RAY TRACING TECHNIQUES FOR INDIRECT REFRACTED ILLUMINATION

by

Henry Albert Bailey
B.C.S., Acadia University, 1997

Thesis
submitted in partial fulfillment of the requirements for
the Degree of Master of Science (Computer Science)

Acadia University
Spring Convocation 1999

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

ABSTRACT

Conventional backward ray tracing can generate beautiful images. However, in scenes where a light indirectly illuminates an object, such as by bouncing off or passing through another object, this method fails. Lack of indirect illumination contributions make an image look unrealistic, such as dark shadows beneath a transparent object rather than lighting patterns or caustics.

There has been much research in improving ray tracing to render indirect light contributions, although much has dealt with reflected light rather than refracted light, or developing methods that restrict the types of objects that can be traced. Some of the techniques that work with refraction produce more realistic images than other methods, and many require enormous amounts of time and resources to render.

Methods for rendering indirect refracted light contributions are examined and implemented for this thesis. Realism, rendering time, complexity, and restrictions are compared to propose a fast, robust method for rendering realistic scenes containing refractive objects.

ACKNOWLEDGMENTS

I would like to thank Dr Rick Giles for his guidance and advice. I would also like to thank Dr Ivan Tomek and Karine Blouin for their encouragement.

Chapter 1

INTRODUCTION

"There is nothing ugly; I never saw an ugly thing in my life: for let the form of an object be what it may, - light, shade, and perspective will always make it beautiful." John Constable, 1776-1837.

Ray Tracing has been used to create some of the most realistic and beautiful images ever generated by a computer. Scenes in movies such as "Toy Story", "A Bug's Life", and "Antz" are produced using ray tracing, as are images in other movies, commercials, and advertisements. Ray tracing is not only a powerful method for rendering realistic images, but can also be applied to any type of geometrical object. Ray tracing is relatively simple to understand and implement. It is also simply modularized, making it relatively simple to expand and to use for multi-threaded processing.

1.1 Preface

Modeling light interactions in a scene is a means of creating realistic computer images. Tracking light as it propagates throughout a scene, however is simply too computationally intensive and inefficient to be used to any extent. The idea of tracing the light backward from the eye, also called ray casting, was developed by Appel [1] and was the first step toward modern day ray tracing. Ray casting was refined later to recursively include reflection and refraction by Whitted [17]. Other improvements such as specular highlights through the Phong model, Lambertian model diffuse lighting, focusing and distribution ray tracing [6], texture mapping, light mapping and bump mapping have been

added to ray tracing techniques. Although the recursive ray-tracing algorithm incorporated these improvements easily, it still could not account for illumination that arrived from other objects in the scene.

In 1985, Kajiya [14] introduced his rendering equation that summarizes indirect illumination into an ambient term and, in 1986, Arvo [2] introduced a bi-directional technique to incorporate indirect illumination into backward ray tracing. More recently, Jensen [13] expanded on Arvo's ideas to propose a photon map that stores not only intensity values for the photons, but also the incoming direction (incidence).

Jensen uses a single global photon map that is traversed during the rendering step to find the closest illumination values. Other techniques that have been applied to indirect illumination are cone/beam/pencil tracing, the accumulation buffer, and radiosity. These other techniques impose restrictions on the types of objects that can be rendered as well as their surface characteristics. The infinitesimally thin rays used in ray tracing are less complicated to implement and understand; therefore, only techniques that utilize rays are examined and implemented.

1.2 Backward Ray Tracing

Backward ray tracing is based on how a camera or eye works. In the case of a camera, light travels through the camera lens and aperture, and its color and intensity are stored on the film. Light travels through the lens and pupil of the eye where the contributions are received by the rods and cones of the eye's retina. The ray-tracing algorithm follows the light backward, from where it entered the lens, to determine how it

made its contributions to the image. Camera tricks such as zooming or focusing can all be performed by manipulating these backward "camera" or "eye" rays.

Each pixel on the screen represents a different ray or combination of rays that can be traced in reverse throughout the scene. The closest object hit by the eye ray is the last object seen before that light strikes the eye. Since only the objects facing toward the ray can be struck, this demonstrates how ray tracing incorporates hidden surface removal and automatic depth sorting as part of its algorithm. Additional light rays are fired from each light source to each intersection point to determine direct light (specular and/or diffuse) contributions at those points, which incorporates shading into the algorithm.

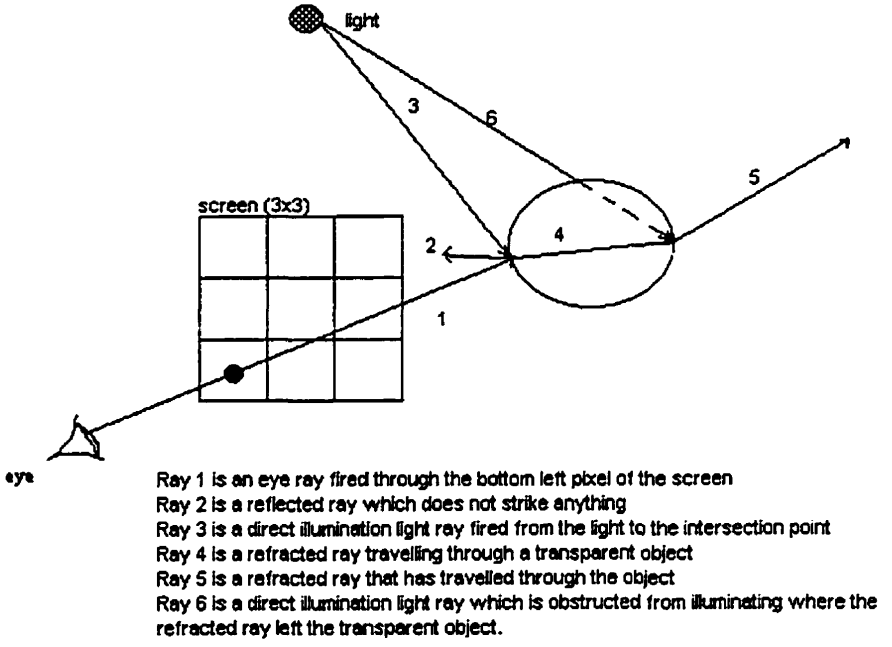


Figure 1 Backward Ray Tracing

Recursive ray tracing tracks the eye rays as they reflect off objects or pass (refract) through them. This permits transparent and reflective objects to be rendered realistically.

Figure 1 shows recursive backward ray tracing. In the figure, a single eye ray has been cast through the lower left pixel of the image. The eye ray intersects a sphere that is reflective and refractive. The figure illustrates the large number of rays that are traced for a single eye ray.

Illumination that reflects or refracts off other objects before striking the intersection point (indirect illumination) is not a direct part of backward ray tracing. Ray 6 in Figure 1 illustrates the problems of indirect illumination encountered by conventional backward ray tracing. Indirect illumination is not calculated but is usually summarized into an ambient value, which is included into the rendering equation. This lack of calculation prevents occurrences such as caustics from refractive objects, and the blending of colors from reflective objects onto one another. Backward ray tracing is view dependent, which greatly reduces the amount of computation, while tracing light as it travels forward from the light sources as it naturally propagates, is view independent and allows for these optical phenomena.

1.2.1 Backward Ray Tracing Pseudo-code

The basic algorithm (in pseudo code) for backward ray tracing is listed below. It consists of a ray casting part where pixels are cast out into the scene and a tracing part where the values for the pixels are calculated.

```
//ray casting section
For each screen pixel {
-Generate one or more rays from the camera to that pixel using the camera information
-Trace the ray(s) and store the returned color (a weighted average) into the final image
}
```

// ray tracing section
Color trace(Ray) {

-Get the first intersection point for the ray. If none, return a background value, otherwise continue
-With the intersection point get the ambient color for the intersection point
-For each light in the scene, determine if an obstructed ray can be fired from that light to the intersection point and if so add the diffuse and specular contribution
-If the intersection point is reflective trace a new reflected ray and add its (scaled) contribution
-If the intersection point is transparent trace a new refracted ray and add its (scaled) contribution
-Return the color (the result of the rendering equation)
}

Many enhancements and improvements can easily be incorporated into backward ray tracing. The casting section can be modified to counter aliasing problems or direct where rays should be fired. The tracing algorithm can be modified to support different types of rendering equations. Ray tracing is also very object oriented in structure so it is very scalable in terms of what objects or techniques can be rendered.

1.3 The Rendering Equation

The value for each pixel is calculated based on the rendering equation used. The rendering equation implemented is modeled after Kajiya's [14] equation:

$$I(p,v,n) = A(p) + \sum (D(p,n) + S(p,v,n)) + R(v') + T(v'')$$

where:
p is the intersection point
v is the ray direction vector
n is the normal at the intersection point.
I(p,v,n) is total illumination
A(p) is the ambient light contribution at a particular point.
D(p,n) is the diffuse contribution for a light source
S(p,v,n) is the specular contribution for a light source
R(v') is the value from tracing a reflected ray, scaled by the reflection co-efficient
T(v'') is the value from tracing a refracted ray, scaled by the transparency co-efficient

The diffuse and specular terms provide the direct illumination contributions to the equation. The reflected and refracted illumination contributions allow for specular contributions from other intersection objects in the scene. The ambient term is used to approximate the amount of diffuse reflection and transmission, meaning light that is reflected or passed through other objects in the scene, or indirect illumination. Techniques for accounting for indirect illumination are examined and implemented.

1.4 Objectives of this Thesis

Indirect illumination is difficult to accurately calculate in backward ray tracing. Illumination is generally calculated as part of shading and shadow ray calculations in ray tracing. Techniques that integrate into the backward ray-tracing algorithm are implemented and are compared for realism, computational cost, complexity, scalability, and restrictions. The main objective of this thesis is to determine which methods are best suited for indirect refracted light, so that a new and more efficient indirect illumination technique can be developed.

Three main regions of interest are examined and implemented, as shown in Table 1 shading models, pre-processors, and indirect illumination (ambient) maps.

Table 1 Indirect Illumination Techniques Assumptions, Comparisons, and Implementations

Method	Assumptions	Comparisons	Implementation
Non-transparent shading model	This method will not render any indirect illumination. The ambient values retrieved by this technique will provide indirect illumination.	View-dependent costs.	This method terminates shadow ray intersection testing when the shadow ray strikes any position other than the point of interest.
Non-refractive shading model	This method will render indirect illumination accurately from transparent objects that do not refract light. It will not render caustics. Variations of this technique can render attenuation of light through transparent objects.	View-dependent costs. Difference in cost (intersection tests) between this method and the non-transparent shading method.	This method terminates shadow ray intersection testing when the shadow ray strikes any opaque point other than the point of interest. Any transparent intersection points are used to create a filter between the light and the point of interest.
Individual ambient maps	When correctly populated these maps will allow indirect illumination to be rendered correctly. Resolution problems may occur.	Population costs. Retrieval costs are the same as inverse mapping so it is not analyzed.	This method stores an individual ambient map with each object. The value in the ambient map indicates the amount of indirect illumination at a particular position. Populating the map requires that over-sampling be restricted, by means of an "initial hit" map.
Global photon map	When correctly populated these maps will allow indirect illumination to be rendered correctly.	Population costs. Retrieval costs.	The value in the global map indicates the amount of indirect illumination at a particular three-dimensional position. Values must be retrieved based on a volume of interest and averaged based on the expected number of photons.

