Real-time Rendering of Acquired BRDF Data Sets

by

Matthew O'Toole

April 30, 2007

Abstract

Provided a set of acquired BRDF data, realistic image synthesis requires evaluation of the rendering equatuation over a hemisphere of directions on a surface. We store the basis representation of the BRDF, acquired directly from the sample material, in a pair of three-dimensional textures to offload the work to the GPU. For materials that produce artifacts in the basis representation, we use data fitted to the D-BRDF model as an alternate method of rendering the BRDF. We discuss the usage of environment maps and importance sampling algorithms as a method of representing complex natual illumination. Through this approach, the solution of the illumination integral can be approximated for real-time and interactive applications that opt to use measured BRDFs.

Contents

List of Figures

Acknowledgements

I would like to thank Abhijeet Ghosh for his time and effort in my mentoring and supervision of this project, and giving me the opportunity to work on such a project. I also thank Assoc. Prof. Wolfgang Heidrich for his confidence and guidance over the past year. I extend my thanks to all those I had the privilege to meet in the PSM Lab.

Chapter 1

Introduction

Photo-realism is a term often used in the sub-field of computer graphics. The term refers to synthesized images that are perceived as real images. The challenge is often associated with realistic illumination, more so than the geometry of the scene.

The classical method of applying illumination to objects is to partition light into an ambient, a diffuse, and a specular component. By applying such limitations to our illumination model, we limit ourselves in rendering a small subset of materials.

The reflectance property of a material can be determined by using an acquisition process that directly extracts the BRDF coefficients of a basis function representation [7]. With a function that describes the reflectance of our sampled material, we can accurately render the material on any given geometry. By using such a process, we widen the set of materials that can be rendered.

An important requirement for realistic renderings is the illumination model. In this thesis, we use an environment map to simulate the light at a given point. We then apply an importance sampling algorithm to fetch a set of sample points

that represent the environment map. By storing the basis representation of the BRDF in a pair of three-dimensional textures and using the set of point lights to describe the environment, we are capable of evaluating the BRDF and the rendering equation on the GPU. The BRDF data, when represented in a global basis such as spherical harmonics, can result in ringing artifacts in the rendering for specular materials. For data sets with visible ringing artifacts, we choose to alternatively use data fitted to a D-BRDF analytical model.

The thesis will describe the methods of implementing a renderer for acquired reflectance functions that can generate images and animations in real-time. The thesis is structured as follows. Chapter 2 discusses the background and theory of computing the illumination integral. The implementation details associated with the renderer will be described in Chapter 3. In Chapter 4, we conclude our findings and discuss the relevance of the work.

Chapter 2

Background

2.1 Overview

Before proceeding to the discussion, we outline a few fundamental concepts that will be referred to in the following sections and chapters.

2.1.1 Probability Density Function

The *probability density function* (PDF) is the probability distribution of any function $f(x)$. A function $f(x) : \mathbf{R} \to \mathbf{R}$ is a probability density function if:

$$f(x) \geq 0 \qquad \text{for all } x \in \mathbf{R} \tag{2.1}$$

$$\int_{-\infty}^{\infty} f(x)dx = 1 \tag{2.2}$$

For the most part, we will be dealing with the discretized PDFs. Given an ordered finite set $\mathbf{X} = \{x_k \mid k = 1, 2, \ldots, N\}$, we define the function $f(x_k) : \mathbf{X} \to \mathbf{R}$ as the discrete probability density function. Both function definitions

will be abreviated by the acronym PDF. Similarly to equation 2.1 and 2.2, the discrete probability density function has the following properties:

$$f(x_k) \geq 0 \tag{2.3}$$

$$\sum_{k=1}^{N} f(x_k) = 1 \tag{2.4}$$

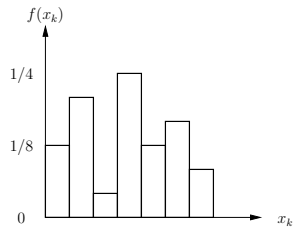


Figure 2.1: Sample Discrete Probability Density Function

2.1.2 Cumulative Density Function

The *cumulative distribution function* (CDF), or *probability distribution function*, is an extension of the probability density function. A function $f(x) : \mathbf{R} \to [0, 1]$ is a cumulative density function if:

$$\lim_{x \to -\infty} f(x) = 0 \tag{2.5}$$

$$\lim_{x \to \infty} f(x) = 1 \tag{2.6}$$

$$f \text{ is monotone} \tag{2.7}$$

Recall that a function $f(x)$ is monotone if $\forall x, y \in \mathbf{R}, x \leq y$ implies that $f(x) \leq f(y)$. We note that the PDF is also the derivative of the CDF. Furthermore, if the PDF is non-zero, the associated CDF function is an injective

function; the inverse cumulative distribution function $f^{-1}(x)$ exists for all x.

As with the PDF, we define the discrete cumulative distribution function $f(x_k) : \mathbf{X} \to \mathbf{R}$. Given the discrete probability density function $p(x_k)$, $f(x_k) = \frac{\sum_{j=1}^{k} p(x_j)}{\sum_{j=1}^{N} p(x_j)}$.

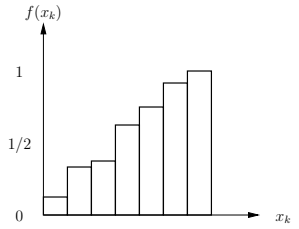


Figure 2.2: Sample Discrete Cumulative Distribution Function

The CDF of figure 2.2 is derived from the PDF of figure 2.1.

2.1.3 Cartesian and Spherical Coordinates

When referring to points or vectors, all coordinates are situated within a three-dimensional space. We often represent the coordinates in two interchangeable coordinate systems: *Cartesian coordinates* and *spherical coordinates*.

We first consider the three dimensional *Cartesian space*, also known as *Euclidean space*. The space is defined by three orthogonal vectors, usually the x-, y-, and z-axis, and each point (x, y, z) is uniquely defined by some linear combination of each axis. These points are defined using the Cartesian coordinate system.

Spherical coordinates span the Euclidean space with an alternate set of parameters, (p, ϕ, θ) where $p \in [0, \infty), \phi \in [0, \pi), \theta \in [0, 2\pi)$. The parameters ϕ and θ spans the sphere of radius p centered at $(0, 0, 0)$.

Definition 1. Spherical to Cartesian coordinates:

$$\begin{aligned} x &= p\sin(\phi)\cos(\theta) \\ y &= p\sin(\phi)\sin(\theta) \\ z &= p\cos(\phi) \end{aligned}$$

Definition 2. Cartesian to spherical coordinates:

$$\begin{aligned} p &= \sqrt{x^2+y^2+z^2} \\ \phi &= \arccos(z/\sqrt{x^2+y^2+z^2}) \\ \theta &= \arctan(y/x) \end{aligned}$$

For all applications of spherical coordinates in this paper, we only consider points on the unit sphere where $p = 1$. The advantage of using spherical coordinates over Cartesian coordinates is that the parametrization of the area over the surface of the unit sphere requires only two variables. Furthermore, for reasons that will be later discussed, we concern ourselves in general with the half sphere, and thus $\phi \in [0, \pi/2)$. For the remainder of this thesis, we define the set $\mathbf{S} = \{\omega = (\theta, \phi) \mid \theta \in [0, 2\pi), \phi \in [0, \pi)\}$ and $\mathbf{S}_{hemi} = \{\omega = (\theta, \phi) \mid \theta \in [0, 2\pi), \phi \in [0, \frac{\pi}{2})\}$.

