Technische Universität Wien
Departamento de Engenharia Informática, Universidade de Coimbra

Implementation of Dispersion Rendering Capabilities in the *Advanced Rendering Toolkit*

Nuno Daniel Raposo Subtil

July 14, 2006

Diploma Thesis for the Degree of Informatics Engineering

This text describes the implementation of dispersion rendering capabilities in the *Advanced Rendering Toolkit*, a spectral renderer developed at the *Technische Universität Wien*. A theoretical framework of physically based rendering techniques is presented, based on Kajyia's rendering and potential equations and physically accurate BRDF/BSDF-based material descriptions. An approximate solution to these equations is then obtained through Monte-Carlo integration techniques. Attention is given to importance-sampling and Russian roulette termination in the context of Monte-Carlo integration.

The physical laws which govern light dispersion within translucent materials are also presented, in the context of simulating them within a spectral renderer. The approach, design and implementation of these features in ART is then discussed, followed by an evaluation of the results obtained. A method for validating these results using photographs of physical objects is also outlined. The obtained results show that the implemented technique worked as intended, creating realistic dispersion effects.

"DID YOU SAY HUMANS DO IT FOR FUN?"
"Some of them get to be very good at it, yes. I'm only an amateur, I'm afraid."
"BUT THEY ONLY LIVE EIGHTY OR NINETY YEARS!"

Terry Pratchett, *The Light Fantastic*

Contents

List of Figures

CHAPTER 1

Introduction

The aim of *image synthesis*, or *rendering* as it is usually known, is to enable the creation of life-like or appealing images from synthetic models. With applications in such varied areas as architecture, medicine, computer games, cinema or television, it is a very interesting field within the realm of computer graphics research.

The problem of rendering realistic images of 3D geometric models has been approached in several distinct ways. For applications where speed is critical, it is common to find techniques based on triangle rasterisation, using very simple illumination models designed to be fast and look "good enough". The limitations of such models are generally masked by clever usage of texture mapping, and the end results can be very pleasing to the eye. Taking into account that most real-time applications will be working at frame rates on the order of 20-30 frames per second, there is also generally not enough time for small details to be noticed — all that is needed is to keep the general environment convincing enough.

For non-interactive rendering, the stakes are raised substantially. Frame rate is not critical, as long as render time is "reasonable" — several hours per frame is common in the movie industry. This allows for more complex lighting and geometric models to be used, increasing realism substantially. However, the general trend is for lighting models to be sufficiently tunable, so that the end result fits the director's artistic vision, even if that means a completely unrealistic (but still convincing) result. It makes perfect sense — movies sometimes call for completely unrealistic imagery as well.

Motivation for physically based rendering

Perceptual, tunable lighting models are however not suitable if what is needed is to accurately predict how a given object will look like under certain lighting conditions. For instance, one may want to render the interior of a building before construction, to

preview how the lighting will interact and what kind of mood it will create inside the building — a classic application of 3D imagery.

In this case, we are not simply interested in generating a pleasing image. Instead, we want an image which is an accurate representation of what the building would look like in reality. This could mean that the image is not even interesting by itself, in which case the architect would probably need to redesign the building rather than try and tune the parameters of the renderer.

Physically-based rendering enables one to achieve just that. By accurately simulating the interactions of light with the objects in the scene to render, it is possible to compute an accurate representation of what the scene would actually look like in reality. All that is required is a good description of the objects and their physical properties; tunable parameters in the renderer, where they exist, serve merely to balance render time versus final image quality. This could actually be perceived as a disadvantage, as it implies that a physically-based renderer lacks flexibility; however, the argument can also be made that, with proper editing tools, it gives the user a lot more freedom in terms of describing the actual materials of the objects that comprise the scene. This is perhaps a more natural way of tuning the final result, as opposed to tweaking abstract values in a shading model applied to a surface.

This report is an account of my work at the *Technische Universität Wien*, Austria. During 5 months I worked with the Photo-realistic Rendering and Modelling group at the *Institut für Computergraphik und Algorithmen*, implementing light dispersion rendering capabilities in the *Advanced Rendering Toolkit* — a physically based renderer developed at the institute. The main objective was to study and implement dispersive rendering capabilities in some of the rendering algorithms available within ART, to enable the renderer to accurately predict light dispersion phenomena based on the material properties and incoming illumination descriptions.

My work began with a preliminary comprehensive study of the underlying theory of physically based rendering. This included an overview of the numerical techniques usually employed to obtain approximate solutions to the complex high-dimensional integrals involved. Details such as importance-sampling and Russian roulette based termination were given special attention, due to the inherent and complicated subtleties of implementing them in practice.

On the technical side, a familiarisation with the architecture and inner workings of ART was required. ART is a highly complex rendering system, consisting of mode than two hundred thousand lines of code, which means this task is not an easy one. The modifications required in order to implement dispersive rendering capabilities included building the required support mechanisms and refactoring some of the rendering code. While not very complicated by themselves, these modifications required careful checking and validation. ART is intended not only to generate interesting images but also, and perhaps more importantly, to output physically correct results. Any changes in the code must thus be properly validated by the underlying theory. This is

why most of the work presented here was not done by editing code on a text editor, but using pencil and paper instead.

In the next chapter of this report, I present the relevant theoretical introduction to physically based rendering, followed by the physical principles of light dispersion in chapter 3. Chapter 4 consists of a quick state of the art report on other renderer implementations available with light dispersion capabilities; chapter 5 contains the details of my work on dispersion rendering within the *Advanced Rendering Toolkit*. Chapter 6 contains a discussion of the results obtained, including an outline of the validation method employed. Chapter 7 presents the conclusions and a note on further work possible on the topic.

CHAPTER 2

Physically Based Rendering

This chapter will be dedicated to building a theoretical framework for the problem of physically-based image synthesis. Although heavily abstract and mathematical in nature, this framework will enable us to analyse and predict the behaviour of several rendering techniques, as well as provide a solid theoretical understanding as to why they work.

Image Synthesis as a Light Transport problem

When an observer looks at a given object, he is perceiving the effects of light in his eyes. Light sources emit light, which interacts with the object (and other objects around it), reflecting and refracting until it eventually reaches the observer. This light is then focused by the eye onto the retina, forming an image there. Special sensors on the retina detect it and convey this information to the brain for processing.

The same happens when we have an observer watching a computer screen. The screen is composed of *pixels*, each of which can be set by software to emit a given "colour", i.e., the emission spectrum of each pixel can be (roughly) controlled. In order to cause the impression of watching the real world, we therefore need to compute the amount and spectrum of light that each pixel should emit towards the observer — we can consider the screen as a "window" through which the observer perceives the world[1].

Let us now assume that the observer is represented by a point in space, and also assume that each pixel is reduced to a single point on the screen. Under these conditions, light can only reach the observer from each pixel on the screen along a single path. We

[1] A camera analogy could also be made, with the same results. Instead of simulating a "window" into the world, we can simulate a camera inside the scene, and compute the amount of light which arrives at each point in the film. Both models are mostly interchangeable and yield the same results.

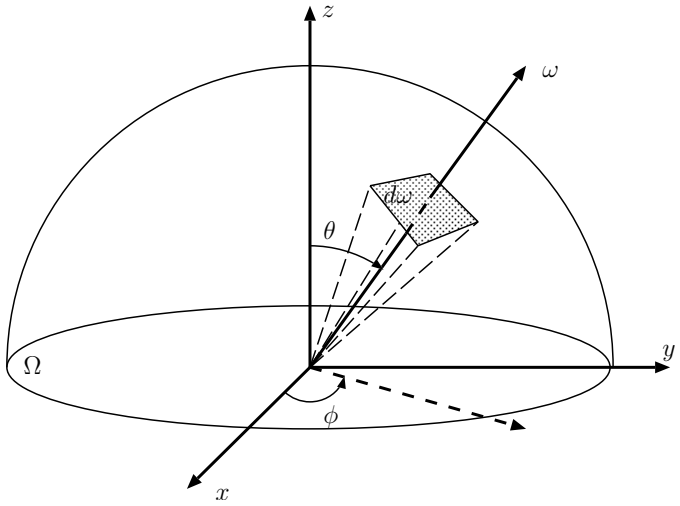


Figure 2.1: Differential solid angles

therefore need to compute, for each pixel, the "amount" of light travelling along this path.

If we follow the path we defined into the scene, we will arrive at a point on the surface of an object (unless we miss all objects in the scene, in which case no light would reach the observer). We therefore need to compute the "amount" of light leaving this point in the direction of the path towards the observer. This will depend both on the surface characteristics of the object as well as the incoming light it receives.

2.1 Definitions

In order to better understand the problem, we will now describe some fundamental concepts.

Directions and Solid Angles

Light emission and transport phenomena exhibit directional dependence, thus it is useful to precisely define the domain of possible emission directions. This can be achieved with the help of solid angles.

For a given point P on a surface, we can establish a set of possible emission directions such that light hitting the surface at point P must be reflected along a direction belonging to this set. For an opaque surface, this set consists of a hemisphere Ω centred on P, with the apex along the direction of the surface normal at P.

A set of directions inside this hemisphere defines a *solid angle*, which can be thought of as a cone or pyramid with the apex at point P. If we consider the hemisphere to be of unit radius, then the *size* of this solid angle is defined by the area of the surface of the hemisphere which lies inside this cone.

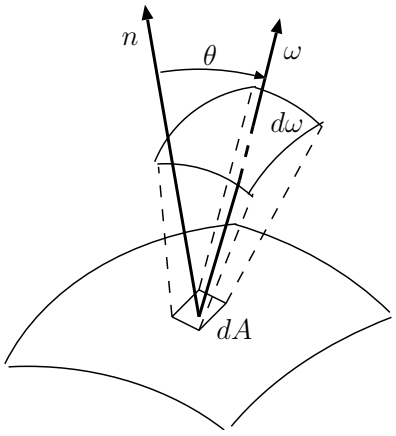


Figure 2.2: Deriving radiance from flux

We can define a spherical coordinate system around P. Suppose P lies on the xy plane, with the z axis pointing along the direction of the normal at P, as shown in figure 2.1. Any direction ω on the hemisphere can be defined by two angles, θ (the angle between ω and the z axis) and ϕ (the angle between the projection of ω on the xy plane and the x axis). Using this coordinate system, we can define a set of directions by appropriate intervals $\Delta\theta$ and $\Delta\phi$ for θ and ϕ. In turn, this allows us to define a direction ω on the hemisphere by a *differential solid angle*, in which $\Delta\theta = d\theta$ and $\Delta\phi = d\phi$ are considered to be infinitesimal.

Flux and Radiance

Consider the problem of computing outgoing light in a given direction ω at a point P of a given surface. In physical terms, the *light power* or *flux*, Φ, is the energy radiated through a given boundary (for example, an object's surface) in all possible directions per unit time.

Radiance is defined as the flux flowing out of a differential patch on the boundary along a differential solid angle per projected area on the surface:

$$L\left(P,\omega\right) = \frac{d^2\Phi\left(P, dA, \omega, d\omega\right)}{d\omega \cdot dA \cdot \cos\theta} \tag{2.1}$$

where dA represents the area of the emitting differential surface patch around P, $d\omega$ is the outgoing differential solid angle around ω, $d^2\Phi(P, dA, \omega, d\omega)$ is the differential flux flowing out of dA along solid angle $d\omega$ and θ is the angle between the surface normal for the emitting patch and ω (figure 2.2). That is, radiance measures the flux leaving point P in direction ω.

By Helmholtz's reciprocity, radiance can also measure the incoming flux at P along a given direction ω. If we consider the point P' visible from P along direction ω, the incoming radiance at P is equal to the outgoing radiance at P' in the opposite direction:

$$L_i(P, \omega) = L(P', -\omega) \tag{2.2}$$

This relationship is sometimes expressed through a *geometric* or *visibility function*: $P' = h\,(P, \omega)$ is the point visible from P in direction ω. Thus:

$$L_i\,(P, \omega) = L\,(h\,(P, \omega)\,, -\omega) \tag{2.3}$$

Modelling surface characteristics

The interaction of light with surfaces can be thought of as a random process — each photon hitting a given surface will be reflected in a random direction, due to the fact that surface irregularities are generally not known in enough microscopic detail.

We can therefore model surface characteristics using a probability distribution function. This function, usually called a *Bidirectional Reflectance Distribution Function* (BRDF), depends on the surface point, the incoming direction and the outgoing direction considered:

$$f_r\,(\omega_i, P, \omega_o) \tag{2.4}$$

where the vector ω_i represents the incoming direction being considered, ω_o the outgoing direction and P is the 3D surface point[2].

In physically-based rendering, BRDFs must obey two fundamental restrictions:

Helmholtz's Reciprocity Law which states that, for any two points A and B with line of sight between them, the only light received at B arriving in the direction of A is the light that A emits towards B. This implies that ω_i and ω_o in 2.4 are interchangeable.

Conservation of Energy which implies that, for a given incoming direction ω, the amount of light scattered in all possible directions must not exceed the amount of incoming light in direction ω. Mathematically,

$$\int_{\Omega} f_r\,(\omega, P, \omega')\,d\omega' \leq 1 \tag{2.5}$$

where Ω represents the hemisphere of possible outgoing directions.

It is also sometimes useful to consider not only reflection but take refraction into account as well. To accomplish this, the BRDF model can be extended: incoming and outgoing directions are no longer limited to the unit hemisphere around the point, but can take any direction from the unit sphere around the point. This model allows for light to be transported to the inside of the object, and is sometimes called *Bidirectional Scattering Distribution Function*, or BSDF.

[2]This notation will be used when it is necessary to distinguish incoming and outgoing directions; in case the distinction is not strictly necessary, ω and ω' will be used instead.

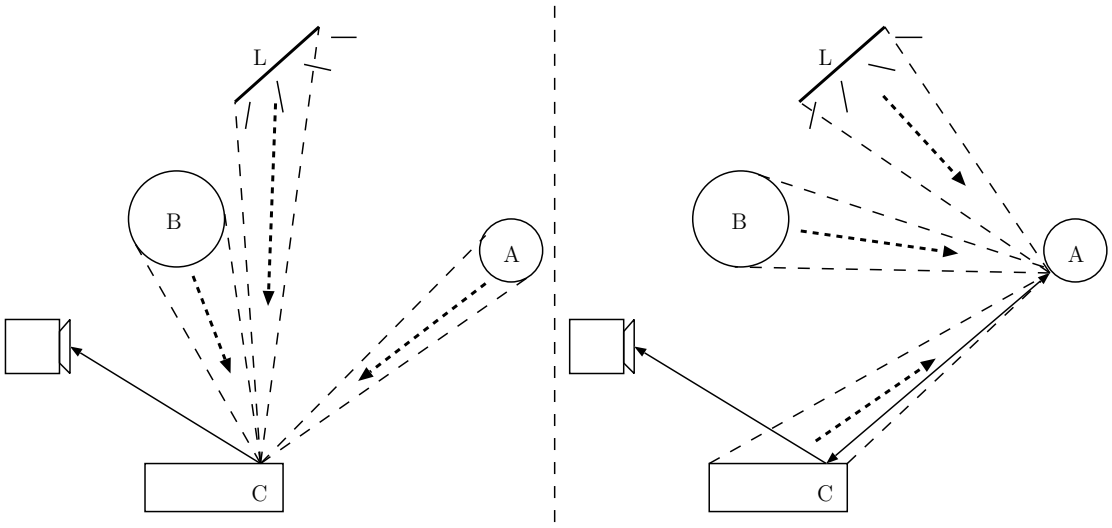


Figure 2.3: The problem of determining radiance. The light emitted at the surface of object C towards the observer depends on the light received at the intersection point from all other objects (left-hand side). However, when we consider another point on the surface of another object, we find that the light it emits towards the first point also depends on the light received from all other objects — including the original object C!

2.2 The Rendering and Potential Equations

The Rendering Equation

Let us now consider the problem of computing the radiance emanating from a point on the surface of an object along a given direction towards the observer, as can be seen on the left side of figure 2.3. It is easy to see that the light emanated from object C depends on the light received in all possible directions from all possible objects — including those that do not emit light but simply reflect it. We could try and evaluate (in discrete terms) all possible incoming directions; however, we would immediately find that, once we attempt to evaluate the radiance emitted at a given point on the surface of another object towards the original intersection point, the whole evaluation process would have to be repeated again in the same way (as can be seen on the right-hand side of figure 2.3).

We shall now formally express this problem. In order to simplify this discussion, we assume that we are dealing with monochromatic light consisting of a single wavelength. This does not imply loss of generality, since polychromatic light can be seen as the sum of monochromatic light at several different wavelengths.

Consider the simple case of light transport where a point P on an emitter patch dA reflects light which is absorbed at another patch dA'. dA receives photons along a given solid angle $d\omega_i$ around direction ω_i, with total power $d^2\Phi_i\left(P, dA, \omega_i, d\omega_i\right)$, reflecting

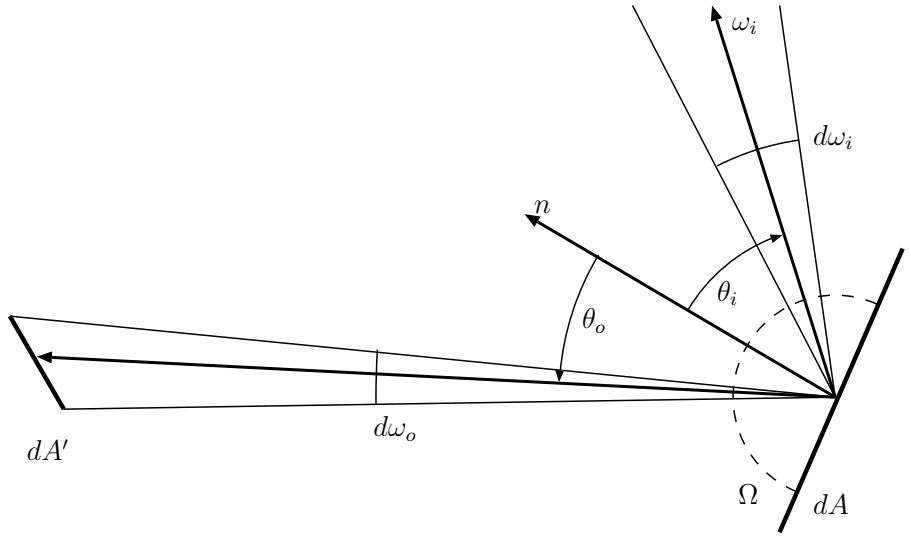


Figure 2.4: Light transport between two surfaces. ω_i "sweeps" along the domain defined by Ω, and the contributions for each direction are added to determine the radiance leaving dA towards dA'

them towards dA' along solid angle $d\omega_o$ around direction ω_o. We can consider a function $F_r(\omega_i, P, \omega_o, d\omega_o)$ which expresses the ratio of reflected power at P along an outgoing solid angle $d\omega_o$ around ω_o, for incoming direction ω_i (figure 2.4).

