Diplomarbeit

Approximative Real-time Soft Shadows and Diffuse Reflections in Dynamic Scenes

ausgeführt am
Institut für Computergraphik und Algorithmen
der Technischen Universität Wien

unter der Anleitung von
Univ.Ass. Dipl.Ing. Dr.techn. Stefan Jeschke

durch

Paul Guerrero
Matrikelnummer 0026761
Sanatoriumstr. 21b 17/3
A-1140 Wien

Unterschrift

Datum

This thesis describes a method for approximative soft shadows and diffuse reflections in dynamic scenes, based on a method by Ren et al. [32]. An overview of precomputed radiance transfer and spherical harmonics is also presented, as well as a short introduction to global illumination. The proposed method uses a low-order spherical harmonics basis to represent incident radiance and visibility on the hemisphere of a receiver point. Diffuse reflecting geometry and shadow casting geometry is represented as sets of spheres. The spheres of an object approximate its shape and diffuse surface color as seen from any viewpoint. In a first pass, the direct illumination of an object is projected to its spheres and stored along with an approximation of the diffuse surface color as SH vectors defined over the surface of each sphere. In a second pass, the average color and the visibility for each sphere at a receiver point is found. The product of average color and visibility is used to approximate the incident radiance from diffuse reflections. Using a sphere set approximation instead of actual geometry for both soft shadows and diffuse reflections allows us to compute the visibility and diffuse reflections of an object on the fly at runtime. This text also describes a GPU implementation of the method and discusses obtained results. Interactive performance with relatively smooth framerates of over 20 fps is achieved for moderately complex scenes.

Contents

1. Introduction

The visual system is, for most humans, the sense that provides the most information about the surrounding world. It takes the light at some point in space and deduces from it a description of the surrounding objects. This transformation is seemingly done without effort and it is very suggestive, i.e. it is difficult (and usually pointless) to see a scene as light spectrum intensities instead of objects or to view an image as a collection of pixels or brush strokes. This ability of the human visual system to intuitively deduce large amounts of information from a very compact description is the starting point for image generating techniques, because it enables them to intuitively communicate large amounts of information to the viewer. This is not a new idea, painters have used the human visual system as medium of choice to communicate their ideas. Computer graphic techniques automate part of this process by generating an image out of some description of the objects that make up the image. Computer graphics are used in medical visualization, video games, the movie industry, computer aided design, architectural visualization and other fields.

The field of computer graphics can be split into two main disciplines, each posing different requirements on the algorithms developed. In offline rendering, quality is more important than the time it takes to generate an image. The main focus is to create images that are either physically accurate or simply convincing to the viewer. Rendering one image can take anywhere from a second to a few days. Offline renderers usually generate the images and store them for later displaying. In real-time or interactive rendering, it is important that the speed of rendering does not drop below a certain threshold. The speed is measured in fps (frames per second) or Hertz. Often, rendering speed is classified as real-time, interactive or non-interactive, although the specific frame rates for each category depend on the application and are not clearly specified. Usually, interactive frame rates start at about 5 fps and real-time refers to the monitor frequency (typically 60 Hz). High rendering speed allows for interactivity in real-time rendering. Frames are created one at a time and are immediately displayed, possibly as a response to user input.

There are several different approaches to rendering an image. They differ in their computational complexity and in the amount of simplification, i.e. which properties of realistic light propagation they ignore or only approximate for the sake of speed.

The simplest method that is widely-used today is scanline rendering. Scanline rendering can be very fast, but it ignores many obvious phenomena of realistic lighting, like reflections and shadows. These phenomena are usually added as special effects and handled separately. But since they are not natively included in scanline rendering, a complete and efficient implementation of any of these phenomena remains to be a challenge.

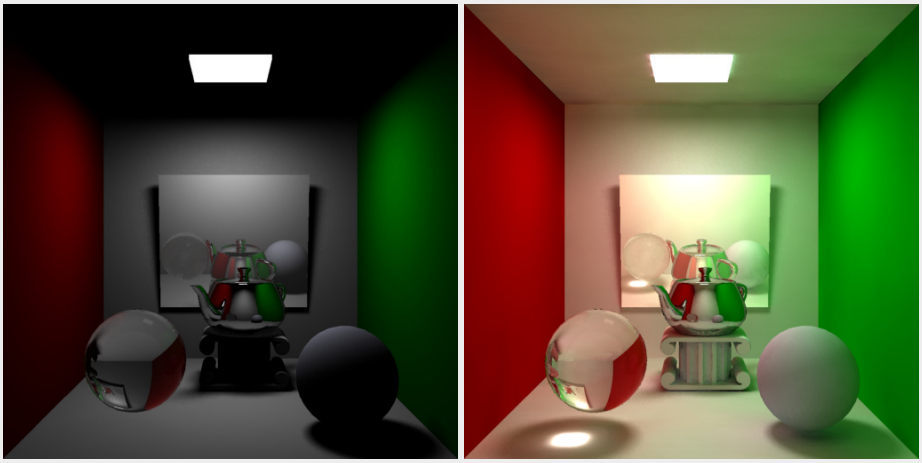


Figure 1.1.: *Left: image rendered with direct lighting only, including reflections and refractions. Right: the same image rendered with global illumination.*

Video games use scanline rendering and it is implemented in the fixed-function pipeline of video cards.

Global Illumination methods more accurately simulate actual light propagation in a scene. They take into account that the light coming from a surface point could potentially be influenced by every other surface in the scene. Thus, such methods can usually simulate phenomena like diffuse and specular reflections, soft shadows, large light sources, etc., but the computational complexity of these methods is relatively high. Figure 1 shows a comparision of global and local illumination rendering.

Traditionally, the broad mass of real-time methods was confined to scanline rendering, as more advanced methods, like the global illumination methods, were too expensive to be computed in real-time and were almost exclusively used in offline rendering. The introduction of video accelerators with a fixed-function pipeline additionally cemented this development, since they could only accelerate applications that used scanline rendering. Other methods would have to rely on the CPU alone and would therefore usually perform much worse. Today, with the advent of programmable GPUs, arbitrary methods can be accelerated by video cards. The massive parallelism found in today's video cards[1] makes the methods run many times faster than on normal CPUs.

In recent years, pushed primarily by the video games industry, the research community for real-time global illumination has grown considerably. The methods developed still suffer from at least one of several limitations, like fixed view or lighting, low frequency lighting, static scenes, approximative geometry, etc., but research is actively working on solving these problems.

This text presents a method for real-time soft shadows and diffuse reflections in dy-

[1] As of this writing, the GeForce 8800 with 128 stream processors and the Radeon HD 2900 with 320 stream processors are the most advanced video cards.

namic scenes with low-frequency lighting environments and approximative geometry. It is largely based on a method for soft shadows presented by Ren et al. [32].

The document is organized as follows: First, in Chapter 2, we will give a short introduction to Global Illumination. In Chapter 3 we will give an overview over similar methods to compute real-time global illumination lighting. There is an introduction to spherical harmonics at the beginning of this chapter since spherical harmonics are the basis for many of the real-time global illumination methods, as well as for the method described in this thesis. In Chapter 4 we describe our method for soft shadows and diffuse reflections. In Chapter 6 we present and discuss results achieved with our method. We conclude in Chapter 7.

2. Global Illumination

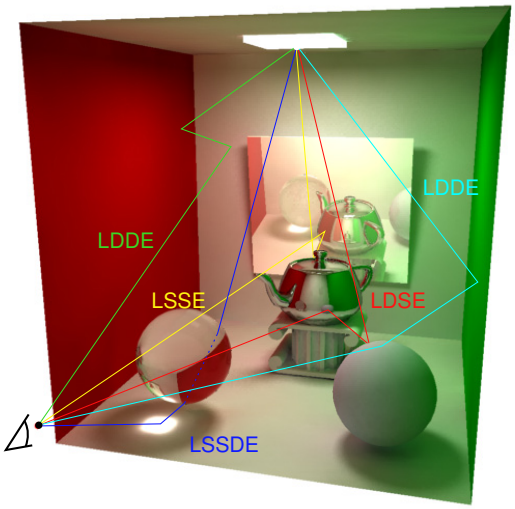


Figure 2.1.: *Example light paths in light transport notation.*

In a realistic scene, light (or more exactly, radiance) propagates from light sources through the scene, bouncing off diffuse and specular surfaces, possibly illuminating regions that are not directly visible to a light source. Any surface can be a light source and indeed, any reflecting surface can be treated as a light source. Thus, shading of a surface does not only depend on local properties, but also on all other surfaces in the scene.