2.2 Rendering Radiance

2.2.1 Lambert's Cosine Law

In reality, light rays interact with curved surfaces. For any given point on the surface, the infinitesimal area of the curved surface becomes a plane. Recall that the plane is defined by $x \cdot n = d$, where normal $n \in \mathbf{R}^3$ and $d \in \mathbf{R}$.

Given a single incident light vector and the normal, the total radiance projected onto that point is directly proportional to $cos(\phi)$, where ϕ is the angle between the normal and the light vector. The law is known as *Lambert's cosine law*.

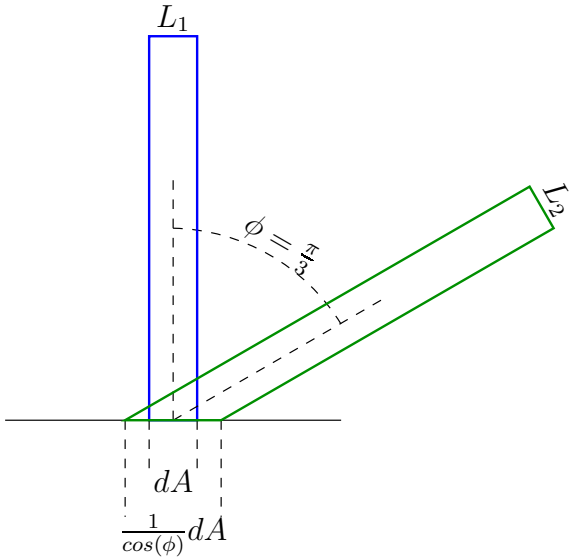


Figure 2.3: Lambert's cosine law

Given a light ray of width dA as in figure 2.3, the radiance of the ray is distributed across an area of size $\frac{1}{\cos(\phi)}dA$. Therefore, the radiance of a light vector is observed to be $\cos(\phi)L_i$ at any point on the plane, where the L_i is the light intensity.

Because the ray intersects the underside of the plane, the light ray does not contribute to the illumination of that point. That is, any ray that exceeds $\phi > \frac{\pi}{2}$ gets culled. For this reason, we defined the set of spherical coordinates $\mathbf{S}_{hemi}$, where $(0, 0) \in \mathbf{S}_{hemi}$ refers to direction of the normal.

2.2.2 Blinn-Phong Analytical Model

The *Phong reflection model* is composed of three components: ambient, diffuse, and specular. The resulting illumination is the summation of all three components.

Definition 3. Phong reflection model

$$I_p = \underbrace{k_a i_a}_{\text{ambient}} + \sum \underbrace{k_d (L \cdot N) i_d}_{\text{diffuse}} + \underbrace{k_s (R \cdot V)^\alpha i_s}_{\text{specular}}$$

The *Blinn-Phong reflection model* is the classical shading model used in graphic pipelines. For most *graphic processor units* (GPUs), the Blinn-Phong analytical function is hardwired into the pipeline.

For efficiency purposes, the Blinn-Phong reflection model is an approximation of the Phong model. The Phong model requires that the reflected light-source R be calculated per pixel. The Blinn-Phong model replaces the angle $R \cdot V$ by $N \cdot H$, where $H = (L + V)/|L + V| = (\hat{L} + \hat{V})/2$.

Definition 4. Blinn-Phong reflection model

$$I_p = \underbrace{k_a i_a}_{\text{ambient}} + \sum \underbrace{k_d (L \cdot N) i_d}_{\text{diffuse}} + \underbrace{k_s (N \cdot H)^\alpha i_s}_{\text{specular}}$$

Notice Lambert's cosine law factor $L \cdot N = \cos(\phi)$ in the diffuse component of both models.

2.2.3 Bidirectional Texture Function

The *bidirectional reflectance distribution function* (BRDF) describes the reflectance properties of a given material. Because we only concern ourselves with the light distribution at a local reference point, BRDFs must assume that

they are invariant with respect to the planar texture coordinates (s, t). Therefore, the BRDF is a four dimensional function parametrized by the spherical coordinates of the incoming light vector and the outgoing view vector.

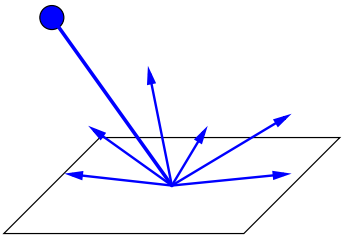


Figure 2.4: Light Ray Reflectance Distribution over Hemisphere

We will define the BRDF as f_r for the remainder of this paper. If we are to assume that the Lambert's cosine law is not implicitly defined by f_r, the area element for reflected radiance is given as follows:

Definition 5. Area element of BRDF Radiance

$$f_r(\theta_o, \phi_o, \theta_i, \phi_i) L_i(\theta_i, \phi_i) \cos(\phi_i) d\omega_i$$

The BRDF is a subset of a higher class of texture functions, called *bidirectional texture functions* (BTFs). The bidirectional texture function, also referred to as the *bidirectional scattering distribution function* (BSDF), is a six dimensional function parametrized by planar texture coordinates (s, t), incoming light spherical angles $\omega_i = (\theta_i, \phi_i) \in \mathbf{S}_{hemi}$, and outgoing view spherical angles $\omega_o = (\theta_o, \phi_o) \in \mathbf{S}_{hemi}$. Provided these parameters, the function can be used to describe inter-reflections, subsurface scattering, self-shadowing, masking, and other nonlocal effects. When a light ray intersects an object, light can be reflected or absorbed. When absorbed, the light can be transmitted to the opposing side or scatter back to the surface. The acquisition process to define a BTF function at a suitable resolution demands large acquisition times and

storage requirements for the six dimensional.

To overcome the difficulties associated with acquiring general BTF functions, we can reduce the problem to a simpler state by eliminating some of the parameters. In particular, there are three subsets of BTFs that are particularly interesting: BSSRDF, BTDF, and the previously discussed BRDF.

The *bidirectional surface scattering reflectance distribution function* (BSSRDF) ignores the specular transmission component of the BTF; the BSSRDF is the contribution of the reflected and the surface scattered light. The *bidirectional transmittance distribution function* (BTDF) defines the distribution of the transmitted light on the opposite side of the surface.

For the purpose of this paper, we will only consider the use of BRDFs.

2.2.4 Illumination Integral

At any given point on the surface, infinitely many light rays interact at different angles and intensities. The emitted illumination is given by the integral of the area element over the surface of the unit sphere [8]. Because we are evaluating the radiance on a planar surface, the domain of integration can be reduced to the hemisphere. That is, no light is transmitted or absorbed, and any coordinate facing away from the surface normal will produce no radiance.

Definition 6. BRDF Radiance

$$L_o = \int_{\mathbf{S}_{hemi}} f_r(\theta_o, \phi_o, \theta_i, \phi_i) L_i(\theta_i, \phi_i) \cos(\phi_i) d\omega_i$$

Provided a suitably represented BRDF f_r and incident light function L_i, rendering realistic images requires the evaluation or approximation of this illumination integral.

2.3 Integral Approximation

2.3.1 Monte Carlo Integration

Among the many method for integral evaluation, *Monte Carlo integration* is a popular choice, despite the slow convergence rate. Monte Carlo ray tracing is often noisy and infeasible for real-time applications.

Given some function and its domain, we can use Monte Carlo to evaluate its integral, $I = \int_{\mathbf{Z}} f(z)dz$.

Definition 7. Monte Carlo Approximation

$$I \approx \frac{1}{n}\sum_{i}^{n}\frac{f(z_i)}{p(z_i)}, \text{ for } n \gg 0$$

By randomly distributing z_i, the PDF $p(z_i) = \frac{1}{V}$, where $V = \int_{\mathbf{Z}} dz$.

$$\frac{1}{n}\sum_{i}^{n}\frac{f(z_i)}{p(z_i)} = \frac{V}{n}\sum_{i}^{n} f(z_i)$$

Monte Carlo integration uses the set of uniformly distributed variables $\{z_i\}$ of size n to evaluate the average of the function $f(z)$. The volume of the domain of integration, V, multiplied with the average provides an estimate of the integral. We now prove that the expected value of the Monte Carlo summation is I (see page 14).