The total power reflected at dA towards dA' is equal to the sum of the incoming power along all possible directions multiplied by the F_r for each possible incoming direction and the considered outgoing solid angle:

$$\phi_r = \int_{\Omega} F_r\left(\omega_i, P, \omega_o, d\omega_o\right) \cdot \phi_i \tag{2.6}$$

where $\phi_r = d^2\Phi_r\left(P, dA, \omega_o, d\omega_o\right)$ and $\phi_i = d^2\Phi_i\left(P, dA, \omega_i, d\omega_i\right)$. The surface may also emit some photons of it's own. The total power $\phi_o = d^2\Phi_o\left(P, dA, \omega_o, d\omega_o\right)$ arriving at dA' coming from dA is thus the sum of the power reflected at dA with the power emitted at dA:

$$\phi_o = \phi_e + \int_{\Omega} F_r\left(\omega_i, P, \omega_o, d\omega_o\right) \cdot \phi_i \tag{2.7}$$

with $\phi_e = d^2\Phi_e\left(P, dA, \omega_o, d\omega_o\right)$.

Following from equation 2.1, we can rewrite ϕ_i, ϕ_e and ϕ_o as:

$$\begin{aligned} \phi_i &= L_i\left(P, \omega_i\right) \cdot d\omega_i \cdot dA \cdot \cos\theta_i \\ \phi_e &= L_e\left(P, \omega_o\right) \cdot d\omega_o \cdot dA \cdot \cos\theta_o \\ \phi_o &= L\left(P, \omega_o\right) \cdot d\omega_o \cdot dA \cdot \cos\theta_o \end{aligned} \tag{2.8}$$

Replacing these terms in equation 2.7 and dividing by $d\omega_o \cdot dA \cdot \cos\theta_o$ yields:

$$L\left(P,\omega_o\right) = L_e\left(P,\omega_o\right) + \int_{\Omega} \frac{F_r\left(\omega_i, P, \omega_o, d\omega_o\right)}{d\omega_o} \cdot L_i\left(P,\omega_i\right) \cdot \cos\theta_i \cdot \cos\theta_o \; d\omega_i \tag{2.9}$$

Now, since the function F_r is taken for a differential solid angle, we can rewrite it in terms of the BRDF function f_r defined previously. If we consider the BRDF function to also include the $\cos\theta_o$ term, we have $f_r = F_r \cdot d\omega_o \cdot \cos\theta_o$, and we arrive at the commonly known form of the Rendering Equation:

$$L\left(P,\omega_o\right) = L_e\left(P,\omega_o\right) + \int_{\Omega} f_r\left(\omega_i, P, \omega_o\right) \cdot L_i\left(P,\omega_i\right) \cdot \cos\theta_i \; d\omega_i \tag{2.10}$$

which neatly expresses the rendering problem from the point of view of the light receiver. An alternate, more compact representation of the rendering equation can be obtained by introducing the following notation for the integral operator:

$$\mathcal{T}\left(P,\omega\right) = \int_{\Omega} L_i\left(P,\omega_i\right) \cdot f_r\left(\omega_i, P, \omega_o\right) \cdot \cos\theta_i \; d\omega_i \tag{2.11}$$

The rendering equation can now be expressed as:

$$L = L_e + \mathcal{T}L \tag{2.12}$$

The Potential Equation

The rendering equation expresses the problem of image synthesis from the point of view of the observer — it establishes a relationship between the scene observed and the light received from it.

We can also establish a relationship between the light emitted and the effect this light will have on the observer. Consider a given surface surface point P, emitting light in a direction ω_o. The amount of light that reaches the observer from that point can be defined by means of *potential*: a measure of the amount of light emitted at a given point along a given direction which eventually reaches the observer, either directly or after any number of reflections.

Denoting potential emitted from point P in direction ω_o by $W\left(P,\omega_o\right)$, we can apply the same reasoning as before to arrive at the potential equation. For the simple case where the light reaches the observer directly:

$$W\left(P,\omega_o\right) = W_e\left(P,\omega_o\right) \tag{2.13}$$

where $W_e(P,\omega_o)$ denotes the potential emitted at point P in direction ω_o. If P lies on a light source, then

$$W_e\left(P,\omega_0\right) = L_e\left(P,\omega_0\right) \cdot \cos\theta \tag{2.14}$$

where θ represents the angle between the surface normal at P and ω_0.

If we have potential emitted from point P on surface A reflecting on point P' on surface A', the amount of light that reaches the observer is defined by the potential emitted towards P' multiplied by the probability that P' reflects that potential towards the observer:

$$W_r(P, \omega_o) = \int_{\Omega} W(P', \omega_i) \cdot f_r(\omega_o, P', \omega_i) \cdot \cos\theta_i \; d\omega_i \tag{2.15}$$

where $P' = h(P, \omega_o)$ is the point visible from P in direction ω_o. We must also account for the possibility of potential reaching the observer directly. Thus the potential equation becomes:

$$W(P, \omega_o) = W_e(P, \omega_o) + \int_{\Omega} W(P', \omega_i) \cdot f_r(\omega_o, P', \omega_i) \cdot \cos\theta_i \; d\omega_i \tag{2.16}$$

As for the rendering equation, a similar alternate notation can be introduced for the integral operator in the potential equation:

$$\begin{aligned} (\mathcal{T}'W)(P, \omega_0) &= \int_{\Omega} W(P', \omega_i) \cdot f_r(\omega_o, P', \omega_i) \cdot \cos\theta_i \; d\omega_i \\ W &= W_e + \mathcal{T}W \end{aligned} \tag{2.17}$$

where $\mathcal{T}'$ becomes the adjoint operator of $\mathcal{T}$.

2.2.1 Different approaches to solving the Rendering and Potential Equations

The rendering equation expresses the fundamental problem of image synthesis. However, since the unknown radiance term L appears both outside and inside the integral, a solution for it is not easily obtained. Several strategies have been proposed to solve the rendering equation.

Inversion

We can rewrite the shortened form of the rendering equation:

$$\begin{aligned} L &= L_e + \mathcal{T}L \\ L - \mathcal{T}L &= L_e \\ L(1 - \mathcal{T}) &= L_e \\ L &= (1 - \mathcal{T})^{-1} L_e \end{aligned} \tag{2.18}$$

This does not directly provide a solution, since $\mathcal{T}$ is infinite-dimensional and can not be inverted. However, inversion can be approximated with finite-element approaches, but this is seldomly done since the resulting algorithms are numerically unstable and highly complex.

Expansion

In the rendering equation, the right-side L can be substituted by $L_e + \mathcal{T}L$, as per definition:

$$L = L_e + \mathcal{T}L = L_e + \mathcal{T}(L_e + \mathcal{T}L) = L_e + \mathcal{T}L_e + \mathcal{T}^2L = ... =$$

$$= \sum_{i=0}^{n} \mathcal{T}^i L_e + \mathcal{T}^{i+1}L \tag{2.19}$$

This is a Neumann series. Since the integral $\mathcal{T}$ represents a light-surface interaction, for physically-based surfaces this implies a reduction in energy — no surface reflects more light than it receives. Thus, successive applications of $\mathcal{T}$ result in increasingly small energy values, and $\mathcal{T}$ is called a *contraction* operator. This implies that $\lim_{n\to\infty} \mathcal{T}^{n+1}L = 0$, and as such:

$$L = \sum_{i=0}^{\infty} \mathcal{T}^i L_e \tag{2.20}$$

Each term of this series describes a light "bounce" as counted from the light source until it reaches the observer (or the point of measurement): $L_0 = L_e$ represents direct light, $L_1 = \mathcal{T}L_e$ represents one bounce on a non-emissive surface, and so on.

The same approach can be applied to the potential equation, resulting in a series of the form

$$W = \sum_{i=0}^{\infty} \mathcal{T}'^i W_e \tag{2.21}$$

Expansion algorithms have the advantage that they do not require the storage of the whole radiance or potential function at each step, and thus require no approximation by finite-element methods. This implies that they can work with the original scene geometry instead of a tessellated mesh of planar polygons. This also implies that they "forget" previously computed values, and thus can not reuse previous results.

Iteration

We can consider an approximate solution for the rendering equation as a fixed point of the series

$$L_n = L_e + \mathcal{T}L_{n-1} \tag{2.22}$$

Since $\mathcal{T}$ is a contraction, L_n converges for any initial value L_0. This is the approach taken with finite-element methods such as diffuse radiosity. The scene is tessellated into small patches, inside which L_n is taken as constant. For each patch, the initial value L_0 is computed (usually as a direct illumination term), and the algorithm iterates over all

patches computing successive terms of the series until a predefined stopping condition is met.

The main drawback of this approach is that it requires storage of L_{n-1} (i.e., the whole approximation to the radiance function) for the whole scene in order to compute L_n. This implies that a solution is computed even for parts of the scene which are not visible. Also, finite-element methods require tessellation of the scene into small discrete elements and, since the quality of the solution depends on the quality of the tessellation, this becomes a non-trivial task. Finally, each iteration step introduces an approximation error, which can be non-negligible.

2.2.2 Numerical methods applied to the Rendering and Potential Equations

Expansion methods translate easily into ray-casting algorithms, where each successive application of $\mathcal{T}$ corresponds to shooting rays out of a point inside the scene and recursing into the intersection points of the rays. However, the integral $\mathcal{T}$ needs to be computed over a continuous hemisphere, which can not be done directly by shooting discrete rays through the scene.

Monte-Carlo quadrature

Numerical quadrature methods exist which approximate integrals as a summation of sample points from the integrand (i.e., discrete rays traced into the scene) multiplied by a weighting value. The trapezoid rule or the Simpson rule are among the simplest methods of numerical quadrature. They converge quickly for one-dimensional integrals, however the convergence slows down exponentially with the dimensionality of the integral.

Monte-Carlo quadrature avoids this dimensional explosion, which is important since $\mathcal{T}$ is a high-dimensional integral. It reduces the problem of computing the integral to that of computing an expected value, by means of averaging randomly distributed samples.

Suppose that we wish to estimate the integral of a function f by means of Monte-Carlo quadrature. Consider a random variable $\mathbf{z}$ uniformly distributed in a given interval V. The probability density function for $\mathbf{z}$ is thus $p(\mathbf{z}) = \frac{1}{V}$. The expected value of $f(\mathbf{z})$ is:

$$E[f(\mathbf{z})] = \int_V f(\mathbf{z}) \cdot p(\mathbf{z}) \, d\mathbf{z} = \int_V f(\mathbf{z}) \cdot \frac{1}{V} \, d\mathbf{z} = \frac{1}{V} \int_V f(\mathbf{z}) \, d\mathbf{z} \tag{2.23}$$

that is, the expected value of $f(\mathbf{z})$ is proportional to the integral of f inside V. Using the theorem of large numbers, we can approximate $E[f(\mathbf{z})]$ with a summation:

$$E[f(\mathbf{z})] \approx \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{z}_i) \tag{2.24}$$

as long as $\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_i$ are independent randomly distributed samples generated using the same uniform probability density p. It can be shown that this results in an estimator whose upper error bound is independent of the dimensionality of the integral to estimate[Szi99].

Quasi-Monte-Carlo quadrature

Having established that Monte-Carlo approximations are possible using randomly distributed sample points, it is also possible to establish certain characteristics for the random samples in an attempt to obtain faster convergence.

It can be shown that the upper bound for the error of Monte-Carlo quadrature is the product of the variation of the integrand and the discrepancy of the random sample set used for integration[Szi99]. This suggests that reducing the discrepancy of the randomly distributed samples would help reduce the error of Monte-Carlo integration.

Several pseudo-random number series exist which that can satisfy this criteria. One such example is the Halton sequence, which can generate points uniformly distributed in the $[0; 1]$ one-dimensional interval. For generating samples in higher-dimensional regions (rectangles, cubes, etc), several independent Halton sequences may be used, one for each axis. This results in uniformly distributed points within the sampled region, avoiding "clustered" samples which can lead to increased error. Monte-Carlo quadrature using such quasi-random sequences receives the name of Quasi-Monte-Carlo quadrature.

Importance Sampling

Another strategy for reducing the error attempts to reduce the variation of the integrand by doing variable transformations, in order to make the integrand more "flat". This is equivalent to altering the probability density function for the random samples in order to have more samples in regions where the value of the integrand is larger. In other words, we are now assuming a non-uniform probability distribution for p instead. The integrand can be rewritten and approximated as:

$$\int_V f(\mathbf{z})\ d\mathbf{z} = \int_V \frac{f(\mathbf{z})}{p(\mathbf{z})} \cdot p(\mathbf{z})\ d\mathbf{z} = E\left[\frac{f(\mathbf{z})}{p(\mathbf{z})}\right] \approx \frac{1}{N} \cdot \sum_{i=1}^{N} \frac{f(\mathbf{z}_i)}{p(\mathbf{z}_i)} \tag{2.25}$$

Intuitively it is easy to see that p should be larger in the same areas where f is large, which results in more samples being taken in those areas. In fact, it can be shown that the variance is minimal when p is proportional to f [Szi99].

2.3 Ray-casting algorithms as Monte-Carlo integrators

Monte-Carlo quadrature yields an approximate value for the rendering and potential equations, as long as there is a way to evaluate the integrand at random points in the integration domain. A strategy for evaluating the integrand will now be presented.

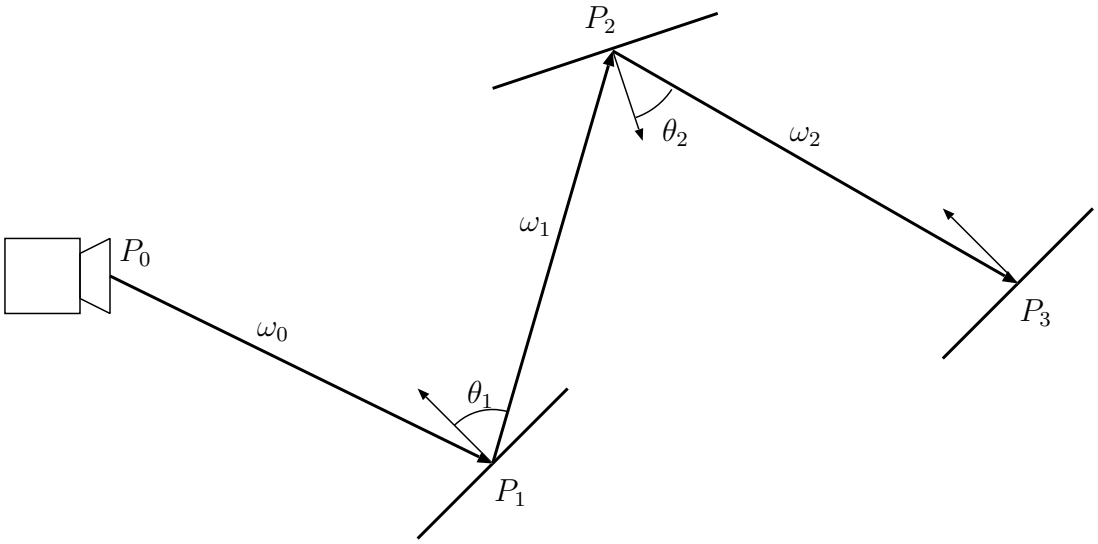


Figure 2.5: Two-step gathering walk. The arrows in $\omega_0, \omega_1, \omega_2$ are reversed from the convention followed in the rendering equation (they do not point away from the surface), to illustrate how the algorithm progresses.

Gathering Walks

We can now take a closer look at the Neumann series for L (equation 2.20).

Consider a simple scene with a light source and two non-emissive surfaces, as in figure 2.5. Let us consider the term $L_2 = \mathcal{T}^2 L_e$, evaluated at the observer. This represents light received after two reflections. Expanding $\mathcal{T}^2$ yields:

$$L_2 = \mathcal{T}_1 \left(\mathcal{T}_2 L_e\right) = \mathcal{T}_1 \left(\int_{\Omega_2} L_e \left(P_3, \omega_2\right) \cdot f_r \left(\omega_2, P_2, \omega_1\right) \cdot \cos\theta_2 \, d\omega_2 \right) =$$

$$= \int_{\Omega_1} \int_{\Omega_2} L_e \left(P_3, \omega_2\right) \cdot f_r \left(\omega_2, P_2, \omega_1\right) \cdot \cos\theta_2 \cdot f_r \left(\omega_1, P_1, \omega_0\right) \cdot \cos\theta_1 \, d\omega_2 d\omega_1 \tag{2.26}$$

If we assume the observer to be a point P_0 in space, we have a single definite starting direction ω_0. Let us fix directions (ω_1, ω_2) in order to evaluate the integrand at a given point in the (bi-dimensional) integration domain. The procedure for computing the value of the integrand becomes:

1. Determine point $P_1 = h\left(P_0, \omega_0\right)$ visible from P_0 in direction ω_0, by casting a ray through the scene and determining the closest intersection to the starting point.

2. From P_1, repeat the process in direction ω_1 to arrive at P_2.

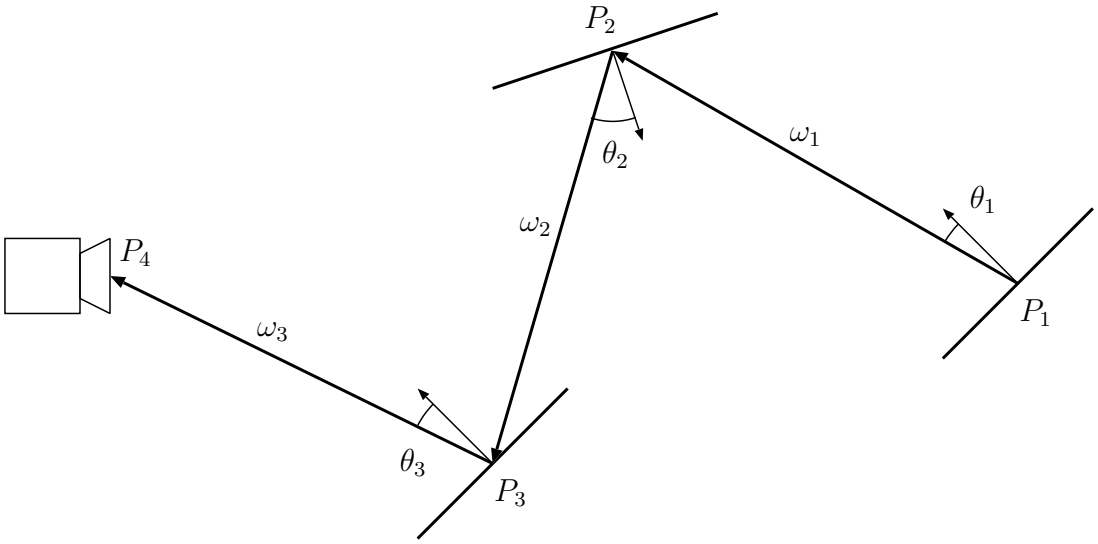


Figure 2.6: Shooting walk of depth two

3. Repeat the process once again from P_2 in direction ω_2, arriving at P_3.

4. Compute the emission intensity of the surface at P_3 in direction $-\omega_2$ and multiply it by the BRDFs and cosine terms for P_2 and P_1.

