The period of redundant work lasts less than a second and is quickly made up in higher parallel performance.

Finding an optimal load balance is then reduced to the bin packing problem which has been shown to be NP-Complete [15]. However, a good approximation can be reached

Table 5.2 Total Photons Processed using Naive Load Balancing Versus Bin Packing. All counts are in thousands of photons.

Processor	Naive Load Balance	Bin Packing
0	47.9	29.4
1	34.5	28.9
2	35.6	29.8
3	25.6	29.4
4	32.7	29.6
5	24.9	29.1
6	35.1	28.7
7	32.8	29.0

using the Best-Fit algorithm. Essentially, a bin is added to the processor with the smallest photon count. While this is a greedy algorithm, it has produced good results. If one processor must process more photons than another, it will take more time per batch and force the remaining nodes to wait. Ideally, all nodes would process the same number of photons and thus not have idle time waiting for communication from an overloaded processor. Table 5.2 shows a comparison of the number of photons processed by each processor using naive load balancing versus bin packing. It shows that load balancing using bin packing is clearly superior to naive balancing. Consider, for example, processor number 5 processes half as many photons as processor 0 using naive load balancing. This means that processor 5 must have spent a lot of idle time waiting for processor 0 to finish. Load balancing using bin packing more evenly distributes the work and thus lessens idle time.

Communication vs. Computation

Another aspect of load balancing in a distributed computing environment is matching communication with computation. Photon simulates light transport in batches of photons followed by a period of communication. If the batches are too large or too small, it could be detrimental to performance. If batches are too small, most of the communi-

Table 5.3 Simulation Batch Sizes

SGI Power Onyx	IBM SP2	SGI Indy Cluster
500	500	500
750	750	750
1125	675	1125
1687	1012	1125
1518	1012	1125
2277	910	1125
3415	1365	1012
3073	1365	1012
4609	1228	1012
4148	1842	1012
6222	1657	1518
7558	1657	1518
11337	1657	1518

cation time will be spent in latency, thus slowing down the simulation. Likewise, overly large batches may spend too much time in transmission, due to large message sizes, and thus slow down the simulation.

Photon attempts to match batch size to communication medium. This is accomplished by a growing batch size to maximize overall simulation speed. Batch size starts with just 500 photons per processor and grows as long as overall speed is increased. When a decrease in simulation speed is detected, the batch size is reduce by 15 percent. Table 5.3 shows the resulting sequence of batch sizes for the three compute platforms. In each case, the simulation was performed on 8 processors using the Harpsichord Practice Room geometry.

Results

Previously, the implementation specifics of two parallel implementations of Photon have been discussed. We now present and compare the resulting performance of the algorithms. Note that in all cases, when a single processor performance is given, it

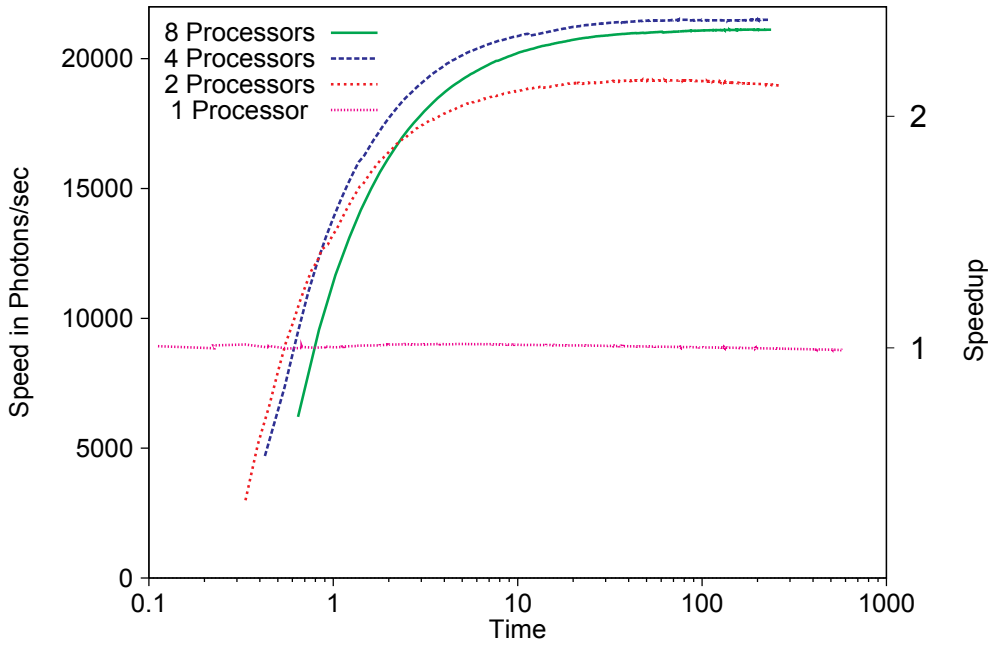


Figure 5.6 Shared Memory Speedup Results (Cornell Box)

is generated using the best serial version of the code. It is not merely the parallel code running on a single processor. Often, researchers present *relative* speedup figures by executing the code on a single processor and repeatedly doubling the number of processors but always using the same code.

Relative speedup is useful to see the scalability of a program, but it does not show if the program was worth parallelizing. Comparison with the best serial algorithm demonstrates the advantages and disadvantages of parallelism. One will notice in the graphs to come that often the serial performance is more than half of the performance on two processors. This is due to the parallel overhead from memory contention or message passing.