Because the method is stochastic, or nondeterministic, the disadvantage of using the Monte Carlo method is noise. In particular, the number of samples required for convergence of the integral to occur has a rate of $O(\sqrt{n})$, although the rate can be improved slightly by using pseudo-random numbers.

$$
\begin{aligned}
\lim_{n\to\infty} E[\frac{V}{n}\sum_{i=1}^{n} f(z_i)] &= \lim_{n\to\infty} \frac{V}{n}\sum_{i=1}^{n} E[f(z_i)] \\
&= \lim_{n\to\infty} \frac{V}{n}\sum_{i=1}^{n} \frac{\int_{\mathbf{Z}} f(z)dz}{\int_{\mathbf{Z}} dz} \\
&= \lim_{n\to\infty} \frac{V}{n}\frac{\int_{\mathbf{Z}} f(z)dz}{V}\sum_{i=1}^{n} 1 \\
&= \lim_{n\to\infty} \frac{I}{n} n \\
&= I
\end{aligned}
$$

2.3.2 Dirac Delta Function

Because of the slow convergence rate, solving the integral problem explicitly in real-time is infeasible. Moreover, a solution with any noise is visually distracting to the human eye. The Dirac delta function can be used to derive a simpler form of the integral that can be evaluated relatively quickly and with no noise.

Definition 8. The Dirac delta function

$$
\begin{aligned}
\delta_z(x) &= \begin{cases} 0 & \text{if } |x| > z \\ \frac{1}{2z} & \text{if } |x| \leq z \end{cases} \\
\delta(x) &= \lim_{z\to 0} \delta_{|z|}(x)
\end{aligned}
$$

The function has the following set of properties:

$$
\delta(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ \infty & \text{if } x = 0 \end{cases} \tag{2.8}
$$

$$
\int_{-\infty}^{\infty} \delta(x)dx = 1 \tag{2.9}
$$

$$
\int_{-\infty}^{\infty} f(x)\delta(x-a)dx = f(a) \qquad \text{for } f \text{ continuous} \tag{2.10}
$$

Because δ is nonnegative for all x and equation 2.9 is satisfied, the Dirac delta function can be interpreted as a probability density function (though technically not a function). Given the set of points $\mathbf{A} = \{a_k \in \mathbf{R} \mid k = 1, 2, \ldots, N\}$, we can extend the Dirac delta function to isolate a finite number of points.

$$\delta_{\mathbf{A}}(x) = \frac{1}{N}\sum_{i=1}^{N}\delta(x - a_i) \tag{2.11}$$

$$\int_{-\infty}^{\infty}\delta_{\mathbf{A}}(x)dx = 1 \tag{2.12}$$

By using such a PDF, we can approximate the integral by isolating the density to finitely many points. That is, we argue that $\int_{-\infty}^{\infty} f(x)dx \approx \int_{-\infty}^{\infty} f(x)\delta_{\mathbf{A}}(x)dx$, though this largely depends on our choice of $\mathbf{A}$.

The Dirac delta function can be extended to fit other sets of coordinates, such as our spherical coordinates on the unit sphere. Our extended Dirac delta function must satisfy the same properties for spherical coordinates.

Definition 9. The extended Dirac delta function

$$\delta_z(\theta, \phi) = \begin{cases} 0 & \text{if } |\phi| > z \\ \frac{1}{A(z)} & \text{if } |\phi| \leq z \end{cases}$$
$$\delta(\theta, \phi) = \lim_{z \to 0} \delta_{|z|}(\theta, \phi)$$

Note that A(z) is the area on the unit sphere spanned by $\theta \in [0, 2\pi)$ and $\phi \leq z$. This function has similar properties to 2.8, 2.9, and 2.10, and we can once again define the set $\mathbf{A} = \{a_k = (a_{k\theta}, a_{k\phi}) \in \mathbf{S} \mid k = 1, 2, \ldots, N\}$ to use for our PDF.

$$\delta_{\mathbf{A}}(\theta,\phi) = \frac{1}{N}\sum_{i=1}^{N}\delta(\theta - a_{i\theta}, \phi - a_{i\phi}) \tag{2.13}$$

$$\int_{\mathbf{S}} \delta_{\mathbf{A}}(\theta,\phi)dx = 1 \tag{2.14}$$

Instead of evaluating the illumination integral directly as with the Monty Carlo or an alternative method, we can use the Dirac delta function as the light function PDF to reduce the integral to a summation. Minimizing the error of the approximation depends on the choice of the finite set **A**.

Definition 10. BRDF Radiance with Dirac delta function

$$\begin{aligned}
L_o &= \int_{\mathbf{S}_{hemi}} f_r(\theta_o,\phi_o,\theta_i,\phi_i)L(\theta_i,\phi_i)\cos(\phi_i)d\omega_i \\
&\approx \int_{\mathbf{S}_{hemi}} f_r(\theta_o,\phi_o,\theta_i,\phi_i)L(\theta_i,\phi_i)\delta_{\mathbf{A}}(\theta_i,\phi_i)\cos(\phi_i)d\omega_i \\
&= \int_{\mathbf{S}_{hemi}} f_r(\theta_o,\phi_o,\theta_i,\phi_i)L(\theta_i,\phi_i)\frac{1}{N}\sum_{k=1}^{N}[\delta(\theta_i - a_{k\theta}, \phi_i - a_{k\phi})]\cos(\phi_i)d\omega_i \\
&= \frac{1}{N}\sum_{k=1}^{N}\int_{\mathbf{S}_{hemi}} f_r(\theta_o,\phi_o,\theta_i,\phi_i)L(\theta_i,\phi_i)\delta(\theta_i - a_{k\theta}, \phi_i - a_{k\phi})\cos(\phi_i)d\omega_i \\
&= \frac{1}{N}\sum_{k=1}^{N} f_r(\theta_o,\phi_o,a_{k\theta},a_{k\phi})L(a_{k\theta},a_{k\phi})\cos(a_{k\phi}), \text{ by property 2.10}
\end{aligned}$$

2.3.3 BRDF Basis Representation

The BRDF data that is given by the acquisition device [7] must be stored in memory. For our BRDF representation, we discuss three methods to fit the data in memory.

Discrete Fitting The BRDF can be stored discretely by creating a four dimensional table, where each 4-tuple coordinate refers to the function value. The advantage is that the BRDF function is explicitly defined. A serious disadvan-

tages is the space requirement. For instance, a low resolution representation of the BRDF may consist of 90 samples for ϕ by 360 samples for θ. With the assumption that we are storing RGB data where each component is a 32 bit float, the memory consumption becomes 11.732 gigabytes.

Basis Fitting As in many situations, we turn to finding a suitable basis set to approximate the BRDF function. Because we are interested in evaluating the BRDF function over the unit sphere, the set of *spherical harmonics basis functions* [10] is a suitable basis to form our BRDF function. The linear combination of a set of continuous basis functions and uniquely determined coefficient functions that fit to the BRDF data provides a visually smooth result.

Model Fitting Model fitting is the process of defining an analytical model, and setting the model's parameters to approximate the acquired data. For instance, the Blinn-Phong illumination model is once again comprised of a diffuse, specular, and ambient term. By determining the diffuse component or specularity of a material, we can then fit the data to the given model by tuning the parameters appropriately. However, we are limited to a model's constraints; because Blinn-Phong cannot model anisotropy, anisotropic materials cannot be represented by that particular model.