2.1. The Light Transport Notation

Following an imaginary light ray or photon inside a scene, its path can be described by the endpoints of the straight parts of the path (the light transport notation, after Paul Heckbert [11]):

- L - light source
- S - a specular reflection
- D - a diffuse reflection
- E - the eye

A complete path from light source to eye is described by a concatenation of L,D,S and E (see Figure 2.1). Regular expressions can be used to describe sets of paths:

S*	-	zero or more specular reflections
D+	-	one or more diffuse reflections
D?	-	one or zero diffuse reflections
(S|D)	-	a diffuse or a specular reflection

L(S|D)*E describes the set of all light paths that are relevant for rendering realistic images. Rendering methods can be characterized by the subset of paths they can handle, e.g. L(S|D)(S|D)E would describe the paths that a global illumination algorithm with one light bounce can handle.

2.2. The Rendering Equation

In 1986, Kajiya [14] formulated an equation that completely describes light transport in a scene:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + L_s(p, \omega_o)$$

$L_o(p, \omega_o)$	is the radiance outgoing from point p in direction ω_o.
$L_e(p, \omega_o)$	is the emitted radiance from point p in direction ω_o.
$L_s(p, \omega_o)$	is the radiance scattered at point p in direction ω_o.

The scattering term $L_s(p, \omega_o)$ can be extended to give the full rendering equation[1]:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_\Omega \rho(p, \omega_i, \omega_o) L_i(p, \omega_i) \cos\theta \; d\omega_i \tag{2.1}$$

$\rho(p, \omega_i, \omega_o)$	is the BRDF (or BSDF when transmittance is included, see Section 2.3) which characterizes the surface reflection properties at point p.
$L_i(p, \omega_i)$	is the incoming radiance at point p from direction ω_i. It can also be formulated as the outgoing radiance at that point p' that is visible to p in direction ω_i, i.e. the first hit of a ray from p in direction ω_i: $L_i(p, \omega_i) = L_o(h(p, \omega_i), -\omega_i) = L_o(p', \omega')$
The angle θ	is the angle between the surface normal at point p and ω_i. Together with the differential angle $d\omega_i$, it describes the geometric relationship between incoming light and the surface at point p.
Ω	is the upper hemisphere of surface point p, or the entire sphere if transmittance is included.

The rendering equation geometry is illustrated in Figure 2.2. Ω is the set of all directions along which a surface point can receive or emit radiance. It is usually taken to be a hemisphere centered around the surface normal. The BRDF describes which parts of the

[1] This is an alternative form of Kajiya's original rendering equation. The original formulation does not incorporate angles or directions

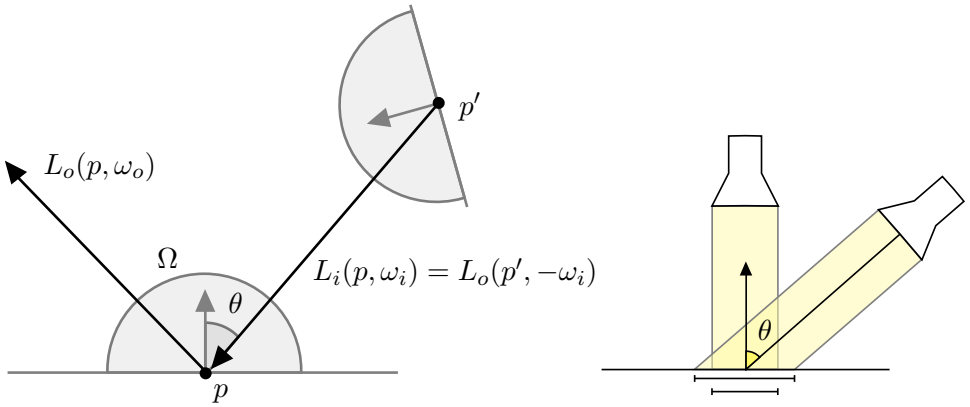


Figure 2.2.: *Left: the rendering equation geometry. Right: the cosine term in the rendering equation describes that light hitting a surface at shallow angeles has less intensity per surface area.*

incoming radiance are reflected in a given direction and depends on material properties of the surface (see section 2.3 for more information on BRDFs). Light hitting the surface at shallow angels (i.e. a large angle with the surface normal) has less intensity per area than light hitting the surface at steep angles (small angle with the surface normal), since the same amount of radiance is dispersed over a larger area (see Figure 2.2, right). The factor for the decrease in light intensity is $\cos\theta$, where θ is the angle between incoming light direction and surface normal and corresponds to the cosine term in the rendering equation.
An algorithm that can efficiently find solutions to the rendering equation would be a solution of the rendering problem. Unfortunately the equation is very difficult to solve analytically. Nevertheless, it can serve as a benchmark for existing algorithms, to determine which parts of the equation they only approximate or completely ignore.

One difficulty of solving the equation is that it is recursively formulated. The incoming light L_i at a point p from a direction ω_i is the outgoing radiance of some point p' that is visible to p in direction ω_i, i.e. p' is the first hit of a ray from p in direction ω_i.

$$L_i(p, \omega_i) = L_o(h(p, \omega_i), -\omega_i) = L_o(p', \omega') \tag{2.2}$$

In its short form, the rendering equation is written:

$$\begin{gathered} L_o = L_e + TL_o \\ (TL_o)(p, \omega_o) = \int_\Omega \rho(p, \omega_i, \omega_o) L_o(h(p, \omega_i), -\omega_i) \cos\theta \ d\omega_i \end{gathered} \tag{2.3}$$

T is a linear operator on L_o. In this form it is clearly visible, that the two sides of the equation are coupled. The radiance $L_o(p, \omega)$ coming from a point p depends on the radiance $L_o(p', \omega')$ coming from p' towards p. The radiance $L_o(p', \omega')$ may in turn depend on the radiance $L_o(p, -\omega')$ from p towards p' or on the radiance from any other point p''. Imagining two parallel mirrors helps visualizing the problem. The short form of the

rendering equation can be expanded accordingly:

$$\begin{aligned} L_o &= L_e + TL_o \\ &= L_e + T(L_e + T(L_e + \dots \end{aligned} \tag{2.4}$$

By the law of the conservation of energy, new energy cannot be created in a reflection T, but existing energy can be absorbed by a material. The deeper the recursion in equation 2.4, the less the influence of the terms on the resulting radiance. This allows us to approximate the rendering equation using only the first few terms of the recursion, i.e. the first few light bounces.

2.3. BRDFs

The *bidirectional reflectance distribution function* characterizes the reflection properties of a material. It describes how a surface point scatters the light coming from a given direction. More precisely, the BRDF is the fraction of energy incident on a surface point from one direction that is reflected in another direction [28]. Traditionally, the BRDF is a four-dimensional function: two dimensions for the incoming direction and two dimensions for the outgoing direction.

$$\rho(\omega_i, \omega_o) = \frac{L_o(\omega_o)}{L_i(\omega_i)\ \cos\theta\ d\omega_i} \tag{2.5}$$

Where $L_o(\omega_o)$ is the radiance reflected in direction ω_o, $L_i(\omega_i)$ the incident radiance coming from ω_i, θ the angle between ω_i and surface normal and $d\omega_i$ the differential solid angle of L_i and L_o. Including dependence on a two-dimensional parametrization of the surface position, the BRDF becomes a function of six variables, as in the rendering equation.

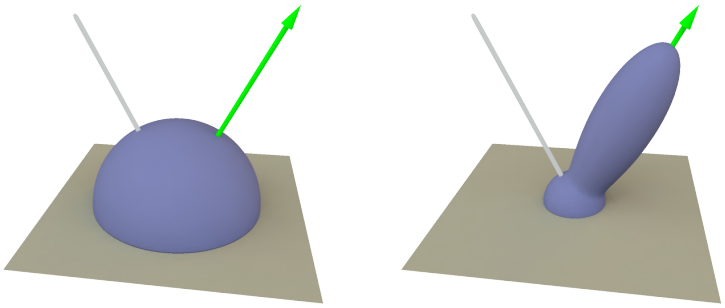


Figure 2.3.: *Polar plots of the reflectance of a perfectly diffuse BRDF (left) and a specular BRDF (right) for light incident along the white ray. The diffuse BRDF scatters light equally in all directions, the specular BRDF reflects stronger around the specular reflection angle (green arrow).*

A BRDF for a specular surface reflects directionally, with large values typically centered around the angle of reflection (see Figure 2.3). For a perfectly diffuse surface, the BRDF

reduces to a four-dimensional function. The light intensity is no longer dependant on the outgoing direction, since all incoming light is scattered equally in all directions. For simplicity, the cosine term in the rendering equation (equation 2.1) is often included in the BRDF, although it is not a property of the material itself.