Method	Assumptions	Comparisons	Implementation
Completely Random Pre-processor	This method should populate the maps uniformly.	Pre-processing costs. Number of rays cast versus number that intersect and the number stored.	A pseudo random number generator is used to fire random rays outward from each light source.
Monte-Carlo Random Pre-processor	This method should populate the maps but may have over-sampling problems. It should be more efficient than the completely random method.	Pre-processing costs. Number of rays cast versus number that intersect and the number stored.	This is an extension to the completely random pre-processor. When a value is stored, additional rays are fired at random around the area where the ray was initially cast.
Brute Force Spherical Interpolation Pre-processor	This method should populate the maps uniformly.	Pre-processing costs. Number of rays cast versus number that intersect and the number stored.	Rays are fired evenly across the surrounding sphere (cube) of each light source.
Adaptive Spherical Interpolation Pre-processor	This method should populate the maps uniformly and at higher resolution than the brute-force method.	Pre-processing costs. Number of rays cast versus number that intersect and the number stored.	Rays are fired evenly across the surrounding sphere (cube) of each light source. When a ray is stored, the region between the ray and its neighbors is sub-divided and additional rays are cast. The sub-division is recursive.
Planar Interpolation	This method is a specialized variation of the interpolation methods that interpolates only across one plane the bounding box for the scene.	Pre-processing costs. Number of rays cast versus number that intersect and the number stored.	Rays are fired evenly across the plane from each light source. The number of hits is related to the efficiency of the bounding box (structure) for the scene.

Chapter 2

MATHEMATICS OF RAY TRACING

Before discussing the problems of correct rendering of indirect refracted illumination, it is important to understand some of the fundamentals of ray tracing. Ray tracing is very computationally intensive, with many floating point operations needed for the most simple of images. This chapter tries to explain some of the necessary mathematics behind ray tracing. Vector mathematics, the geometry of reflection and refraction, inverse mapping, and quadrics intersections are all discussed.

2.1 Vectors

Three-dimensional vectors and points are the core components of any ray tracer. A ray consists of a starting point and a direction vector. A vector can sometimes be represented as four components, the fourth being the length of the vector, $|V|$. Forcing the length of a vector to be equal to one is called normalizing the vector. This can be done by dividing each component in the vector by the vector's length. Normalized vectors reduce the amount of computation in some ray tracing equations and are therefore more efficient than arbitrary length vectors. For example, the intersection distance when using a normalized direction vector is equal to the absolute distance from the intersection point and therefore the closest intersection point when ray tracing.

Another benefit of normalized vectors is in calculating the dot product. The dot product of two normalized vectors equals the cosine of the angle between them. The dot

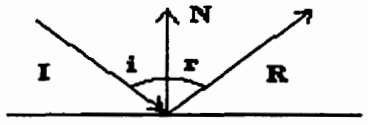
product is used by many ray-tracing equations for reflection, refraction and illumination calculations such as Phong specular and Lambertian diffuse.

The dot product of reflected light ray and the view direction is used to calculate the Phong illumination value. The Phong specular equation is equivalent to the dot product raised to a specular-reflection exponent. This causes a steeper exponential curve for higher powers and produces the specular highlights seen on a shiny surface. In most ray-tracers, the exponent is a surface property for an object. The implementation uses a different method by instead storing the exponent with the light source. The reason for this design change is that the specular exponent attempts to give finite size to an infinitely small light source, particularly in the case of a point light source. The light source should be constant between all objects in the scene, so the exponent simulates a finite size for the light source. Furthermore, the uses of bump maps and distributed reflection or refraction as surface parameters can be used to control the relative size of the highlights. The amount of specular contribution is based on the reflectivity of the surface. A non-reflective surface will have no specular contribution.

There are many different diffuse surface models. In this thesis, the Lambertian surface model has been implemented. Lambertian diffuse illumination is a model that evenly distributes light scattering as it strikes a surface. The amount of Lambertian diffuse illumination is the dot product of the incoming light ray and the surface normal, which makes it independent of the viewing angle.

2.2 Reflection

A ray striking a reflective object creates a specularly reflected ray. The angle of incidence from the incoming ray equals the angle of reflection of the outgoing ray. Figure 2 demonstrates reflection and Equation 1 explains how the reflected vector is calculated using normalized direction and surface normal vectors. Glassner [8] demonstrates how the specularly reflected vector computed in Equation 1 will be of unit length (normalized).



I is the incident vector
N is the surface normal
R is the reflected vector
i is the angle of incidence
r is the angle of reflectance

Figure 2 Reflection

$$R = I - 2(N \bullet I)N$$

Equation 1 Calculating the reflection vector

Perfect specular reflection makes some objects look unnatural and therefore unrealistic. Cook's [6] distributed ray produces gloss (blurry reflection) which better simulates how light actually reflects off certain real world surfaces. Although distributed reflected rays are not implemented, distributed camera rays, which produce depth of field (focusing) effects, have been implemented.

The amount of light contributed by a reflected ray is a function of the reflectivity of the surface. Some surfaces, such as a mirror may have a large amount of reflectivity, while others may absorb most of the light, giving a lower reflectivity value. In the programs for this thesis, reflectivity is implemented using a single value with a range of zero to one. The type of reflection for a particular surface is more realistically modeled with a separate value for each light wavelength sample. This is because a surface may reflect or absorb red light in a different way than blue light or any of the other colors in the spectrum.

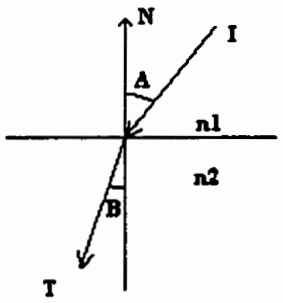
2.3 Transparency and Refraction

Ray tracing transparent objects is much more complicated than tracing reflective surfaces. Rays may travel through objects, leading to algorithmic changes such as negating the surface normal, or attenuating the intensity based on the distance traveled through a medium. The calculations needed to calculate transparency rays, however, are straightforward.

For a transparent object that has the same level of refraction as its surrounding medium, the transparent ray simply continues through the object until it intersects the other side or a nested object. Each transparent object has a transparency (or opacity) coefficient for each surface intersection point. This co-efficient is similar to the reflection coefficient, except it determines the scaling amount for the transparency ray. An object may have a different transparency coefficient for each wavelength sample, to account for how certain wavelengths of light may be blocked, while others are transmitted. Realistically, the sum of the reflection and transparency coefficients for a particular wavelength would never

exceed one, since that would cause the objects to gain too much illumination from subsequent recursive tracing.

In real life, transparent objects have refractive properties, so simply extending the incoming ray through the object is incorrect. Each (semi) transparent object has an index of refraction, which describes the speed of light for that medium compared to a vacuum. The change in the speed of light as it crosses from one medium causes the light to bend, giving what is known as refraction. Equation 2, Snell's Law, and Figure 3 demonstrate how the transmitted ray is bent based on the indexes of refraction of the two objects.



I is the incident vector
T is the transmitted vector
N is the surface normal
n1 is the index of refraction for the incident medium
n2 is the index of refraction for the transmitted medium
A is the incident angle
B is the transmitted angle

Figure 3 Refraction

$$\frac{\sin(A)}{\sin(B)} = \frac{n2}{n1}$$

Equation 2 Snell's Law

The formula for the calculation of a transmitted ray is shown in Equation 3.

$$T = \frac{n1}{n2} I - \left(\cos(B) + \frac{n1}{n2} \cos(A) \right) N$$

Equation 3 Refracted Ray Calculation

It is possible for the transmitted ray to be reflected rather than refracted, when the ray is transmitted from a dense medium to a less dense one. This phenomenon is called total internal reflection and results because the transmitted angle exceeds ninety degrees. The incident angle that results in a transmitted angle of exactly ninety degrees is called the critical angle.

A refractive object causes light to bend as light travels through it, making illumination contributions from lights that are obscured by transparent objects difficult to calculate. Several shading models and algorithms are examined and implemented to compare how they deal with this problem.

2.4 Inverse Mapping

Inverse mapping allows the storage of texture, light, bumps, reflection, transparency and other surface characteristics in a two-dimensional table (map). The intersection point and normal are used to calculate their corresponding u and v co-ordinates in the map. The inverse mapping structure can be a plane, cylinder, sphere, or any other object appropriate object. The plane works well for a polygon, and the sphere works well for most other structures. Spherical and planar inverse mappings are used throughout the implementation. Haines [9] explains spherical mapping in Equation 4 and planar mapping.

$$\Phi = \arccos(-Sn \bullet Sp)$$
$$v = \Phi / \pi$$
$$\theta = \frac{\arccos((Se \bullet Sn)/\sin(\Phi))}{2 * \pi}$$
$$if\,((Sp \otimes Se) \bullet Sn) > 0$$
$$then \ldots u = \theta$$
$$else \ldots u = 1 - \theta$$

Equation 4 Spherical Inverse Mapping

Sn is the surface normal

Se is the equator vector (right)

Sp is the polar vector (up)

u is the map x co-ordinate range of 0..1

v is the map y co-ordinate range of 0..1

Spherical inverse mapping is relatively simple to compute. However, objects that overlap, such as tori, may not uniquely inverse map using this method. Spherical inverse mapping also wastes storage when applied to certain objects such as single sided polygons, since many index values will never be used. Planar (convex quadrilateral) inverse mapping is a better method for polygons than spherical inverse mapping.

Infinite or unrestrained objects such as planes do not map efficiently with either method, as there are no maximum dimensions for these objects. Inverse mapping techniques are also subject to aliasing problems when the resolution is low. The use of inverse ambient mapping does however allow for preprocessing of the indirect illumination values, which provides an efficient and potentially accurate ambient term during the rendering of the image.