This allows the evaluation of the integrand in a given point of the integration domain. This process essentially builds a path along which radiance can travel from a light source towards the observer — it is thus known as a *gathering walk*. It is easy to generalise this algorithm from the two dimensions considered to an arbitrary number of dimensions.

Shooting Walks

The same expansion can also be applied to the Neumann series for W (equation 2.21).

Using the same scene (figure 2.5) as for the gathering walk, we shall now consider the expansion of W_2:

$$W_2 = \mathcal{T}'^2 W_e = \mathcal{T}'_1 \left(\mathcal{T}'_2 W_e\right) = \mathcal{T}'_1 \int_{\Omega_1} W_e\left(P_1, \omega_1\right) \cdot f_r\left(\omega_1, P_2, \omega_2\right) \cdot \cos\theta_2 \, d\omega_1 \tag{2.27}$$

Since P_1 lies on a light source, we have

$$W_2 = \mathcal{T}'_1 \int_{\Omega_1} L_e\left(P_1, \omega_1\right) \cdot \cos\theta_1 \cdot f_r\left(\omega_1, P_2, \omega_2\right) \cdot \cos\theta_2 \, d\omega_1 \tag{2.28}$$

which further expands to

$$W_2 = \int_{\Omega_2} \int_{\Omega_1} L_e\left(P_1, \omega_1\right) \cdot \cos\theta_1 \cdot f_r\left(\omega_1, P_2, \omega_2\right) \cdot \cos\theta_2 \cdot f_r\left(\omega_2, P_1, \omega_3\right) \cdot \cos\theta_3 \, d\omega_1 d\omega_2 \quad (2.29)$$

Following the same reasoning, we can fix a starting point P_1 and two directions (ω_1, ω_2) in order to compute the value of the integrand at a given point in the integration domain[3]:

1. Compute the cosine-weighted emission of P_1 along ω_1.
2. Determine point P_2 visible from P_1 along ω_1 by casting a ray through the scene and computing the closest intersection.
3. Repeat the procedure at P_2 along ω_2 to determine point P_3.
4. Determine if the observer can be reached from P_3, and compute direction ω_3.
5. Multiply the emission at P_1 by the accumulated BRDFs and cosine terms.

This procedure is also straightforward to generalise for an N-dimensional integral, thus it is possible to evaluate the integrand at a given point in the integration domain. As opposed to gathering walks, this procedure generates a path along which light is "shot" from the light source reaching the observer — a *shooting walk*.

Russian Roulette

We now have the means to evaluate the integrands on both the rendering and the potential equations at defined points in the integration domain. However, the integral is still infinite-dimensional, thus a suitable strategy for terminating evaluation of the integrand must be found.

A very naive strategy would be to simply truncate the Neumann series at a given depth limit. This, however, introduces some bias, especially in the case of highly reflective scenes (where $\mathcal{T}$ is "not very contractive"). A better alternative is the Russian Roulette.

The Neumann series expands to a series of integrals with the following general form:

$$L = \int_{\Omega} L_i\left(P, \omega_i\right) \cdot f_r\left(\omega_i, P, \omega_o\right) \cdot \cos\theta_i \, d\omega_i \quad (2.30)$$

In the context of Monte-Carlo integration, the parameters are random samples. Assume $w = f_r\left(\omega_i, P, \omega_o\right) \cdot \cos\theta_i$. Rewriting L yields:

$$L = \int_{[0,1]^2} w\left(\mathbf{z}\right) \cdot L_i\left(\mathbf{z}\right) \, d\mathbf{z} \quad (2.31)$$

[3]Direction ω_3 is already fixed since by definition the observer is considered as a single point in space.

Algorithm 1 First-hit ray tracing

```
function Trace(ray)
  P = intersect-with-scene(ray)
  if P not found
    return 0
  else
    Le = emission(P,-ray.direction)
    Ll = direct-lightsource-sample(P)
    return Le + Ll
  endif
end
```

that is, we are now integrating over a domain of bi-dimensional random samples, which yield random directions on the unit hemisphere after applying the proper transformation (as per the spherical coordinate system defined earlier). Monte-Carlo quadrature would approximate the value of this integral as an average of the value of the integrand taken at random sample points. The idea of Russian roulette is to further randomise this process: at each step, another independent random sample is taken in order to decide whether to actually evaluate the integrand or instead assume it to be zero, thereby terminating the process. We can express this within the integrand by introducing a different function L_{ref}:

$$L_{ref} = \begin{cases} \frac{w(\mathbf{z}) \cdot L_i(\mathbf{z})}{s} & \text{if} \quad \text{sample is used} \\ 0 & \text{if} \quad \text{sample is not used} \end{cases} \tag{2.32}$$

where s is the random sample used to decide whether to "kill" the integral. The expected value of L_{ref} then becomes:

$$E\left[L_{ref}\right] = s \cdot E\left[L_{ref} | \text{sample is used}\right] + (1-s) \cdot E\left[L_{ref} | \text{sample is not used}\right] =$$

$$s \cdot E\left[\frac{w \cdot L_i}{s}\right] + (1-s) \cdot 0 = E\left[w \cdot L_i\right] = L \tag{2.33}$$

Therefore, Russian roulette introduces no bias to the estimator if the value of the integrand at each sampled point is divided by the random sample s. The same reasoning can be applied to the potential equation, leading to the same results.

We can now present some well-known rendering algorithms and discuss how they can fit into this framework.

2.3.1 First-Hit Ray Tracer

First-hit ray tracing is an extremely primitive form of a gathering-walk algorithm. Starting from the observer, it simply computes the first visible point for each pixel and uses

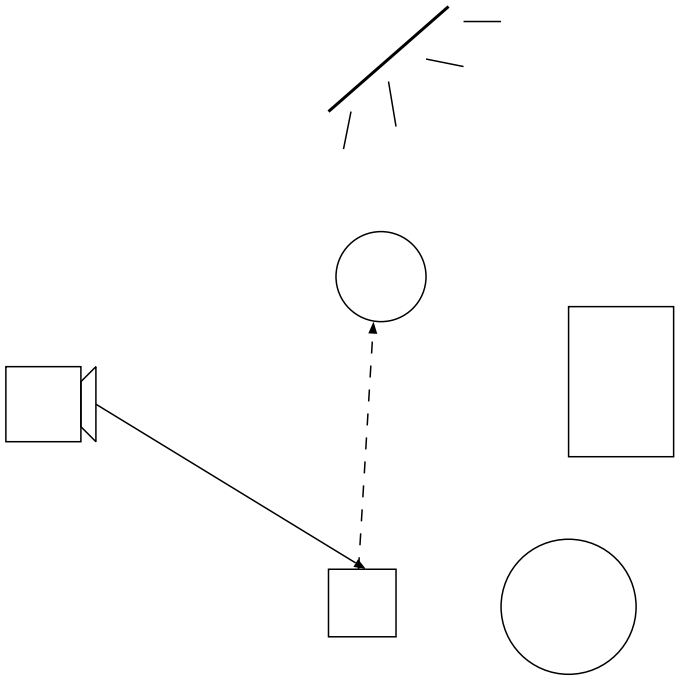


Figure 2.7: First-Hit Ray-Tracing

shadow rays to sample the light sources from that point in order to determine the outgoing radiance towards the observer (see figure 2.7). This is the same as simplifying the rendering equation to the form

$$L\left(P,\omega_0\right) = L_e\left(P,\omega_o\right) + \int_{\Omega} L_{light}\left(h\left(P,-\omega_i\right),\omega_i\right) \cdot f_r\left(\omega_i,P,\omega_o\right) \cdot \cos\theta_i \, d\omega_i \tag{2.34}$$

where L_{light} is the emission at a light source. Algorithm 1 shows pseudo-code which describes the function for tracing a ray through the scene for a first-hit ray-tracer. This very primitive algorithm is effectively what is implemented in most real-time rendering toolkits, such as OpenGL or OpenRT. The actual implementation may or may not involve ray tracing (OpenGL is a polygon-based rasteriser, however this is still equivalent to a first-hit ray-tracer), and in most cases the light source sampling is heavily simplified. In OpenGL, for example, the light sources are assumed to be points in space (the light source sampling is done in a single defined direction) and the geometry function h is omitted (which means that it does not render shadows).

2.3.2 Whitted Ray Tracer

Whitted ray-tracing (figure 2.8) is a simplified gathering-walk algorithm, which follows multiple light bounces only for perfectly reflective or translucent surfaces. It stops as soon as it reaches a surface with a diffuse component.

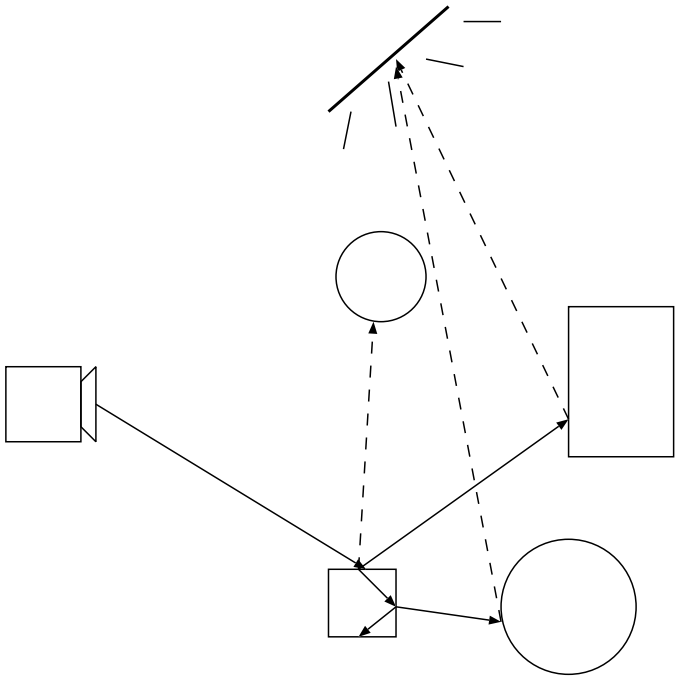


Figure 2.8: Whitted Ray-Tracing

Algorithm 2 Whitted ray-tracing

```
function Trace(ray)
  P = intersect-with-scene(ray)
  if P not found
    return 0
  else
    Le = emission(P,-ray.direction)
    Llight = direct-lightsource-sample(P)
    L = Le + Llight * BRDF(P, Llight)
    if P.Kr > 0
      L += P.Kr * Trace(reflect(ray, P))
    endif
    if P.Kt > 0
      L += P.Kt * Trace(refract(ray, P))
    endif

    return L
  endif
end
```

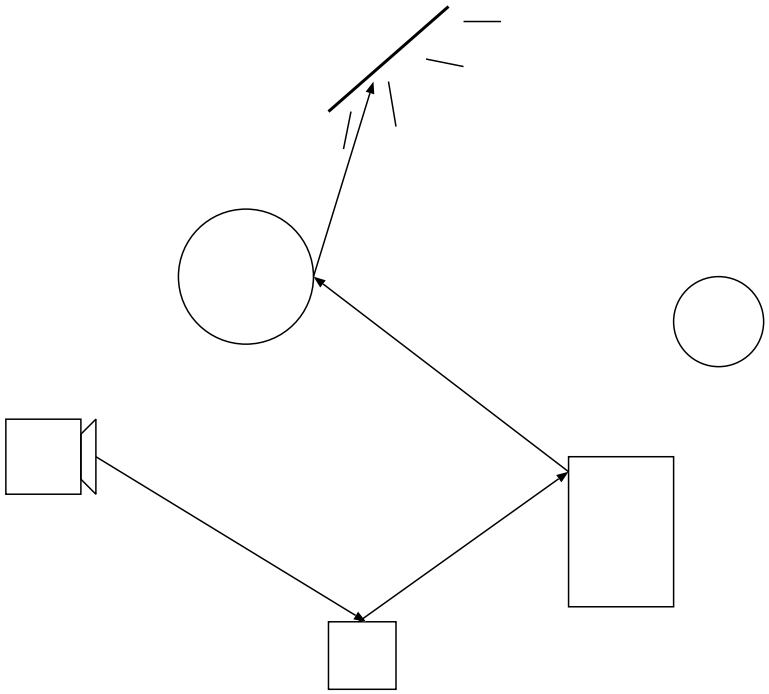


Figure 2.9: Path tracing

Starting from the observer, this method traces light paths through the scene. At each step along the way, it computes a light source contribution by testing for light source visibility using a "shadow ray". Since it only follows perfectly specular reflections or refractions, it generates deterministic paths. This is equivalent to simplifying the rendering equation as:

$$L(P,\omega_o) = L_e(P,\omega_o) + \int_\Omega L_{light}(h(P,-\omega_i),\omega_i)\cdot f_r(\omega_i,P,\omega_o)\cdot\cos\theta_i\, d\omega_i + \\ + k_r\cdot L(h(P,\omega_r),-\omega_r) + k_t\cdot L(h(P,\omega_t),\omega_t) \tag{2.35}$$

where k_r, k_t are the specular reflection and transmission coefficients for each surface (which usually depend on several parameters not shown here, such as angle of incidence, outgoing angle and position on the surface). That is, it adds the emission term for the surface, the light source sampling term and the incoming radiance through the perfectly specular paths scaled by the k_r and k_t coefficients. The function for tracing a ray through the scene using this method would be as shown in algorithm 2.

2.3.3 Path Tracer

Path tracing is the most straightforward application of the concepts discussed earlier for a gathering-walk type algorithm. It incorporates importance-based sampling and

Algorithm 3 Path tracing

```
function Trace(ray, filterSoFar)
  P = intersect-with-scene(ray)
  if P not found
    return 0
  endif

  if P is lightsource
    return filterSoFar * emission(P, -ray)
  else
    (newray, t) = randomBRDFSample(ray, P)
    a = luminosity(filterSoFar)
    if russian-roulette(a) then
       nextFilter = filterSoFar * W(-ray, P, newray) / (a * t)
       return Trace(ray, nextFilter)
    else
       return 0
  endif
end
```

Russian roulette termination of the generated paths.

This method essentially creates random paths along which radiance can reach the observer. At each step along the path, instead of spawning new rays it uses importance sampling to choose a new direction for the ray. Ultimately, this results in a random path along which light emanating from a light source can reach the observer (figure 2.9).

The probability density function used for the importance sampling should be proportional to the value of the integrand. In most cases, the function used is proportional to the BRDF of the surface. This works since the integral is a product of the BRDF and the incoming radiance, which means that it should be (roughly) proportional to the BRDF function.

When using Russian roulette, the random walk is continued with a probability a_i. The probability is usually selected as a value that reflects the amount of light absorbed through the path so far, which can be computed by multiplying the BSDF terms of the intersection points. This increases the probability of terminating paths which no longer contribute significant radiance to the integral. Paths are also usually terminated once they reach a light source.

Let t_i be the density used for importance sampling. The radiance accumulated along a path is measured by the path tracer as

$$L = L_{e_1} + \frac{w_1}{t_1 \cdot a_1} L_{e_2} + \frac{w_1}{t_1 \cdot a_1} \cdot \frac{w_2}{t_2 \cdot a_2} L_{e_3} + ... \tag{2.36}$$

where w_i are the cosine-weighted BSDF terms, and L_{e_i} are the emission terms for each intersection point (usually 0 except at light sources). This leads to very high variance

in case the light sources are small, since the probability of reaching them decreases. For the same reason, the path tracer is also not very well suited for rendering caustics, especially when they are small. Darker caustics are generally not easily visible in the output, and bright caustics are rendered with high variance.

The 'Trace' function for the path tracer would be implemented as per algorithm 3. It should however be noted that for the case of perfectly specular or translucent materials the BRDF takes a different (deterministic) form, which must be treated differently.

An alternate version of the path tracing algorithm incorporates direct light-source sampling at each step along the path. This helps to reduce the variance, especially in the case of small light sources. However it introduces complexity in that it mixes two different kinds of importance sampling, requiring some care in the weighting factors. A well-known solution for this problem is proposed in [VG95]. Figure 2.11 illustrates the differences in the output of these two techniques.

Although path tracing is a very simple "brute-force" algorithm, it offers several advantages over other techniques. Being very simple means that it is easier to implement and verify than other algorithms. It also imposes very little constraints on the scene to render, achieving correct results on mostly any scene, given enough computational time. It does not, however, provide very fast convergence.

There are variants of this algorithm, such as bidirectional path tracing and Metropolis light transport, which generate paths starting from both the observer and the light sources, and try to combine them in order to generate a full path for radiance to reach the observer. This generally increases convergence speed, however such algorithms are several orders of magnitude more complex and extremely difficult to implement.

2.3.4 Photon Tracer

Photon tracing is a shooting-walk algorithm which can be seen as the inverse of Whitted ray-tracing. It makes use of the same simplifying assumptions[4]. The algorithm starts at light sources and shoots random particles, or "photons". It then follows their path through the scene, stopping as soon as a surface with a diffuse component is reached — when this happens, the algorithm deposits the energy carried by the photon at that point, in an appropriate data structure. In pseudo-code, the 'Shoot' function for tracing a photon through the scene could be described as per algorithm 4, where k_r and k_t have the same meaning as for the Whitted ray tracer, and k_d is the diffuse component.

This algorithm essentially computes an approximate solution to the potential equation. That is, it approximates a radiance function for the whole scene, which can then be used to estimate the radiance reaching the observer from a given point. This estimate is usually used by gathering walk algorithms such as the Whitted ray tracer in order to estimate indirect illumination: at each intersection with a diffuse surface, the data

[4] 'Photon Tracing' can mean different things for different authors. One common alternate definition describes a full algorithm for image synthesis which can be seen as the exact inverse of path tracing: rays are shot randomly from light sources, and whenever they reach the observer their contribution to a given pixel on the image is recorded. The description provided here matches the ART implementation.