BTDF is the bidirectional transmittance and is defined similar to the BRDF, but on the opposite hemisphere. Together with the BRDF it forms the *bidirectional scattering distribution function* BSDF, although it is rarely used.

BRDFs are either measured from real materials and stored in tables, or an analytical expression is used. Several analytical models have been developed that approximate the reflectance properties of real materials[29, 3, 6].

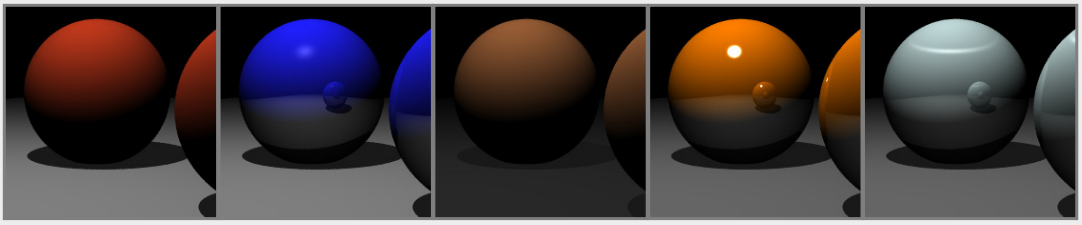


Figure 2.4.: *A few common analytical BRDFs applied to spheres (images from [2]).*

3. State of the Art

3.1. Spherical Harmonics

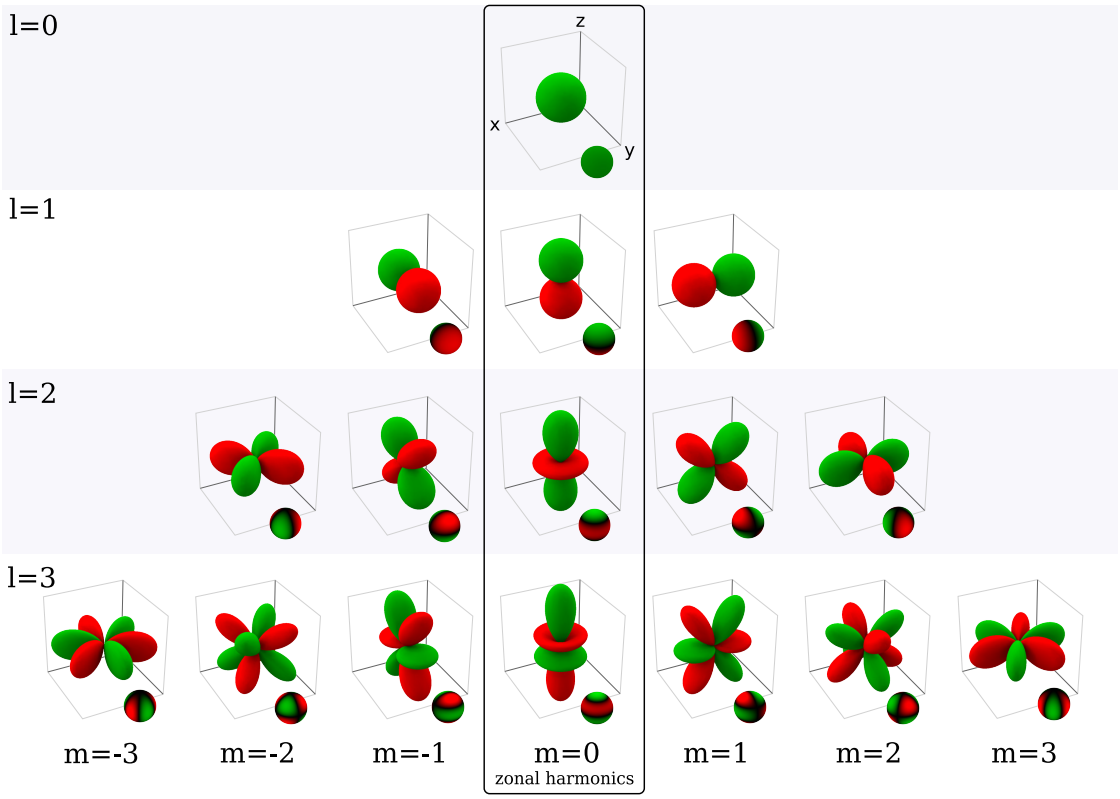


Figure 3.1.: *The spherical harmonics basis functions of the first four bands.*

Spherical harmonic lighting is a technique for lighting geometry from large area light sources. It was introduced by Sloan, Kautz and Snyder at Siggraph 2002 [36]. The key feature of spherical harmonic lighting is to represent functions over the hemisphere of a receiver point in the spherical harmonics basis (in similar techniques, other bases are used, like the wavelet basis [26, 27, 23, 42, 39]). Although this incurs approximation error, hemisphere functions can be represented as a single vector of coefficients. Operations on SH coefficient vectors are usually very efficient.

The spherical harmonics basis is a set of orthonormal functions defined over the surface

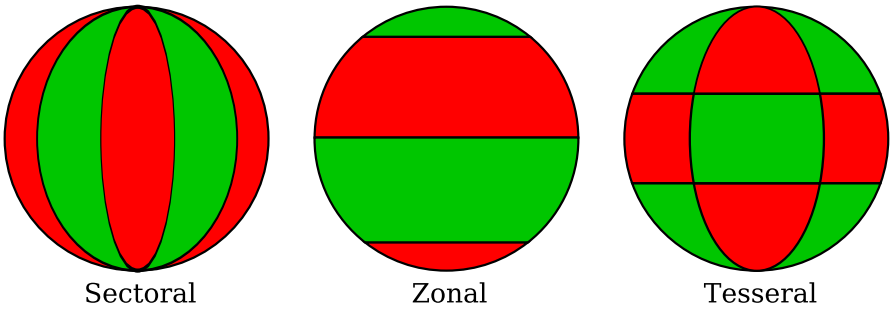


Figure 3.2.: *SH basis function types. Red areas are negative function values, green areas positive values.*

of a sphere. The set of functions is ordered into bands, which group the functions by frequency. Starting at band 0, which contains just one constant function, the frequency increases with the band number. The number of functions in each band also increases with the band number, each band has two more functions than the next lower band. Spherical harmonics basis functions are written

$$\mathbf{y}_l^m(\theta,\varphi) \;\; with \;\; l \in \mathbf{R}^+ \;\; and \;\; -l \leq m \leq l$$

where l is the band number, m the position inside the band and (θ, φ) polar coordinates on the sphere surface. For convenience, the functions are sometimes indexed in a specific order

$$\mathbf{y}_i(\theta,\varphi) = \mathbf{y}_l^m(\theta,\varphi) \;\; with \;\; i = l(l+1)+m$$

Figure 3.1 shows the spherical harmonics basis functions for the first four bands. Positive values are colored light green, negative values dark red.

SH basis functions are also distinguished by the way they divide the sphere surface into zones or segments of positive and negative values. Latitudinal divisions are called zones and the corresponding set of basis functions $\mathbf{y}_l^0$ *zonal harmonics* (see Figure 3.2). Divisions along the meridians are called sectors and the corresponding set of basis functions $\mathbf{y}_l^{-l}$ and $\mathbf{y}_l^l$ *sectoral harmonics*. The set of all remaining basis functions are called *tesseral harmonics*.

Mathematically, spherical harmonics are the angular portion of a set of solutions to Laplace's equation in spherical coordinates. They are calculated using the Associated Legendre Polynomials. For details on the equations and sample code see appendix A.