2.5 Quadric

A quadric is a second-degree equation whose ray intersection point can be solved using the quadratic function. Special emphasis is placed on the quadrics, since certain optimizations can be performed. Quadrics are commonly used in ray tracing and can be written in the following form:

$$ax^2 + 2bxy + 2cxz + 2dx + ey^2 + 2fyz + 2gy + hz^2 + 2iz + j = 0$$

Haines [9] mentions a number of ways quadrics can be optimized to reduce floating point calculations. Among his ideas are storage of scaled values (for example variable a and 2*a), and common factoring. Commonly used quadrics, such as ellipsoids, can be optimized by using separate, specialized intersection and normal calculations. One additional optimization that can be applied to these specialized quadrics is based on the knowledge that certain variables must never be equal to zero. This allows the entire equation to be reduced by that variable and have the variable replaced with the value of one. Any multiplication that involves that variable can be removed, saving a floating-point calculation. For example the ellipsoid must always have a value for the x, y and z radius, otherwise it is an ellipse rather than an ellipsoid. After substituting the ellipsoid function into the quadric formula, the variables [a, e, h] are never equal to zero, or they will violate the constraints on the object. This means that the quadric function can be reduced by any one of these variables, when the object is being created, making calculations more efficient.

Chapter 3

PROBLEMS WITH RAY TRACING

Developers and users of ray tracers face many problems such as demand on computation and memory, aliasing, and lack of indirect lighting. More efficient intersection algorithms have been developed to counter some of the computation costs, as well as bounding volumes, and other object space improvements. Procedural texture maps require less memory than conventional inverse maps. Aliasing can be countered by super sampling each pixel and averaging the result, at the expense of many more computations. Radiosity hybrid techniques have allowed indirect illumination to be more accurately incorporated into ray tracing.

Unfortunately, radiosity is even more computationally intensive than ray tracing, and requires the scene to be partitioned into patches. Much research has been done in the field of radiosity, but its techniques have not been implemented. It enforces restrictions that backward ray tracing is not subject to such as partitioning objects into patches. The forward tracing technique examined does however, use a technique similar to that required by radiosity.

3.1 Super Sampling and Path Tracing

Ray tracing often produces visual artifacts when the value for a pixel represents only a part of its area. Super sampling is performed to find an average value over the area of a pixel, using infinitely thin rays.

At any particular intersection point, both a reflective and refractive ray may be traced. This results in many recursive calls for the tracing of each initial eye ray. Kajiya [14] introduced path tracing, which probabilistically chooses to follow either a reflective or a refractive branch at each intersection point. This technique concentrates more on the differences between first level intersections rather than higher levels of recursion. Conventional ray tracing methods spend a large amount of computation on those higher levels whose values contribute much less to the equation. Path tracing, though efficient, is not implemented because it reduces the contributions from reflected and refracted sources.

The principle behind path tracing could, however be applied to refracted objects when forward tracing. This would eliminate computation associated with reflection, for the purposes of studying only indirect refracted light.

3.2 Shading Methods

Once an intersection point for an eye ray has been determined, a new ray is cast from each light source toward the intersection point. If a light intersects the object at the same location, the direct illumination values for the light are contributed for that point. If the light strikes an opaque object before reaching the intersection point, the light is blocked and therefore not contributed to the intersection point. Transparent and refractive surfaces create difficulties with this algorithm.

3.2.1 Ignoring Transmission Of Light

A common shading method is to ignore all transmitted light during direct lighting calculations. When direct illumination is calculated for a light ray, if the light strikes any object that is closer than the distance from the light source to the point of interest, it is

considered to be blocked. This method treats opaque, transparent objects the same, and allows the shading algorithm to immediately stop performing intersection tests between the light and objects if the light ray is blocked by anything. The order of object intersection tests can be modified so objects that are more likely to block are tested first, thus making the algorithm even more efficient. A relatively straightforward and efficient speedup for this algorithm comes from the observation that when light is blocked for a particular intersection point, points nearby will probably also be blocked by that object. The blocking object can therefore be stored within the shading algorithm to be immediately tested or updated during all shading calculations. This speedup applies best to the intersection points for the initial eye rays.

Ignoring all transmission of light, by itself, is the most efficient and least realistic shading method examined and implemented. It correctly calculates direct lighting contributions, but relies solely on the ambient value for indirect illumination. Many ray tracers implement this shading method because it is so efficient and because transparent objects and indirect illumination may not be required for the rendered image.

3.2.2 Ignoring Refraction of Transmitted Light

Another way in which indirect illumination can be calculated during backward ray tracing is to allow transmission of light, but ignore how it refracts (bends) as it travels from the light source to the point of interest. Completely opaque objects will still block the light source, but transparent objects will filter the light before it reaches the point of interest. A problem with this method is that light generally diminishes as it travels through a partially

transparent object. The further the distance through the object, the less it contributes, because light is absorbed or dispersed by the object.

A simple extension to this technique is to allow for attenuation of the light based on the type of medium and distance light travels through it. Many light sources are implemented using attenuation and the additional transparency based attenuation can easily be incorporated.

Ignoring refraction of light allows objects that are partially blocked by transparent objects to still be illuminated. Although this method is more realistic than ignoring all transparency, it is only accurate for scenes that contain objects that do not refract light. It does not allow for effects such as caustics. It is less efficient than ignoring all transparency, whose optimizations cannot be uniformly applied to this method. Object intersection implementations must also be modified so that all intersection points, their transparency values, and/or attenuation coefficients are returned for each light ray. Another implementation problem is that objects that partially block a light for a point of interest should be calculated based on the order in which the light travels through them. This further reduces the efficiency of this technique. This technique and its variations, such as ignoring attenuation or ordering of intersected objects, can be applied to still produce a realistic and relatively efficient shading model for allowing indirect transmitted light. The non-refractive shading model does not, however model indirect refracted illumination.

3.2.3 Other Shading techniques

Other techniques can be used to render caustics, but have not been examined and implemented. Firing multiple shadow rays from the light source around the intersection

point in a Monte-Carlo fashion is called bi-directional ray tracing. The rays that reach the intersection point contribute indirect illumination. This technique is very expensive but can however produce realistic images, and is similar to a forward light tracing pre-processor method(s) implemented, except that it does not use ambient maps.

The cost associated with bi-directional techniques is incurred at render time and is view dependent. In addition to the high rendering cost, implementation of a bi-directional ray-tracer is complicated and restricts indirect illumination sources to the region surrounding the intersection point.

Another technique, implemented in other ray tracers such as POV Ray is based on the angle between the light ray and the surface normal. When the angle between these vectors is small, the light is not bent very much and the surface blocked by the transparent object receives the light. As the angle increases, the contribution decreases. This technique restricts the caustics to the area shadowed by the transparent object.

Caustics may occur outside a shadow area, so this technique is not always accurate. Furthermore, caustics are created by light rays grouping together, rather than light rays which are close to the surface normal, so this technique is not realistic.

3.3 Ambient Map

The shading model that treats all objects as opaque is very efficient and can be used in association with the ambient term for an object to produce realistic images that include indirect refracted illumination. A constant ambient value may be fine for objects that do

not receive any refracted illumination, but generally, the indirect illumination for an object varies across its surface. An ambient map allows this deviation to be represented.

The values for an ambient map can sometimes be specified without requiring the scene to be pre-processed. Objects with luminous areas, such as algae on a wall, can have their ambient maps specified accurately when the scene is being created. Indirect illumination from light striking other bodies in a scene, however, is dependant on where the other lights and objects in the scene are located. An estimated ambient map is only as accurate as its specification by the scene creator. A scene dependant calculation of the ambient map is the only way to ensure it is accurate.

Chapter 4

FORWARD RAY TRACING

Backward ray tracing calculates direct lighting contributions accurately and efficiently. Forward light tracing calculates all light contributions, direct and indirect, accurately but inefficiently. Bi-directional ray tracing attempts to confine forward light tracing to the regions of interest calculated during the backward tracing phase. The pre-processors implemented are part of a two pass rendering scheme. The light sources are forward traced in a pre-processing phase, and their indirect illumination is stored as ambient light with each object. The pre-processing phase is view independent, while the rendering phase uses backward ray tracing to calculate direct lighting and retrieves the indirect illumination based on the ambient value.

4.1 Pre-Processing

The forward tracing pre-processor phase consists of tracing a large number of rays from each light source and storing their values with objects that they intersect. Since only indirect illumination is being stored, the primary intersection for each ray does not need to be stored, only its reflected or refracted rays. Distribution ray tracing can easily be incorporated into the forward ray-tracing phase. The diffuse value for the intersection is stored in the ambient map. As mentioned in a previous chapter, path tracing rather than ray tracing could be used to calculate only reflected or refracted indirect illumination contributions, or to make the forward tracing phase more efficient. Another useful side-effect of the forward ray-tracing pre-processing phase, is that information about direct

lighting can be obtained and used to reduce the number of direct (shadow) rays for backward ray-tracing.

4.2 Pre-processing Using Random Rays

The light rays may be selected at random or through a selective process. One technique implemented uses a large number of random rays fired from each light source. This completely random technique is inefficient and results in many light rays not striking any objects. A slightly more efficient extension to the completely random technique is to fire additional random (Monte Carlo) rays around the area where the initial light rays are stored, to further saturate areas where the rays will most likely be stored. Although the rays are generated by a uniformly distributed random number generator, there is still the problem of resolution.

4.3 Indirect Light Density and Resolution

Backward ray tracing eliminates light resolution problems since each intersection point receives the equivalent of one light ray per light source. Distributed (area) light sources use multiple rays, but their overall intensity is the equivalent of a single light source. Forward light tracing for use in ambient maps creates the problem of correct density and resolution of the contributions. Refraction and reflection from other objects adds complexity that a given location may have more than one contribution from the same light source. Radiosity corrects for this resolution problem by tessellating objects into patches and retrieving indirect illumination based on intersection with a patch. Tessellation causes much additional overhead and computation, in addition to storage concerns. The resolution problem now becomes related to the patch sizes.

Another attempt at indirect illumination is Mitchell [15], who developed a system for indirect reflection contributions from curved reflectors but not refractive objects. The accumulation buffer [16] tessellates surfaces, projects their corners onto other surfaces, and calculates the illumination based on the change in surface area. As with all tessellation techniques, the objects must be simplified, the polygons only approximate the original object, the amount of computation increases and surface characteristics such as bump mapping become restrictive or difficult. Resolution problems plague the pre-processor implementations, while the shading models discussed earlier are free of resolution problems.

4.4 Individual Ambient Maps

Heckbert [12] developed a method in which objects are pre-processed through forward ray tracing and individual ambient maps are stored with each object. The inverse mapping method used restricts which maps can be applied to which objects. A planar or spherical map works with almost any (convex) object. The difficulty with this technique is in choosing the correct map resolution. Each object must also provide an ambient map as part of its implementation, which must move and rotate as the object changes. This technique quickly overruns memory resources for large numbers of objects.

Individual ambient maps also have the problem of resolution. Infinite objects such as a plane cannot be represented through this system. The same principle applies to the shape of the object or its location relative to the camera. The forward (pre-processing) phase is view independent, so object resolution and determining which objects need maps may require an additional view-dependant pre-pre-processing stage.