2.3.4 Spherical Harmonics

The set of spherical harmonic functions is an orthogonal basis commonly used in computer graphics because it is parametrized by spherical coordinates on the unit sphere. The spherical harmonics, defined by $Y_l^m(\theta, \phi)$, is the angular portion of the solution to Laplace's equation.

Definition 11. Spherical Harmonics

$$Y_l^m(\theta,\phi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}}P_l^m(\cos\theta)e^{im\phi}$$
$$\text{where } l \in \{0,1,2,\ldots\} \text{ and } m \in= \{-l,-l+1,\ldots,l-1,l\}$$

We can thus use them to approximate our BRDF function by linearly combining a finite set of these spherical harmonic functions. As with any orthogonal basis, we must find and store the set of coefficients for each l and m that approximate our BRDF.

Definition 12. BRDF Function Represented by Spherical Harmonics. Incident light coordinates are given by (θ_i, ϕ_i) and exitant light coordinates, the view vector, is given by (θ_o, ϕ_o).

$$\sum_{l\in\{0,1,2,\ldots\}} \sum_{m\in=\{-l,\ldots,l\}} c(\theta_o,\phi_o)Y_l^m(\theta_i,\phi_i)$$

Note the simplicity of the BRDF function. Given the coefficient and basis values, the BRDF function value is produced by a few multiplications.

Using a basis function to determine the approximation of the BRDF function does have consequences. For higher-order data, ringing artifacts can be introduced, an unwanted oscillation of the BRDF values. Specular materials are often the culprit because they require the higher-order basis functions. To resolve the ringing artifacts, we can fit the acquired data to an analytical model.

For $\phi > \frac{\pi}{2}$, the light has no direct illumination onto the surface. Therefore, any light satisfying $\phi > \frac{\pi}{2}$ can be removed from the illumination integral equation. There is no need in storing the spherical harmonics for $\phi > \frac{\pi}{2}$, and so each order of spherical harmonics only represents half of the function values. In figure 2.5, the values are parametrized for $(s,t) \in [0,1] \times [0,1]$ such that texture coordinate $(0,0)$ (bottom-left) is spherical coordinates $(0,0)$ and texture

coordinate $(1, 1)$ (upper-right) is spherical coordinate $(2\pi, \frac{\pi}{2})$.

Figure 2.5 is scaled such that a function value of 0 is represented by black and a function value of 1 is represented by white. Any values outside of the range are clamped to $[0, 1]$. In particular, the black areas consist of negative function values.

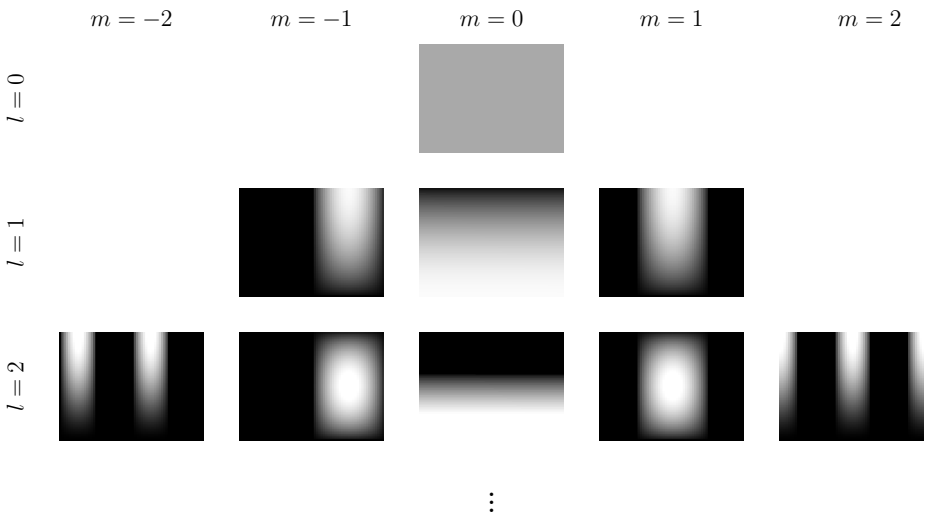


Figure 2.5: Spherical Harmonics

2.3.5 Ashikhmin-Shirley Model

The *Ashikhmin-Shirley model* [2] is another analytical model developed to represent materials that the classical models cannot. The Anisotropic Phong BRDF model has the following properties:

- Energy conservations and reciprocity laws
- Anisotropic reflections
- Fresnel behaviour
- Non-constant diffuse term

The energy conservations and reciprocity laws define that no light energy is lost or gained; it is either reflected, absorbed, or transmitted. Anisotropic reflections allows a brushed metal appearance, where light is not reflected equally along different axes. The Fresnel term defines the reflectance of the material as the angle of view approaches a grazing angle. The non-constant diffuse term provides more variation to the diffuse reflection.

The model is another classical sum of the often used diffuse and specular term, $p(k_1, k_2) = p_s(k_1, k_2) + p_d(k_1, k_2)$ where k_1 is the normalized light vector and k_2 is the normalized view vector. The model diverges from the Phong or Blinn-Phong model by redefining the specular component analytical model.

$$p_s(k_1, k_2) = \frac{\sqrt{(n_u + 1)(n_v + 1)}}{8\pi} \frac{(n \cdot h)^{n_u \cos^2 \phi + n_v \sin^2 \phi}}{(h \cdot k) \max((n \cdot k_1), (n \cdot k_2))} F((k \cdot h))$$

The Fresnel fraction F is defined as $F((k \cdot h)) = R_s + (1 - R_s)(1 - (k \cdot h))^5$ where R_s is a 3-component RGB color that specifies the specular reflectance. Note that n is the surface normal, h is the normalized half-vector between k_1 and k_2, k is either k_1 or k_2, and n_u, n_v control the shape of the specular lobe.

Similarly, the diffuse term is redefined as follows, where R_d is a 3-component color RGB describing the diffuse reflectance:

$$p_d(k_1, k_2) = \frac{28 R_d}{23\pi}(1 - R_s)(1 - (1 - \frac{(n \cdot k_1)}{2})^5)(1 - (1 - \frac{(n \cdot k_2)}{2})^5)$$

2.3.6 Distribution-based BRDF model

Although the Ashikhmin-Shirley model has many desirable properties, it does not offer sufficient flexibility for many materials. The *Distribution-based BRDF model* (D-BRDF) [1] is a generalization of the Ashikhmin-Shirley analytical model.

The D-BRDF model is defined as follows:

$$p(k_1, k_2) = \frac{cp(h)F((k \cdot h))}{(k_1 \cdot n) + (k_2 \cdot n) - (k_1 \cdot n)(k_2 \cdot n)}$$

Again, F represents the Fresnel term discussed in the Ashikhmin-Shirley section. $p(h)$ is the anisotropic Phong function parametrized by the half-vector h covering the hemisphere. Given the table $p(h)$ and the coefficient c from fitting the acquired data to the model, the D-BRDF model defines a simpler and more flexible method to represent the BRDF.

Chapter 3

Implementation

Given a set of BRDF coefficients and spherical harmonic values for the basis representation, we store the data set in a pair of three-dimensional textures. By converting the Cartesian coordinates for incident and exitant light to the appropriate BRDF texture coordinates, we can sample the coefficients and basis function values from the look-up tables. Given a set of point lights from an environment map or otherwise, we can perform the illumination in real or interactive time.

3.1 BRDF Coordinates

Given a set of Cartesian coordinates, we must determine the texture coordinates to sample the appropriate texel. We can generalize the transformation of coordinates to four phases: Cartesian coordinates, spherical coordinates, texel coordinates, and texture coordinates.