SGI Power Onyx

Figures 5.6 through 5.8 show the resulting performance for the shared memory version of Photon on an 8-processor SGI Power Onyx. These graphs show a typical parallel

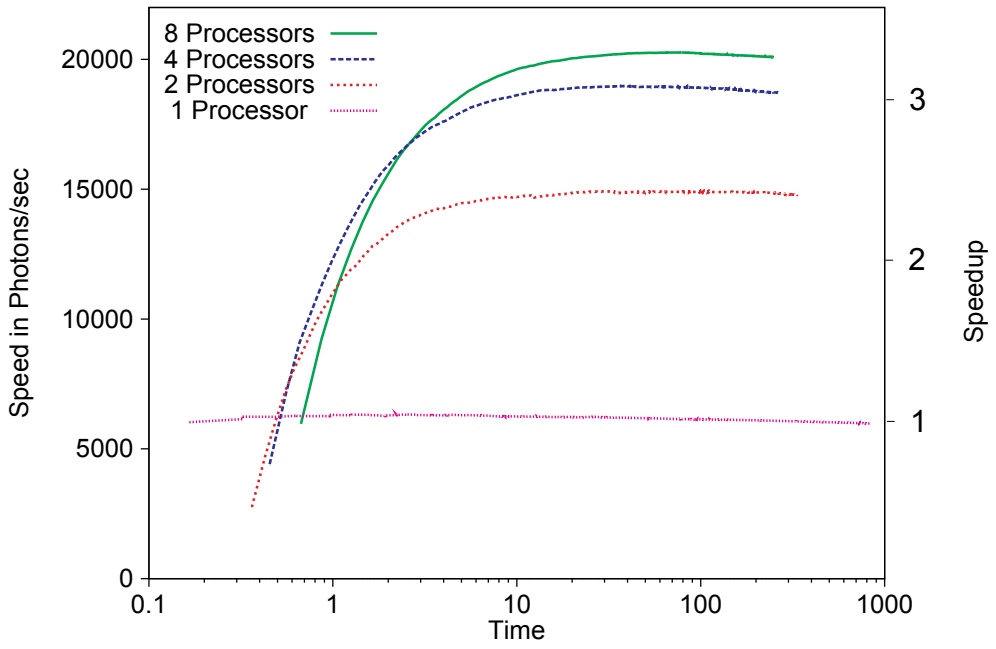


Figure 5.7 Shared Memory Speedup Results (Harpischord Practice Room)

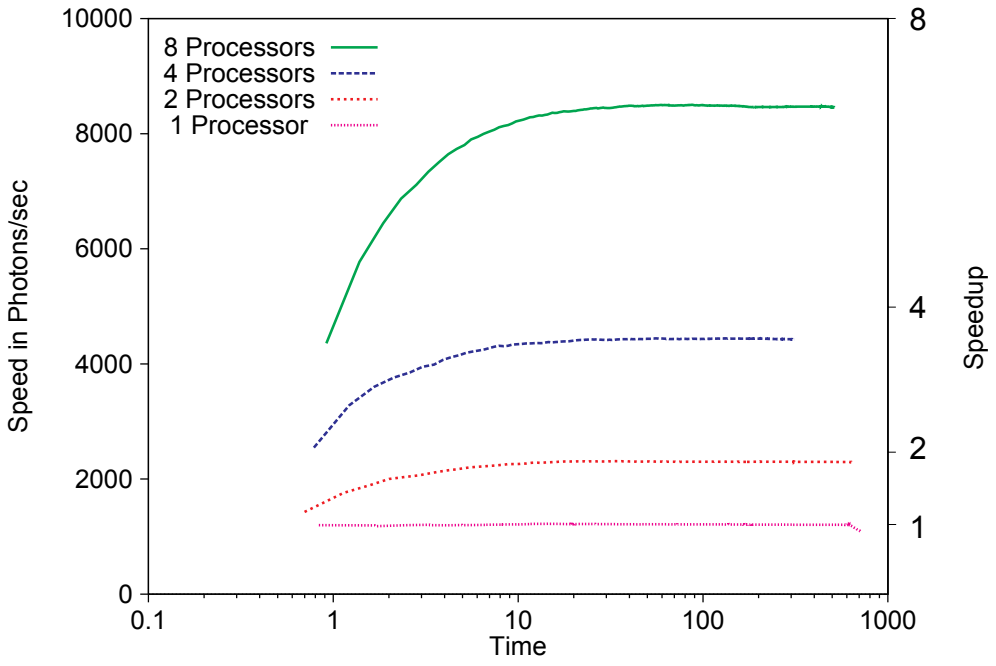


Figure 5.8 Shared Memory Speedup Results (Computer Laboratory)

processing phenomenon: As the geometry size increases, so also does the scalability. For small geometries, using more than two processors is a waste. Large geometries, on the other hand, seem to have excellent scaling properties. This can be attributed to the fact that with a large geometry, processors spend more time in other areas of the bin forest or program code. One might think that subdividing a small geometry to create more polygons would improve speedup and thus improve performance. This is not the case. Notice that as the geometry size increases, the scalability increases, but the absolute performance is reduced.

SGI Indy Cluster

The SGI Indy cluster demonstrates the scalability of Photon in a message passing environment. This version of the code queues photons to be sent to other processors in the all-to-all communication phase. Notice that communication overhead and slower processors force the initial time to the right and reduce performance. Although performance is lost, scalability is increased. Often, when a program is parallelized for distributed memory, the memory bottlenecks are removed, making it possible to run faster and scale better. This is the case with Photon. Each processor is free to work in its own memory space and thereby reduce memory contention.