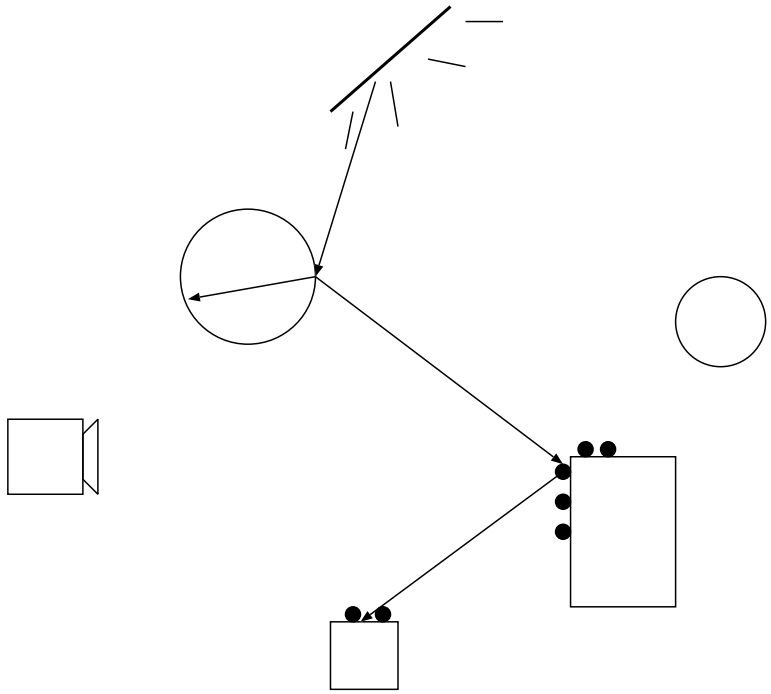


Figure 2.10: Photon Tracing

Algorithm 4 Photon tracing

```
function Shoot(ray, power)
  P = intersect-with-scene(ray)
  if P not found
    return
  endif
  if P.Kd > 0
    Deposit(P, power)
    newray = randomreflect(ray, P)
    Shoot(newray, P.Kd * power)
  endif

  if P.Kt > 0
    Shoot(refract(ray, P), P.Kt * power)
  endif
  if P.Kr > 0
    Shoot(reflect(ray, P), P.Kr * power)
  endif
end
```

structure generated by the photon tracer is checked, and a radiance estimate is computed based both on direct illumination and data recovered from the photon tracer.

The main problem with this approach is that of reconstructing the radiance function from the estimate. This function is generally highly complex, with severe discontinuities in shadow areas, and reconstructing it from discrete samples is usually extremely difficult. The choice of data structure used to store this information (usually light maps or photon maps) has severe implications on the reconstruction performance as well as storage requirements.

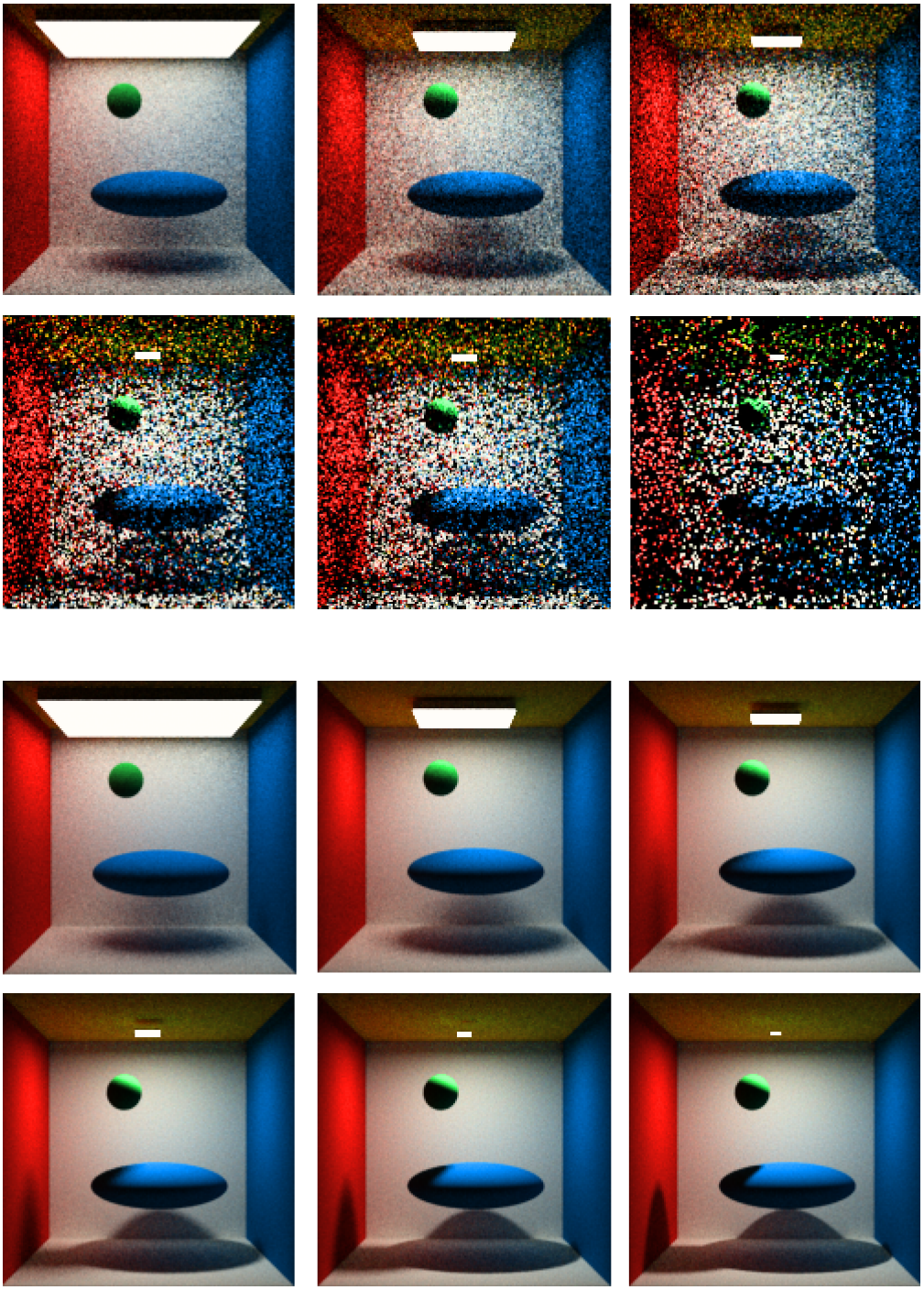


Figure 2.11: Comparison of simple path tracing and path tracing with light source sampling. The top two rows show the same scene rendered with 6 different light sources (increasingly smaller from left to right and top to bottom), using the simple version of the path tracing algorithm. The bottom two rows show the same scene rendered using the path tracer with light source sampling. (Rendered with ART, images courtesy of Alexander Wilkie)

CHAPTER 3

Light Propagation in Dielectric Materials

In order to properly simulate dispersion effects, besides having knowledge of the theoretical framework behind physically based rendering in general, it is also essential to understand the physical principles which govern dispersion itself. These principles shall now be presented.

In physics, a dielectric is defined as a substance which is highly resistant to the flow of electric current, such that it causes space to "seem bigger" or "smaller" to electrical charges or electromagnetic waves flowing through it. When an electromagnetic wave, such as light, travels through a dielectric, its velocity is reduced and it behaves as if it had a shorter wavelength.

This is the case with optically translucent materials. Light entering a translucent material will slow down, causing all sorts of interesting optical phenomena such as refraction and dispersion.

The optical principles that govern these phenomena will now be reviewed, focusing on those that cause dispersion of light. This text will mostly follow the optical laws, as opposed to being a full electromagnetic treatment of light propagation, since this is enough for the purposes of simulating dispersion in translucent materials within the context of image synthesis.

3.1 Reflection and Refraction

When a beam of light encounters an interface between two translucent materials, it is generally split into two beams. One of them is reflected at the media interface and travels in the mirror direction. This is generally called *specular* reflection, when occurring at a perfectly smooth surface (as opposed to *diffuse* reflection, which occurs at a rough surface and causes the reflected light to scatter in several directions).

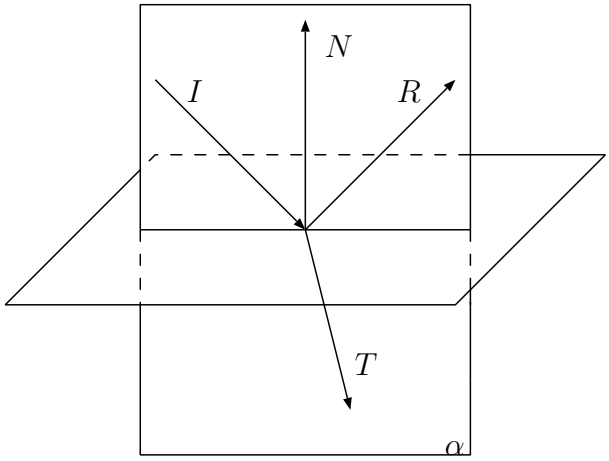


Figure 3.1: The plane of incidence. The incident ray I, the refracted ray T, the reflected ray R and the geometric normal N at the point of incidence all lie in the same plane α

The second beam suffers a change of direction and speed, and penetrates the material. This phenomenon is called *refraction*. The incident, reflected, refracted and surface normal vector lie on the same plane, called the *plane of incidence* (see figure 3.1).

Reflection

For reflection in the mirror direction, the angle between the incident ray and the normal is the same as the angle between the normal and the reflected ray. This, combined with the fact that the reflected vector must lie on the plane of incidence, allows for the following vector formula for computing the reflection vector $\vec{R}$, assuming $\vec{I}$ (the incident vector) and $\vec{N}$ (the surface normal vector) are unit-length vectors:

$$\vec{R} = 2\left(-\vec{I} \cdot \vec{N}\right) \cdot \vec{N} + \vec{I} \tag{3.1}$$

Reflection is thus not a wavelength-dependant effect.

Refraction

Refraction is caused by light slowing down as it penetrates a material with a different *refractive index*. This index is essentially the factor by which an electromagnetic wave is slowed down upon entering the media as compared to the propagation speed in vacuum:

$$n = \frac{c_0}{v} \tag{3.2}$$

where n is the material's index of refraction, c_0 is the propagation speed in vacuum and v is the *phase-velocity* of the wave inside the material (that is, the velocity at which

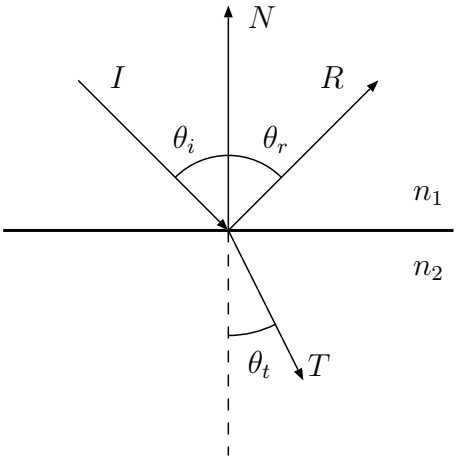


Figure 3.2: Refraction of a beam of light at the interface between two media

a given phase of the wave propagates). This implies that the index of refraction of vacuum is 1.0, and since nothing can move faster than light in a vacuum, $n > 1.0$ for all other materials[1].

The relationship between the angle of incidence and angle of refraction is obtained through Snell's law (figure 3.2):

$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{n_1}{n_2} \tag{3.3}$$

Thus, light penetrating a denser material (higher n) deviates towards the surface normal, and light exiting a denser material into a less dense media (lower n) deviates away from the surface normal. Above a *critical angle* $\theta_i = \arcsin\left(\frac{n_2}{n_1}\right)$, for $n_2 < n_1$ (that is, light exiting a dense material into a less dense one), Snell's law predicts that the refraction direction would actually prevent light from exiting altogether, instead making it travel along the boundary between the two materials. What effectively happens is that refraction does not occur altogether, and all the light is reflected. This phenomenon is known as *total internal reflection*. Fiber-optic cables, for instance, make use of this phenomenon to enable light to travel through them: they are built in such a way that a beam of light travelling from the inside towards the outside of a fiber-optic cable always strikes the interface at such an angle as to cause total internal reflection, reflecting it back inside.

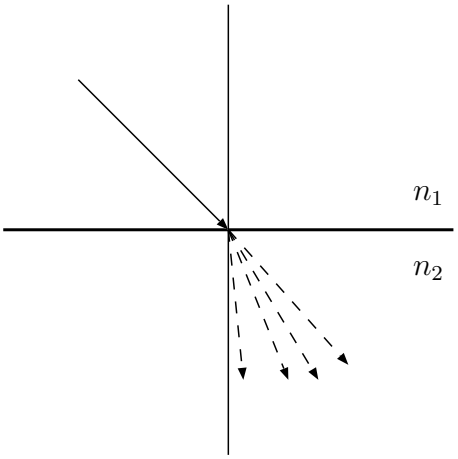


Figure 3.3: Dispersion of light at an interface between two translucent materials. A single polychromatic beam of light is separated into monochromatic rays, where each follows a different refraction direction as governed by the wavelength-dependant version of Snell's law

Wavelength Dependency

Earlier, it has been stated that the index of refraction for a given material depends on the phase-velocity of light propagating inside the material. This is the speed at which a given phase on the wave (for instance, a crest) propagates inside the material. For longer wavelengths, it is easy to see that this speed decreases, therefore increasing the refraction index. Thus, n may be seen as a wavelength-dependant function:

$$n(\lambda) = \frac{c_0}{v(\lambda)} \tag{3.4}$$

which leads us to a wavelength-dependant version of Snell's law:

$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{n_1(\lambda)}{n_2(\lambda)} \tag{3.5}$$

This implies that different wavelengths are refracted in different directions, causing dispersion (figure 3.3). This causes the well-known "rainbow" effect of a beam of light travelling through a translucent prism.

[1] This is actually not true for every possible wavelength in every material, however it holds true for the majority of translucent materials and for the visible spectrum. So-called *meta-materials* exist which effectively refract visible light "backwards", where the refraction vector effectively follows a direction with a negative angle towards the surface normal.

3.2 Fresnel Terms

Earlier it was stated that a beam of light encountering an interface between two translucent materials is divided into a reflected beam R and a refracted \or transmitted beam T. The amount of light which is reflected and refracted is governed by the Fresnel terms.

The Fresnel terms are a consequence of applying Maxwell's formula to the radiant intensities of light following the reflection and refraction paths. The resulting expression can then be decomposed into two components, one with polarisation parallel to the plane of incidence (*p-polarised*), and another with perpendicular polarisation (*s-polarised*, from the German "senkrecht", meaning perpendicular). This derivation is outside the scope of this text, however the end result is important.

Let $r_{\parallel}$ and $t_{\parallel}$ be the relative intensities of the p-polarised reflected and refracted beams; let and $r_{\perp}$ and $t_{\perp}$ represent the relative intensities of the s-polarised reflected and refracted beams. For unpolarised light we have

$$\begin{aligned} R &= \frac{r_{\parallel} + r_{\perp}}{2} \\ T &= \frac{t_{\parallel} + t_{\perp}}{2} \end{aligned} \tag{3.6}$$

Conservation of energy implies that

$$R + T = 1 \tag{3.7}$$

that is, the amount of reflected and refracted light must not be more than the incoming light. The actual reflected and refracted components are determined by the Fresnel terms:

$$\begin{aligned} r_{\parallel} &= \left[\frac{\tan(\theta_t - \theta_i)}{\tan(\theta_t + \theta_i)}\right]^2 \\ t_{\parallel} &= 1 - r_{\parallel} \\ r_{\perp} &= \left[\frac{\sin(\theta_t - \theta_i)}{\sin(\theta_t + \theta_i)}\right]^2 \\ t_{\perp} &= 1 - r_{\perp} \end{aligned} \tag{3.8}$$

It should be noted that, when $\theta_t + \theta_i = \frac{\pi}{2}$, the value of $\tan(\theta_t + \theta_i)$ in the expression for $r_{\parallel}$ goes to infinity. The actual meaning of this is that no p-polarised light is reflected.

We can determine the incident angle θ_B for which no p-polarised light will be reflected. Rearranging the previous condition:

$$\theta_t = \frac{\pi}{2} - \theta_B \tag{3.9}$$

Applying Snell's law yields:

$$\begin{aligned} n_1 \sin(\theta_B) &= n_2 \sin\left(\frac{\pi}{2} - \theta_B\right) \\ n_1 \sin(\theta_B) &= n_2 \cos(\theta_B) \\ \frac{\sin(\theta_B)}{\cos(\theta_B)} &= \frac{n_2}{n_1} \\ \theta_B &= \arctan\left(\frac{n_2}{n_1}\right) \end{aligned} \tag{3.10}$$

This angle θ_B is called the *Brewster* angle, and equation 3.10 is commonly known as *Brewster's Law*.

Wavelength Dependency in Fresnel Terms

Using Snell's law and trigonometric identities, we can rewrite the Fresnel terms $r_{\parallel}$and $r_{\perp}$:

$$\begin{aligned} r_{\parallel} &= \left[\frac{n_1(\lambda)\cos(\theta_i) - n_2(\lambda)\cos(\theta_t)}{n_1(\lambda)\cos(\theta_i) + n_2(\lambda)\cos(\theta_t)}\right]^2 \\ r_{\perp} &= \left[\frac{n_1(\lambda)\cos(\theta_t) - n_2(\lambda)\cos(\theta_i)}{n_1(\lambda)\cos(\theta_t) + n_2(\lambda)\cos(\theta_i)}\right]^2 \end{aligned} \tag{3.11}$$

This shows that the Fresnel terms also depend on the index of refraction for the materials. This in turn depends on the wavelength, thus the Fresnel terms are different for different wavelengths. This is an important result which will affect the implementation of dispersion rendering capabilities.

3.3 The Sellmeier Equation

Having obtained the optical laws which govern the effect of light dispersion, we now have to be able to determine the index of refraction for a given material for a desired wavelength.

The index of refraction is a non-linear, non-monotonous function of the wavelength. A sample representation of this is shown in figure 3.4. While it is quite easy to explain, physically exact analytical models for this phenomenon are difficult to find.

A good empirical approximation to this for the visible spectrum is the *Sellmeier equation*:

$$n^2(\lambda) = 1 + \frac{B_1\lambda^2}{\lambda^2 - C_1} + \frac{B_2\lambda^2}{\lambda^2 - C_2} + \frac{B_3\lambda^2}{\lambda^2 - C_3} \tag{3.12}$$

where $B_1, B_2, B_3, C_1, C_2, C_3$, the *Sellmeier coefficients*, are obtained through measurements for a given material. This is actually a specific form of the Sellmeier equation,

where the most general form is a series with as many coefficients as required. It can also be used with as little as 4 coefficients (B_1, B_2, C_1, C_2) yielding less precise but still acceptable results.

A plot of the Sellmeier equation for a common material is shown in figure 3.5. The variation of n with the wavelength is what generates dispersion — materials with a steeper variation will exhibit a more pronounced dispersion effect.