The implementation of individual ambient maps uses a number of techniques that sabotage the one-to-n relationship seen in direct light contributions. The implementation for this thesis gives all objects equal resolution. It attempts to restrict bunching of non-refracted light rays by only allowing one reflected and/or refracted secondary ray for each intersection area. This is done by flagging a second map with each object when it is struck by a primary light ray, so that two almost equal primary rays cannot both contribute to the ambient maps. This technique can easily be extended to allow "n" rays per area and averaging their contribution. This implementation also averages or blurs the maps so that under-sampling errors are reduced and there are not large differences between adjoining regions of the map.

This implementation can produce realistic images of scenes containing refractive objects. As it is currently implemented, it has many problems, namely resolution, accuracy, under-sampling, memory constraints, inappropriate inverse map types, and aliasing. When adequately sampled and blurred, this implementation does render indirect illumination such as caustics to produce realistic images. The images produced are more realistic than those rendered using previously implemented backward ray tracing shading methods.

4.5 Photon Map

Procedural texture maps require less memory and produce fewer aliasing problems than conventional texture maps. Similarly, photon maps provide the same advantages over conventional ambient maps. The photon map, as developed by Jensen [13], maps indirect illumination into a structure, which is traversed at render time to calculate the ambient

contribution for an intersection point. The structure used is a k-d tree, or k-dimensional tree. In this case, k is equal to three, corresponding to the x, y and z co-ordinates of the intersection point. Jensen uses the photon map as a tool for radiosity and as a replacement for direct lighting calculations. In his implementation, the incident light value is stored along with the light contribution. The implemented photon map does not use that approach. Only the contribution is stored, as direct light and recursive tracing are performed to calculate all direct illumination, specular reflection, and specular refraction contributions.

The implemented photon map suffers from resolution problems relating to improper sampling (too much or too little). The problems relating to sampling can be controlled through varying the region (volume) of interest and the expected number of photons per region. The volume and the expected number of photons are currently hard coded values separate from the number of photons in the map and their density, although greater accuracy should result in relating them.

The k-d tree results in an efficient retrieval of the indirect light contributions with little loss of accuracy during view dependant rendering. The cost of retrieval is based on the number of photons in the map, and the size of the volume of interest. This cost occurs for each intersection point where the rendering equation is calculated. Trimming of the k-d tree based on the volume boundary greatly reduces the number of nodes that must be evaluated. Smaller boundary volumes result in more efficient trimming. The photon map tests in Chapter 5 compare the efficiency of the photon map.

A further improvement can be obtained from balancing the tree. This optimization is not implemented, as the tree is randomly populated and therefore relatively balanced. A balanced tree does, however reduce the number of node comparisons. Another potential optimization is to direct the creation of the photon map based on view dependence, to cause trimming higher in the tree and reduce the number of comparisons.

As mentioned above, the photon map is constructed from randomly created light rays. The pre-processing phase implemented is relatively inefficient in that many rays do not strike any objects. Adding control to the number and density of the pre-processing rays allows a more even distribution and reduces the problems that are apparent in the photon map and individual ambient maps.

4.6 Forward Ray Tracing Through Grid-like Interpolation

A second forward ray-tracing pre-processor has been implemented which interpolates the light rays evenly across a bounded plane, which is constructed based on the bounding box for the virtual scene. The size of the steps used to interpolate across the plane determines the size of the volume of interest used by the photon map. The volume used is slightly larger than the interpolation steps, so a value for the expected number of rays is hard coded into the photon map so that ambient values blend or blur more smoothly between adjoining regions.

The problem of samples bunching together (over-sampling) is controlled through the interpolation pre-processor, but the number of samples to use remains unclear. More samples mean a larger and less efficient map, but more accurately depict the indirect illumination. The number of steps is dependent on a parameter passed to the pre-processor,

but the size of the steps is calculated from the number of steps and the size of the bounding box. A larger box implies that the steps are further apart. The number and size of steps should be constant between all light sources, so this implementation may become less accurate when used by multiple light sources, or sources located within the bounding box.

Another problem with this method is that the bounding box spans all objects, but the pre-processor is only concerned with reflective or refractive objects. Clearly, a second bounding box based only on reflective and refractive objects is needed to reduce the number of unused rays. The implemented bounding box structure is oriented along the x, y and z-axis. A more efficient implementation would allow for an arbitrarily aligned box, or a collection of bounding slabs.

A further improvement in efficiency relates to the interpolation plane used to determine which forward light rays to fire. The structure currently implemented uses minimum and maximum u and v co-ordinates, but is semi-efficient in that it only interpolates along two dimensions rather than three. The inefficiency is a result of the interpolation plane boundary being a square rather than a convex polygon or a non-intersecting polygon.

The use of a boundary square means that the rays that are fired based on the boundary plane may not even strike the bounding box, let alone any reflective or refractive objects in the scene. This results in wasted light ray instantiations and more importantly, unnecessary intersection tests. Luckily, this design problem is confined to the pre-processing stage, and excess computations are incurred only during that phase, rather than repeatedly occurring during the backward rendering phase.

The bounding plane implementation is an effort to control the problem of firing an infinite number of rays outward from each light source. The rays are evenly distributed and fired at a finite resolution. The implementation using a single bounding plane has many restrictions that need to be eliminated to provide a robust, efficient and accurate pre-processing phase.

4.7 Spherical Interpolation Around Light Sources

The pre-processor that uses planar interpolation needs to be expanded in order to interpolate around a unit sphere surrounding the light source. One way in which this can be done is through interpolating around the sphere using trigonometric functions. The use of a unit sphere eliminates the need to normalize the light rays, as they will already be of unit length. The unit sphere may, however cause bunching together of rays and involves costly trigonometric calculations. The trigonometric functions can be stored in lookup tables at a large memory expense, depending on the number of interpolations. A different type of interpolation volume is implemented as part of this thesis to provide a robust pre-processor.

The interpolation structure implemented is a cube whose eight corners are located on the surface of the sphere. The six sides of the cube are interpolated along using a pre-determined sampling rate. The sampling rate can also be used to specify the resolution of the ambient maps or photon map retrieval parameters. Interpolation methods eliminate bunching (over-sampling) of light rays, but are susceptible to under-sampling. This interpolation method also results in a large number of rays whose values are not stored, and is therefore inefficient in its naïve (brute force) implementation.

4.7.1 Adaptive Interpolation

The efficiency of this method can be improved by adaptively (recursively) sampling at an increased resolution around the light rays that are stored. This adaptive modification allows a lower resolution to be used along the sides of the cube, and have the resolution recursively increased for regions where the rays are stored. An implementation that uses adaptive cube interpolation improves the efficiency of the brute-force cube implementation, prevents over-sampling and controls the amount of under-sampling. The implementation has a number of small problems relating to how it adaptively sub-divides the sampling regions, but these problems are due to implementation-specific decisions, rather than the adaptive sub-division technique.

4.8 Photon Map Problems

The Monte-Carlo pre-processor randomly populates the k-d-tree so balancing the tree is not crucial. The structured way that an interpolation pre-processor populates the k-d-tree means that the tree may be much less balanced than through random population and is therefore less efficient in its unbalanced form.

There are several other problems with the implementation of the photon map. The values stored in the photon map are single precision floating-point numbers for both intensity and position, yielding six single precision floating-point numbers per photon. This provides reasonable precision but double precision may be required for sampling areas that are extremely close together. The position values are compared based on the intersection position and a range value. Generally, this range value is only a few decimal places in size, therefore the extra precision gained by using a double is not needed.

Double precision numbers are used throughout the ray tracer code for the color (red, green, blue) values as well. These numbers are constrained to the range of zero to one, and are eventually converted to the integer range of 0-255, therefore single precision photon map colors do not create any problems.

The main problem with the current implementation of the photon map deals with the retrieval of photons using a volume of interest. All photons in a particular volume are retrieved and are averaged together based on the expected number of photons in that volume. The problem occurs when a photon that has been deposited in the map upon striking one surface is retrieved as part of the ambient value for another surface. This is noticeable in the scene where photons have been deposited from one side of a two-sided, non-transparent polygon, and retrieved at render time from the other side of the polygon.

The volume of interest encapsulates the area on both sides of the infinitely thin polygon, and allows the wrong photons to be retrieved. Jensen [13] stores the incident angle of the incoming photon. In the implementation for this thesis, that extra storage will not fix the problem. The intersection normal must be stored with the photon, and its dot product with the rendering ray calculated at render time. This adds extra computation and storage for a relatively minor problem and adds complications when the surface is transparent. A more convenient fix for this problem is to not allow opaque polygons to be two sided.

The complexity of populating a photon map (k-dimensional tree) is the same as for any binary tree. A binary tree with N nodes has height of at least $\log_2 N$ and therefore insertion of a new node requires at least $\log_2 N$ comparisons and has an upper bound

$O(N)$. The complexity associated with retrieving the ambient value from a photon map is more complicated. Multiple nodes are retrieved from the k-dimensional tree based on the volume of interest. In a situation where the volume of interest encapsulates the entire map, N values are retrieved requiring N node comparisons. The upper bound is therefore $O(N)$. The volume of interest implementation for photon retrieval makes the photon map very inefficient. The tests performed in Chapter 5 illustrate the costs of photon map retrieval. Implementing a more efficient retrieval method is an area for future work.

Chapter 5

TESTS AND COMPARISONS

5.1 Test Environment

Each image is rendered at 200 x 200 pixels, with one eye ray per pixel, for a total of 40000 eye rays per image. The tests have been performed under the conditions shown in Table 2.

Table 2 Test Environment

Operating System	Windows 95 (4.00.950)
CPU	Pentium II 350
RAM	64 MB
Java Virtual Machine	JDK 1.1.7 (with JIT)
Java Compiler	Visual Age for Java V2.0 (JDK 1.1 compatible)

The test environment will only affect timing results and these have not been presented since times will vary between different machines. The metrics presented in the following tables will be consistent under any capable test environment.

5.2 Explanation of Metrics

Each test scene has four tables of metrics associated with it. The first table displays the costs of rendering using two different shading models. The second table demonstrates the pre-processor costs of using individual ambient maps with each scene object. The third table demonstrates the pre-processor costs of using a global photon map for the ambient values of each scene object. The fourth table outlines the cost of using the global photon map at render time.

The first and fourth tables are view dependent, meaning that their values depend on the camera (eye) configuration and image resolution. The second and third tables are independent of the rendering phase. The values in the fourth table do, however depend on the size and organization of the photon map constructed in the pre-processing stage of the third table.

The number of each type of ray is the same for both shading models since those number are based on tracing eye rays rather than shadow rays. The difference in (object) intersection tests is of more importance since it highlights the view dependent costs of the different shading models. The pre-processor techniques use the "Non-Transparent" shading model at render time, so its values are the view dependent costs for direct lighting, and the lookup costs of the ambient map type is the indirect illumination cost.