IBM SP-2

Figures 5.12 through 5.14 show the performance graphs for the IBM SP-2. The results are similar to the SGI Indy cluster, except that the reduced scaling between 2 and 4 processors is not expected. Beyond 4 processors, the graphs show that Photon seems to scale well. The reason for the performance hit that is taken by moving from 2 to 4 processors is theorized to be due to communication overhead. Unlike the Indy cluster, the SP-2 requires that asynchronous messaging be buffered, which adds an extra memory copy and buffer management overhead to each message. In a configuration of 2 nodes,

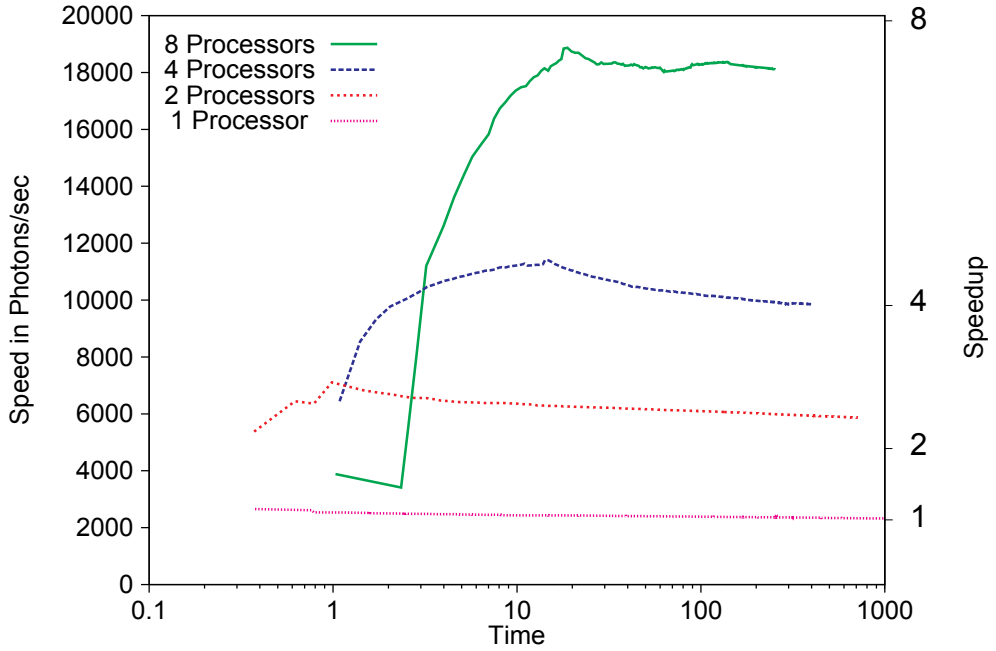


Figure 5.9 Indy Cluster Speedup Results (Cornell Box)

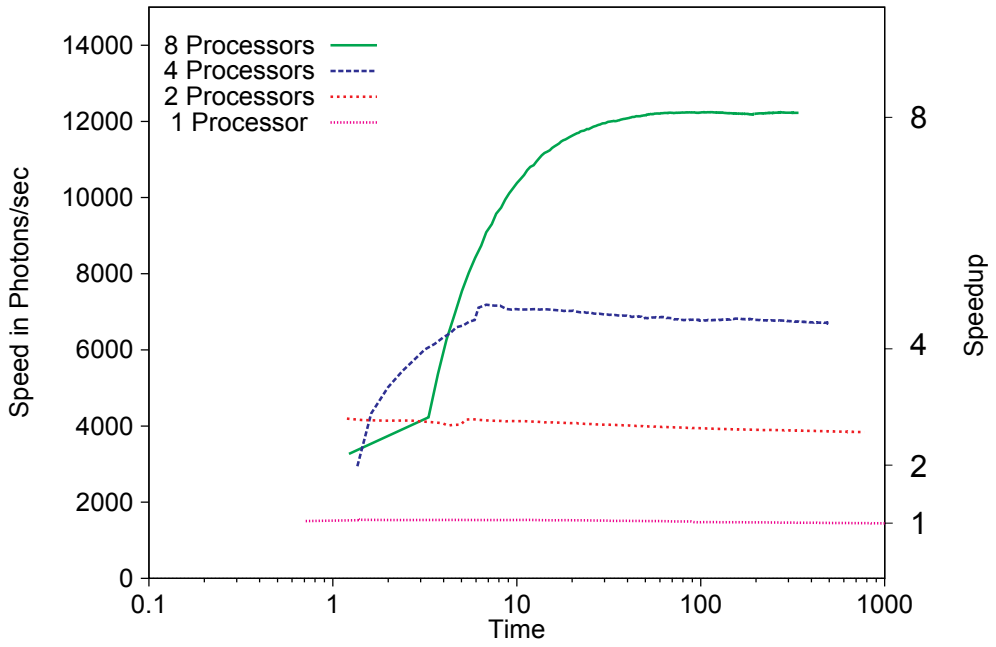


Figure 5.10 Indy Cluster Speedup Results (Harpsichord Practice Room)