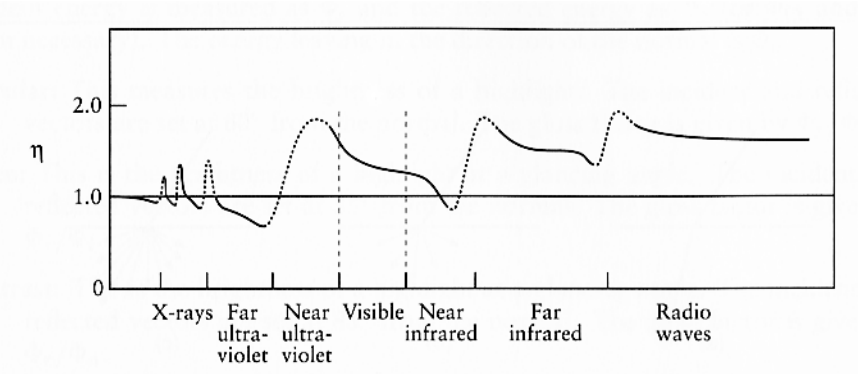


Figure 3.4: Schematic representation of the index of refraction plotted as a function of wavelength over a range of the electromagnetic spectrum. Reprinted from [Glas95]

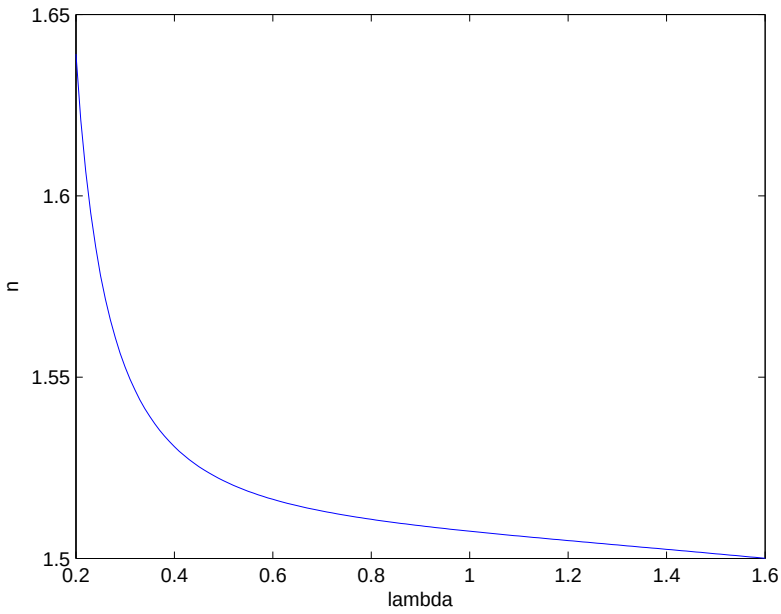


Figure 3.5: Plot of the Sellmeier equation for borosilicate crown glass BK7. The index of refraction (Y axis) is plotted as a function of wavelength (X axis)

CHAPTER 4

Prior Art

Dispersion is an interesting optical phenomenon which has been extensively studied. The optical laws governing dispersion are quite well understood, and even relatively simple.

The simulation of dispersion in Computer Graphics has also been investigated to some extent, and some literature is available. However, actual implementations of dispersive rendering algorithms are hard to come by. One reason for this is that proper dispersive rendering requires a renderer which can deal with a spectral representation of light, rather than the usual RGB-colour based approach. A spectral renderer is thus not only more complex to implement, but also more computationally expensive.

4.1 Previous Research

The first implementation of dispersive rendering was presented in [Tho86]. This was implemented on a standard RGB-colour based ray-tracer. The technique implemented consisted of dividing the spectrum into two sub-rays upon encountering a dispersive interface. Each sub-ray would correspond to one endpoint of the spectrum carried by the incident ray and would be refracted according to the corresponding wavelength-dependant index of refraction. The angular spread between the two sub-rays would then be analysed, and if necessary (if the angular spread was too large) the incident ray would be adaptively subdivided into more sub-rays, up to a pre-defined limit.

While this method would yield good results in most cases, there is a very big trade-off in terms of efficiency when applying a limit to the number of sub-rays generated. This method also has problems handling the case where different sub-rays intersect different faces of the object.

This model was improved upon in [YKIS88]. By carefully inspecting the "common case" occurrences for dispersion, the authors found that in most cases an initial subdivi-

sion into 3 sub-rays is enough to generate realistic images. Furthermore, they propose a model in which the ray carries information about the angular spread as well, and whenever necessary can be further subdivided (for example, when the angular spread extent for one sub-ray intersects an edge of the model, meaning that different sub-rays intersect different faces in the model). When subdividing the ray, the parameters for the new ray are computed based on linear interpolation of the parameters of the old rays. This method improves computational efficiency and solves the problem of the previous method where different sub-rays strike different faces of the model.

Finally, [SFD00] proposes the use of a composite spectral model in order to render dispersive effects. In this article, the authors describe a spectral rendering model which samples the spectrum as a sum of a smooth component and a series of "spikes". This system is easy to extend in order to support dispersive ray tracing, since a monochrome ray can be represented as a spike with no smooth component. The proposed approach does dispersive ray tracing with no adaptive subdivision, since this would have little influence on the final result. The effects of "dispersive aliasing" (resulting from too few monochromatic samples taken at a dispersive interface) are also presented.

4.2 Implementations

Readily-available software which can do dispersive rendering is not common, and can be broadly divided in two categories: RGB-colour based renderers and spectral renderers.

RGB-colour based rendering techniques are the most common. Most of the currently available renderers (either commercial or open-source) are RGB-colour based, meaning that they deal with RGB colour values only.

One of the few examples of dispersion-capable RGB renderers is the open-source *Persistence of Vision Raytracer* (POV-Ray), available at `http://www.povray.org/`. This renderer simulates light dispersion in RGB colour space, resulting in obvious colour banding. Nevertheless, it is one of the few available implementations. A sample image rendered with POV-Ray is presented in figure 4.1, with a blown-up of a dispersive effect detail within the image.

Another option which recently became available is *Maxwell Render*, from Next Limit Technologies (`http://www.maxwellrender.com/`). Released in April of 2006, Maxwell Render is a physically-based spectral renderer which can reportedly simulate most kinds of light interactions, including dispersion. Being a spectral renderer, it is reasonable to expect good results with dispersive rendering, with no visible banding as in POV-Ray. This has not been tested, however, as this renderer is not freely available.

As far as we know, these are the only readily-available implementations of dispersive renderers.

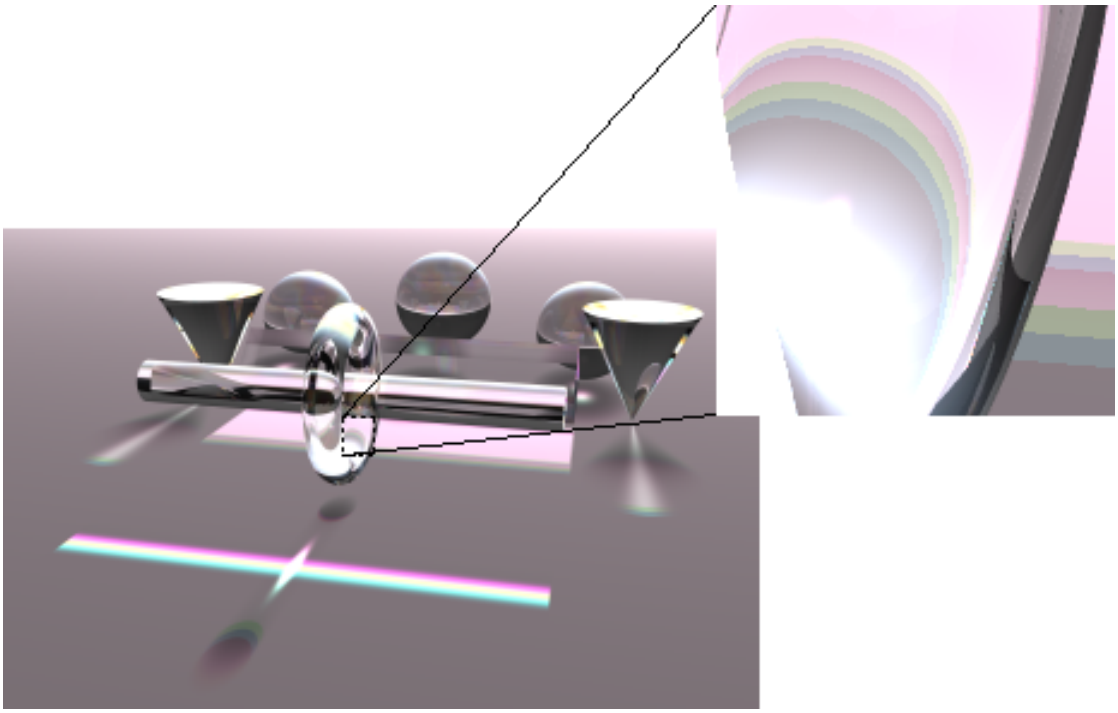


Figure 4.1: Image computed using the Whitted ray tracer from POV-Ray, exhibiting dispersive materials.

CHAPTER 5

Dispersion in the *Advanced Rendering Toolkit*

Having described the underlying theory in the previous sections, this chapter details what work has been done in order to achieve dispersive rendering within the *Advanced Rendering Toolkit*.

5.1 ART Overview

The *Advanced Rendering Toolkit* (ART) is a spectral renderer in development at the Institute for Computer Graphics of the *Technische Universität Wien*. It is intended to be a complete physically correct rendering system, which does image synthesis based solely on the underlying physical principles of light propagation.

Although the project initially started in June 1996, it is still under active development.

5.1.1 Architecture

ART uses a highly modular architecture. It is split into 22 different modules, each of which exists as a static library after compilation. This allows the ART rendering engine to be incorporated into other programs if so desired.

Most of ART is written in Objective-C, which provides object-oriented programming as a simple superset of ANSI C. It allows C and Objective-C code to fully inter-operate, which provides for a very flexible OO environment. Most of the performance-critical parts of ART are actually written in C, where the parts which are deemed to benefit from better structuring are written in Objective-C. Several other factors influenced the decision to use Objective-C, such as the general lack of compatibility between C++ compilers, the somewhat simple and elegant OO implementation present in Objective-C, the presence of a runtime system and the general familiarity of the first developers

of ART with the NeXTStep APIs, which were written in Objective-C and were found to be very elegant in practice.

ART thus exists as a collection of libraries and programs which make use of them, serving as an interface to the end user. The functionality needed for generating images is generally split among several relatively simple programs: a scene compiler, the actual renderer and a gamut- and tone-mapper which can output image files. However, the main `artist` binary encapsulates the whole process, from reading and compiling the scene file up to outputting a final RGB image file.

5.1.2 The Spectral Rendering Approach

ART is a spectral renderer. As such, it (usually) works not with RGB colour values but with a representation of light spectrum. One of the main issues with a spectral renderer is that of actually representing spectral samples.

Spectral Samples

Several approaches exist for this representation. One is by means of basis functions, which provide a "smooth" continuous approximation to a given spectrum. This approach is, however, not suitable for representing a wide range of light sources, such as fluorescent lamps, which consist essentially of "spikes" in the spectral distribution.

The "opposite" approach is a point-sampled spectrum. This simple method samples the spectrum at pre-determined wavelengths. It can be coupled with actual wavelength information in order to more precisely define spikes or monochromatic samples. This is the approach taken within ART, since it reduces computational complexity while still maintaining reasonable accuracy. It also makes it easier to trade off accuracy with speed, by simply increasing the number of sampling points in a spectral sample. ART makes this possible by abstracting the data structures used to store spectral samples, in such a way that any distribution of spectral sample information may be achieved by simply defining compile-time constants. By default, ART supports several spectral sample distributions: Spectrum8 (using 8 samples spread across the visible spectrum), Spectrum16 (16 samples), Spectrum45 (45 samples) and RGB (which takes 3 samples at wavelengths corresponding to red, green and blue colours).

Hybrid spectral representation methods exist in the literature, such as for instance the method proposed in [SFD00], where smooth basis functions are combined with a series of "spikes". This allows for representation of most spectral emission patterns accurately, however the computational cost rises. These methods are not implemented in ART.

The Spectral Rendering Pipeline

The output of a spectral renderer such as ART is not an actual image. Rather, what the renderer generates is a frame-buffer filled with spectral samples, which correspond to the predicted radiance received at each point in the simulated sensor.

Converting this information into an actual RGB image is not straightforward, as there is usually no direct correlation between spectral samples and RGB values. The raw spectral samples are first converted into the CIE XYZ colour space, where gamut and tone mapping are applied. The gamut mapper essentially compresses the colour range to fit within the CIE XYZ colour gamut, while the tone mapper compresses the luminance to fit within the acceptable luminance range. Finally, the CIE XYZ colour values are converted to RGB colour values and the final image is generated.

ART Scene File Format

The scene file format for ART consists essentially of Objective-C code. A scene description file is an Objective-C source file with a specially-named function which, when called, is expected to set up the scene and rendering parameters, returning a scene object. This object contains information on scene geometry and material properties, the camera used to render it, and the *action sequence* which will perform the rendering.

Every step in the rendering process within ART is described by a corresponding action object. These objects define which steps are taken (such as setting up acceleration structures, running pre-processing steps such as photon tracing, which renderer should be used to synthesise the image, whether to run the tone-mapper, etc) and also the parameters for each step. These actions are stored in a stack-based data structure called the action sequence. ART then executes this action sequence by popping the stack and executing each action in order. It is the job of the user to provide a valid and meaningful action sequence in the scene file.

5.2 Proposed Approach for Dispersion Rendering

The proposed approach for implementing dispersion in ART is akin to that described in [SFD00]. A polychrome ray, upon encountering an interface between two materials with different indexes of refraction, will be split into a series of monochrome rays, one for each spectral sample considered.

Each monochrome ray will be sampled from the parent polychrome ray, such that only wavelengths which are represented on the parent polychrome ray will be generated. The wavelength for each generated ray shall be used to compute the index of refraction for the material, using the Sellmeier equation. This IOR will then be used with Snell's law to compute the direction of each monochrome ray. Wavelength-dependant Fresnel terms also need to be applied to each monochrome ray.

After this dispersive fan-out computation, it is necessary to modify the renderers / photon tracer appropriately in order to follow the monochrome rays. This will depend on each specific renderer, but generally consists of determining if the polychrome ray contains a monochrome fan-out list, and deciding which monochrome ray to follow (either one, several or all of them).

5.3 Preliminary work

The preliminary work for the dispersion rendering implementation consisted of verifying and implementing the required functionalities in order to build the monochrome fan-out list.

ART treats reflections and refractions equally. Generally, a reflection / refraction direction is associated with a *filter*, which is a spectral colour value that contains information about the wavelengths absorbed at that reflection / refraction. This is called a *Bidirectional Scattering Distribution Function sample*, or BSDF sample for short.

The BSDF sample structure in ART is generally used for polychrome rays. However, it does contain an optionally-used pointer to another list, the *compound BSDF sample list*, which is intended to be used for this kind of monochrome fan-out. A refractive BSDF sample with dispersion will thus consist of the ordinary polychrome ray, computed as usual using the average IOR for the material (no wavelength dependency). This polychrome ray will in turn contain a pointer to a linked-list of monochrome rays as the compound BSDF sample list. The decision of using this list or not is left to the renderer.

5.3.1 Monochrome BSDF sample reconstruction

In order to build the monochrome fan-out list, we need to split a polychrome ray into several representative monochrome rays. To ensure coverage of the whole visible spectrum, the wavelengths for each monochrome ray should be chosen stochastically, in a way that ensures coverage of all wavelengths within the sampled spectral range.

This consists of a sampling and reconstruction problem. Given a spectral sample consisting of several energy samples at pre-defined wavelengths of the spectrum, we need to construct several spectral samples each representing a single wavelength within the original sample, which is not itself directly sampled. Moreover, we must guarantee that the resulting set of monochrome samples carries no more power than the original polychrome refraction ray.

The approach that was implemented consists of generating a random relative shift from the sampled wavelengths, within the interval $[-0.5; 0.5]$. This shift is then applied to each sampled wavelength, multiplied by half the distance to the next and to the previous wavelength, thus ensuring statistical coverage of all the sampled spectrum range without requiring the samples to be taken uniformly.

After generating the wavelength to sample, the algorithm then proceeds to generate a sample for that wavelength. This is done by computing the relative distance to the left and right samples, and splitting the energy in the sampled wavelength λ_i in the original polychrome sample by wavelengths λ_i and λ_{i+1} (or λ_{i-1} for negative shifts) in the reconstructed sample. The amount of energy which is allocated to each of the two wavelengths depends on the distance to the shifted wavelengths: if the shift is 0, all energy remains in the original wavelength λ_i, whereas if the shift is 0.5, the energy is equally split among both λ_i and λ_{i+1}. This process is illustrated in figure 5.1.

The end result is that, for spectral samples with N sampled wavelengths, N monochrome

rays are generated, each with energy expressed in only two wavelengths λ_i and λ_{i+1}. By definition, if we split the energy in each wavelength among two separate wavelengths, the sum of all generated samples will have the same power as the original polychrome sample, thus guaranteeing energy conservation.

5.3.2 Fresnel term scaling

The Fresnel terms need to be applied to the monochrome rays as well as the polychrome rays, to ensure that, whichever ray is chosen by the renderer, it will have a correct contribution.

When the incident ray is split into reflection and (polychrome) refraction, both the refraction direction and Fresnel transmittance term are computed using the average IOR of the material. As before, when computing the Fresnel terms, we must ensure that we get conservation of energy, i.e., the sum of the individual powers of all monochrome rays must not be above the power carried by the original polychrome refraction ray, after applying the Fresnel terms.

However, we also need to ensure that the sum of the polychrome reflection ray and the monochrome refraction rays do not carry more energy than the polychrome incident ray. This in turn implies that we need to make sure that the monochrome Fresnel terms (computed using the exact IOR of the material for each wavelength considered) add up to be the same as the Fresnel term for the polychrome refraction ray.