5.3 Control Scene

The control scene for the renderings contains a transparent, but non-refractive object that lies between the light source and the other objects in the scene, demonstrating the need for indirect illumination. The scene has one light and three objects. The scene

contains a transparent plane with a number of waves stored through a bump map. A point light source is located far above the transparent plane. Beneath the transparent plane is a large green plane with a red sphere on it.

The "Non-Transparent" shading model creates a large dark area where the transparent object blocks the light (cross-reference). The "Non-Refractive" shading model allows light to be filtered and transmitted in a non-refracted path and therefore renders this scene accurately (cross-reference). The "Non-Refractive" shading model provides a benchmark to compare the accuracy of the pre-processors. The control scene demonstrates the usefulness of the "Non-Refractive" shading model in specialized cases and exposes the problems encountered by the (in this case unnecessary) bi-directional techniques.

Table 3 Control Scene Shading Costs

Method	Number of Intersection Tests	Number of Shadow Rays	Number of Reflection Rays	Number of Refraction Rays
Non-Transparent Shading Model (Figure 7)	201,996	21,696	592	5,180
Non-Refractive Shading Model (Figure 8)	201,996	21,696	592	5,180

Table 3 has identical values for both tests. This is due partly to the simplicity of the scene (very few objects) and the order in which the shadow ray intersections are evaluated. Additional shadow optimizations that are not implemented, such as evaluating the last

shadow intersection object first would better illustrate the additional costs of the "Non-Refractive" shading model.

The five pre-processors tested in Table 4 and Table 5 render the control scene as close as possible to the "Non-Refractive" shading model rendering. Not all pre-processors were capable of rendering a realistic image using finite resources. An upper limit of approximately 1,000,000 light rays was used for these pre-processors.

Table 4 Pre-Processing the Control Scene Using Individual Ambient Maps

Method	Number of Intersection Tests	Object Intersections	Light Values Stored	Number of Recursively Traced Rays	Percentage (%) of Rays That Intersect An Object	Percentage (%) of Rays That Are Stored
Completely Random (Figure 9)	3,000,741	1,120	247	1,000,247	0.11	0.02
Monte-Carlo Random (Figure 11)	727,836	42,022	11,612	242,612	17.32	4.79
Brute Force Planar Interpolation (Figure 13)	583,593	95,896	17,290	194,531	49.30	8.89
Naïve / Brute Force Spherical Interpolation (Figure 15)	3,597,354	1,276	264	1,199,118	0.11	0.02
Adaptive Spherical Interpolation (Figure 17)	1,765,410	107,625	19,551	588,470	18.29	3.32

The completely random and naïve, brute force spherical interpolation pre-processors were unable to produce an accurate image. In order to be more accurate with these techniques more rays must be cast. The intersection and storage percentages indicate that neither method is a reasonable pre-processor.

The Monte-Carlo, planar interpolation, and adaptive spherical interpolation techniques all produce an accurate control scene image. The planar interpolation technique appears to be the most efficient pre-processor for the control scene. Table 5 illustrates the costs of pre-processing the control scene using a global photon map.

Table 5 Pre-Processing the Control Scene Using a Global Photon Map

Method	Number of Intersection Tests	Object Intersections	Photons Stored	Number of Recursively Traced Rays	Percentage of Rays That Intersect An Object	Percentage of Rays That Are Stored
Completely Random (Figure 10)	3,000,744	1,121	248	1,000,248	0.11	0.02
Monte-Carlo Random (Figure 12)	853,152	83,387	31,384	284,384	29.32	11.04
Brute Force Planar Interpolation (Figure 14)	598,503	100,866	22,260	199,501	50.56	11.16
Naïve / Brute Force Spherical Interpolation (Figure 16)	3,597,354	1,276	264	1,199,118	0.11	0.02
Adaptive Spherical Interpolation (Figure 18)	2,012,238	189,901	93,311	670,746	28.31	13.91

The planar interpolation and adaptive spherical interpolation techniques produce an accurate control scene image. The Monte-Carlo implementation produces an erroneous image and demonstrates the problem of over sampling. Table 6 illustrates the view-dependent cost of using a global photon map based on the photons stored in

Table 1. The number of photons used is an indicator of the density of the photons, while the number of nodes examined relates to the size of the map and the efficiency of its structure.

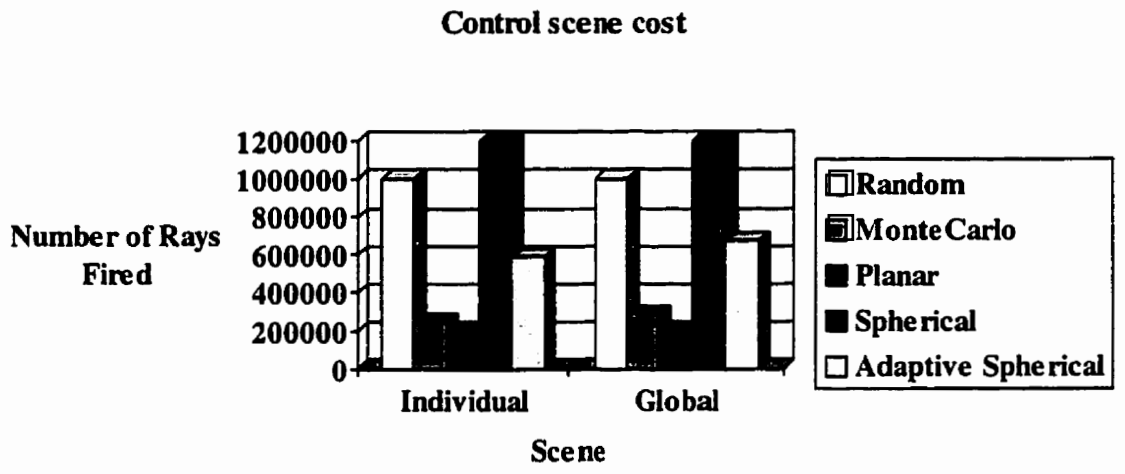
Table 6 View Dependent Global Photon Map Retrieval Costs for the Control Scene

Method	Number of Photon Nodes Examined	Number of Photons Used
Completely Random	177,838	136
Monte-Carlo Random	1,244,251	86,977
Brute Force Planar Interpolation	7,710,996	64,197
Naïve / Brute Force Spherical Interpolation	421,840	764
Adaptive Spherical Interpolation	17,333,273	264,604

The number of node comparisons is an indication of the efficiency (balancing) of the map and the size of the volume of interest. The volume of interest is kept the same for all pre-processors tested to promote consistency. Balancing the map has not been implemented, so even though the map populated using Monte-Carlo techniques is roughly the same size as the planar interpolation map, it is more efficient in node comparisons. The differences in node comparisons for similar sized maps would be much less for a balanced tree.

The completely random and naïve/brute force spherical interpolation techniques were inefficient in pre-processing even a simple scene. They are not included in further tests. They are, however the basis for the other pre-processors which have proven that they can render indirect illumination as accurately as the non-refractive shading model. Figure 4 illustrates the costs of the different pre-processors and ambient map types for the control scene.

Figure 4 Pre-processing cost for the control scene



5.4 Refraction Scene

The control scene did not have any refractive objects, and so the "Non-Refractive" shading model rendered it accurately. Many scenes do contain refraction and so a simple shading model cannot properly render them. The refraction scene is identical to the control scene except that the transparent plane with the wavy bump map has an index of refraction of water (1.33) [8] rather than an index of refraction of a vacuum (1.0).

Table 7 Refraction Scene Shading Model Costs

Method	Number of Intersection Tests	Number of Shadow Rays	Number of Reflection Rays	Number of Refraction Rays
Non-Transparent Shading Model (Figure 19)	185,114	19,018	1,042	1,786
Non-Refractive Shading Model (Figure 20)	185,114	19,018	1,042	1,786

Table 7 has identical values for both tests, since it is essentially the same as the control scene. The differences between it and Table 3 are due to the refractive plane. Neither of the images rendered using the shading models accurately depicts how the final image should appear. The three pre-processors tested with the refraction scene produce a more realistic image. The costs of using the pre-processors are recorded in the following three tables.

Table 8 Pre-Processing the Refraction Scene Using Individual Ambient Maps

Method	Number of Intersection Tests	Object Intersections	Light Values Stored	Number of Recursively Traced Rays	Percentage of Rays That Intersect An Object	Percentage of Rays That Are Stored
Monte-Carlo Random (Figure 21)	708,801	35,720	10,267	236,267	15.12	4.34
Brute Force Planar Interpolation (Figure 23)	583,593	95,896	17,290	194,531	49.30	8.89
Adaptive Spherical Interpolation (Figure 25)	1,765,410	107,625	19,551	588,470	18.29	3.32

The values in Table 8 indicate that the most efficient technique uses planar interpolation while the adaptive technique is least efficient. The images rendered from the individual ambient maps appear to be accurate as well. These images are an indication of why caustics and indirect illumination are important to consider when rendering.

Table 9 Pre-Processing the Refraction Scene Using a Global Photon Map

Method	Number of Intersection Tests	Object Intersections	Photons Stored	Number of Recursively Traced Rays	Percentage of Rays That Intersect An Object	Percentage of Rays That Are Stored
Monte-Carlo Random (Figure 22)	853,707	83,477	31,569	284,569	29.33	11.09
Brute Force Planar Interpolation (Figure 24)	598,503	100,866	22,260	199,501	50.56	11.16
Adaptive Spherical Interpolation (Figure 26)	2,012,238	189,901	93,311	670,746	28.31	13.91

The values in Table 9 indicate that the adaptive technique is the most efficient. This is a reversal of what Table 8 indicates. The difference is the result of how the photon map is populated compared to the individual ambient maps. The individual maps are implemented to prevent over sampling, while the photon map is not. The adaptive technique restricts over sampling when the rays are cast, rather than when they strike an object. The images rendered from using a global map appear similar to those rendered using individual maps.

Table 10 displays the costs of using the global photon map produced in Table 9. The number of photons used indicates the density of the photons in the map and the amount

of blurring (averaging) used at each volume of interest to calculate the indirect illumination contribution.