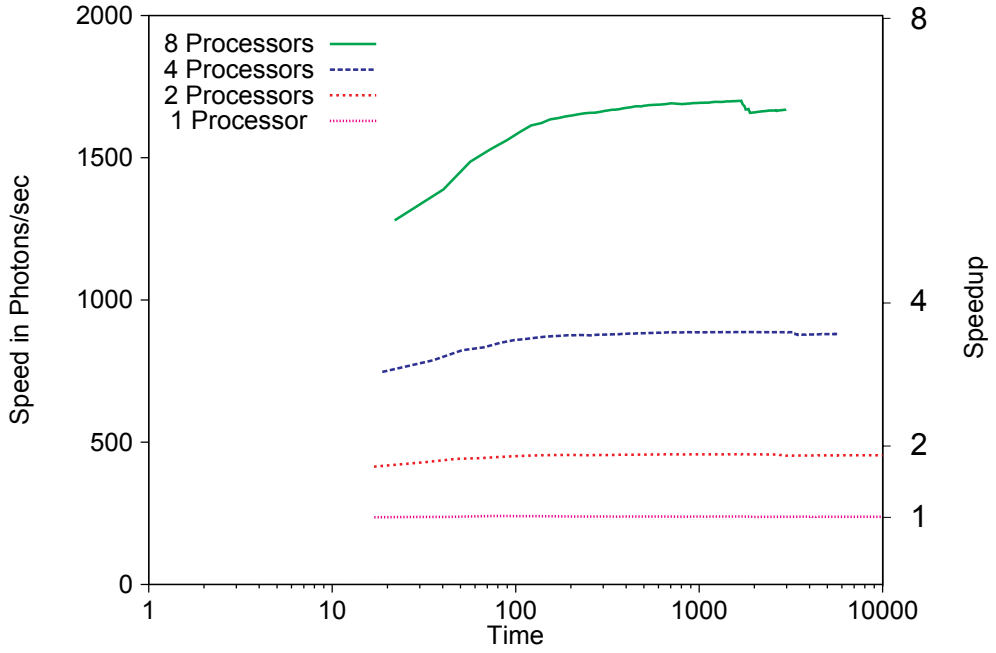


Figure 5.11 Indy Cluster Speedup Results (Computer Laboratory)

each processor only sends one message per batch. In this case the communication may be overlapped with computation and thus hidden. Increasing the number of processors increases the buffer management and memory copy overhead to the point that it can no longer be hidden. This is why the absolute performance of configurations of more than two processors is shifted down. However, performance after the shift appears to scale well.

Visualizing Performance

Performance differences are often hard to express and visualize. In this section, we present two new ways of visualizing and comparing computer performance. We believe they are unique in their representation and highly informative.

First, Figure 5.15 shows a table of performance graphs, a 4-dimensional graph-of-graphs data representation. Each graph is a log-log plot of the data on the same xy scale. The horizontal outer-graph scale increases in complexity of the geometry scene,

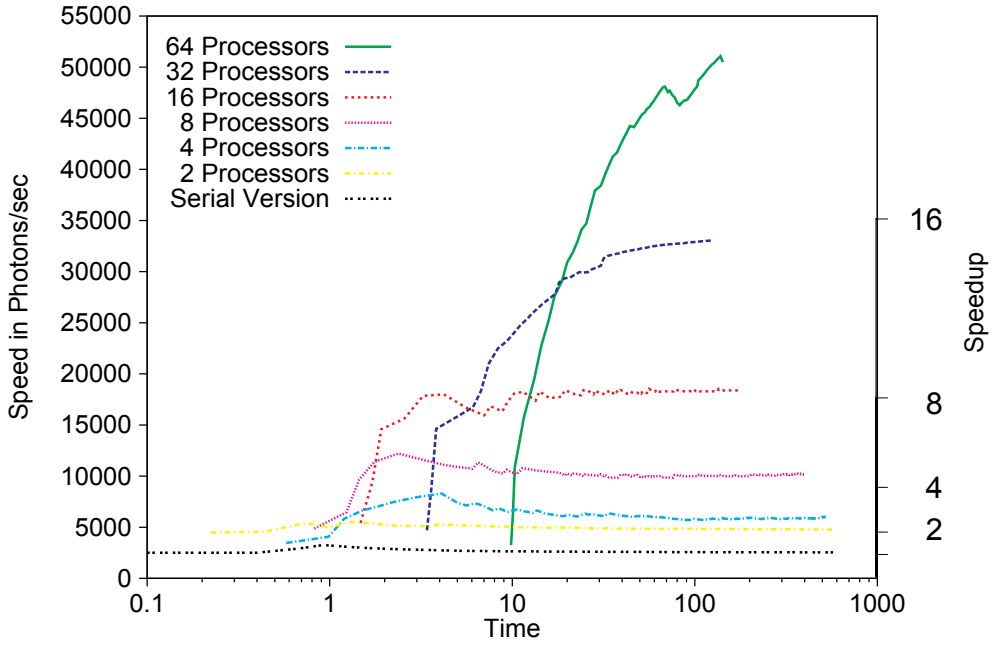


Figure 5.12 SP-2 Speedup Results (Cornell Box)

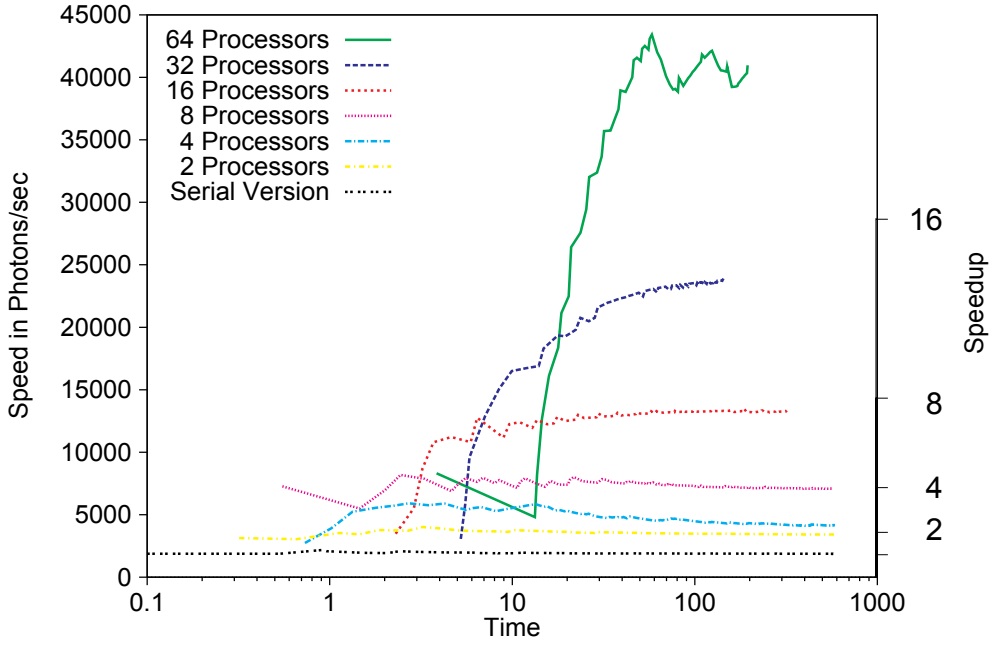


Figure 5.13 SP-2 Speedup Results (Harpsichord Practice Room)