A spherical function $f(s)$ can be approximated with a linear combination of the spherical harmonics basis functions

$$f(s) = \sum_{i=0}^{\infty} c_i \mathbf{y}_i(s) \tag{3.1}$$

where s is some parametrization of the sphere surface. The coefficients c_i of the linear combination form the SH coefficient vector (or simply SH vector, from here on written in

boldface italic):

$$\boldsymbol{f} = (c_0, c_1, c_2, c_3, \dots)$$

The vector of corresponding SH basis functions will be denoted $\mathbf{y}(s)$. An order n SH vector has n^2 coefficients, corresponding to the basis functions of first n bands, all other coefficients are zero. A low-order SH vector can only capture the low-frequency behaviour $\widetilde{f}(s)$ of a spherical function $f(s)$.

$$\widetilde{f}(s) = \sum_{i=0}^{n^2-1} c_i \mathbf{y}_i(s) = \boldsymbol{f} \cdot \mathbf{y}(s) \tag{3.2}$$

Properties of SH Functions

The spherical harmonics basis has some properties that make it well suited for calculating global illumination type of lighting.
SH functions are rotationally invariant, meaning that the SH approximation of a rotated function $g(s) = f(R(s))$ is just like the SH approximation of the original function $f(s)$ with rotated input.

$$\widetilde{g}(s) = \widetilde{f}(R(s)) \tag{3.3}$$

Where $R(s)$ is an arbitrary rotation on the sphere. Rotations of SH functions do not deform the functions in any undesired way. This is an important property when using the SH basis for scene lighting, because it can guarantee that there are no objectionable aliasing artifacts, like light intensity fluctuations, when rotating objects.

The second property stems from the orthonormality of the SH basis. Given two SH functions $\widetilde{f}(s)$ and $\widetilde{g}(s)$, the integral of the two function's product reduces to the dot product of their coefficient vectors $\boldsymbol{f}$ and $\boldsymbol{g}$.

$$\int_S \widetilde{f}(s)\widetilde{g}(s)ds = \boldsymbol{f} \cdot \boldsymbol{g} \tag{3.4}$$

This is a very convenient property for SH lighting, because integrals like the exit radiance at a diffuse receiver point can be calculated in the SH basis using a single dot product.

$$\int_S L(s)t(s)ds \approx \int_S \widetilde{L}(s)\widetilde{t}(s)ds = \boldsymbol{L} \cdot \boldsymbol{t}$$

Where $L(s)$ is the incoming light and $t(s)$ the transfer function.

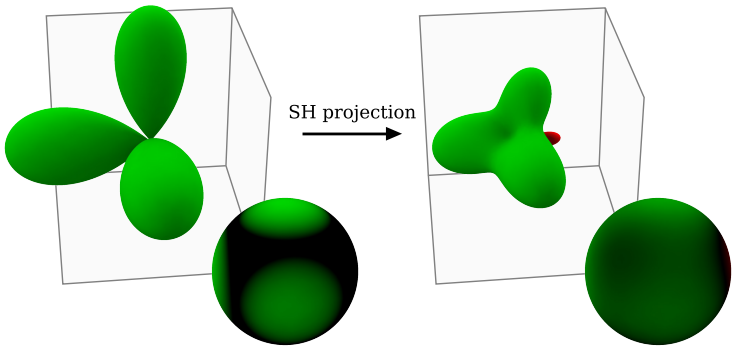


Figure 3.3.: *SH projection of a simple spherical function. The projected function only retains the low-frequency behaviour of the original function. The small negative valued fin on the back of the projected function is a ringing artifact caused by high frequencies at the joint of the three lobes.*

3.1.1. Projecting a Function

Given a spherical function $f(s)$, the SH vector $\boldsymbol{f}$ representing its low-frequency behaviour is obtained by *projecting* the function to the SH basis via

$$\boldsymbol{f} = \int_S f(s)\mathbf{y}(s)ds \tag{3.5}$$

where **y**(s) is a vector of SH basis functions of the same dimension as $\boldsymbol{f}$. The higher the order of the SH vector $\boldsymbol{f}$, the closer the *reconstructed* function $\widetilde{f}(s)$ (see equation 3.2) corresponds to the original function $f(s)$.

$$\widetilde{f}(s) = \boldsymbol{f} \cdot \mathbf{y}(s) \approx f(s) \tag{3.6}$$

Figure 3.3 shows a simple spherical function consisting of three lobes projected to the SH basis using the first four SH bands.

The integral in equation 3.5 can be solved using Monte Carlo integration [21]. In Monte Carlo integration, a large number of randomly distributed samples of the function to be integrated is collected. The choice of probability distribution of the samples is important, as it affects the accuracy and number of samples needed for integration. The samples are then scaled by the probability density $p(s)$ at each sample location, summed up and divided by the number of samples:

$$\int f(s)ds \approx \frac{1}{N}\sum_{j=0}^{N} f(s_j)w(s_j) \tag{3.7}$$

Where N is the number of samples, $w(s) = \frac{1}{p(s)}$ and s_j the location of sample j. If the samples are uniformly distributed, then the weight function $w(s)$ is a constant w and

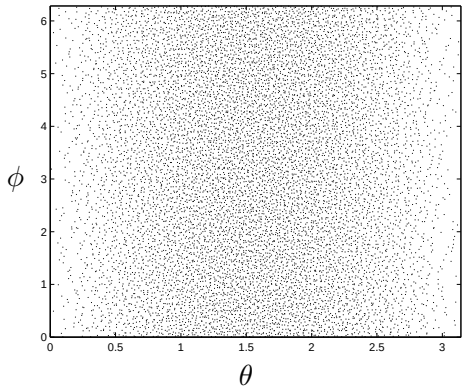


Figure 3.4.: *10.000 samples generated using jittered stratification.*

equation 3.7 becomes

$$\int f(s)ds \approx \frac{w}{N}\sum_{j=0}^{N} f(s_j) \tag{3.8}$$

Since $f(s)$ is a spherical function, a uniform distribution over the surface of a sphere is needed. Using two uniformly distributed random variables over the domain of the polar coordinates (θ, ϕ) results in a sample distribution that is too dense at the poles of the sphere and too sparse in between. To correct this, the latitudinal polar coordinate θ has to be adjusted. The following transformation from uniformly distributed random variables ξ_1 and ξ_2 to polar coordinates (θ, ϕ) results in a uniform sample distribution over the sphere:

$$\theta = 2\arccos(\sqrt{1-\xi_1}) \text{ and } \varphi = 2\pi\xi_2 \tag{3.9}$$

This sample distribution suffers from large variance, deteriorating the accuracy of the integration if samples are not approximately equally spaced.

Jittered Stratification [21] (see Figure 3.4) reduces the variance and results in a more reliable equal spacing of samples. The range of the random variables is divided into cells or *strata* and one sample is picked in each strata. The sum of variances of each strata is at most as high as the variance of random sampling over the whole sphere and usually much smaller.

With a uniform sample probability of $p(s) = \frac{1}{4\pi}$ over the whole sphere, the Monte Carlo Integration becomes:

$$\int f(s)ds \approx \frac{4\pi}{N}\sum_{j=0}^{N} f(s_j) \tag{3.10}$$

Using this form of Monte Carlo integration to solve the integral in equation 3.5, the equation for projecting a polar function $f(s)$ to the SH basis becomes

$$\boldsymbol{f} \approx \frac{4\pi}{N}\sum_{j=0}^{N} f(s_j)\mathbf{y}(s_j) \tag{3.11}$$

3.1.2. SH Rotations

The rotation invariance property of the SH basis (see Section 3.1) states that given an SH vector $\boldsymbol{f}$ of order n with reconstructed function $\widetilde{f}(s)$, we can find an SH vector $\boldsymbol{f}'$ which has a reconstructed function $\widetilde{f}'(s)$ that is a perfect rotation of $\widetilde{f}(s)$. In other words, $\widetilde{f}'(s) = \widetilde{f}(Rs)$.

In a naive approach, $\widetilde{f}(s)$ is reconstructed from $\boldsymbol{f}$ and projected back to the SH basis with rotated input.

$$\boldsymbol{f}'_i = \int_S \widetilde{f}(Rs)\mathbf{y}_i(s)ds = \int_S \sum_{j=0}^{n} \boldsymbol{f}_j \mathbf{y}_j(Rs)\mathbf{y}_i(s)ds \tag{3.12}$$

This is a relatively inefficient approach, as it requires SH reconstruction and projection.