Consider the sum of monochrome Fresnel terms:

$$T_M = \sum_{i=1}^{N} t_i \tag{5.1}$$

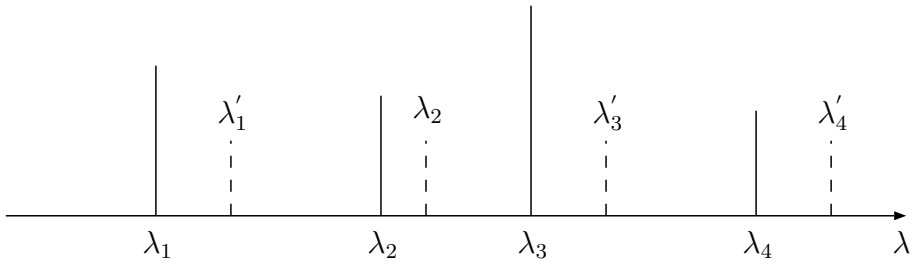
where T_M is the sum of all monochrome Fresnel terms t_i. We need to ensure that:

$$k \cdot T_M = T_P \tag{5.2}$$

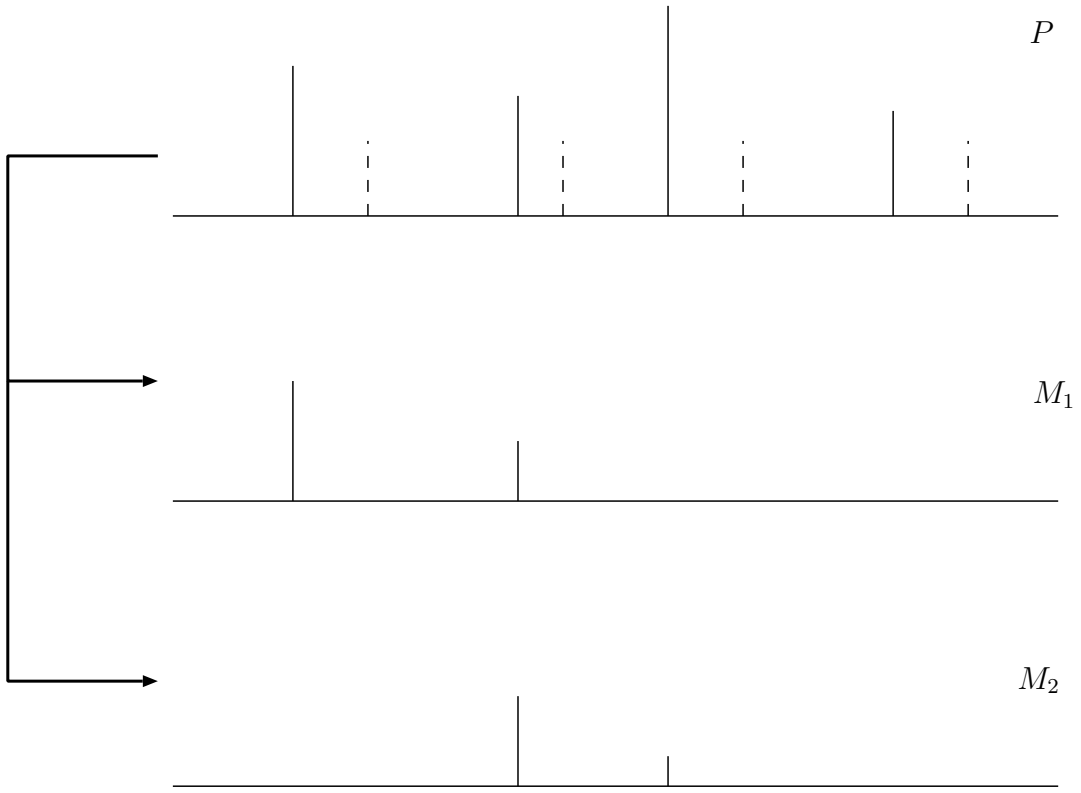
where k is a constant scale factor and T_P is the Fresnel transmittance computed for the polychrome ray. Solving for k yields:

$$k = \frac{T_P}{\sum_{i=1}^{N} t_i} \tag{5.3}$$

Thus, when computing the Fresnel terms for the monochrome rays, it is necessary to compute k as per eq. (5.3), and multiply each monochrome Fresnel term by k. This will ensure that the sum of the Fresnel terms does not exceed the Fresnel term for the polychrome refraction ray.



$$\lambda'_i = \lambda_i + S\left(\frac{\lambda_{i+1} - \lambda_i}{2} + \frac{\lambda_i - \lambda_{i-1}}{2}\right)$$



...

$$\begin{aligned} M_i(\lambda_i) &= (1.0 - S) \times P(\lambda_i) \\ M_i(\lambda_{i+1}) &= S \times P(\lambda_i) \end{aligned}$$

Figure 5.1: Monochrome sample reconstruction. A random relative shift S is computed, and the shifted wavelengths λ'_i are determined by applying this random shift to each sampled wavelength λ_i (top figure). The energy in the sampled wavelengths from the original polychrome sample P is then split among the M_i monochrome samples generated. For each M_i sample, the energy on wavelength λ_i in the original sample P is distributed among λ_i and λ_{i+1} (or λ_{i-1}, in case of a negative shift), in inverse proportion to the distance to the shifted wavelength (bottom figure).

5.4 Dispersion in the Simple Path Tracer

The simple path tracer is an implementation of the algorithm described in section 2.3.3. It is perhaps the simplest renderer in ART which can simulate most kinds of light interactions, not relying on any simplifying assumptions in order to generate images. It was an obvious choice to start with implementing dispersion as the renderer is relatively simple and easy to verify.

Overview of the Simple Path Tracer in ART

The simple path tracer algorithm in ART works by recursively tracing several rays per pixel into the scene. At each recursion level, the renderer keeps track of the current accumulated filter value (corresponding to the $\frac{w_i}{t_i \cdot a_i}$ terms in eq. 2.36 on page 29), the current ray being traced and the current recursion depth. The recursive trace returns two values: a direct illumination contribution for that recursion level, and an indirect illumination contribution which is obtained by further recursion into the scene.

The tracing process begins by computing the first intersection for the ray. If no intersection is found, the light scattered into the viewing direction from "infinity" (from, e.g., a skylight illumination model) is returned and recursion terminates, returning no direct contribution and the scattered light as the indirect contribution.

Otherwise, the hit point is then evaluated to determine if it lies on a light source. Should this be the case, the emission along the ray path is computed and stored as the direct contribution for this recursion level. Otherwise, the direct contribution is determined to be zero.

A material filter for the intersection point is computed (corresponding to the w_i term from eq. 2.36), according to the material properties of the surface.

The renderer then checks if the current recursion level is above the maximal level set by the user, in which case it terminates the recursion, returning no indirect illumination and the stored direct contribution value. Otherwise, the process continues, with the renderer generating a BSDF sample list from the hit surface. One of the samples is chosen randomly by importance sampling, using the average filter power of each sample as a probability for that sample being chosen. This measure of probability was chosen since it should yield more samples in brighter areas, where noise is expected to be more visible. The choice probability (corresponding to the t_i term) is stored.

The filter for the next recursion level is then computed, and the average power of the accumulated filter is compared to a pre-determined threshold. When it is lower than the threshold, Russian roulette is applied to decide if the ray should be killed: if it is killed, recursion is terminated and the direct light component is returned. Otherwise, the probability from the Russian roulette (corresponding to the a_i term) is stored.

A recursion ray is then assembled, using the direction from the chosen BSDF sample. The trace function then recurses, obtaining direct and indirect light values from the recursive trace.

Finally, the accumulated filter value is scaled by the Russian roulette probability and the random choice probability, as per equation 2.3 on page 24. The direct and indirect

light from the recursion into the scene are added together and multiplied by the scaled filter value. This is then returned as the indirect light contribution, along with the direct contribution computed earlier.

Our approach to handle dispersion

The initial approach intended to allow for only one monochrome ray to be traced into the scene. The random choice mechanism would pick between the polychrome reflection and refraction rays. Then, if the chosen ray contained a component BSDF sample list, another random choice would be taken and one of these rays would be followed.

It was later deemed necessary to allow for following all the monochrome rays and add their contributions together, in order to reduce variance expressed as strong colour noise in the resulting images. This was done at the first intersection with a dispersive material only. Even in the case that later intersections would result in another monochrome fan-out (which could happen with scenes containing fluorescent surfaces), only one of the monochrome rays would be followed. This was intended to reduce colour noise without an explosion in the computational complexity of the algorithm.

Design Issues and Implemented Modifications

The simple path tracer was written under the (normally valid) assumption that only one ray would ever be traced into the scene. This caused problems when attempting to follow all the monochrome fan-out rays at a dispersive interface.

To solve this problem, the renderer was modified and split into two steps. First, the choice of rays to be followed (polychrome reflection, single or multiple monochrome refractions) is made, and a list of all the rays to be followed is compiled. Then the renderer goes through the list in order to trace all the chosen rays.

The random , as per the Monte-Carlo importance sampling technique outlined earlier, requires the rays to be scaled according to the selection probability. This scaling factor is applied to each ray before it is moved to the list. Care must be taken when moving monochrome rays into the list, as these have been subjected to *two* random choices — one for the refraction / reflection case, and if the refraction was chosen, another random choice for picking the monochrome ray. It is thus necessary to apply two scaling terms to the monochrome rays.

The path tracer also uses Russian roulette in order to decide when to kill a ray. This process was re-implemented as a decision step when compiling the ray list, remaining equivalent to the non-dispersive version. Russian roulette is applied to the parent polychrome ray for all rays below a (configurable) minimal contribution threshold, in order to avoid killing "potentially bright" rays (which would otherwise increase the variance too much). The scaling factor is then stored, and any rays added to the ray list have their filters divided by this scaling factor. This ensures that the renderer complies with the conditions set forth previously in equation 2.32, since the filter will eventually be multiplied by the emitted radiance at a light source.

5.5 Dispersion in the Photon Tracer

The photon tracer in ART consists of an implementation of the photon tracing algorithm described in 2.3.4. It contributes to the rendering process by approximating a solution to the potential equation, thus describing the radiance state of the whole scene. This approximation is then sampled by the renderers which are capable of doing so, in order to estimate the indirect illumination of the scene.

Storage

Two data structures exist in ART for storing the information generated by the photon tracer — light maps and photon maps.

The photon map is essentially an array which stores every photon hit, along with directional and energy information. It is usually organised as a kD-tree, which then enables quick retrieval of the photons within a given distance from a point during the gathering step. The approximation of the radiance function for a given point is not implicitly calculated by this data structure.

A light map, on the other hand, behaves as a texture in which each texel contains information about the incoming light only. Whenever a photon strikes a surface, the photon energy is added to the light map texel at that point. Interpolation schemes may be used for determining the actual energy for a point covered by the light map, in a similar fashion to what happens with regular textures. This implies that the light map can not usually represent abrupt discontinuities in the radiance function (like sharp shadow boundaries). Depending on the light map resolution, this can lead to serious artifacts in the images.

Overview of the Photon Tracer

The photon tracer in ART recursively traces photons through the scene, starting at the light sources. The trace algorithm itself is similar to that of the path tracer.

The total number of photons to generate is split among the light sources, according to the relative power of each light source. Each photon carries a fraction of the power of the light source that emitted it, such that all photons emitted by one light source add up to the power of that light source.

Each photon is then shot from a random point on the light source along a random emission direction, with the power scaled once again according to the probability density function used to sample the light source (generally a cosine distribution, leading to a cosine factor).

The recursive tracing step then begins, using the emission direction as the initial ray for tracing the photon through the scene. The initial action taken is to compute the first intersection of the ray with the scene. If this is not found, recursion terminates.

Having found an intersection, the material filter for the surface at that intersection is computed and the photon's energy is scaled by this filter. If the surface is diffuse, then the photon energy is deposited into the storage data structure at this point unless the

current recursion depth is below the minimal recursion threshold. The recursion level is also tested against the maximum recursion limit, and in case it exceeds it, recursion is terminated and the algorithm returns.

Otherwise, a BSDF sample list is computed for the surface at the intersection point. One of these samples is then chosen randomly, using the average filter power as the probability density function. The energy that will arrive at the next recursion level is computed (by pre-multiplying the filter on the chosen BSDF sample with the current photon energy). Should this value be above a pre-defined threshold, the next ray is assembled and followed by recursion.

The photon tracer does not return any values, it simply generates data in the photon store.

Our approach

The same approach was taken as with the simple path tracer. A polychrome ray reaching an interface between two translucent media is split into several monochrome rays, which shall then be followed by the photon tracer.

It was intended to allow for following several (potentially all) monochrome rays, following the same reasoning as with the simple path tracer. However, correct results can only be obtained when only a single ray is followed. This is because following several rays would imply generating more photons than previously predicted, which would then influence the scaling factors which are applied to all photons. A post-processing stage could be implemented which would re-scale the energy in each photon store according to the final number of photons generated, after the photon tracer runs. Due to time constraints, this post-processing stage could not be implemented. However it was decided to implement this functionality in the photon tracer anyway, in order to leave this possibility open.

Design Issues and Implemented Modifications

As for the simple path tracer, it was necessary to modify the photon tracer to enable it to follow several rays. This was accomplished by separating the photon tracer in two steps. The first step builds a list of all the rays to follow. The rays are chosen randomly, once again using the average filter power as a probability. The filters for each ray are immediately multiplied by the scaling factor derived from the probability density function (as per the Monte-Carlo importance sampling technique), and as before monochrome rays must be scaled twice.

The photon tracer then goes through this list and recursively follows each ray. Russian roulette is not applied since all rays will contribute to the final solution of the potential equation as long as they hit any surface within the scene. Therefore, threshold- and recursion-depth-based culling is sufficient.

CHAPTER 6

Evaluation

Validating the results from a rendering system is an inherently difficult task. The output is generally an image, which is not very well suited to automatic validation due to the fact that external references which precisely match the output are not easy to obtain.

The fact that ART is a physically based renderer makes this somewhat easier in theory, since it inherently aims to simulate reality instead of simply generating pleasing images as the output. A photograph could, in theory, be a valid reference to compare with the output from ART. However, there are still too many variables to easily perform a pixel-by-pixel comparison using images obtained from common photographic equipment. There is ongoing research to enable validation of a spectral renderer using consumer cameras and lenses, but no complete framework has been yet developed.

In spite of this, it was decided to build a system to enable subjective comparisons with at least some reasonable scientific basis. A specially constructed viewing booth available at the institute was used to take photographs of reference objects inside it. A rudimentary system was devised to allow for reasonably accurate measurements of object and camera positions. This chapter discusses the methods and results obtained.

6.1 Obtaining reference images

In order to obtain reference images which can be compared to the output from ART, photographs must be taken under carefully controlled conditions.

The most reasonable way to achieve this was by using the viewing booth available at the Institute. The viewing booth consists of a wooden enclosure, about 1.2m wide by 0.5m tall and 0.5m depth, with one side wall removed. The walls are painted with a diffuse grey paint. There is a configurable light source on the top, covered by a white diffuser filter. The light source spans the whole width and depth of the box. Several light source types are available, such as a D65 and a UV light, selectable by the controls

on the viewing booth. These controls also allow for setting different light intensities.

Modelling the Viewing Booth

The first step was building an accurate model of the viewing booth in ART. There was already an initial version of the model available, however it was lacking true measurements for the dimensions and colour of the walls. The dimensions were measured using a simple measuring tape, following the common technique of taking several measurements and using the average.

ART allows for specifying material colours as diffuse samples. Light emissions can also be specified directly by their emission spectrum. The Institute also has a spectrophotometer available, a "Spectrolino" from GretagMacBeth. This is a hand-held unit which can take both reflectance and emission measurements. Special software developed at the Institute can be used to interface with the Spectrolino, and output measurements in a format directly readable by ART. After correcting some problems with the software, it became possible to take reflectance measurements from the walls. Emissive measurements for the light source, however, could never be done due to communication problems with the unit.

Nevertheless, the dimensions and the reflectance measurements were incorporated into the model, and light source intensities were adjusted by hand. This was not deemed to be a crucial barrier to using the viewing booth for result validation, since setting different camera exposures would also result in different brightness levels in the photographs, and there is currently no way of simulating camera exposure and lens aperture settings in ART.

Obtaining distance and positioning measurements

Obtaining accurate camera positioning proved to be a difficult task. The camera was mounted on a steady tripod and placed in front of the viewing booth in a suitable position. Several attempts to obtain the position by simple measurements, however, proved to be ineffective.

The solution found was to print a millimetric grid on A3 paper, glued to cardboard. This was then placed in a known measured position along the back wall of the viewing booth. The camera was set up in the desired position, and oriented as much as possible such that the focal plane would be parallel to the back wall. Making sure that the reference grid would at least roughly cover the centre area of the image in the viewfinder, a reference photograph for the camera position was taken, as shown in figure 6.1.

A special metal ruler was also built, with a flat base, such that it could be placed on the back wall and measure distance perpendicularly to it. This was used to measure the distance of the focal plane marking on the camera to the back wall.

The camera position was then determined from the reference photo and the measured distance to the back wall. The reference picture was loaded on an image editing program. The centre pixel of the image was then located, which would correspond to the exact centre of the focal plane within the camera. Since the cardboard grid was

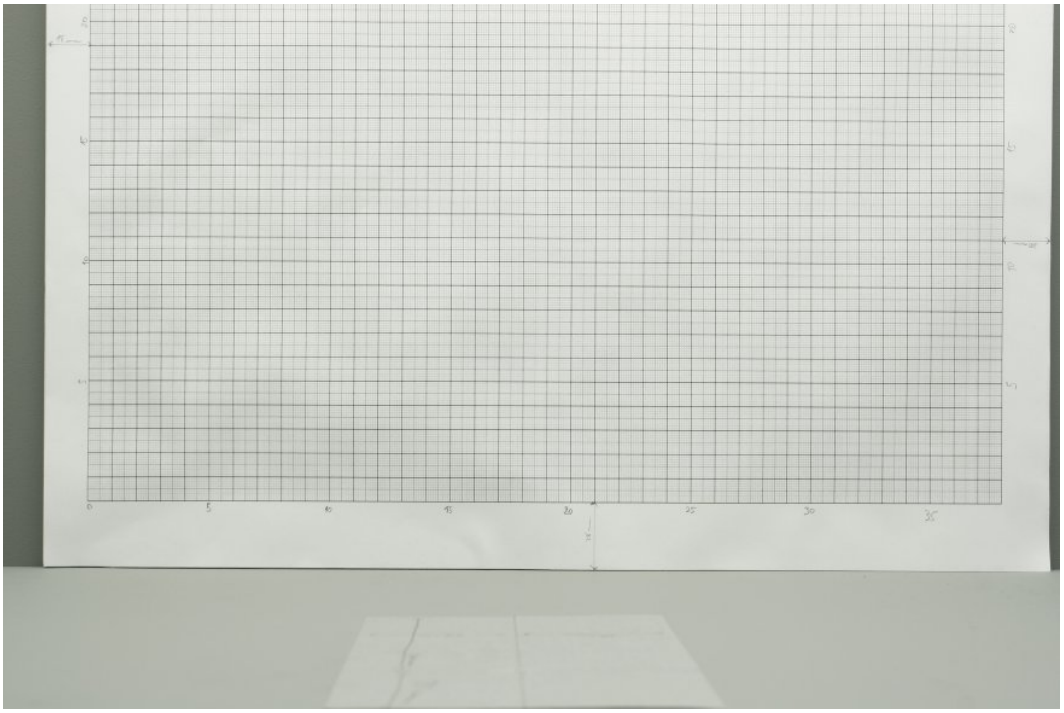


Figure 6.1: Reference photo for camera positioning

positioned in order to cover the central area of the image, this pixel would lie on the grid. The grid was then used to determine the position of the centre of the focal plane relative to the viewing booth, by tracing the grid lines to their markings on the edges of the millimetric grid. Knowing the position of the cardboard, this would then yield the actual position of the camera, when combined with the distance measurement relative to the back wall.