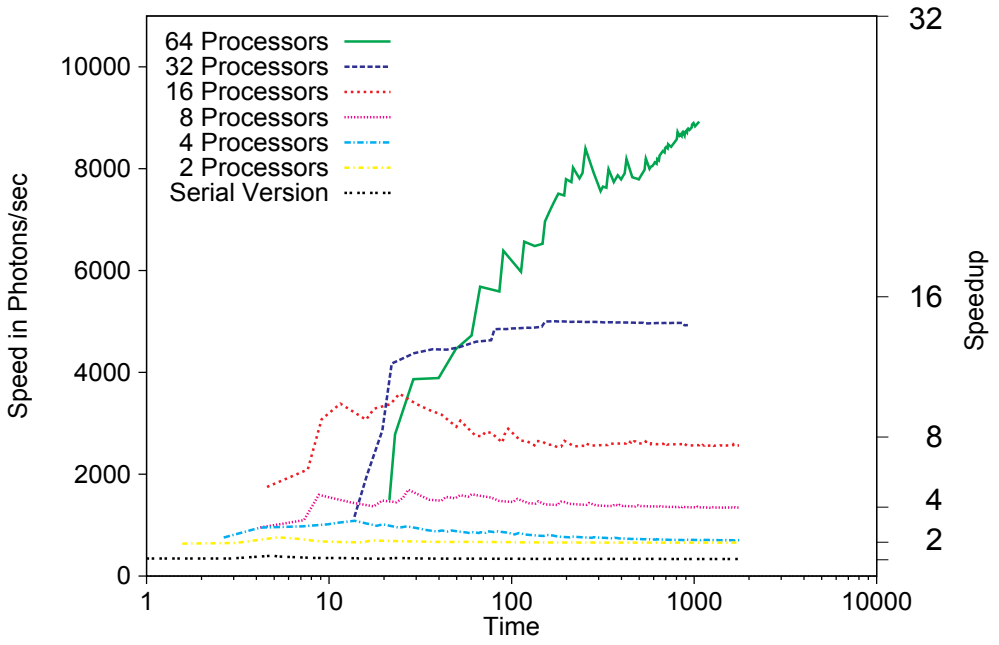


Figure 5.14 SP-2 Speedup Results (Computer Laboratory)

and the vertical outer-graph scale increases in processor coupling. Note how the time to the first data point increases as coupling decreases. This is due to slower message passing speeds. The graph shows that as the complexity of the geometry scene increases, scalability also increases. However, the overall performance is decreased.

Finally, Figure 5.16 gives a visual feel for speedup. Photon was used to generate a scene on 2, 4, and 8 processors. In each case the simulation was run on an SGI Power Challenge for approximately two minutes using the distributed memory version of Photon. It is easy to see the improved quality due to higher photon simulation counts. Note especially the improvement in the mirror, and in shadows under the harpsichord and skylight.

Summary

Photon has been parallelized for both shared memory and message-passing environments. The shared memory version of Photon works best on large problem sets where