Since the SH basis is orthonormal, an SH rotation is a linear transformation on the coefficients of an SH vector. It is possible to find a matrix M_R that directly transforms the coefficients of $\boldsymbol{f}$ to $\boldsymbol{f}'$ [8]. The orthonormality of the SH basis implies that M_R is a block diagonal sparse matrix and that coefficients of different bands do not interact [10]. Figure 3.13 shows the composition of an SH rotation matrix.

$$\mathsf{M}_R = \left(\begin{array}{c|ccc|ccccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ \hline 0 & X & X & X & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & X & X & X & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & X & X & X & 0 & 0 & 0 & 0 & 0 & \dots \\ \hline 0 & 0 & 0 & 0 & X & X & X & X & X & \dots \\ 0 & 0 & 0 & 0 & X & X & X & X & X & \dots \\ 0 & 0 & 0 & 0 & X & X & X & X & X & \dots \\ 0 & 0 & 0 & 0 & X & X & X & X & X & \dots \\ 0 & 0 & 0 & 0 & X & X & X & X & X & \dots \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}\right) \tag{3.13}$$

Rotating an SH vector using M_R requires only one matrix multiplication with a sparse matrix and is more efficient than the naive approach.

$$\boldsymbol{f}' = \mathsf{M}_R \boldsymbol{f} \tag{3.14}$$

The Problem, however, is finding an efficient method to construct the matrix M_R given a rotation R. M_R can be constructed by projecting the rotated SH basis functions to the SH basis:

$$(\mathsf{M}_R)_{ij} = \int_S \mathbf{y}_i(Rs)\mathbf{y}_j(s)ds \tag{3.15}$$

For low-order SH vectors, a useful method for finding the $(\mathsf{M}_R)_{ij}$ as a function of R's components is to do symbolic integration on the above equation. For higher-order SH vectors however, this method becomes increasingly inefficient.

Several methods have been proposed to speed up SH rotations. The method by Kautz et al.[17] is more efficient for higher-order SH vectors. The rotation R is decomposed into

its zyz rotation components (α, β, γ). The rotation around y is further decomposed into a rotation around x by $\frac{\pi}{2}$, followed by a rotation around z by β and a rotation back around x by $-\frac{\pi}{2}$.

$$\mathsf{M}_R = \mathsf{M}_{Rz}\mathsf{M}_{Rx}\mathsf{M}_{Rz}\mathsf{M}_{Rx}\mathsf{M}_{Rz} \tag{3.16}$$

The rotations around x have a fixed angle and the matrix components $(\mathsf{M}_R)_{ij}$ for these rotations can be precomputed. The matrix components for z-axis rotations can be found using a simple formula [10, 17]. Křivánek et al. [19] describe an algorithm for rotating SH vectors around the z-axis without constructing a matrix.

Ivanic and Ruedenberg [12, 13] use recursive functions to build rotation matrices of order $n+1$ from rotation matrices of order n.

Křivánek et al. [19] propose a fast, approximative method for SH rotations. The y-rotation in the zyz rotation decomposition is approximated by its truncated Taylor expansion. The computational complexity is reduced from $O(n^3)$ for Ivanic and Ruedenberg's method and the $zxzxz$ method of Kautz et al. to $O(n^2)$. The downside is that the method only accurately approximates SH rotations about the y-axis for small rotation angles.

Sloan et al. [37] propose working with only a subset of the SH basis functions that can be rotated more efficiently: the zonal harmonics $\mathbf{y}_l^0$ (see Figure 3.1). An order-n projection $\boldsymbol{g}$ of a function to zonal harmonics has n components, since there is one zonal harmonic basis function in each SH band. Zonal harmonic basis functions are all circularly symmetric around the z-axis, so the range of functions they can approximate is limited to the functions that are, too, circularly symmetric around the z-axis. Sloan et al. approximate general functions using combinations of rotated basis functions. ZH projections have the advantage, that they can be rotated from the z-axis to an arbitrary axis s^* by a simple formula:

$$(\boldsymbol{g}')_l^m = \mathbf{y}_l^m(s^*)\sqrt{\frac{4\pi}{2l+1}}\boldsymbol{g}_l \tag{3.17}$$

where $\boldsymbol{g}'$ is the SH rotation of $\boldsymbol{g}$, in other words $\widetilde{g}'(s) = \widetilde{g}(R^{-1}s)$. Note that $\boldsymbol{g}'$ has to be an SH vector, since the rotated function $\widetilde{g}'(s)$ is not circularly symmetric around the z-axis anymore.

3.1.3. SH Products and Squares

Products of functions that are represented in the SH basis are useful when computing the visibility effects of multiple overlapping blockers. Calculating the visibility directly in the SH basis avoids costly SH projections.

Given two order-n coefficient vectors $\boldsymbol{f}$ and $\boldsymbol{g}$, the product of their SH functions can be approximated by projecting the product of the reconstructed functions $\widetilde{f}(s)$ and $\widetilde{g}(s)$ back to an order-n coefficient vector $\boldsymbol{f} * \boldsymbol{g}$, called the *SH Product* [32]:

$$\boldsymbol{f} * \boldsymbol{g} = \int_s \widetilde{f}(s)\widetilde{g}(s)\mathbf{y}(s)ds \tag{3.18}$$

The product $\widetilde{f}(s)\widetilde{g}(s)$ of the reconstructed functions may require a coefficient vector of order up to $2n-1$ to be represented accurately in the SH basis. The order-n projection $\boldsymbol{f} * \boldsymbol{g}$ is a low-frequency approximation.

Substituting the right part of equation 3.2 for $\widetilde{f}(s)$ and $\widetilde{g}(s)$ and rearranging gives the following equation:

$$(\boldsymbol{f} * \boldsymbol{g})_i = \sum_{jk} \boldsymbol{f}_j \boldsymbol{g}_k \int_s \mathbf{y}_i(s)\mathbf{y}_j(s)\mathbf{y}_k(s)ds = \sum_{jk} \boldsymbol{f}_j \boldsymbol{g}_k \Gamma_{ijk} \tag{3.19}$$

Γ_{ijk} are the tripling coefficients [27]. They form a sparse, symmetric order-3 tensor Γ, called the *SH Triple Product Tensor* [32], defined as:

$$\Gamma_{ijk} = \int_s \mathbf{y}_i(s)\mathbf{y}_j(s)\mathbf{y}_k(s)ds \tag{3.20}$$

The Γ_{ijk} correspond to the well studied Clebsch-Gordan coefficients [8], but can also be computed by numerically solving the Integral using Monte Carlo Integration (see Section 3.1.1).

In a naive approach using each Γ_{ijk}, the SH product of two order-n SH vectors is $O(n^6)$, since Γ has n^6 components. Taking advantage of the sparsity of Γ, the computational complexity can be reduced to $O(n^5)$ [27]. For real-time applications, the Γ_{ijk} are usually precomputed to avoid having to evaluate the integral at run-time.

The SH product $\boldsymbol{f} * \boldsymbol{g}$ of the SH vector $\boldsymbol{f}$ with an arbitrary vector $\boldsymbol{g}$ is a linear transformation of the components of $\boldsymbol{g}$. Hence, the *SH product matrix* (or *transfer matrix* when used for shadowing) $\mathsf{M}_{\boldsymbol{f}}$ can be defined, which describes the linear transform of the components of an arbitrary SH vector $\boldsymbol{g}$ to match $\boldsymbol{f} * \boldsymbol{g}$ [36, 32].

$$\boldsymbol{f} * \boldsymbol{g} = \mathsf{M}_{\boldsymbol{f}} \boldsymbol{g} \text{ for any } \boldsymbol{g} \tag{3.21}$$

$\mathsf{M}_{\boldsymbol{f}}$ is defined by

$$(\mathsf{M}_{\boldsymbol{f}})_{ij} = \int_S \widetilde{f}(s)\mathbf{y}_i(s)\mathbf{y}_j(s) = \sum_k \boldsymbol{f}_k \Gamma_{ijk} \tag{3.22}$$

Using the product matrix, the computational complexity of an SH product is reduced to $O(n^4)$, but the product matrix $\mathsf{M}_{\boldsymbol{f}}$ of one of the SH vectors needs to be known in advance.