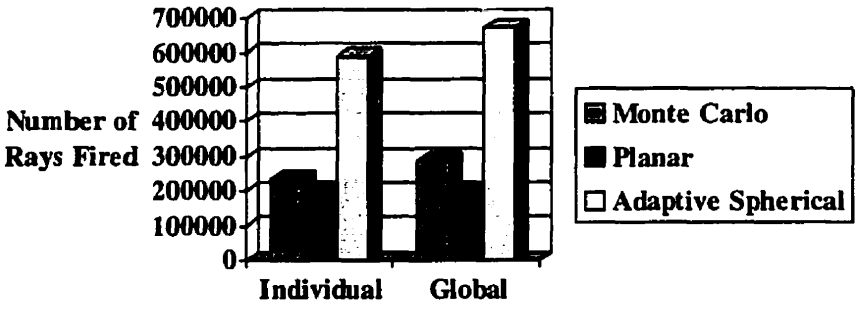
Table 10 View Dependent Global Photon Map Retrieval Costs for the Refraction Scene

Method	Number of Photon Nodes Examined	Number of Photons Used
Monte-Carlo Random	1,017,676	100,102
Brute Force Planar Interpolation	2,061,132	72,075
Adaptive Spherical Interpolation	4,808,309	300,105

The lack of balancing results in the inefficiency of the planar interpolation technique. The sheer size of the photon map populated by adaptive spherical interpolation is the reason why its node comparisons and number of photons used are higher than the other methods.

The control scene and refraction scene are trivial compared to the complex scenes that can be rendered. Figure 5 illustrates the costs of the different pre-processors and ambient map types for the control scene. A larger, non-trivial scene is used for the final test.

Figure 5 Pre-processing cost for the refraction scene



5.5 Complex Scene

A complex scene is used to administer the final test of which technique is best for indirect illumination. The scene consists of two point lights, and nine objects. Both lights are located close to one another, but using two lights doubles the amount of pre-processing work and number of shadow rays. The camera is located within a box (6 polygon objects) with the top (surface) being a transparent, two-sided polygon with a wavy surface. A transparent ellipsoid and a semi transparent sphere are also located in the box along with a fractal object. The fractal object uses a hierarchical bounding volume structure to improve its efficiency.

Table 11 Complex Scene Shading Model Costs

Method	Number of Intersection Tests	Number of Shadow Rays	Number of Reflection Rays	Number of Refraction Rays
Non-Transparent Shading Model (Not shown)	2,226,208	85,492	266	2,492
Non-Refractive Shading Model (Figure 27)	2,257,544	85,492	266	2,492

The "Non-Transparent Shading Model" produces a completely black image. The increase in rays and intersection tests over previous scenes is due to the increase in complexity of this scene. The difference between the intersection tests of the non-transparent shading model and the non-refractive shading model illustrates how the non-transparent shading model is more efficient.

Table 12 Pre-Processing the Complex Scene Using Individual Ambient Maps

Method	Number of Intersection Tests	Object Intersections	Light Values Stored	Number of Recursively Traced Rays	Percentage of Rays That Intersect An Object	Percentage of Rays That Are Stored
Monte-Carlo Random (Figure 28)	12,875,302	330,654	119,392	793,399	41.68	15.05
Brute Force Planar Interpolation (Figure 30)	9,762,016	368,402	155,648	599,701	61.43	25.95
Adaptive Spherical Interpolation (Figure 32)	30,734,182	927,937	174,470	1,893,058	49.02	9.22

As with previous tests, the planar interpolation technique is the most efficient at hitting objects and storing its values. The adaptive technique stores a relatively small amount more than the planar interpolation technique, but requires roughly three times as many rays.

The images produced using the individual maps appear blocky, because the resolution of the maps is low compared to the resolution of the camera. The view-independent pre-processing phase is therefore dependent on the view-dependent rendering phase to determine the correct resolution of the individual maps. The problem with resolution imposes an unwanted restriction on the use of individual ambient maps.

Table 13 Pre-Processing the Complex Scene Using a Global Photon Map

Method	Number of Intersection Tests	Object Intersections	Photons Stored	Number of Recursively Traced Rays	Percentage of Rays That Intersect An Object	Percentage of Rays That Are Stored
Monte-Carlo Random (Figure 29)	20,028,562	724,937	357,511	1,225,078	59.17	29.18
Brute Force Planar Interpolation (Figure 31)	10,521,966	410,622	197,868	645,921	63.57	30.63
Adaptive Spherical Interpolation (Figure 33)	43,054,080	1,666,268	835,427	2,640,756	63.10	31.64

Table 13 indicates that the three methods are roughly equivalent in efficiency with regard to light ray hit and photon storage. The numbers also indicate that the adaptive spherical interpolation technique creates a much larger map, resulting in smoother ambient values between adjacent regions. Although time is not shown, the amount of time needed to pre-process using the adaptive spherical technique was many times longer than the other techniques. The Monte-Carlo technique took approximately twice as long as the planar interpolation technique. The adaptive spherical technique took approximately six times longer than the planar interpolation technique.

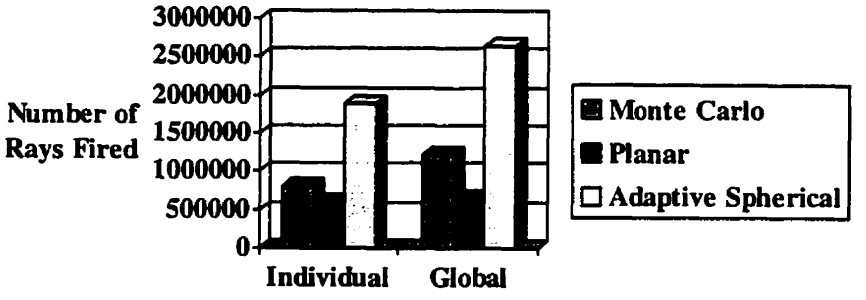
Table 14 View Dependent Global Photon Map Retrieval Costs for the Complex Scene

Method	Number of Photon Nodes Examined	Number of Photons Used
Monte-Carlo Random	7,596,240	1,511,262
Brute Force Planar Interpolation	26,128,615	742,213
Adaptive Spherical Interpolation	69,499,037	3,188,513

Monte-Carlo is more efficient than planar interpolation due to how the photon map is populated (balancing). Adaptive spherical interpolation is inefficient with regards to node comparisons due to its large size and lack of balancing.

The test scenes have shown which methods calculate indirect illumination faster, more accurately and more robustly than others do. They have also exposed which methods have potential, but need to be improved in either their design or implementation. Figure 6 illustrates the costs of the different pre-processors and ambient map types for the control scene.

Figure 6 Pre-processing cost for the complex scene



Chapter 6

CONCLUSIONS

Indirect illumination from refractive objects plays an important role in ray tracing. Techniques that calculate indirect refractive illumination may work more efficiently, accurately, or robustly than others. Another concern is whether a technique requires significant design and implementation changes to be incorporated to use it.

6.1 Efficiency

The Non-Transparent shading model is the fastest but does not render indirect illumination. The Non-Refractive shading model is the fastest technique for indirect transparent illumination, as opposed to indirect refractive illumination. The cost is incurred at render time, but is much less than the other pre-processor alternatives for a 200 x 200 image. Increasing the number of camera rays reduces the relative savings of the non-refractive shading model over the forward light tracing pre-processors.

If indirect refractive illumination is needed (for realism) but the camera (eye) is not located very close to the objects, individual ambient maps can be used. The planar interpolation technique is the most efficient method for populating individual ambient maps.

If resolution problems occur, the planar interpolation and Monte-Carlo methods are the most efficient at populating a global photon map.

6.2 Robustness

The planar interpolation is a specialized case and is not robust. It can, however be expanded to make it robust and even more efficient when using individual ambient maps. It can be modified to interpolate along the individual ambient maps of the transparent (and reflective) objects based on their resolution. This would also eliminate the implementation restriction of min/max values for objects and a bounding box for the entire scene.

The adaptive spherical interpolation technique is the most robust method implemented. It works with both individual and global maps. It does not impose restrictions on the objects or scene, and can work with other light sources such as directional or spotlights with little (if any) implementation change. The adaptive technique ensures a consistent sampling density, as opposed to Monte-Carlo, which can create areas of super-sampling in a global map.

The global photon map is the most robust map type implemented. It works with any type of object, without the problems associated with inverse mapping of individual ambient maps. There are still some problems with using a global photon map such as memory, speed, and obstructed photons in the volume of interest.

6.3 Accuracy / Realism

The shading methods alone cannot produce realistic images in scenes where refractive objects are rendered. The pre-processors can render indirect refractive illumination as well as illumination from indirect reflective sources.

The individual ambient maps can produce realistic images, but may have resolution problems or an incorrect inverse mapping structure. The inverse maps are blurred, which can create blocks of color depending on the blur size. The contributions between adjacent areas may vary greatly because ambient contributions are blurred based on objects rather than by area.

The photon map produces the most accurate and realistic images, when correctly populated. Ambient contributions blend between adjacent regions of interest and any type of object can be accurately mapped.

6.4 Conclusion

The adaptive spherical interpolation technique using a global photon map is the best technique for rendering indirect refractive illumination. It is robust, imposes very few (if any) implementation restrictions and renders realistic images. It is relatively efficient and its light ray concentration controllable through sampling and adaptive depth parameters.

As a compromise, the non-refractive shading model can be used for evaluating object / camera / light placement in a scene. The Monte-Carlo technique using individual ambient maps can be used to provide a preliminary view of the final image, and the adaptive spherical interpolation using a global photon map can be used to render the final image.

6.5 Further Work

Object instantiation is a major performance bottleneck in Java. The current implementation of the photon map requires the creation of thousands of individual photon objects when it is populated. The release of the Java HotSpot™ virtual machine, which will increase performance of this ray tracer, reduces the cost of instantiating objects. Another implementation change is balancing the photon map to make retrieval of nodes more efficient.

Implementing a more efficient photon map retrieval method is needed to make the photon map a more useful indirect illumination map. The use of procedural texture maps reduces the resolution problems of inverse mapping. Construction of a procedural (fractal compression) photon map based on populated values or individual ambient map values would eliminate resolution problems from ambient maps. A procedural photon map would be expensive to construct but more efficient at retrieving values than the current photon map.

Much of the future work associated with the implementation will be in improving the efficiency of algorithms and implementing additional features such as displacement mapping, constructive solid geometry, and free-form deformation.

BIBLIOGRAPHY

[1] Appel, A., "Some Techniques for Shading Machine Renderings of Solids", *AFIPS Conference Proceedings*, 32, 37-45, 1968.

[2] Arvo, James, "Backward Ray Tracing," *SIGGRAPH '86 Developments in Ray Tracing* seminar notes, Vol. 12, August 1986.

[3] Biggs, N., Discrete Mathematics (Revised Edition), New York: Oxford University Press, 1990

[4] Burger, P. and Gillies, D., Interactive Computer Graphics, England: Addison Wesley, 1990.

[5] Cheah, S.C., "An Implementation of a Recursive Ray Tracer That renders Caustic Lighting Effects," Independent Study: University of Maryland, 1996.

[6] Cook, R.L., "Stochastic Sampling and Distributed Ray Tracing," Introduction to Ray Tracing, USA: Academic Press, 1991.