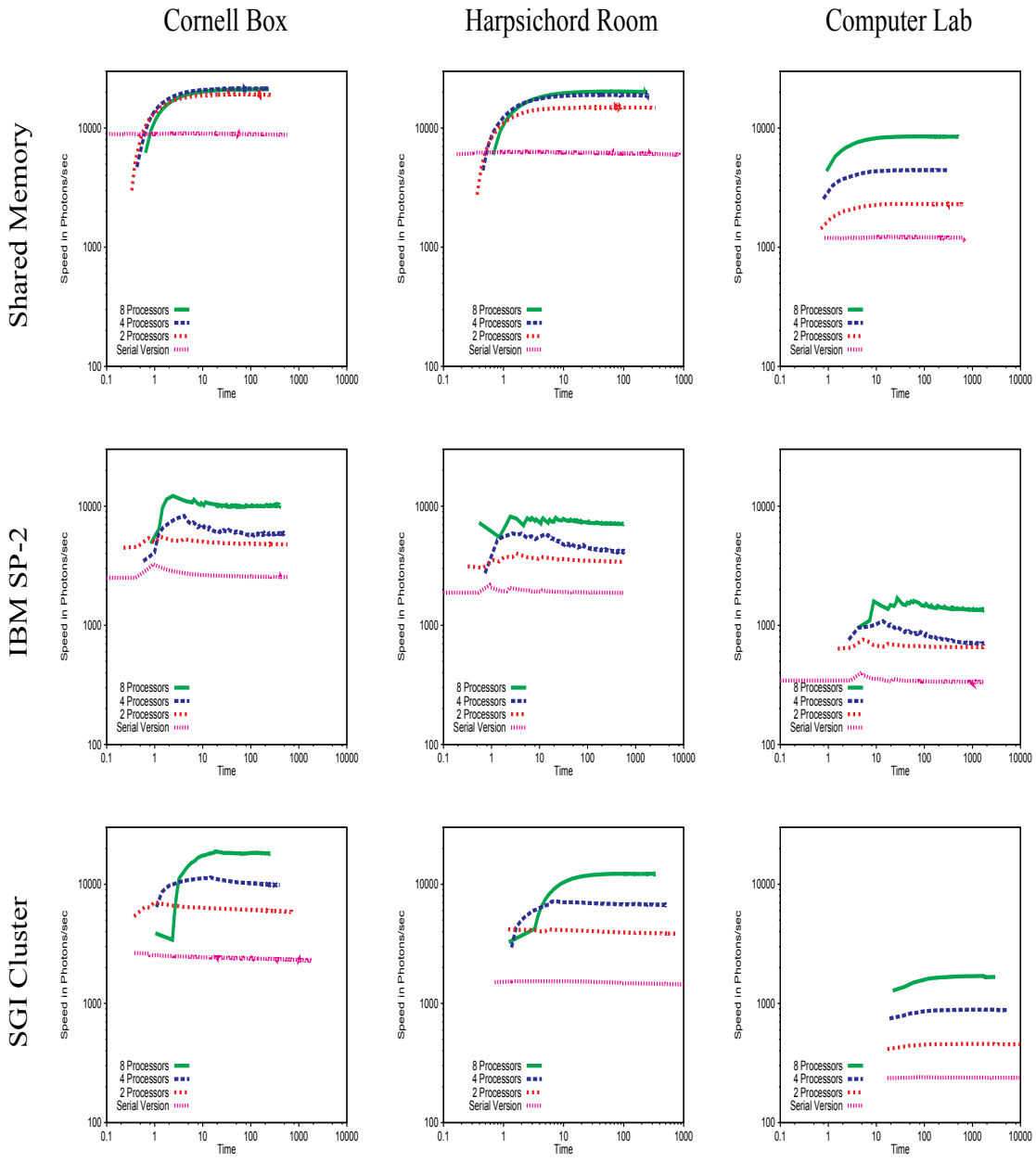
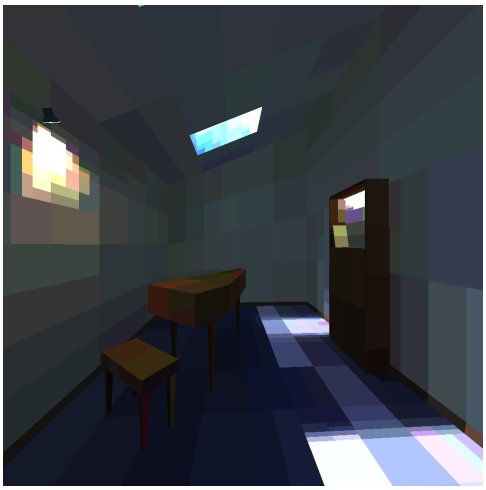
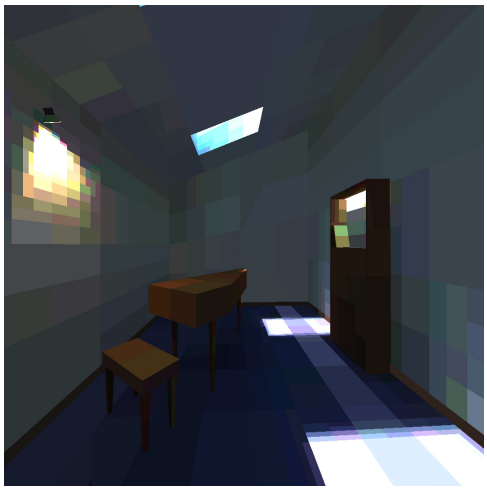


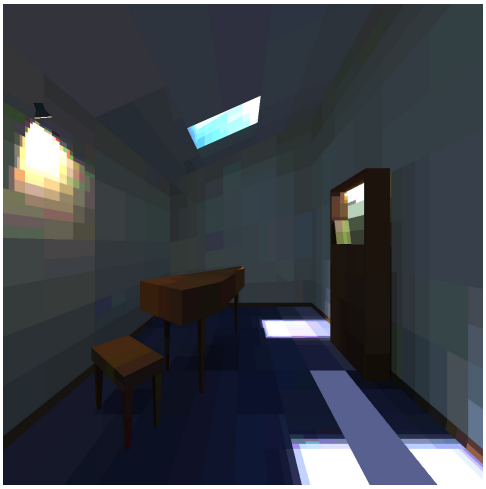
Figure 5.15 Performance and Speedup vs. Complexity



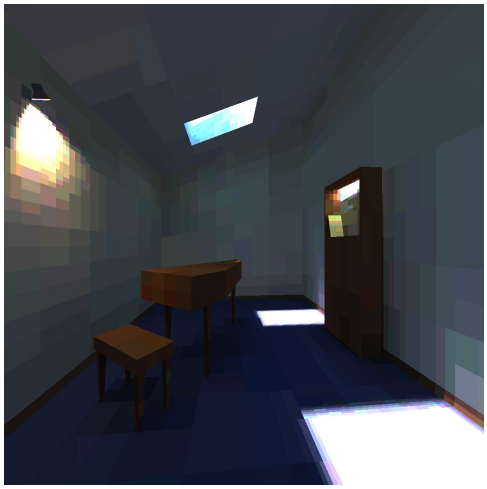
2 minute run on 1 processor.



2 minute run on 2 processors.



2 minute run on 4 processors.



2 minute run on 8 processors.

Figure 5.16 Visual Speedup

memory contention is minimal. The distributed memory parallel algorithm is scalable. Finally, two new methods of visualizing performance indicate several dimensions of parallelism, throughput, and answer quality. We believe these are very useful representations of the advantages and disadvantages of parallelism.