3.1.4. SH Exponential and Logarithm

SH exponentials are useful when accumulating blocker visibility. Zhou et al. [44] finds the total blocked visibility $\boldsymbol{g}$ at a receiver point by computing the product of blocker visibility functions $\boldsymbol{g}[i]$ of the individual blockers in the SH basis.

$$\boldsymbol{g} = \boldsymbol{g}[1] * \boldsymbol{g}[2] * \cdots * \boldsymbol{g}[m] \tag{3.23}$$

where m is the number of blockers and $\boldsymbol{g}[i]$ the SH projection of the blocker visibility function $g[i](s)$ of blocker i, defined over incident illumination directions of a receiver point.

$g[i](s)$ takes on values of 0 for directions where blocker i is blocking and 1 everywhere else.

Instead of using expensive SH products, Ren et al. [32] accumulate the log of blocker visibilities using inexpensive sums.

$$\boldsymbol{g} = \exp(\log(\boldsymbol{g}[1]) + \log(\boldsymbol{g}[2]) + \cdots + \log(\boldsymbol{g}[m])) \tag{3.24}$$

The logarithm is usually precomputed offline and a fast approximation is used for the exponential. The remainder of this section will describe the *SH exponential* and *SH logarithm* methods proposed by Ren et al. [32].

The SH Exponential is typically computed at run-time, so it has to be a fast operation. Using the Volterra Series [34] and Taylor expansion, Ren at al. show that

$$\begin{aligned} \exp(\boldsymbol{f}) &= \mathbf{1} + \boldsymbol{f} + \frac{\boldsymbol{f}^2}{2} + \frac{\boldsymbol{f}^3}{3!} + \dots \\ \approx \exp_*(\boldsymbol{f}) &= \mathbf{1} + \boldsymbol{f} + \frac{\boldsymbol{f} * \boldsymbol{f}}{2} + \frac{\boldsymbol{f} * \boldsymbol{f} * \boldsymbol{f}}{3!} + \dots \end{aligned} \tag{3.25}$$

where $\mathbf{1}$ is the unit SH vector $(\sqrt{4\pi}, 0, 0, \dots, 0)$. The error of the approximation

$$\boldsymbol{f}^p \approx \underbrace{\boldsymbol{f} * \boldsymbol{f} * \cdots * \boldsymbol{f}}_{\text{p times}}$$

is acceptable if the reconstructed function $\widetilde{f}(s)$ of $\boldsymbol{f}$ is bound in a small interval, e.g. $[0, 1]$, as in the blocker functions.

When using a finite number of terms of the Taylor expansion, error increases as the magnitude $\|\boldsymbol{f}\|$ increases. To keep the magnitude low, $\boldsymbol{f}$'s DC component $\boldsymbol{f}_0$, the coefficient of the constant SH basis function $\mathbf{y}_0^0$, is isolated and handled separately.

$$\begin{aligned} \boldsymbol{f} &= \left(\frac{\boldsymbol{f}_0}{\sqrt{4\pi}}\right)\mathbf{1} + \widehat{\boldsymbol{f}} \\ \widehat{\boldsymbol{f}} &= (0, \boldsymbol{f}_1, \boldsymbol{f}_2, \dots, \boldsymbol{f}_{n^2-1}) \end{aligned} \tag{3.26}$$

The SH exponential is then

$$\exp_*(\boldsymbol{f}) = \exp\left(\frac{\boldsymbol{f}_0}{\sqrt{4\pi}}\right)\exp_*(\widehat{\boldsymbol{f}}) \tag{3.27}$$

When using DC isolation on SH vectors of order 4 or lower, the first two terms of the Taylor expansion in equation 3.25 provide sufficient accuracy. Replacing the SH exponential in equation 3.27 with the two-term Taylor expansion, we get:

$$\exp_*(\boldsymbol{f}) = \exp\left(\frac{\boldsymbol{f}_0}{\sqrt{4\pi}}\right)\left(a(\widehat{\boldsymbol{f}})\mathbf{1} + b(\widehat{\boldsymbol{f}})\widehat{\boldsymbol{f}}\right) \tag{3.28}$$

The coefficients $a(\widehat{\boldsymbol{f}})$ and $b(\widehat{\boldsymbol{f}})$ are chosen to provide the least-squares best projection of $\exp_*(\widehat{\boldsymbol{f}})$ onto the vectors $\mathbf{1}$ and $\widehat{\boldsymbol{f}}$.

$$a(\widehat{\boldsymbol{f}}) = \frac{\exp_*(\widehat{\boldsymbol{f}}) * \mathbf{1}}{\mathbf{1} * \mathbf{1}} \qquad b(\widehat{\boldsymbol{f}}) = \frac{\exp_*(\widehat{\boldsymbol{f}}) * \widehat{\boldsymbol{f}}}{\widehat{\boldsymbol{f}} * \widehat{\boldsymbol{f}}} \tag{3.29}$$

The value of $a(\widehat{\boldsymbol{f}})$ and $b(\widehat{\boldsymbol{f}})$ usually depends on the SH vector $\widehat{\boldsymbol{f}}$, that is, on the blocker model. But experiments by Ren et al. have shown, that $a(\widehat{\boldsymbol{f}})$ and $b(\widehat{\boldsymbol{f}})$ agree on their values for SH vectors $\widehat{\boldsymbol{f}}$ with the same magnitude when the magnitude is small, roughly $\|\widehat{\boldsymbol{f}}\| < 4.8$. This makes it possible to precompute the a, b values and tabulate them by SH vector magnitude $\|\widehat{\boldsymbol{f}}\|$.

$$\exp_*(\boldsymbol{f}) = \exp\left(\frac{\boldsymbol{f}_0}{\sqrt{4\pi}}\right)\left(a(\|\widehat{\boldsymbol{f}}\|)\mathbf{1} + b(\|\widehat{\boldsymbol{f}}\|)\widehat{\boldsymbol{f}}\right) \tag{3.30}$$

This approximation of the SH exponential is $O(n^2)$, with n the number of bands of $\boldsymbol{f}$. Accurate results are only guaranteed for $\|\widehat{\boldsymbol{f}}\| < 4.8$.
Using the relation

$$\exp(x) = \left(\exp\left(\frac{x}{2^p}\right)\right)^{2^p} \tag{3.31}$$

SH vectors of larger magnitude $\|\widehat{\boldsymbol{f}}\|$ can be used with this method. First, the input SH vector $\boldsymbol{f}$ is scaled by 2^{-p} before using it in equation 3.30. The result is repeatedly squared p times to undo the scaling.

$$\exp_*(\boldsymbol{f}) = \left(\exp_*\left(\frac{\boldsymbol{f}}{2^p}\right)\right)^{2^p *} \tag{3.32}$$

where $\boldsymbol{f}^{2^p *}$ denotes p repeated squarings of $\boldsymbol{f}$ using SH products. SH products are $O(n^5)$, so the SH exponential becomes $O(n^5)$, when using the scaling/squaring method.

The SH Logarithm should closely match the inverse of the approximation used for the SH exponential. Starting from the Taylor expansion in equation 3.25

$$\begin{aligned} \boldsymbol{g} = \exp(\boldsymbol{f}) &= \mathbf{1} + \boldsymbol{f} + \frac{\boldsymbol{f} * \boldsymbol{f}}{2} + \frac{\boldsymbol{f} * \boldsymbol{f} * \boldsymbol{f}}{3!} + \dots \\ &= \mathbf{1} + \boldsymbol{f} + \frac{\mathsf{M}_{\boldsymbol{f}}\,\boldsymbol{f}}{2} + \frac{\mathsf{M}^2_{\boldsymbol{f}}\,\boldsymbol{f}}{3!} + \dots \end{aligned} \tag{3.33}$$

where $\mathsf{M}_{\boldsymbol{f}}$ is the SH product matrix of $\boldsymbol{f}$ (see Section 3.1.3), Ren et al. show that

$$\begin{aligned} \log(\boldsymbol{g}) &= \mathsf{R}^T_{\boldsymbol{g}}\, q'(\mathsf{D}_{\boldsymbol{g}})\, \mathsf{R}_{\boldsymbol{g}}(\boldsymbol{g} - \mathbf{1}) \\ q'(x) &= \frac{\log(x)}{x - 1} \end{aligned} \tag{3.34}$$

where $\mathsf{R}_{\boldsymbol{g}}^T \mathsf{D}_{\boldsymbol{g}} \mathsf{R}_{\boldsymbol{g}}$ is the result of an eigenanalysis of the product matrix $\mathsf{M}_{\boldsymbol{g}}$ of $\boldsymbol{g}$. $\mathsf{R}_{\boldsymbol{g}}$ is a rotation matrix that projects an input vector to a basis of eigenvectors and $\mathsf{D}_{\boldsymbol{g}}$ is the diagonal matrix of associated eigenvalues. $q'(x)$ is applied to each component of the diagonal of $\mathsf{D}_{\boldsymbol{g}}$. The eigenvalues on the diagonal of $\mathsf{D}_{\boldsymbol{g}}$ are clipped to a small ε to avoid applying the log to negative values or to values close to zero.