[7] Foley, J., van Dam, A., Feiner, S., Hughes, J., Phillips, R., Introduction to Computer Graphics, USA: Addison Wesley, 1994.

[8] Glassner, A., An Introduction To Ray Tracing, USA: Academic Press, 1991.

[9] Haines, E., "Essential Ray Tracing Algorithms,"Introduction to Ray Tracing, USA: Academic Press, 1991.

[10] Hall, R., Illumination and Color in Computer Generated Imagery, New York: Springer-Verlag, 1989.

[11] Hearn, D., and Baker, M. P., Computer Graphics C Version, 2nd Ed., New Jersey: Prentice Hall, 1997.

[12] Heckbert, P., "Adaptive Radiosity Textures for Bi-Directional Ray Tracing," *Computer Graphics*, Vol. 18, No. 4, pp. 145-154, 1990.

[13] Jensen, H.W., "Global Illumination Using Photon Maps," *Rendering Techniques '96*, Eds. Pueyo, X. and Schroder, P.: Springer-Verlag, pp. 21-30, 1996.

[14] Kajiya, J., "The Rendering Equation", *Computer Graphics*, Vol. 20 No. 4, pp. 143-149, 1986.

[15] Mitchell, D., Hanrahan, P., "Illumination from Curved Reflectors," *Computer Graphics*, Vol. 26, No. 2, pp. 283-291, 1992.

[16] Nishita, T. and Nakamae, E. "method of Displaying Optical Effects within Water using Accumulation Buffer," *Computer Graphics Proceedings, Siggraph '94*, pp. 373-379, 1994.

[17] *The Oxford Dictionary of Quotations*, 3d ed. New York: Oxford University Press, 1980.

[18] Ward, G.J., Rubenstein, F.M. and Clear, R.D., "A Ray Tracing Solution for Diffuse Interreflection," *Computer Graphics*, Vol. 22, No. 4, pp. 85-92, 1988.

[19] Watkins,C., Coy, S., and Finlay, M., Photorealism and Ray Tracing in C, USA: M&T Publishing Inc., 1992.

[20] Watt, A., Watt, M., Advanced Animation and Rendering Techniques *Theory and Practice*, England: Addison Wesley, 1992.

[21] Watt, A., Policarpo, F., The Computer Image, New York: Addison Wesley, 1998.

[22] Whitted, T., "An Improved Illumination Model for Shaded Display", *Communications of the ACM*, 26(6), 342-349, 1980.

APPENDIX A - GENERATED IMAGES

The images generated by the ray tracer are saved in true color TGA format. They have been converted to DIB format in order to be stored with this document, however there is no change in size or quality from this conversion.

Control Scene Images

The control scene images rendered using different shading models:

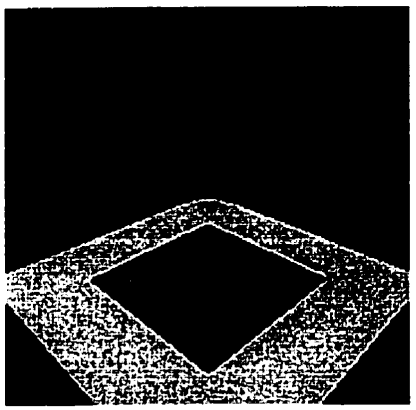


Figure 7 The control scene rendered using a non-transparent shading model

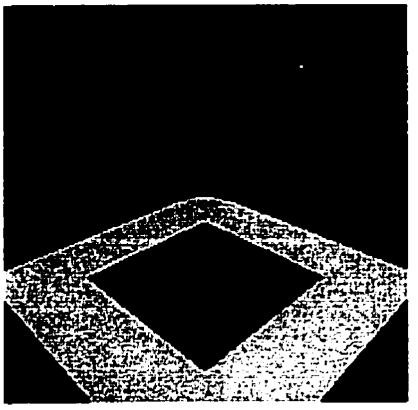


Figure 8 The control scene rendered using a non-refractive shading model.

The control scene images rendered using five different forward-tracing pre-processors.

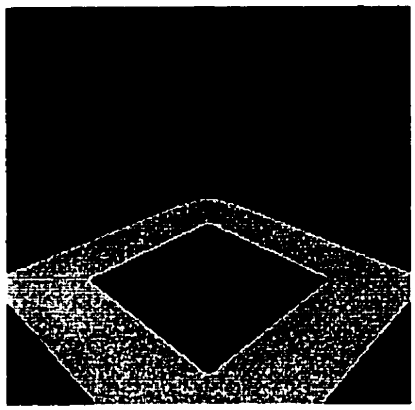


Figure 9 The control scene rendered using individual ambient maps populated by a completely random forward light tracer.

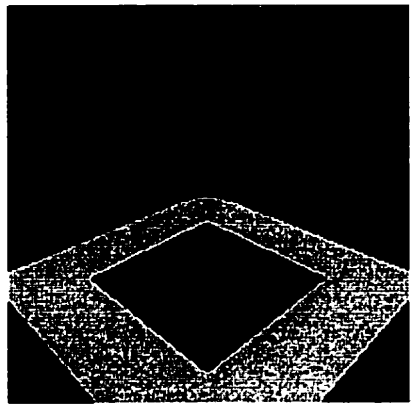


Figure 10 The control scene rendered using a global photon map populated by a completely random forward light tracer

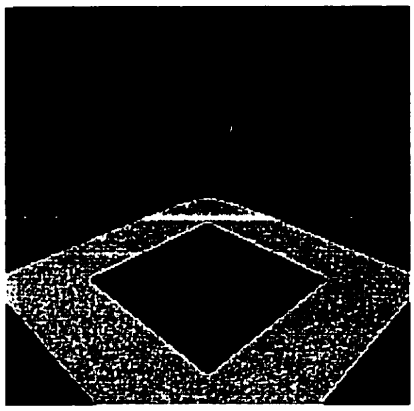


Figure 11 The control scene rendered using individual ambient maps populated by a Monte-Carlo forward light tracer

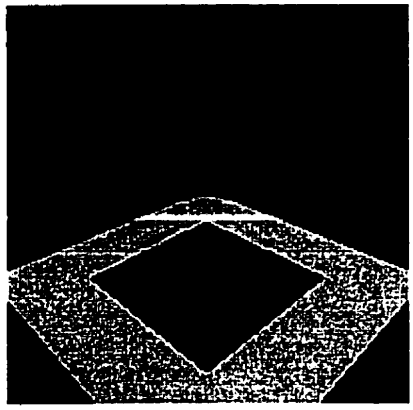


Figure 12 The control scene rendered using a global photon map populated by a Monte-Carlo forward light tracer

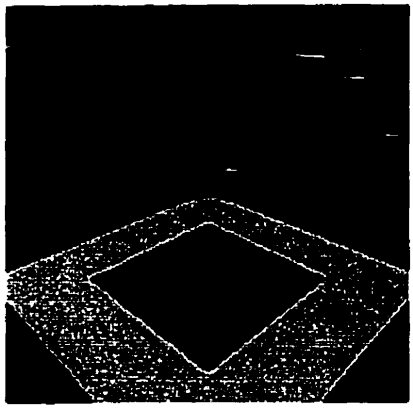


Figure 13 The control scene rendered using individual ambient maps populated by a planar interpolation forward light tracer

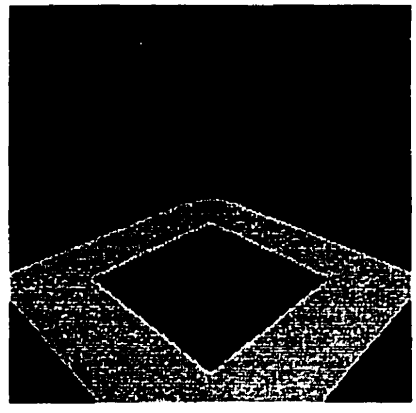


Figure 14 The control scene rendered using a global photon map populated by a planar interpolation forward light tracer

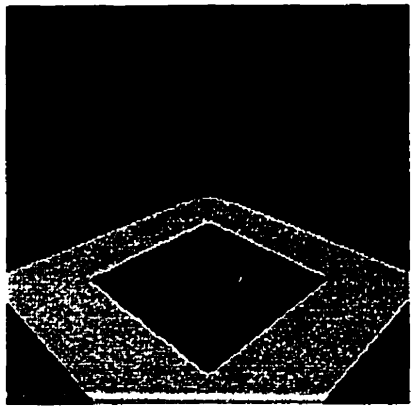


Figure 15 The control scene rendered using individual ambient maps populated by a naïve/brute force spherical interpolation forward light tracer

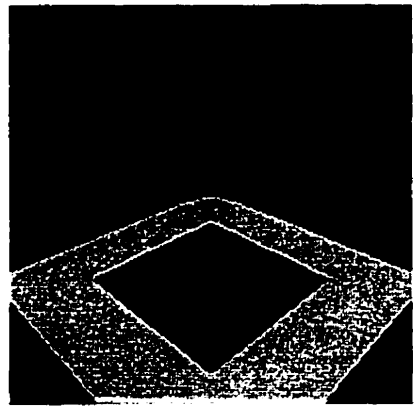


Figure 16 The control scene rendered using a global photon map populated by a naïve/brute force spherical interpolation forward light tracer

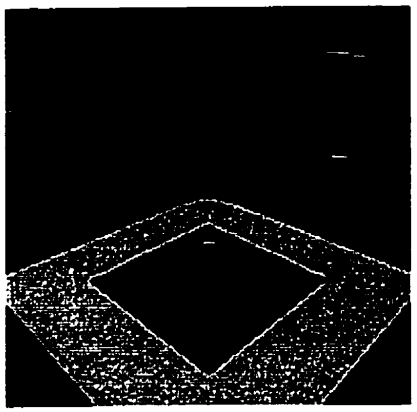


Figure 17 The control scene rendered using individual ambient maps populated by an adaptive spherical interpolation forward light tracer

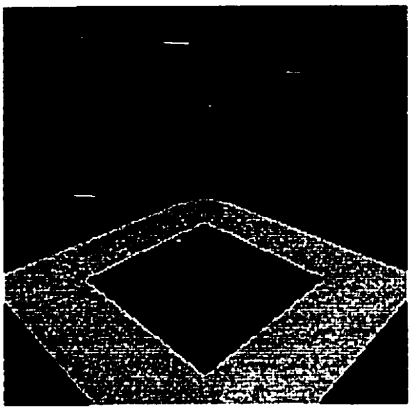


Figure 18 The control scene rendered using a global photon map populated by an adaptive spherical interpolation forward light tracer

Refraction Scene Images

The refraction scene images rendered using different shading models:

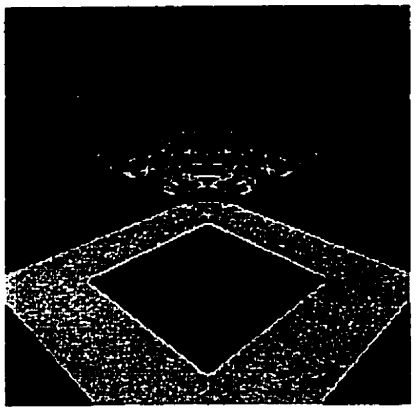


Figure 19 The refraction scene rendered using a non-transparent shading model.