The code for section 3.1 is written using Cg, a programming language based on C for the purpose of writing vertex, geometry, and fragment shaders.

3.1.1 Cartesian Coordinates

The geometry of the scene is usually comprised of a set of models and light sources. The models and lights are given in local coordinates, and are transformed to global coordinates.

Code 1.

```
...
result.light    = mul(LightModelView, light);
result.normal   = mul(ModelViewInvTrans, normal);
result.position = mul(ModelView, position);
...
```

By performing these matrix transformations, we transform each vector such that they are relative to each other. That is, the light direction can be determined by the difference of model position and the light. Similarly, the view vector is given by the negated position, such that the view vector at the given position points towards $(0, 0, 0)$.

Code 2.

```
...
light  =  normalize(light - position);
normal =  normalize(normal);
view   = -normalize(position);
...
```

Provided these normalized vectors, we can determine the spherical coordinates of the light and view vector with respect to the normal vector.

3.1.2 Spherical Coordinates

Given the Cartesian coordinates for the light vector and the view vector to evaluate the BRDFs, we must transform them into spherical coordinates. For the

Cg code, the `ViewSphericalCoordinates` and `LightSphericalCoordinates` variables contain the texture coordinates for sampling the coefficient and spherical harmonic functions. The x- and y-components of the variables provide the $s \in [0,1]$ and $t \in [0,1]$ texture coordinate, respectfully.

The ϕ value is uniquely determined, and is given by $\cos^{-1}(view \cdot normal)$. The θ coordinate is not uniquely determined, and a user-defined vector is provided to allow a unique choice of θ. We decide on using the vector $(0,1,0)$, because the conversion from Cartesian to spherical coordinates only becomes ill-conditioned near grazing angles.

To determine the coordinates of a vector with respect to the normal and the user-defined vector $(0,1,0)$, we perform a backface culling operation on the view and light vectors. If $v \cdot n < 0$ where n is the normal and v is the view or light vector, the pixel either faces away from the viewer or the light does not illuminate the pixel. Otherwise, if $v \cdot n \geq 0$, we can compute the spherical coordinates as follows:

$$\phi = \cos^{-1}\left(\frac{v \cdot n}{\|v\|\|n\|}\right) \tag{3.1}$$

$$e_2 = (0,1,0)$$

$$proj_n(v) = \frac{v \cdot n}{\|n\|^2} n \tag{3.2}$$

$$a = v - proj_n(v), \quad b = e_2 - proj_n(e_2)$$

$$\theta = \cos^{-1}\left(\frac{a \cdot b}{\|a\|\|b\|}\right) \tag{3.3}$$

The $proj_n(v)$ function defined by equation 3.2 projects the vector v onto the vector n.

Code 3.

```
...
upVector = float3(0.0f, 1.0f, 0.0f);
upVector = normalize(upVector - dot(upVector, normal) * normal);

viewSphericalCoordinates.x = 0.5f * acos(dot(normalize(view - dot(view,
normal) * normal), upVector)) / 3.141592654f;
viewSphericalCoordinates.y = acos(dot(view, normal)) * 2.0f / 3.141592654f;

lightSphericalCoordinates.x = 0.5f * acos(dot(normalize(light - dot(light,
normal) * normal), upVector)) / 3.141592654f;
lightSphericalCoordinates.y = acos(dot(light, normal)) * 2.0f / 3.141592654f;
...
```

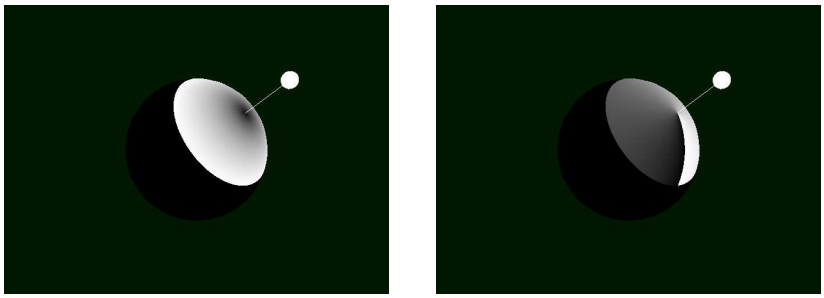


Figure 3.1: Spherical Coordinates for Point Light. The figure visualizes ϕ values per pixel (left) and θ values per pixel (right).

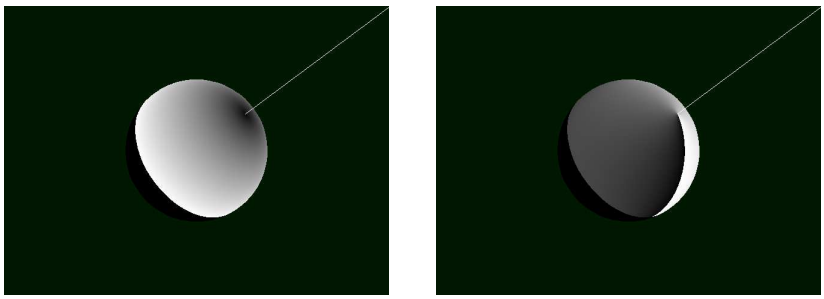


Figure 3.2: Spherical Coordinates for Directional Light. The figure visualizes ϕ values per pixel (left) and θ values per pixel (right). Note that the light vector coordinates is constant across all pixels.

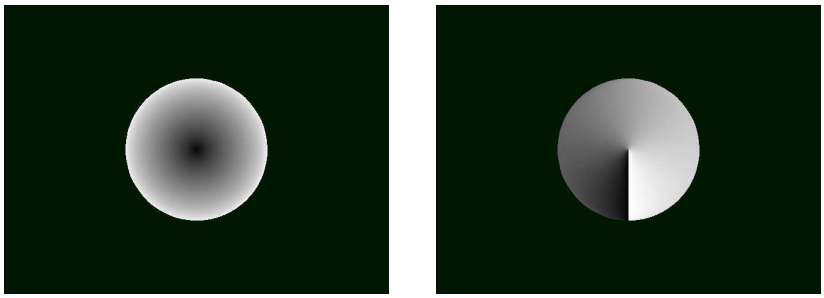


Figure 3.3: Spherical Coordinates for View Vector. The figure visualizes ϕ values per pixel (left) and θ values per pixel (right).

3.1.3 Texel Coordinates

Given the (θ, ϕ) values, we can map them to the appropriate texel coordinates. That is, we define that texel $(0, 0)$ holds the value for spherical coordinates $(0, 0)$. Similarly, texel $(n-1, m-1)$ (upper-right texel) contains the data for spherical coordinate $(2\pi(1-\frac{1}{n}), \frac{\pi}{2}(1-\frac{1}{m}))$, as demonstrated in figure 3.4. Therefore, we scale the coordinates by multiplying each spherical coordinate by $(\frac{n}{2\pi}, \frac{2m}{\pi})$.

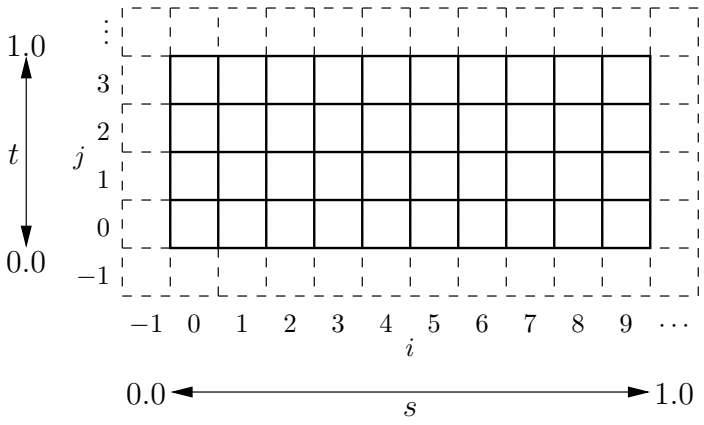


Figure 3.4: Texel Coordinates

3.1.4 Texture Coordinates

We discretely store the BRDF data as texture data using the OpenGL API [9]. The texture coordinates $(s,t) \in [0,1] \times [0,1]$ uniformly maps to the spherical coordinates. The purpose of storing the compressed BRDF data in textures is such that the memory is accessible by fragment programs implemented on the GPU for the hardware-accelerated computation of each pixel fragment.