6 CONCLUSION AND FUTURE WORK

We have presented a parallel implementation of a global illumination algorithm. The speedup graphs show that it is scalable. Photon solves the Rendering Equation by determining the radiance at each point, and it does this without the need for storing huge ray history files. Photon is written using MPI, and can therefore be run on a variety of supercomputers as well as inexpensive clusters of workstations.

Photo-realism

The question may be asked, "Does Photon solve the Rendering Equation?" Since Photon is based on quantum light transport simulation from the light source, it has the potential to account for all lighting effects. At this time polarization is being added, and we foresee the ability to add fluorescence. It is our belief that polarization will play a large role in the realism of a rendered scene.

Only those algorithms that account for *all* lighting effects can truly claim to solve the Rendering Equation. Photon correctly solves for the radiance for each discrete area and direction. As the discrete areas and angle ranges shrink, Photon converges to a solution for the radiance at every point in a scene, and therefore will converge to a solution to the Rendering Equation.

Massive Parallelism

Currently, the octree representation of the geometry is replicated on all nodes. This could limit the size of the input geometry. Distribution of the geometry would allow computation of a global illumination solution for very complex scenes. It would also lend itself to more massive parallelism.

The choice of an octree representation of the geometry is especially beneficial for distribution. The octree data structure orders the intersection testing for a given photon such that we only test polygons in the space the photon is traveling through. When an intersection is detected, it is the closest intersection and further testing is not needed. In a distributed environment, a photon is then only passed to those processors that are responsible for the space the photon is traveling through. The photons can then be queued and sent in a batch to the appropriate processors, thus reducing communication overhead. A bounding box data structure would require all processors to calculate intersection points, and then the closest intersection must be determined. The result is a global reduction operation for each photon, which is far too expensive.

Photon is a new and unique approach to computer graphics. If computers continue to follow Moore's Law, many algorithms that seem barely viable today will become routine for computers of the future. We expect that the approach to rendering described herein will be used more widely as memory sizes grow and computer performance increases.

APPENDIX

BIBLIOGRAPHY

[1] S. Aluru, J. Gustafson, and G. Prabhu, "A Random Number Generator for Parallel Computers," *Parallel Computing*, vol. 18, 1992, pages 839-847.

[2] A. Appel, "An Efficient Program for Many-Body Simulation," *SIAM Journal on Scientific and Statistical Computing*, vol. 6, no. 1, 1985, pages 85-103.

[3] J. Arvo and D. Kirk, "Particle Transport and Image Synthesis," *SIGGRAPH '90 Proceedings*, vol. 24, no. 4, August 1990, pages 63-66.

[4] L. Aupperle, *Hierarchical Algorithms for Illumination*, Ph.D. thesis, Princeton University, Department of Computer Science, Princeton, New Jersey, November 1993.

[5] D. Bailey *et al.*, "The NAS Parallel Benchmarks," *Report RNR-91-002*, NASA Ames Research Center, January 1991.

[6] P. Bjorstad and E. Boman, "A New Algorithm for the SLALOM Benchmark," Technical Report No. 55, Department of Informatics, University of Bergen, Norway, May 1991.

[7] M. Carter, *Parallel Hierarchical Radiosity Rendering*, Ph.D. thesis, Iowa State University, Department of Electrical and Computer Engineering, Ames, Iowa, 1993.

[8] M. Cohen and D. Greenberg, "The Hemi-Cube: A Radiosity Solution for Complex Environments," *SIGGRAPH '85 Proceedings*, vol. 19, no. 3, Aug. 1985, pages 31-40.

[9] R. Cook, "Stochastic Sampling in Computer Graphics," *ACM Transactions on Graphics*, vol. 5, no. 1, Jan. 1986, pages 51-72.

[10] R. Cook, T. Porter, and L. Carpenter, "Distributed Ray Tracing," *Computer Graphics*, vol. 18, no. 3, 1984, pages 137-145.

[11] R. Cook, and K. Torrance, "A Reflectance Model for Computer Graphics," *ACM Transactions on Graphics*, vol. 1, 1982, pages 7-24.

[12] I. Deak, "Uniform Random Number Generators for Parallel Computers," *Parallel Computing*, vol. 15, 1990, pages 155-164.

[13] W. Feller, *An Introduction to Probability Theory and Its Applications*, John Wiley and Sons Inc., New York, New York, 1968.

[14] S. Foo and K. Torrance, "Equipment Acquisition for the Light Measurement Laboratory of the Cornell Program of Computer Graphics," *Cornell University Program of Computer Graphics Technical Report*, 1995.

[15] M. Garey and D. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, New York, 1979.

[16] A. Glassner, *An Introduction to Ray Tracing*, Academic Press Limited, 1989.

[17] A. Glassner, "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graphics and Applications*, vol. 4, no. 10, Oct. 1984, pages 15-24.

[18] J. Gustafson and Q. Snell, "Reevaluating Global Illumination Methods," *Ames Laboratory Technical Report, USDOE* (in preparation).

[19] J. Gustafson, "The Consequences of Fixed Time Performance Measurement," *Proceedings of the 25th Hawaii International Conference on System Sciences*, IEEE Computer Society Press, January 1992, pages 113-124.

[20] J. Gustafson *et al.*, "A Radar Simulation Program for a 1024-Processor Hypercube," *Proceedings of Supercomputing '89*, pages 96-105.

[21] J. Gustafson, "Reevaluating Amdahl's Law," *Communications of the ACM*, vol. 32, no. 5, May 1988, pages 532-533.

[22] J. Hammersley and D. Handscomb, *Monte Carlo Methods*, Wiley, New York, NY, 1964.

[23] P. Hanrahan, D. Salzman, and L. Aupperle, "A Rapid Hierarchical Radiosity Algorithm," *Computer Graphics*, vol. 25, no. 4, 1991, pages 197-206.