This method was relatively easy to implement, and was also relatively robust to lens distortion aberrations, since these generally occur towards the edges of the image and never break line continuity. By following the lines on the image from the centre pixel, the actual position of the camera could be obtained even in the presence of barrel distortion effects from the lens.

Several other methods of obtaining the distance to the back wall were also tested, using various measurement devices, but yielding unsatisfactory results. An attempt was also made to use the focusing system on the camera to convey the distance measurement. However, the Canon EOS 20D camera available at the Institute would not write this information to the image files, even though it reportedly reads this measurement from the lens and uses it for light metering. A Nikon D70s camera was also tested, and although distance measurements could be obtained from the camera, they were extremely inaccurate, as the camera only seems to need a rough estimate of the distance to the subject.

Having obtained the camera position, it was then a matter of placing the objects in

known positions inside the viewing booth. This was easily managed using measuring tape and a square ruler, with satisfactory results. A perfect match on the positioning between the photographs and the corresponding models in ART was not obtained, however the results were more than good enough to perform a subjective comparison of the two.

Camera exposure settings

As stated previously, the issue of matching the overall luminosity between reference images and ART renders is still open. There is currently no mechanism in ART which can simulate aperture values and exposure times for the camera. In addition, it was not possible to obtain emissive measurements of the light source in the viewing booth. As such, it was essentially impossible to match them accurately, so the luminosity values were tuned by hand in order to try to match the photograph.

The walls on the viewing booth was found to be roughly comparable to the standard 18% neutral gray reference used in photography. This corresponds to the standard calibration of a light meter inside a camera: when metering for a given area of the image, the meter will compute appropriate exposure values such that this area appears on a black and white picture as a 18% neutral gray. As such, exposure measurements were taken on the walls of the viewing booth, and used directly with no compensation. The lens was generally stopped down to f/16, since the depth of field at the working distances considered would be too small to keep the whole object in focus at larger apertures. Smaller apertures would reveal dust on the sensor, and were thus not used.

6.2 Simple Path Tracer

Validation

The validation of the simple path tracer consisted of rendering translucent objects inside the viewing booth model, and comparing them to controlled photographs of the same scene. This posed a problem for the validation of the dispersive rendering capabilities implemented, in that it turned out to be completely impossible to generate any visible dispersion effects inside the viewing booth.

Validation was done nevertheless, where the renderer correctly predicted that no dispersion would be visible on the two objects considered. Subjective comparison shows no missing features on the renders compared to the photographs, nor strange artifacts on the renders not present on the photographs.

The first set of photographs depict a translucent sphere, of an unknown glass material. This was tested with the dispersive renderer using a non-dispersive material, to make sure that the behaviour of the renderer was consistent when rendering a material with a constant IOR. This is shown in figure 6.2. The material does not appear very refractive, and it was not possible to see any dispersion effects through the sphere in any other conditions outside the viewing booth, so no further testing was made with this object.

The second test object used was a crystal made by Swarovski. This object is a lot more complex, and slight variations in positioning can yield somewhat different results, which was evident in the comparison of the renders against the photographs. Care was taken to align the crystal as well as possible with the camera and viewing booth, however a perfect match was never obtained. Still, it was possible to compare the photographs with the renderer output, which once more showed no missing features or errors from the renderer, as shown in figure 6.3. Again, the renderer correctly predicted no dispersion would be visible. One visible difference is a blue cast visible on the top of the crystal in the photographs, which is not present on rendered images. This is caused by a blue felt applied to the base of the crystal by the manufacturer, which was not in the ART model.

These tests are, of course, not a complete validation of the dispersive rendering capabilities of the path tracer. It was not possible to create visible dispersion effects under controlled conditions at the Institute, which made proper validation difficult. Further work could be attempted, using perhaps a dark enclosure with a small and bright light source shining on the objects. The requirement of having controlled conditions which can be reproduced in ART however meant that this testing could not be carried out with the available equipment within the project time frame.

Results

The path tracer was used to render the Swarovski crystal in different settings. Shown in figure 6.3 is the crystal placed inside a diffuse box, with a grid texture overlayed on the wall surfaces. This crystal is illuminated by a diffuse area light source placed on top of the box. The image clearly shows the rainbow-coloured effects created by light refracting and reflecting within the prism. Colour fringing is also visible on the edges of the black stripes on the wall texture, when viewed through the prism.

6.3 Photon Tracer

The photon tracer's main strength for dispersion rendering lies with providing information on caustics to the renderer. It can not predict the effect of dispersion directly visible inside the translucent objects.

Validation

Validating the photon tracer proved to be harder than for the path tracer. Obtaining photographs of dispersive caustics in controlled conditions is extremely difficult. It requires the use of a collimated light source, which is not available at the Institute. Instead, a simple "collimator" device was built using a telescope eyepiece, however the results were completely unsatisfactory.

As such, the only collimated light source available was the Sun itself. Making use of a window curtain to improvise a slit through which sunlight would shine, some photographs were made of the rainbow caustics caused by a triangular glass prism. These

were useful only as a quick "sanity-test" for the photon tracer, as they were only used to make sure that the caustics generated were not completely wrong. It was impossible to control and reproduce the conditions under which the pictures were taken, and as such they were of limited value for validating the photon tracer. One such photograph is shown in figure 6.5.

Results

Test scenes were set up in ART with a triangular glass prism inside a box with a collimated light source. These were rendered using the Whitted ray tracer, with light maps generated by the photon tracer.

The first image, shown in figure 6.6, was rendered using a diffuse light source, in order to show the geometry of the scene. The dispersion effect, although clearly perceptible, is not very pronounced (as expected with a diffuse light source).

The next image, figure 6.7, shows the result of rendering the same scene after replacing the light source with a very narrow collimated light source. No renderer in ART is currently capable of gathering direct illumination from this kind of light source, since light source sampling will sample the collimated emission direction with zero probability. As such, most of the image is black. It would be possible to modify renderers which do direct light source sampling (such as the light-source-sampling path tracer or the Whitted ray tracer) in order to correctly render scenes with collimated light sources. However, for the Whitted ray tracer this is of limited value, as it does not compute indirect illumination. Due to the positioning of the objects in this scene — the collimated light source shines directly on the top surface of the prism, where it is never visible by reflection for the observer — the final output would remain completely black when rendered with the Whitted ray tracer.

However, the photon tracer can be used nevertheless to compute indirect illumination and caustics for this scene. The result is the rainbow-coloured caustic visible on the right wall. All the rendering parameters for this scene were the same as for the previous scene with the Lambert light source. The only difference was that the photon tracer recursion depth was limited, to avoid generating noise on the other surfaces (by diffuse reflection of the caustic).

The two images in figure 6.8 depict the same caustic as in figure 6.7, but rendered at a much higher resolution. The light map resolution was also increased, since at this level of detail the light map texel boundaries would otherwise become visible.

The image in figure 6.9 shows a complex caustic with light dispersion effects caused by the Swarovski crystal. It uses a diffuse light source which shines on the crystal, oriented sideways on top of a diffuse plane which receives caustics. The effect of dispersion is subtle but visible on the caustic patterns.

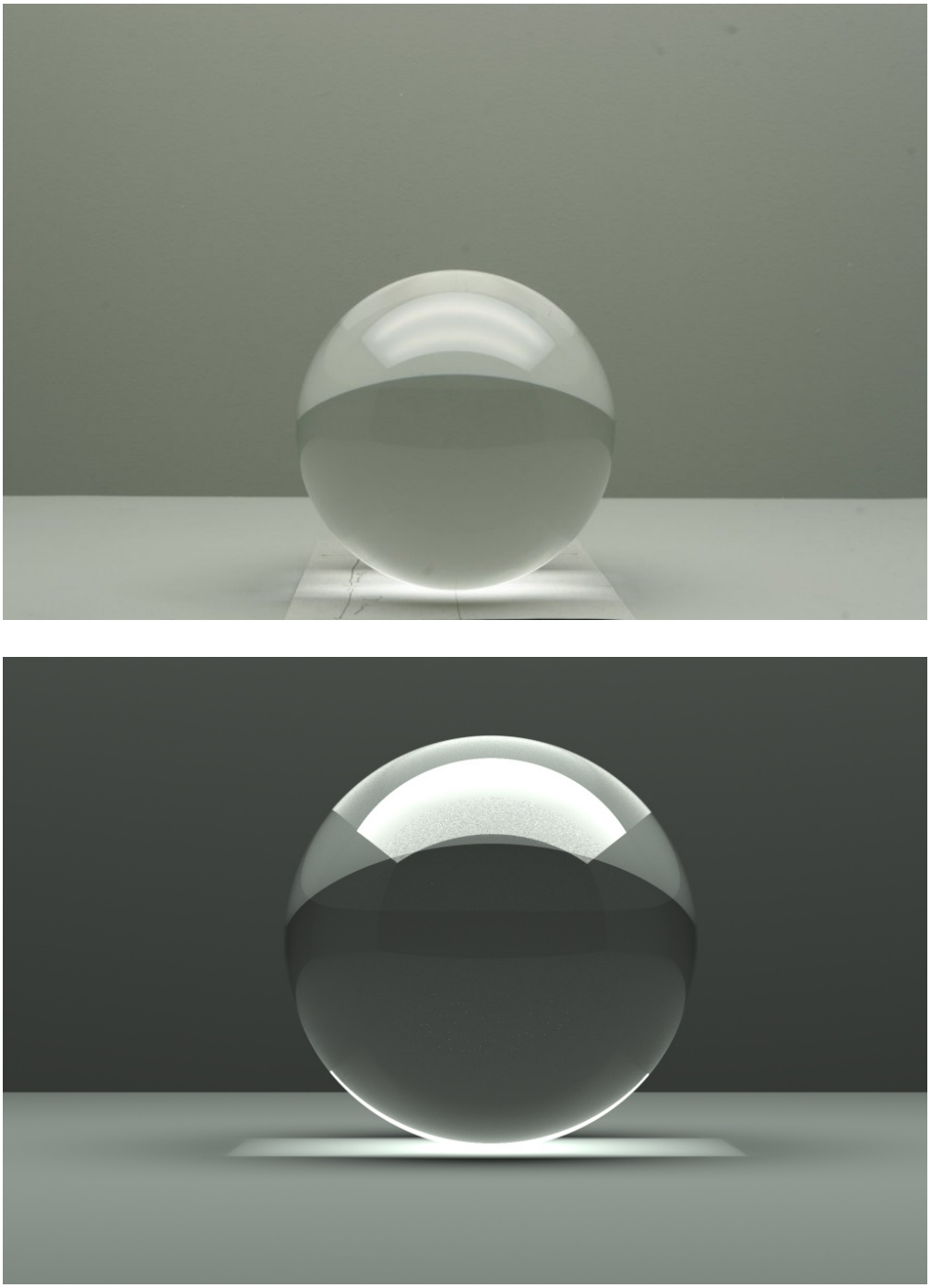


Figure 6.2: A glass sphere inside the viewing booth
Top image: Canon EOS 20D, 18-40mm @34mm, f/16.0, 1s. D65 light source at 100% intensity.
Bottom image: ART rendering of the same scene. Simple path tracer with dispersion, 2048 samples per pixel. Non-dispersive material (constant IOR = 1.5)

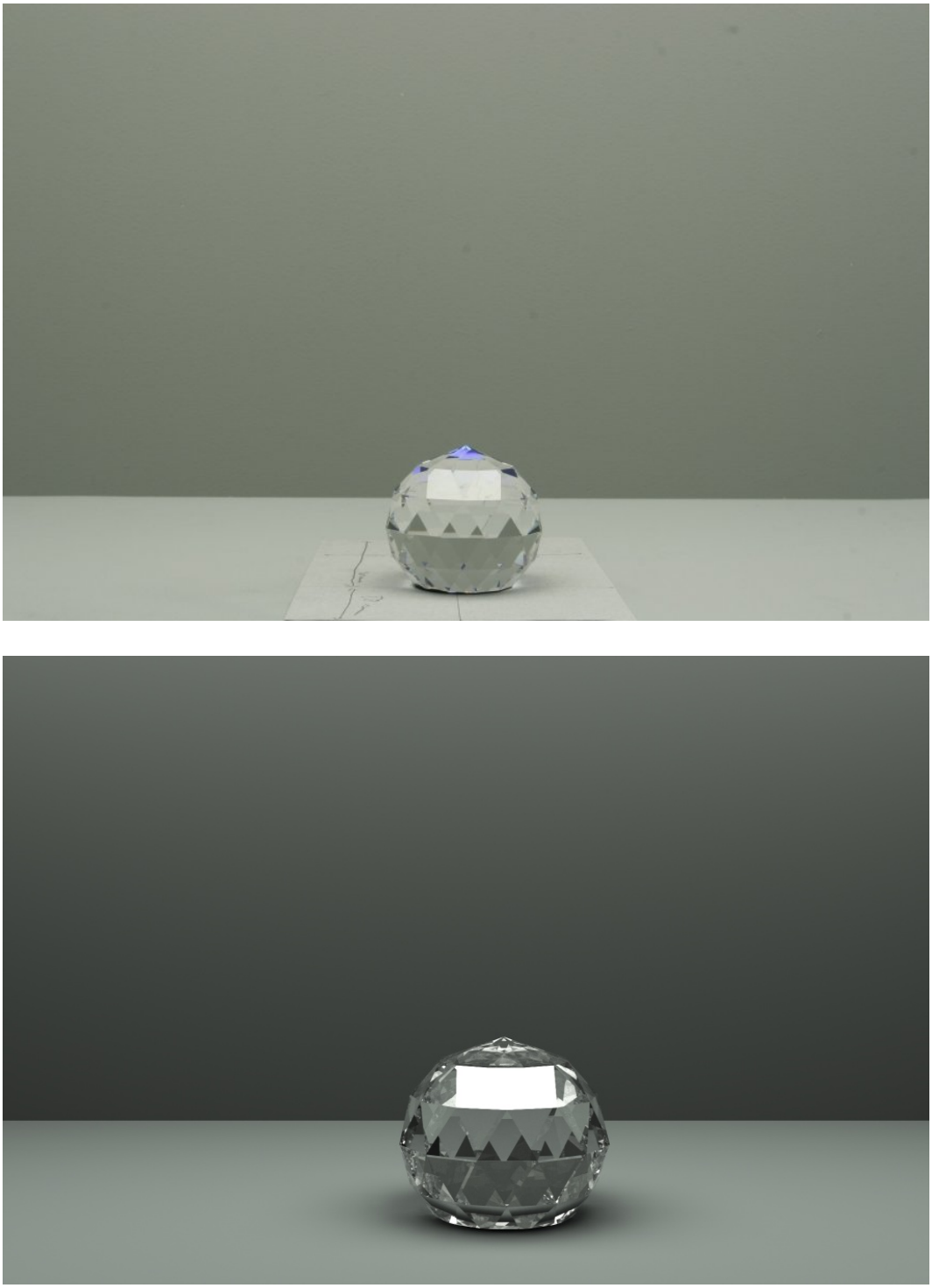


Figure 6.3: Swarovski crystal inside the viewing booth.
Top image: Canon EOS 20D, 18-40mm @34mm, f/16.0, 1s. D65 light source at 100% intensity.
Bottom image: ART rendering of the same scene. Simple path tracer with dispersion, 2048 samples per pixel. Schott optical glass material (LaSF 35).

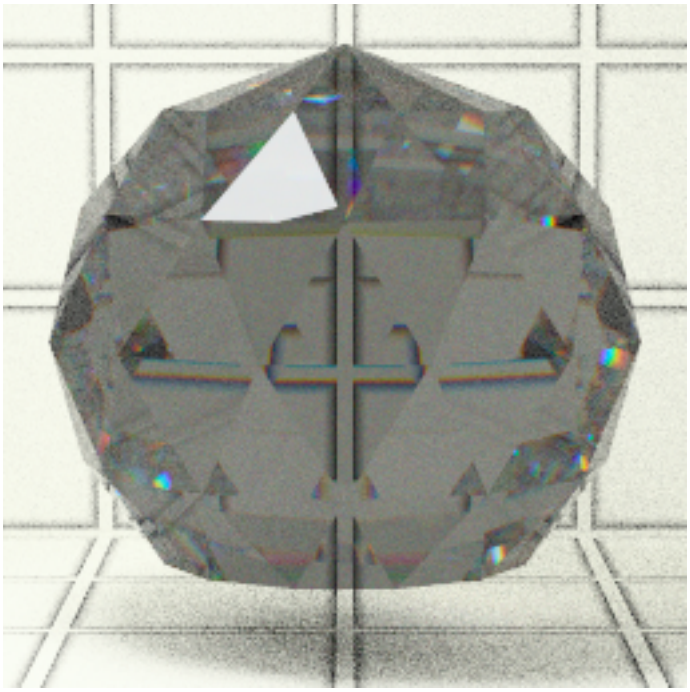


Figure 6.4: Swarovski crystal inside a box with textured walls. Dispersion is visible both as rainbow-coloured light effects within the prism as well as colour fringing on the black stripes on the wall viewed through the prism. (Simple path tracer with 2048 samples per pixel)

Figure 6.5: Rainbow caustic caused by sunlight refracting through a triangular glass prism