Because of the eigenanalysis involved, finding the SH logarithm is an expensive operation typically not done at run-time.

3.2. Precomputed Radiance Transfer

Figure 3.5.: *Model with environment lighting only (left) and with self-shadows and interreflections using precomputed radiance transfer (right). (Images from [36].)*

The paper by Sloan et al. [36] was the first in a row of papers dealing with precomputed radiance transfer. It introduced spherical harmonics as a novel method to store precomputed values for a receiver point, thereby reducing the storage cost.

Under the assumption of an infinitely distant light source (e.g. an environment map), the incident light at a receiver point p can be described as the product of the radiance coming from the light source and a local transfer function T_p:

$$L_i(p, \omega_o) = L_{env}(\omega_i) T_p(\omega_i) \tag{3.35}$$

where L_{env} is the infinitely distant light source. T_p describes which parts of the radiance from the light source L_{env} arrive at the receiver point p. When only calculating the shadowed response of a receiver point, T_p is a visibility function $V_p(\omega_i)$ which takes on values of 1 everywhere the light source is visible and 0 everywhere else (see Figure 3.6). When including reflections, T_p also describes which parts of L_{env} arrive at the receiver point by way of reflections.

$$T_p(\omega_i) = V_p(\omega_i) + R_p(\omega_i) \tag{3.36}$$

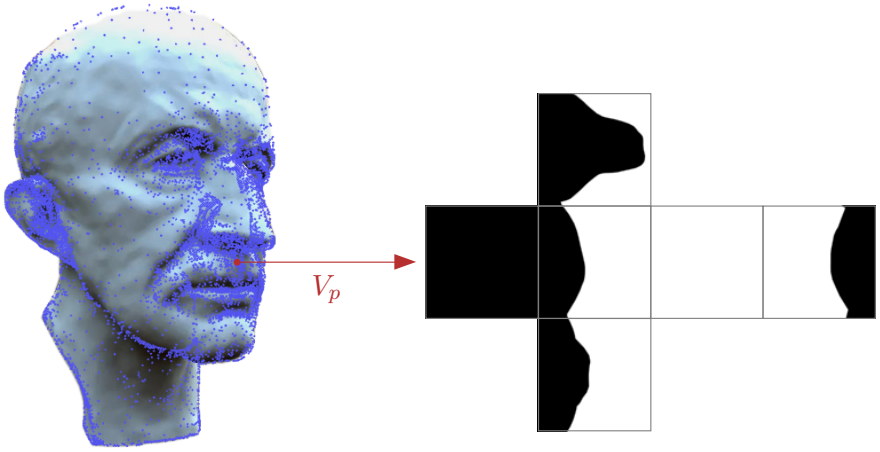


Figure 3.6.: *The visibility function V_p of the red receiver point displayed as a cubemap. The blue dots are other receiver points on the model.*

where $R_p(\omega_i)$ is the fraction of the radiance of L_{env} arriving at p through reflections. In both cases, T_p is a linear transform on L_{env}.

Using this in the rendering equation (see equation 2.1), we get its reduced form for precomputed radiance transfer:

$$L_o(p, \omega_o) = \int_\Omega \rho(p, \omega_i, \omega_o) L_{env}(\omega_i) T_p(\omega_i) \tag{3.37}$$

The cosine term has been included in the BRDF ρ. Integration is done over the entire sphere, but the lower hemisphere us usually zeroed out by the BRDF. Note that there is no emittance term, since we assume that light is only emitted from the infinitely distant light source L_{env}.

In a scene with static objects, the transfer functions can be precomputed at densely sampled receiver points on an object's surface to describe the light transport on that object. However, storing these functions is expensive and has been a problem in prior methods. Here, spherical harmonics provide a compact way to represent the transfer functions. Usually no more than 5 bands, i.e. 25 coefficients, are needed per transfer function.

Diffuse BRDFs

Diffuse BRDFs can be included in the precomputed transfer function, since they remain constant at run time. If the distant lighting is represented in the spherical harmonics basis, too, calculation of the outgoing light at a receiver point (see equation 3.37) simplifies

to:

$$\begin{aligned} L_o(p) &= \boldsymbol{l} \cdot \boldsymbol{f}_p \\ \boldsymbol{l} &= \int_\Omega L_{env}(\omega_i)\ \mathbf{y}(\omega_i)\ d\omega_i \\ \boldsymbol{f}_p &= \int_\Omega \rho_p(\omega_i) T_p(\omega_i)\ \mathbf{y}(\omega_i)\ d\omega_i \end{aligned} \tag{3.38}$$

A simple dot product between the SH vector $\boldsymbol{l}$, approximating the distant lighting and the SH vector $\boldsymbol{f}_p$, approximating the precomputed product of transfer function and diffuse BRDF. Note that $\boldsymbol{l}$ and $\boldsymbol{t}$ can only capture the low-frequency behaviour of the light source and the transfer function.

Glossy BRDFs

Glossy BRDFs are more difficult to compute. Since glossy BRDFs are dependent on view direction, they cannot be precomputed and stored in the transfer vector like diffuse BRDFs. In their original method [36], Sloan et al. can only handle a special kind of isotropic glossy BRDF. Kautz et al. [17] and later Sloan et al. [35] improve the method to handle arbitrary glossy BRDFs. The glossy BRDF is stored as a table of SH vectors over all view directions. Each coefficient of the stored SH vectors can be described as a function over all view directions, which can itself be projected to the SH basis, resulting in a $n^2 \times n^2$ matrix for a glossy BRDF, where n is the number of SH bands.

$$\mathsf{B}_{ij} = \int_{\omega_o} \int_{\omega_i} \mathbf{y}_i(\omega_o) \mathbf{y}_j(\omega_i) \rho(\omega_i, \omega_o) d\omega_i d\omega_o \tag{3.39}$$

where B is the BRDF matrix for the glossy BRDF B. The outgoing light at a glossy receiver point can then be calculated with

$$L_o(p, \omega_o) = \mathbf{y}(\omega_o)(\mathsf{B}\mathsf{R}_p\mathsf{T}_p)\boldsymbol{l} \tag{3.40}$$

where T_p is the SH product matrix (see Section 3.1.3) of the transfer vector. T_p transforms the distant radiance $\boldsymbol{l}$ to radiance incident at the point p, including shadows and interreflections. R_p is an SH rotation (see Section 3.1.2) rotating the incident radiance to the BRDF's local coordinate frame. Only moderately glossy BRDFs can be handled in this way. A prohibitive amount of SH coefficients would be necessary to get useful approximations of highly specular BRDFs.

Local light sources

When only using their method for shadowing, Sloan et al. [36] show that it is possible to handle local, dynamic light sources. The incident radiance field, i.e. the radiance coming

directly from light sources is calculated at key points[1] on an object's surface. In between, the radiance field is interpolated. The assumption of this method is, that lighting variation is not too high over the surface of an object. Since reflections are ignored, the radiance coming from a receiver point depends only on the interpolated incident radiance field at the position of the receiver point and on the transfer vector. Radiance incident at a receiver due to reflections would be dependent on the incident radiance field of the point that reflected the light and could therefore not be calculated locally at the receiver point. Note that this is only an issue with local lights. When using environment lighting only, the incident radiance field is constant everywhere in the scene.

Volumetric Data

In their paper, Sloan et al. [36] also show how transfer vectors can be precomputed for volumetric data, like clouds, and how shadows and reflections from dynamically moving objects can be transferred to other objects by precomputing a field of transfer matrices around the moving object.

With the method of Sloan et al. [36], extended by [17, 35], real-time performance can be achieved for diffuse and moderately glossy materials in static scenes with dynamic view and lighting changes.