[24] X. He *et al.*, "A Comprehensive Physical Model for Light Reflection," *SIGGRAPH '91 Proceedings*, vol. 25, no. 4, July 1991, pages 175-186.

[25] P. Heckbert and P. Hanrahan, "Beam Tracing Polygonal Objects," *Computer Graphics*, vol. 18, no. 3, 1984, pages 119-127.

[26] A. Heirich and J. Arvo, "Scalable Photorealistic Rendering of Complex Scenes," *First Eurographics Workshop on Parallel Graphics and Visualization*, Bristol, U.K., September 1996.

[27] D. Immel, M. Cohen, and D. Greenberg, "A Radiosity Method for Non-Diffuse Environments," *SIGGRAPH '86 Proceedings*, vol. 20, no. 4, August 1986, pages 133-142.

[28] E. Isaacson and H. Keller, *Analysis of Numerical Methods*, John Wiley and Sons, Inc., New York, New York, 1966.

[29] J. Kajiya, "The Rendering Equation," *SIGGRAPH '86 Proceedings*, vol. 20, no. 4, August 1986, pages 143-149.

[30] D. Knuth, *Seminumerical Algorithms*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1981.

[31] I. Lux and L. Koblinger, *Monte Carlo Particle Transport Methods: Neutron and Photon Calculations*, CRC Press, Boca Raton, Florida, 1991.

[32] M. Maekawa, A. Oldehoeft, and R. Oldehoeft, *Operating Systems, Advanced Concepts*, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA, 1987.

[33] N. Magnenat-Thalmann and D. Thalmann, *Image Synthesis Theory and Practice*, Springer-Verlag, Tokyo, 1987.

[34] Message Passing Interface Forum, "MPI: A Message Passing Interface Standard," *International Journal of Supercomputer Applications*, vol. 8, no. 3/4, 1994.

[35] A. Mood, F. Graybill, and D. Boes, *Introduction to the Theory of Statistics*, McGraw-Hill, New York, New York, 1974.

[36] S. Pattaniak and S. Madur, "Computation of Global Illumination by Monte Carlo Simulation of the Particle Model of Light," *Third Eurographics Workshop on Rendering*, May 1992, pages 71-83.

[37] B. Phong, "Illumination for Computer Generated Pictures," *Communications of the ACM*, vol. 18, no. 6, June 1975, pages 311-317.

[38] S. Rubin and T. Whitted, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *SIGGRAPH '80 Proceedings*, vol. 14, no. 3, July 1980, pages 110-116.

[39] P. Schroder and P. Hanrahan, "On the Form Factor between Two Polygons," *SIGGRAPH '93 Proceedings*, August 1993, pages 163-164.

[40] P. Shirley *et al.*, "Global Illumination via Density Estimation," *Proceedings of the Sixth Eurographics Workshop on Rendering*, June 1995, pages 187-199.

[41] P. Shirley, "Monte Carlo Simulation and Integration," *Global Illumination*, ACM SIGGRAPH '92 Course Notes.

[42] P. Shirley, *Physically Based Lighting Calculations for Computer Graphics*, Ph.D. thesis, University of Illinois at Urbana-Champaign, Department of Computer Science, Urbana, Illinois, 1991.

[43] Y. Shreider, *The Monte Carlo Method*, Pergamon Press, New York, NY, 1966.

[44] F. Sillion, *et al.*, "A Global Illumination Solution for General Reflectance Distributions," *SIGGRAPH '91 Proceedings*, vol. 25, no. 4, July 1991, pages 187-196.

[45] F. Sillion and C. Puech, *Radiosity and Global Illumination*, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1994.

[46] F. Sillion and C. Puech, "A General Two-Pass Method Integrating Specular and Diffuse Reflection," *Computer Graphics*, vol. 23, no. 3, 1989, pages 335-344.

[47] M. Singhal and N. Shivaratri, *Advanced Concepts in Operating Systems*, McGraw-Hill, Inc., New York, NY, 1994.

[48] B. Smits, J. Arvo, and D. Salesin, "An Importance Driven Radiosity Algorithm," *SIGGRAPH '92 Proceedings*, vol. 26, no. 2, July 1992, pages 273-282.

[49] I. Sobol, *A Primer for the Monte Carlo Method*, CRC Press, Boca Raton, Florida, 1994.

[50] E. Sparrow and R. Cess, *Radiation Heat Transfer*, Hemisphere Publishing Corporation, Washington, 1978.

[51] W. Stürzlinger, G. Schaufler, and J. Volkert, "Load Balancing for a Parallel Radiosity Algorithm," *Proceedings of the 1995 Parallel Rendering Symposium*, ACM SIGGRAPH, pages 39-45.

[52] E. Veach and L. Guibas, "Optimally Combining Sampling Techniques for Monte Carlo Rendering," *Proceedings of SIGGRAPH '95*, ACM SIGGRAPH, pages 419-428.

[53] J. Wallace, M. Cohen, D. Greenberg, "A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray-Tracing and Radiosity Methods," *SIGGRAPH '87 Proceedings*, vol. 21, no. 4, 1987, pages 311-320.

[54] G. Ward, "Measuring and Modeling Anisotropic Reflection," *SIGGRAPH '92 Proceedings*, vol. 26, no. 2, July 1992, pages 265-272.

[55] T. Whitted "An Improved Illumination Model for Shaded Display," *Communications of the ACM*, vol. 23, no. 6, June 1980, pages 343-349.

[56] S. Yakowitz, *Computational Probability and Simulation*, Addison-Wesley, New York, NY, 1977.

[57] D. Zareski *et al.*, "Efficient Parallel Global Illumination using Density Estimation," *Proceedings of the 1995 Parallel Rendering Symposium*, ACM SIGGRAPH, pages 47-54.