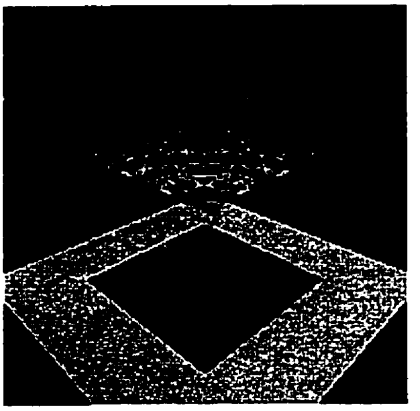


Figure 20 The refraction scene rendered using a non-refractive shading model

The refraction scene images rendered using three different forward-tracing pre-processors:

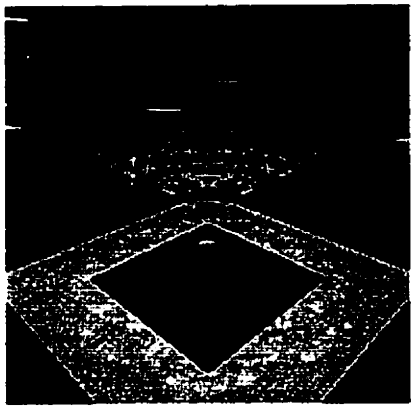


Figure 21 The refraction scene rendered using individual ambient maps populated by a Monte-Carlo forward light tracer.

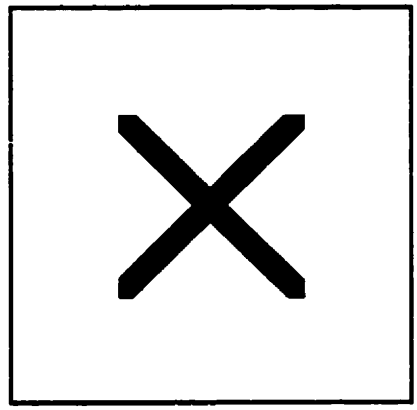


Figure 22 The refraction scene rendered using a global photon map populated by a Monte-Carlo forward light tracer.

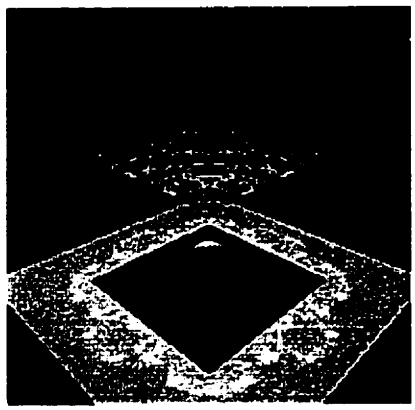


Figure 23 The refraction scene rendered using individual ambient maps populated by a planar interpolation forward light tracer

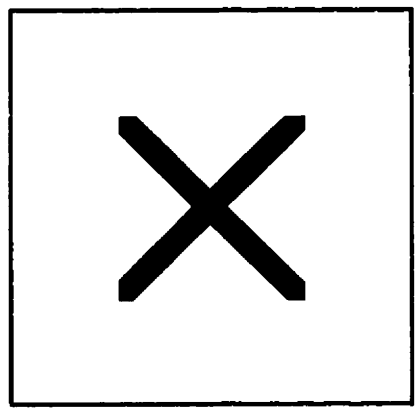


Figure 24 The refraction scene rendered using a global photon map populated by a planar interpolation forward light tracer

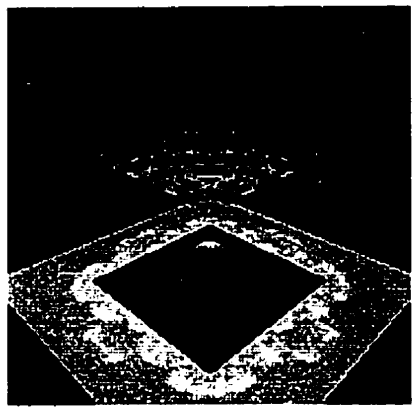


Figure 25 The refraction scene rendered using individual ambient maps populated by an adaptive spherical interpolation forward light tracer.

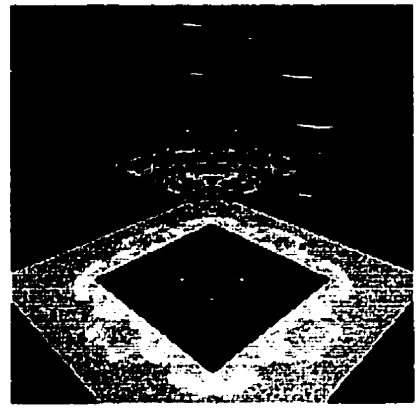


Figure 26 The refraction scene rendered using a global photon map populated by an adaptive spherical interpolation forward light tracer

Complex Scene Images

The complex scene images rendered using different shading models. The non-transparent shading model rendering creates a completely black image, and so it is not included.

Figure 27 The complex scene rendered using a non-refractive shading model

The complex scene images rendered using three different forward-tracing pre-processors:

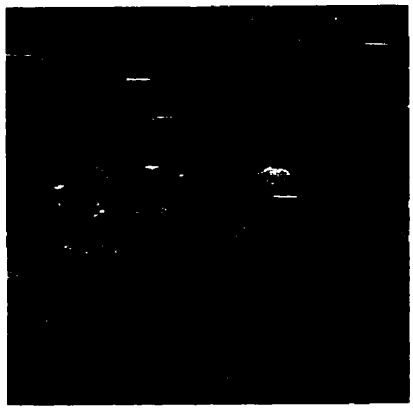


Figure 28 The complex scene rendered using individual ambient maps populated by a Monte-Carlo forward light tracer

Figure 29 The complex scene rendered using a global photon map populated by a Monte-Carlo forward light tracer

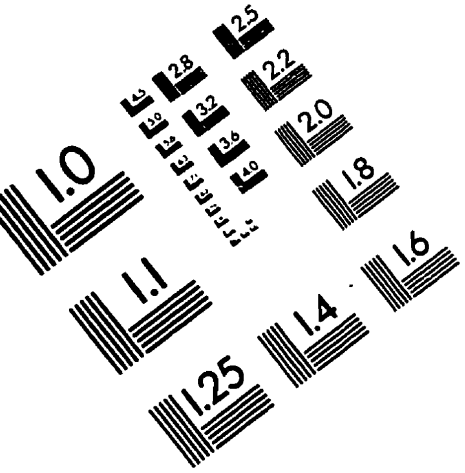
Figure 30 The complex scene rendered using individual ambient maps populated by a planar interpolation forward light tracer

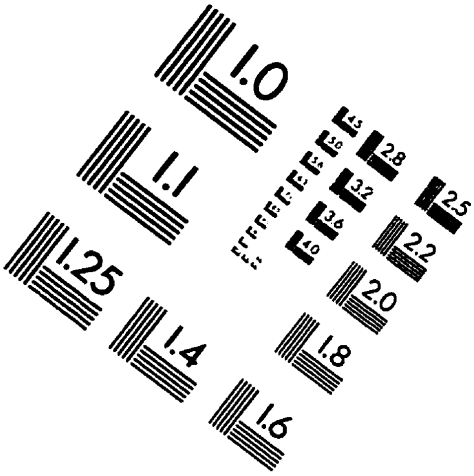
Figure 31 The complex scene rendered using a global photon map populated by a planar interpolation forward light tracer

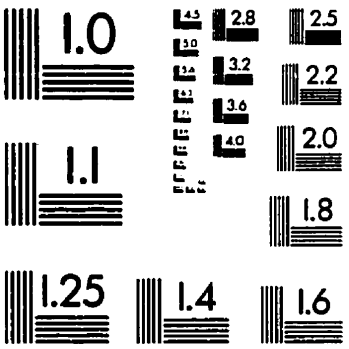
Figure 32 The complex scene rendered using individual ambient maps populated by an adaptive spherical interpolation forward light tracer

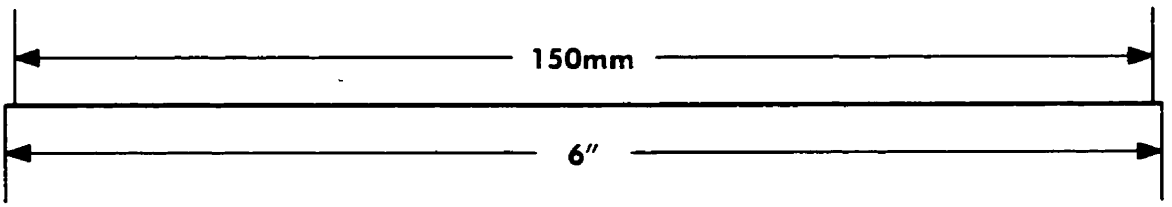
Figure 33 The complex scene rendered using a global photon map populated by an adaptive spherical interpolation forward light tracer

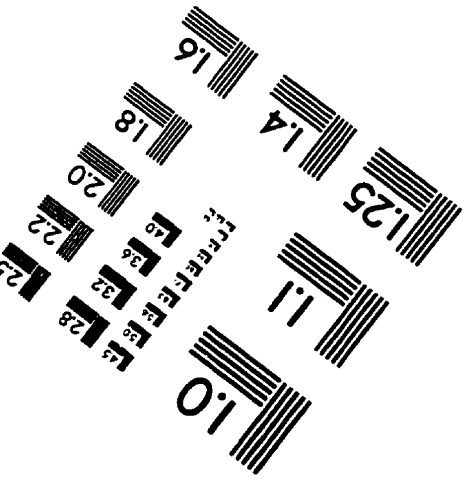
IMAGE EVALUATION
TEST TARGET (QA-3)

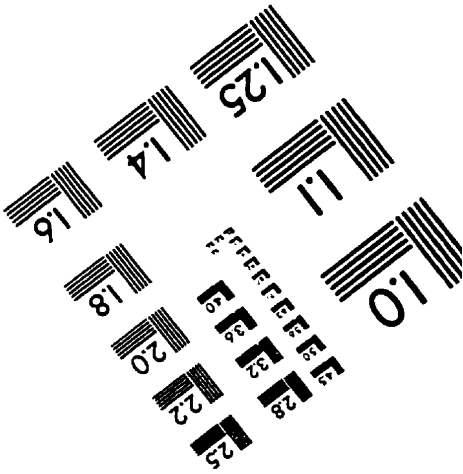












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989