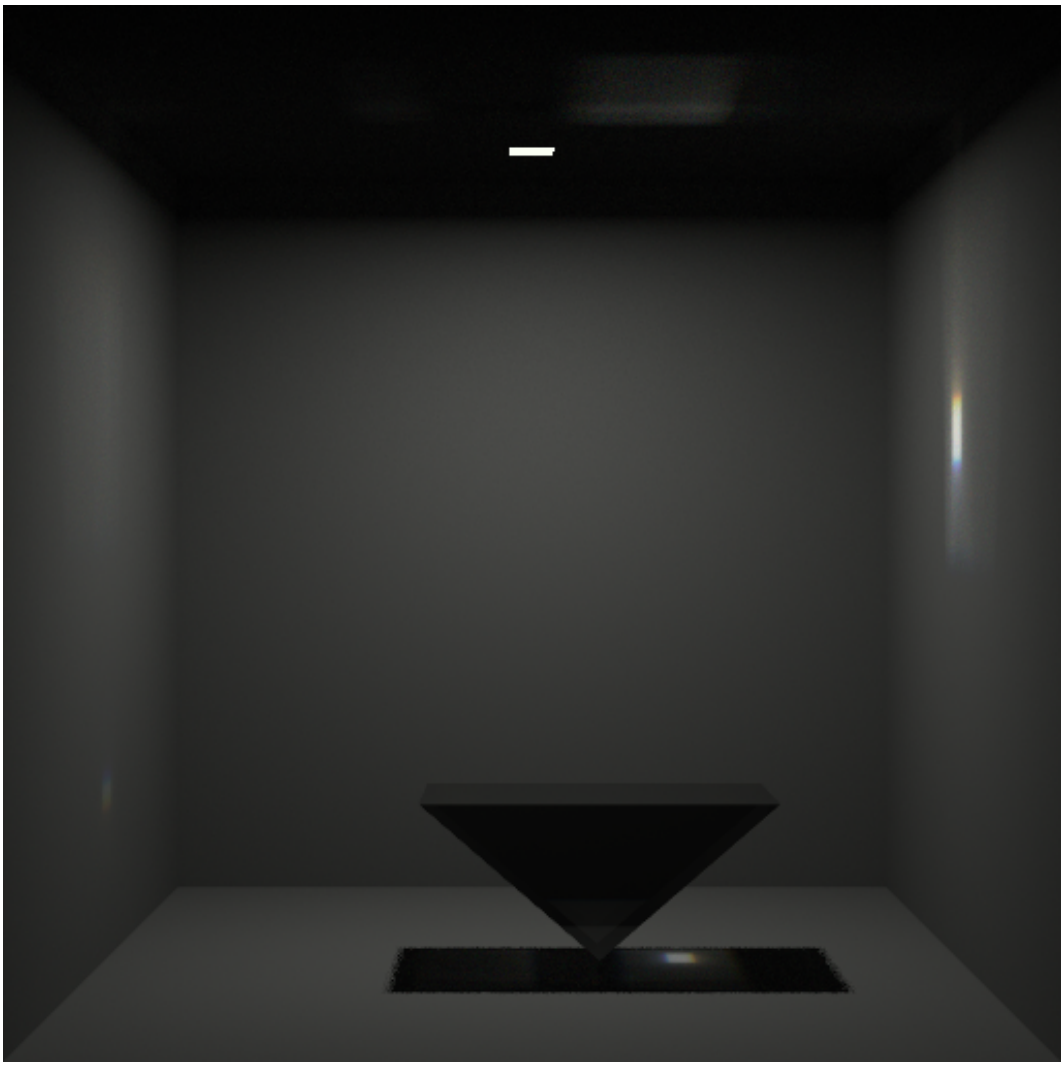


Figure 6.6: Test scene for the photon tracer, with a Lambert diffuse light source
Prism material: Schott Optical Glass LaSF 35
Rendered using the Whitted ray tracer and light maps
Photon tracing parameters: 100 million photons, light map resolution of 400 texels per unit square in UV space

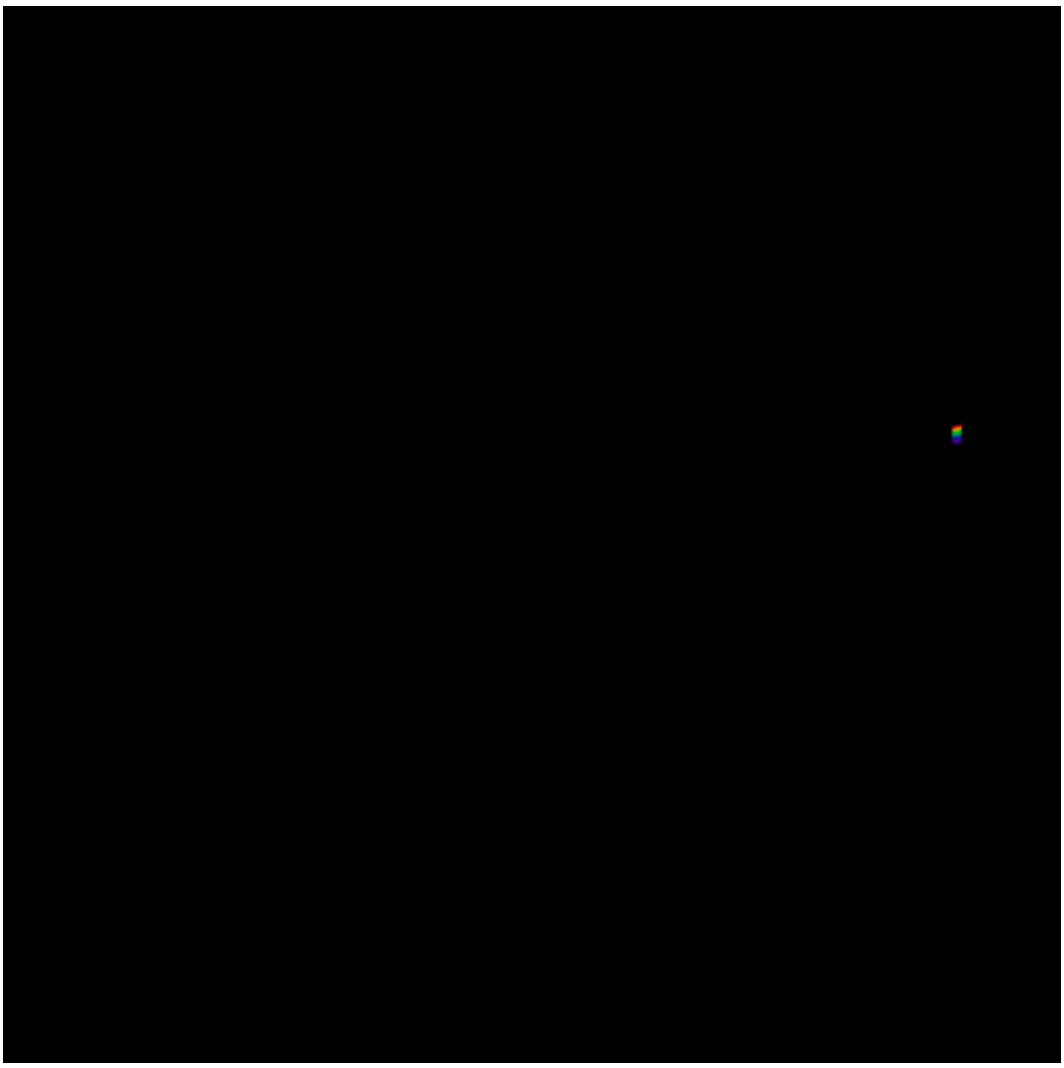


Figure 6.7: Test scene for the photon tracer. Rendered with a narrow collimated light source and limited recursion in the photon tracer.

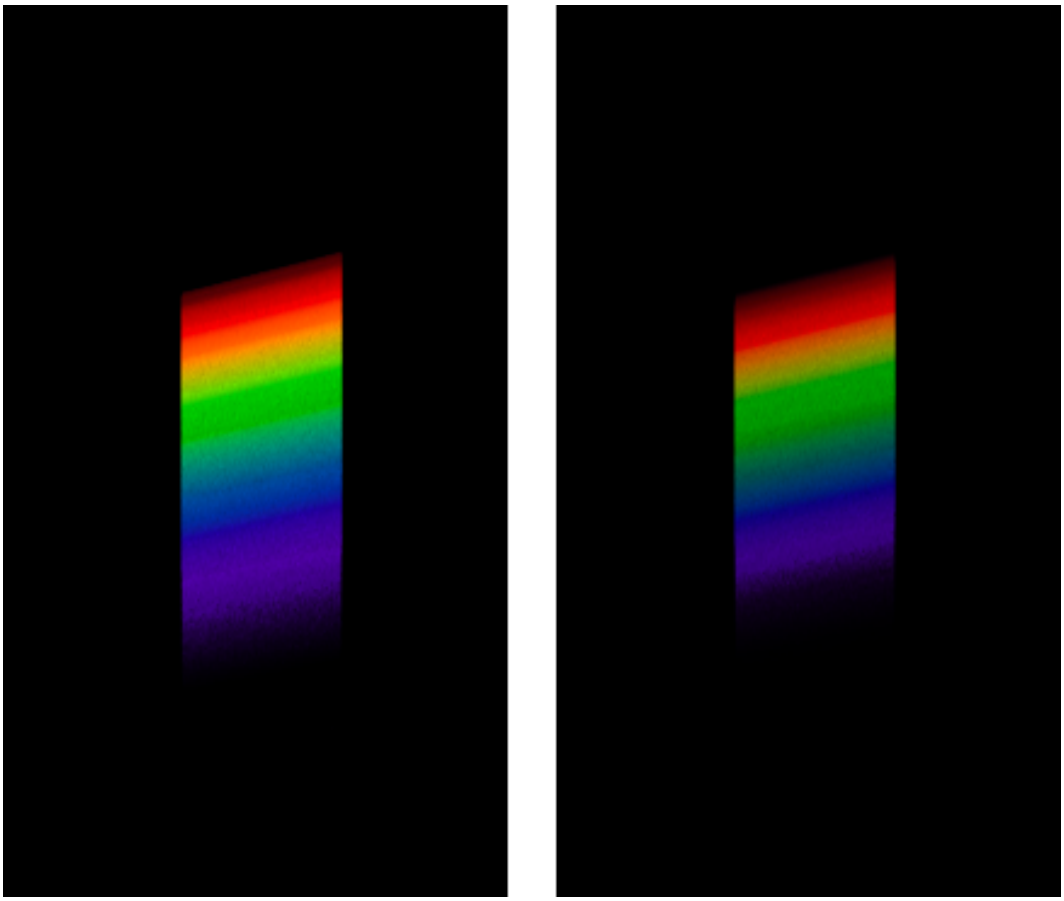


Figure 6.8: Detail of the caustic from figure 6.7. Rendered with 8 spectral samples (left image) and 16 spectral samples (right image). Light map resolution was increased to 1000 texels / unit UV square.

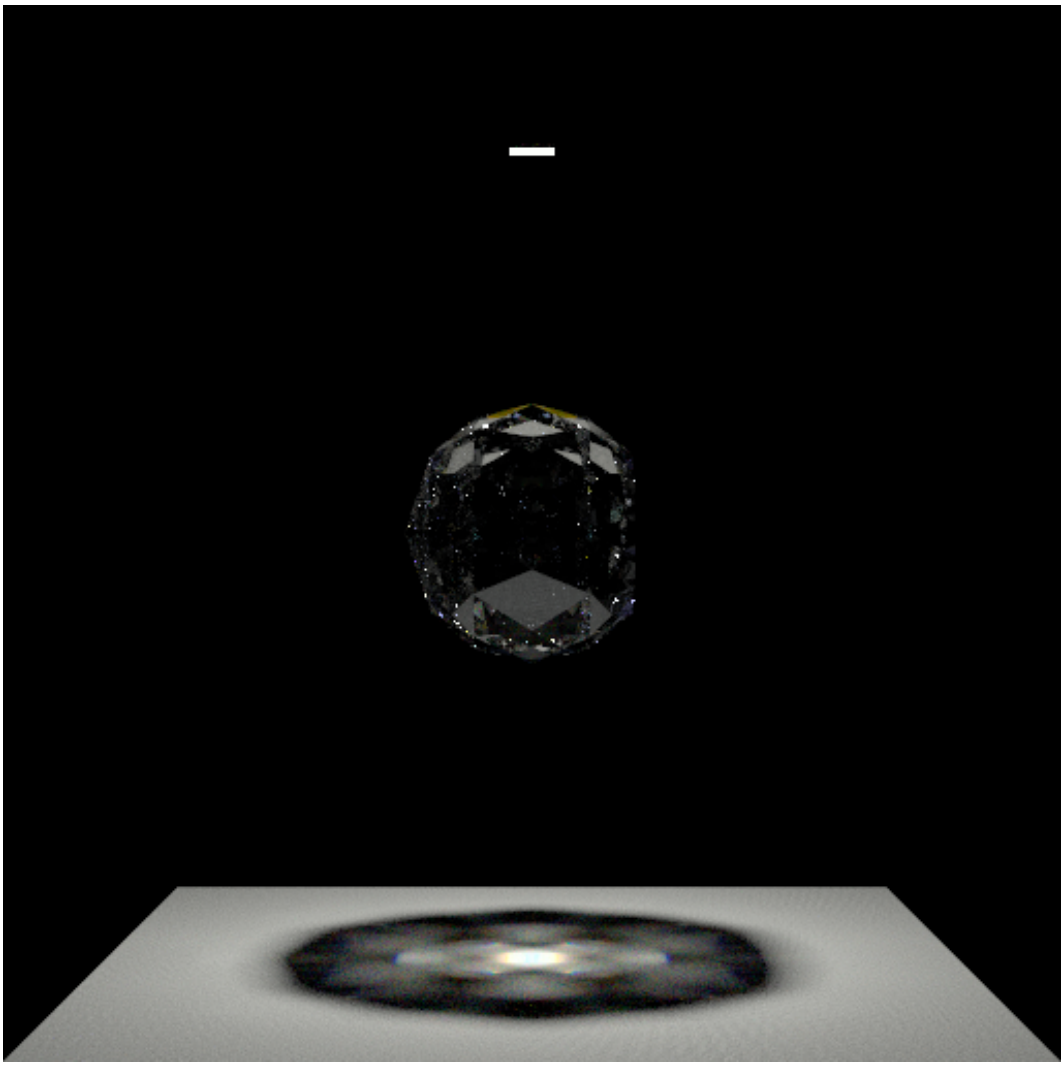


Figure 6.9: Complex caustics caused by the Swarovski crystal. Whitted ray tracer with photon tracer using light maps (10 million photons, 800 texels / unit UV square)

CHAPTER 7

Conclusion and Outlook

The problem of rendering dispersion effects is relatively well understood. The underlying physical principles are not new, having been studied for quite some time before. Several research papers exist about the problems faced when implementing dispersive rendering capabilities in an existing rendering system.

The work presented here consists of an implementation of dispersive rendering capabilities on two fronts. The modifications to the simple path tracer enable ART to simulate dispersion while approximating a solution to the rendering equation. In practice, this means ART is now capable of simulating dispersion effects visible while looking through a translucent object.

On the other hand, modifying the photon tracer allows ART to predict dispersive effects while computing an approximate radiance state of the whole scene, i.e., solving the potential equation. With this, ART can effectively simulate the effect of a beam of light being scattered into monochromatic colours by a translucent object and producing a rainbow caustic on another surface.

By implementing these two features, this project has provided dispersive rendering capabilities to an open-source spectral renderer, something which did not exist beforehand. Also, it has proven that ART is capable of fully simulating light dispersion effects, which could pave the way for further work in the area.

The project deadline, as often happens, creates a sense of leaving some unfinished work behind. As I finish my stay with the Institute, it has not yet been possible to actually implement a fully dispersive renderer which can both render dispersion effects visible through a prism as well as the caustics the prism causes on other surfaces. The simple path tracer does not make use of the information provided by the photon tracer; the Whitted ray tracer, while capable of using the approximate radiance function computed by the photon tracer, does not currently simulate dispersion in the ray casting process. This would be the logical next step, in order to obtain images which

contain at once all dispersion effects that ART can currently simulate. A quick implementation of dispersion in the Whitted ray tracer was actually done, and the results looked promising; this was however not properly designed or tested, and it suffered from a few problems. It was intended only as a quick proof-of-concept, left behind as a "pointer" towards further work on the topic.

Another possible line of work lies with the random selection of monochrome rays in the photon tracer. One of the ideas which was also partially tested during my stay was that of using a different probability distribution function for this selection. We considered selecting the rays according to the CIE luminous efficiency function, which describes the response levels of the human eye to different wavelengths in the visible spectrum. It resembles a Gaussian function, with the peak around the green area of the visible spectrum. The idea would be to concentrate more samples in the areas of the spectrum to which the eye is more sensitive, reducing noise in those areas (where it would in theory be more visible) while increasing noise in the rest of the spectrum (where it would not be as easily detected by the eye). Preliminary testing of this proved to be inconclusive, however, due to numerous technical difficulties encountered.

The results presented here, while apparently good in terms of correctness, can not be said to have been definitively verified. However, the limited tests performed suggest that the implementation is sound. The results are not completely conclusive, but do not show any evident problems or artifacts. The images also convey a sense of realism to the viewer. When rendered in semi-realistic settings, they gain that unexplainable quality of not immediately being perceived as obviously computer-generated. This conveys some confidence in the results. Only proper testing would be able to validate them, of course, but we are confident that they could probably withstand the scrutiny.

This project has been a tremendously gratifying endeavour. The amount of knowledge and experience gained in the area of computer graphics are perhaps the greatest achievement of the past 5 months. As such, and despite having obtained results that are not definitive, I leave the Institute with a sense of accomplishment, though well aware that there would probably be enough work left to do for another 5 months.

Bibliography

[Ric01] Austin Richards. *Alien Vision — Exploring the Electromagnetic Spectrum with Imaging Technology*. SPIE Press, 2001

[Glas95] Andrew S. Glassner. *Principles of Digital Image Synthesis*, volume 2. Morgan Kaufmann Publishers, Inc., 1995

[KF86] Miles V. Klein, Thomas E. Furtak. *Optics*, Second Edition. John Wiley & Sons, 1986.

[Szi99] Szirmay-Kalos László. *Monte-Carlo Methods in Global Illumination*. Institute of Compute Graphics, *Technische Universität Wien*, 1999. (Available at script `http://www.fsz.bme.hu/~szirmay/script.pdf` as of 24 June 2006).

[Kaj86] James T. Kajyia. *The Rendering Equation*. SIGGRAPH '86: Proceedings of the 13th annual conference on Computer Graphics and Interactive Techniques, p. 143-150. ACM Press.

[VG95] E. Veach and L. Guibas. *Bidirectional estimators for light transport*. In Computer Graphics (SIGGRAPH '95 Proceedings), pages 419-428, 1995.

[Tho86] Spencer W. Thomas. *Dispersive Refraction in ray tracing*. The Visual Computer, 2(1):3-8, January 1986.

[YKIS88] YingYuan, Tosiyasu L. Kunii, Naota Inamoto and Lining Sun. *GemstoneFire: Adaptive Dispersive Ray Tracing of Polyhedrons.* The Visual Computer, 4(5):259-270, 1988.

[AW05] Andrea Weidlich. *Physically-based Rendering of Light Propagation in Uniaxial Crystals*.

[SFD00] Yinlong Sun, F. David Fracchia, Mark S. Drew. *Rendering Light Dispersion with a Composite Spectral Model*. International Conference on Color in Graphics and Image Processing — CGIP'2000.

[Fol97] James D. Foley, Andires van Dam, Steven K. Feiner, John F. Hughes, Richard L. Phillips. *Introduction to Computer Graphics*. Addison-Wesley Publishing Company, 1997.