The texture memory is defined by GL_RGBA_FLOAT32_ATI, a 4-component 32-bit float. Given any texel parametrized by (θ, ϕ), three components are used to store the coefficients for the BRDF, and the fourth component stores the basis value for the spherical harmonics function. We can decouple the BRDF coefficients from the spherical harmonics table by separately defining texture memory of type GL_RGB_FLOAT32_ATI and GL_ALPHA_FLOAT32_ATI.

Given any set of values $\phi \in [0, \pi/2]$ and $\theta \in [0, 2\pi]$, $s = \frac{\theta}{2\pi}$ and $t = \frac{\phi}{\pi/2}$. Recall that coordinates with $\phi \in (-\pi/2, 0)$ fail the backface culling test.

In OpenGL, we can sample textures using a combination of texture filters. Because the texture type is GL_*_FLOAT32_ATI, our hardware requires that GL_TEXTURE_MAG_FILTER is set to GL_NEAREST. For hardware without the limitation, we can consider using GL_LINEAR to linearly interpolate between the BRDF data. Otherwise, linear interpolation is written manually into the fragment shader.

When sampling a texture, we must be aware that the texture coordinates $(s,t) = (0,0)$ does not sample at the center of a texel. Figure 3.4 demonstrates that the center for texel $(0,0)$ is given by a nonzero texture coordinate value. For this example, each texel has width $1/10$ and height $1/4$. Therefore, the value of texel $(0,0)$ is given by texture coordinates $(\frac{1}{2}\frac{1}{10}, \frac{1}{2}\frac{1}{4})$. We must be cautious when transforming the coordinates so that they sample the correct pixel.

3.1.5 BRDF evaluation

The basis representation of the BRDF described by definition 12 is implemented by a for-loop in Cg. For each l and m, we are given the table of values parametrized by (θ, ϕ). By using GL_TEXTURE_3D to store each layer of data into a single three-dimensional texture, we can iterate through each layer and sample appropriately.

Note that `DepthInfo.x` $= \frac{1}{N}$, where N is set to the number of layers. As with two-dimensional (s, t) coordinates, caution is required when attempting to sample from the appropriate layer. Note that to sample the layer i from a three-dimensional texture, the coordinates are set to $(s, t, (\frac{1}{2} + i)\frac{1}{N})$.

Code 4.

```
...
color = 0.0f;

viewSphericalCoordinates.z  = 0.5f * DepthInfo.x;
lightSphericalCoordinates.z = 0.5f * DepthInfo.x;

for (int i = 0; i < DepthInfo.y; i++) {
    coefficient = tex3D(coefficientTexture, viewSphericalCoordinates);
    sh          = tex3D(shTexture, lightSphericalCoordinates);

    color += coefficient * sh;

    viewSphericalCoordinates.z  += DepthInfo.x;
    lightSphericalCoordinates.z += DepthInfo.x;
}
...
```

Because the coefficients texture utilizes only a single component, a reduction to the number of texture samples can be made by encoding four coefficient values into one texel. A similar strategy can be used to take advantage of the additional alpha channel in spherical harmonics texture, or by packing additional data into each component of either texture. This effectively reduces the number of texel

fetches, a bottleneck in the performance of the fragment shader.

3.2 Environment Mapping

We choose to use a set of point samples to represent the environment map. This finite subset of points to represent L_i follow from the discussion of the Dirac delta function. The algorithms in this section are written in MATLAB.

3.2.1 Light Probe

Debevec defines a light probe image to be an omnidirectional high-dynamic range image [6]. The light probe defines the incident illumination at a given point in space. With the light probe defined, we can sample the incoming light at any direction at the defined point. By pre-computing the incident light at a particular point, we can localize the rendering of an object.

Formally, a light probe, also known as an environment map, is a function $e : \mathbf{S} \rightarrow \mathbf{R}^3$. The set $\mathbf{R}^3$ defines the three nonnegative red, green, and blue components of a pixel. Recall that we define $\mathbf{S}$ to be the set of directions parametrized by θ and ϕ.

There are two variants to the physical storage of the light probe function. The cube environment map stores the values over the six sides of a cube centered at the given point of the light probe. The alternate method is mapping the function to a matrix $\mathbf{A}$, where $\mathbf{A} \in \mathbf{R}^{m \times n}$ and $a_{ij} = e(2\pi\frac{j}{n}, \pi\frac{i}{m} - \frac{\pi}{2})$. The storage for this method is referred to as a spherical environment map. Note that this method is essentially equivalent to the storage of the BRDF coefficient and spherical harmonic function values.

The disadvantage of spherical environment maps, or simply environment maps, is that the mapping gets increasingly "stretched" near the poles, where $\phi = \pm\frac{\pi}{2}$. However, the coordinates of the texture are easier to decipher than the

alternate cube environment map. Figure 3.5 represents the mapping between the sphere and the block of memory $[-\frac{\pi}{2}, \frac{\pi}{2}] \times [0, 2\pi]$.

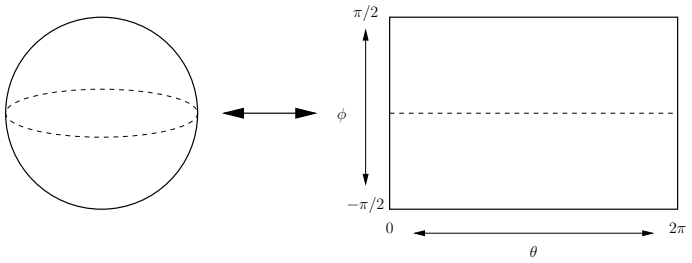


Figure 3.5: Spherical Environment Map

For our applications, we use the high-dynamic range spherical environment map acquired from Grace Cathedral. The environment map is characteristically dark, except for a few sparse bright areas. Moreover, we will define $L_i = e$ in order to evaluate the illumination integral.

3.2.2 Importance Sampling

Randomly selecting a set of points for each surface point would be an example of unbiased sampling. This method of sampling would be very expensive for real-time rendering applications, and suffers from high variance in the estimate between surface points.

Instead, we choose a biased representation of the environment illumination by collapsing the environment map into a set of point lights. These points are then used for all surface points in the rendering. By pre-computing the set of points, this method can be applied to real-time rendering and removes the variance in the integral estimate between adjacent surface points.

Unweighted Sampling We can randomly select a set of directions to sample the environment map. By randomly selecting $x \in R^3$, the direction $d = \frac{x}{\|x\|_2}$ is

uniformly selected. By mapping the Cartesian coordinates to spherical coordinates, we can directly sample the spherical environment map.

Another method to is randomly select coordinates and sample directly from the spherical environment map. If we attempted this, we would notice that multiple points congregate near the poles of the unit sphere; the spherical environment is not distributed uniformly across the entire $[-\frac{\pi}{2}, \frac{\pi}{2}] \times [0, 2\pi]$ area. Instead, we notice that we have a probability distribution $\mathbf{P} \in \mathbf{R}^{m \times n}$ where $p_{ij} = cos(\pi \frac{i}{m} - \frac{\pi}{2}) = sin(\pi \frac{i}{m})$. Therefore, we can define the uniformly distributed spherical environment map $\hat{\mathbf{A}}$ where $\hat{a}_{ij} = a_{ij} p_{ij}$. Even though randomly selecting from $\hat{\mathbf{A}}$ also produces clustering at the poles, the points are uniformly distributed in terms of pixel intensity.

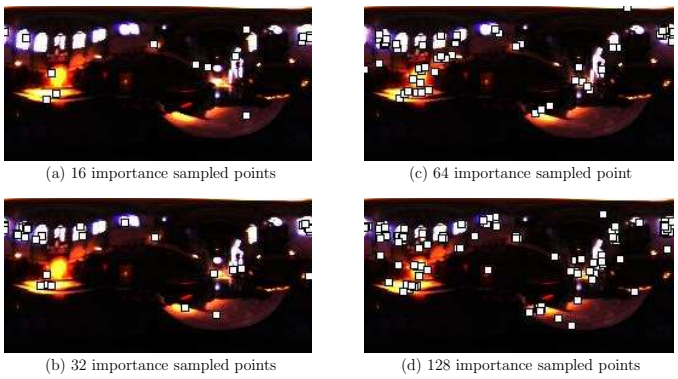
Weighted Sampling In Grace Cathedral and other environment maps, there are certain areas in the environment map that are brighter. Grace Cathedral is a particularly compelling contrast between bright and dark areas. By concentrating sampling to the brighter areas, we create a better approximation of the illumination integral using fewer points.

Algorithm 1. Importance Sampling Algorithm

1. Define $\mathbf{A}$ as the monochrome environment map.
2. Suppose $e = (\underbrace{1, 1, \ldots, 1}_{n})$. Compute a vector $r = \mathbf{A}e$ of row intensities. Define a PDF by normalizing r, and define the corresponding CDF.
3. Similarly, each row is defined as:

$$\mathbf{A} = \begin{pmatrix} - & \mathbf{a}_1^T & - \\ & \vdots & \\ - & \mathbf{a}_m^T & - \end{pmatrix}$$

   Define a CDF for each $\mathbf{a}_i$.
4. Generate a set of random tuples in $[0, 1] \times [0, 1]$ for each point.
5. By using binary search or otherwise, find the value i that maps, according to the CDF of r, to the randomly generated value.
6. Given i, find the value j that maps, according to the CDF of $\mathbf{a}_i$, to the other randomly generated value. a_{ij} is the sampled point. Repeat steps 4 to 5 for every random tuple.



(a) 16 importance sampled points

(c) 64 importance sampled point

(b) 32 importance sampled points

(d) 128 importance sampled points

Figure 3.6: Importance Sampling on Grace Cathedral

In figure 3.6, we notice the clustering of points near the windows and alter. We also notice the sparsity of points in the darker areas of the environment map.

Because the importance sampling is biased towards pixel intensities, sampled pixels tend to group around bright areas. For the Grace Cathedral, many points cluster around the windows. Therefore, we get a good approximation where the light is focused at the top hemisphere.

The relative darkness of the rest of the Cathedral forces most light sources to be situated at the upper half of the spherical environment map texture. Because we lack light sources with $\phi < 0$, any pixel with a normal pointing towards the floor will have a poor approximation.

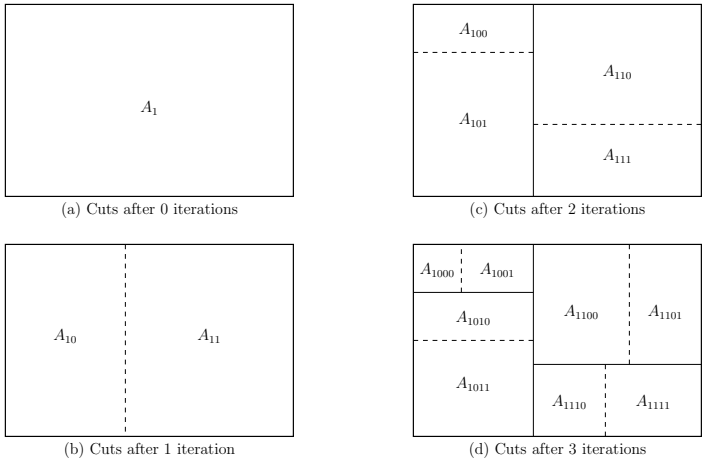
```
function result = ImportanceSampling(img, fileinfo, n)

% Weighted average of color channels following ITU-R Recommendation
% BT.709
monocrome = img(:,:,1)*0.2125 + img(:,:,2)*0.7154 + img(:,:,3)*0.0721;

N = fileinfo.width;
M = fileinfo.height;

x = 1:N;
y = 1:M;

phi = linspace(-pi/2,pi/2,M);

rowPDF = sum(monocrome, 2)' .* cos(phi);
rowPDF = rowPDF ./ sum(rowPDF);
rowCDF = cumsum(rowPDF);

colPDF = monocrome ./ repmat(sum(monocrome, 2),1,N);
colCDF = cumsum(colPDF,2);

result = img; % zeros(M,N);

for (i = 1:n)
    c1 = min(find(rand <= rowCDF));
    c2 = min(find(rand <= colCDF(c1,:)));

    result(max(c1-3,1):min(c1+3,M),max(c2-3,1):min(c2+3,N),:) = 0;
    result(max(c1-2,1):min(c1+2,M),max(c2-2,1):min(c2+2,N),:) = 255;
end

end
```

3.2.3 Median Cut Algorithm

Although there are many importance sampling algorithms that can be designed to decide on particular sampled points, our discussion moves towards the deterministic *Median Cut algorithm* [5]. Because of the sparsity of points in the importance sampled results, certain reflective angles will be approximated with fewer light sources. The Median Cut algorithm is biased towards light intensity and distribution over space.

Algorithm 2. Median Cut Algorithm.

1. Add the entire monochrome light probe image to the region list as a single region.
2. For each region in the list, subdivide along the longest dimension such that its light energy is divided evenly.
3. If the number of iterations is less than n, return to step 2.
4. Place a light source at the center or centroid of each region, and set the light source color to the sum of pixel values within the region.



(a) Cuts after 0 iterations

(b) Cuts after 1 iteration

(c) Cuts after 2 iterations

(d) Cuts after 3 iterations

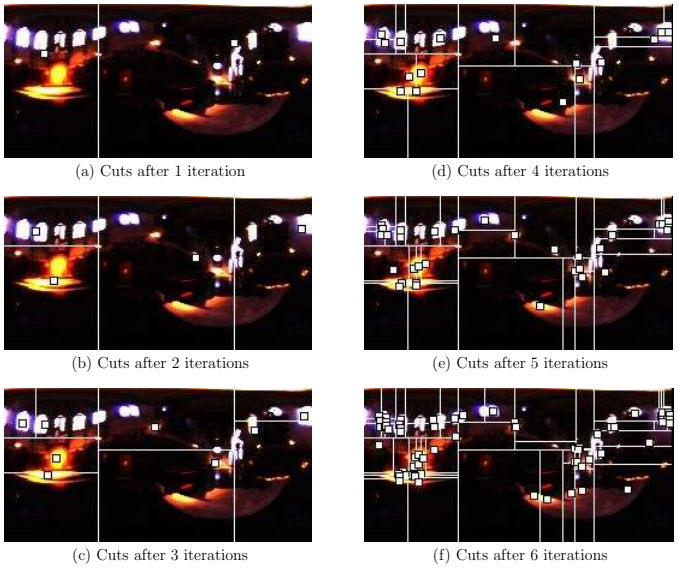
Figure 3.7: Median Cut Algorithm

At each iteration of the algorithm, the region $\mathbf{A}_b$ is partitioned into regions $\mathbf{A}_{b0}$ and $\mathbf{A}_{b1}$, such that $\sum_{a_{ij} \in \mathbf{A}_{b0}} a_{ij} \approx \sum_{a_{ij} \in \mathbf{A}_{b1}} a_{ij}$. That is, the environment

map is subdividing the intensities of each region. Effectively, the resulting set of regions have equal intensities.

Notice in figure 3.7 that the region $\mathbf{A}_{101}$ is subdivided horizontally, depicting the subdivision of the longest dimension.

For spherical environment maps, recall that the mapping of the environment to the discretized region is not a uniform distribution. To compensate, we perform the scalar multiplication of matrices $\mathbf{A}$ and $\mathbf{P}$ before proceeding.



(a) Cuts after 1 iteration

(d) Cuts after 4 iterations

(b) Cuts after 2 iterations

(e) Cuts after 5 iterations

(c) Cuts after 3 iterations

(f) Cuts after 6 iterations

Figure 3.8: Median Cut Algorithm on Grace Cathedral

By comparing figure 3.6 to figure 3.8, we notice that points congregate near bright areas. However, the Median Cut algorithm attributes more points to relatively darker regions. We achieve a better distribution of points over the entire environment map.

Recall in definition 10 that there is a factor of $\frac{1}{N}$ associated with the BRDF basis representation. Because we sum the pixel values of each region, we are effectively evaluating every discrete light value in the environment map. There-

fore, even though $N = 2^i$ lights are generated for some $i \in \mathbf{Z}^+$, the scaling factor becomes $\frac{1}{\sum p_{ij}}$.

```
% MedianCutAlgorithm
%
% Written by Matthew O'Toole (mpotoole@interchange.ubc.ca)
%
% The algorithm is based on Paul Debevec's "A Median Cut Algorithm for
% Light Probe Sampling".
%
%
function result = MedianCutAlgorithm(img, fileinfo, n)

% Weighted average of color channels following ITU-R Recommendation
% BT.709
monocrome = img(:,:,1)*0.2125 + img(:,:,2)*0.7154 + img(:,:,3)*0.0721;

N = fileinfo.width;
M = fileinfo.height;

x = 1:N;
y = 1:M;

phi = linspace(-pi/2,pi/2,M);

scale = repmat(cos(phi)',1,N);
mat = monocrome .* scale;

result = img; % zeros(M,N);
result = Helper(result, mat, n, 0, 0);

end


function result = Helper(result, mat, n, x, y)

half_intensity_x = 0;
half_intensity_y = 0;
total_intensity = sum(sum(mat));

i = 0;
j = 0;

[M,N] = size(mat);

while(half_intensity_x < total_intensity / 2)
    i = i + 1;
    half_intensity_x = half_intensity_x + sum(mat(:,i));
end
while(half_intensity_y < total_intensity / 2)
    j = j + 1;
    half_intensity_y = half_intensity_y + sum(mat(j,:));
end
```

```
if (n == 0)
    c1 = y+j;
    c2 = x+i;

    [M,N] = size(result);

    result(max(c1-3,1):min(c1+3,M),max(c2-3,1):min(c2+3,N),:) = 0;
    result(max(c1-2,1):min(c1+2,M),max(c2-2,1):min(c2+2,N),:) = 255;
else if (M < N)
    result(y+1:y+M,x+i,:) = total_intensity;

    result = Helper(result, mat(:,1:i),   n-1, x,   y);
    result = Helper(result, mat(:,i+1:N), n-1, x+i, y);
else
    result(y+j,x+1:x+N,:) = total_intensity;

    result = Helper(result, mat(1:j,:),   n-1, x, y);
    result = Helper(result, mat(j+1:M,:), n-1, x, y+j);
end

end
```

3.3 Results

The remainder of the implementation is given to writing the Cg fragment shaders that compute the sum of products for the BRDF basis representation, and evaluate the D-BRDF model for fitted BRDF data. This section lists a set of figures that demonstrate a subset of the acquired BRDF data sets.

The performance averages at 30 FPS (real-time performance) for five orders of spherical harmonics under a point light source, and 1-2 FPS (interactive performance) for 128 directional light sources. For higher-order data sets of seven or eight orders, the performance is affected by a drop to 15-20 FPS for the point light source and 1 FPS for environment lighting. The D-BRDF model does not suffer from such performance penalties, and can render over 100 FPS. The performance was recorded on a 640 by 480 window, using a GeForce 7600 GT as the GPU.

Figure 3.9: Blue Synthetic Fabric. A directional light vector illuminates the teapot from above (left), and a set of 128 lights determined by using the Median-Cut algorithm illuminates the teapot with the Grace Cathedral light probe (right).

Figure 3.10: Blue Silk. The rendered images of blue silk applies the same illumination model described in figure 3.9. The blue silk fabric is an example of the strong anisotropic properties encoded into the material's BRDF.

Figure 3.11: Lindt Chocolate Wrapping Paper (D-BRDF). The data for this specular paper is acquired using a higher order set of the basis functions, and fitted to the analytical D-BRDF model.

Figure 3.12: Gold Dust Paint. The gold dust paint is a highly specular material represented using the basis representation of the BRDF. Because the material is highly specular, the data is acquired using a higher order set of spherical harmonic basis functions.

Figure 3.13: Teal Paint. As with figure 3.12, the green teal paint is an example of a highly specular material.

Figure 3.14: Magenta Plastic. The plastic material demonstrates the diffuse property of the BRDF data.

Figure 3.15: Red Velvet. The renderings demonstrate the BRDF for the red velvet material.

Chapter 4

Conclusion

This thesis focuses on the methods of rendering acquired BRDF data sets in real-time. By identifying certain key concept that simplify the rendering equation, we offload pixel illumination computations to the GPU. The result is a set of generated images that are synthesized significantly faster than conventional ray tracing approaches, at the cost of providing a rough approximation of the solution.

With the ability to render images efficiently, a set of animated sequences were generated using the renderer described in this thesis. The visualized data was extracted by observing the coefficients of the BRDF directly in a basis function during the acquisition process [7].

The work presented in this paper allows real-time applications to utilize acquired BRDF data sets. The cost reduction of using BRDFs allows photo-realistic illuminations to be approximated in applications where offline rendering is not an option. By enhancing the aspect of realism in such applications, we provide a more immersive and entertaining visual experience to the user.

Bibliography

[1] M. Ashikhmin. Distribution-based BRDFs. http://jesper.kalliope.org/blog/library/dbrdfs.pdf.

[2] M. Ashikhmin and P. Shirley. An anisotropic Phong BRDF model. *J. Graph. Tools*, 5(2):2532, 2000.

[3] D. Burke. Bidirectional importance sampling for illumination from environment maps. Master's thesis, The University of British Columbia, 2004.

[4] J. M. Cohen and P. Debevec. The LightGen HDRShop plugin. http://www.hdrshop.com/main-pages/plugins.html.

[5] P. Debevec. A median cut algorithm for light probe sampling, 2005. SIGGRAPH 2005 Poster.

[6] P. Debevec, Light probe image gallery. http://www.debevec.org/Probes/.

[7] A. Ghosh, S. Achutha, W. Heidrich, and M. O'Toole. BRDF Acquisition with Basis Illumination. UBC Computer Science Technical Report TR-2007-10.

[8] J. T. Kajiya. The rendering equation. In SIGGRAPH 86: *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143150, 1986.

[9] M. Segal and K. Akeley. The OpenGL graphics system: a specification (version 2.1 - December 1, 2006).

[10] E. W. Weisstein. Wolfram MathWorld. http://mathworld.wolfram.com/.