3.3. Ambient Occlusion

Figure 3.7.: *Comparision of images generated with and without ambient occlusion. Left: image lit by environment lighting without using ambient occlusion. Right: image generated using ambient occlusion. (Images from [20].)*

Ambient occlusion is a term used to describe the inaccessibility of a receiver point, i.e. how much of its hemisphere is occluded. The method was originally used to darken the

[1]The sample points for the incident radiance field are found by using the iterated closest point algorithm[22]

classical ambient term [29] in areas like creases or corners that are not fully exposed to the environment, giving objects a more convincing appearance.

Although Landis [20] coined the term ambient occlusion as a cheap alternative to global illumination for offline rendering, a similar technique was used before by Zhukov et al. [45] to calculate the obscurance of a point, which is a generalization of ambient occlusion:

$$W(p) = \frac{1}{\pi} \int_{\Omega} f(h(p,\omega)) \cos\theta \; d\omega \tag{3.41}$$

where f is a function of this distance $h(p,\omega)$ of a point p to the first hit of a ray cast from p in direction ω. θ is the angle between ω and the surface normal and $\int_{\Omega}$ integrates over the upper hemisphere of p. Intuitively, the closer objects are to the receiver point, the more it is obscured by them.

Ambient occlusion simplifies the obscurance technique by replacing the function f with a visibility function V, which takes on values of zero in directions where no geometry is visible and one everywhere else:

$$A(p) = \frac{1}{\pi} \int_{\Omega} V(p,\omega) \cos\theta \; d\omega \tag{3.42}$$

When using local illumination rendering (like in most current games and other real-time applications), the ambient occlusion term A can be used to darken inaccessible regions of an object, giving it a more global-illumination like appearance. (see Figure 3.3).

The ambient occlusion term is usually computed using raytracing to determine the occluded directions at each receiver point. This limits the use of ambient occlusion in real-time rendering to static objects, where the ambient occlusion term can be precomputed.

The bent normal

Often it is also useful to determine the "bent normal" at each receiver point [20]. The bent normal represents the average unoccluded direction, i.e. the average direction of incoming light at a receiver point. It can be found by normalizing the average of all unoccluded rays during raytracing. The bent normal is useful when calculating the ambient occlusion with environment lighting. An average of the environment light in the direction of the bent normal approximates the incident environment light at a receiver point. Although this is not always correct (the bent normal may point to a direction that is actually occluded), the results are usually visually pleasing.

Ambient Occlusion for dynamic scenes with indirect lighting

Recently, ambient occlusion has been extended in several methods. Bunnell [4] calculates the ambient occlusion terms dynamically at run-time by approximating the surfaces of scene geometry with small discs. He also describes how to incorporate diffuse indirect lighting in additional rendering passes.

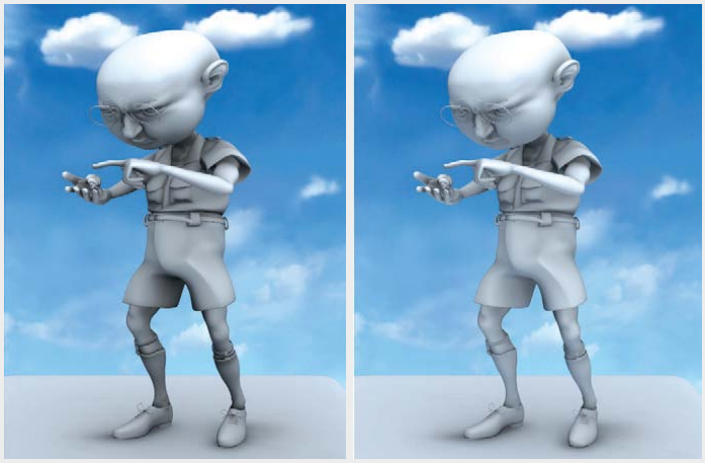


Figure 3.8.: *Left: image lit by environment lighting using ambient occlusion. Right: The same image with indirect lighting. (Images from [4].)*

Each object in the scene is covered with discs, one disc for each vertex. The size of the disc is determined by the size of adjacent faces. The ambient occlusion caused by one disc at a receiver point can be calculated analytically, derived from the solid angle subtended by the disc on the receiver point's hemisphere. Figure 3.9 shows the geometric relation between disc and receiver point. The ambient occlusion term of a single disc d is given by:

$$A(p,d) = 1 - \frac{r_{pd}\cos\theta_d \max(1, 4\cos\theta_p)}{\sqrt{\frac{Ar_d}{\pi} + {r_{pd}}^2}} \tag{3.43}$$

where r_{pd} is the distance between the receiver point and the disc center and Ar_d the area of the disc. θ_d is the angle between disc normal and the direction towards the receiver point, θ_p the angle between the surface normal at the receiver point and the direction towards the disc.

In a first pass, the ambient occlusion terms of all discs in the upper hemisphere are summed up at each receiver point. This overestimates the real ambient occlusion values since overlapping discs are not handled correctly, resulting in shadows that are too dark. In a second pass, the same shadowing procedure is repeated, but this time, the ambient occlusion caused by a disc is multiplied by the occlusion of the disc itself, i.e. the ambient occlusion value from the first pass. The result is an underestimation of the real ambient occlusion values. The true ambient occlusion values are approximated by taking the average of first and second pass.

Indirect lighting is handled in separate passes. One pass is needed for one light bounce. In each pass, the diffuse radiance is calculated for each disc, possibly using information from previous passes. The form factor used for indirect illumination is different from the

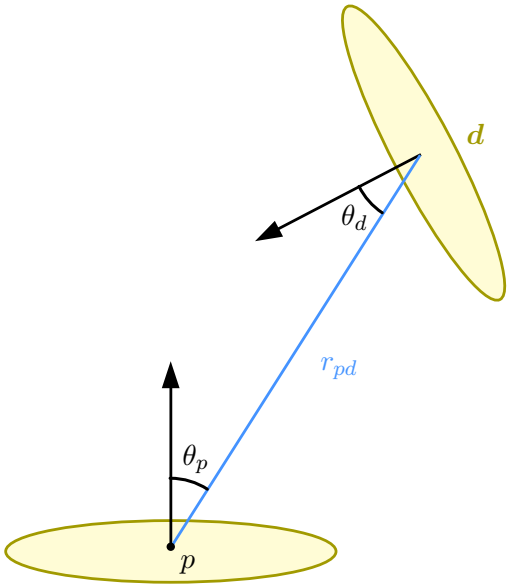


Figure 3.9.: *The geometric relation between receiver disc and emitter disc.*

one used for shadowing (equation 3.43):

$$T(p,d) = \frac{Ar_d \cos\theta_d \cos\theta_p}{\pi {r_{pd}}^2 + Ar_d} \tag{3.44}$$

Shadowing for indirect illumination is done the same way as when calculating the ambient occlusion term, but the light transfer form factor T is used.

Ambient Occlusion fields

Kontkanen and Laine [18] and Malmer et al. [25] both describe methods for computing shadows from dynamic objects using ambient occlusion fields.

In Kontkanen and Laine's method, an approximate ambient occlusion value is calculated at each receiver point. The visibility function $V(p,\omega)$ of an object as viewed from a receiver point p is approximated by the visibility of a spherical cap $V_{cap}(p,w)$ in the mean direction of the object (see Figure 3.10). The approximation is accurate for one object, as long as the solid angle subtended by the cap is approximately equal to the solid angle subtended by the object.

$$A(p) = \frac{1}{\pi}\int_\Omega V(p,\omega)\cos\theta\, d\omega \approx \frac{1}{\pi}\int_\Omega V_{cap}(p,\omega)\cos\theta\, d\omega \tag{3.45}$$

A spherical cap has two parameters, the average direction of occlusion and the size of the cap, i.e. the solid angle subtended on the hemisphere of the receiver point. These two parameters are stored for each direction around an object as functions of the distance from

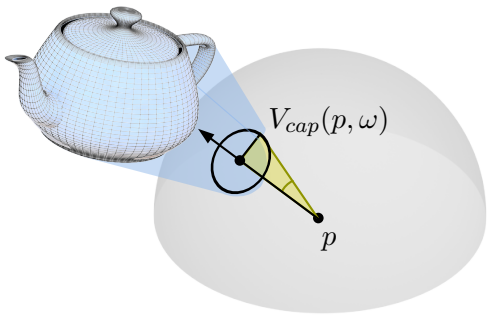


Figure 3.10.: *The spherical cap for a teapot object. The arrow points in direction ω and the angular radius is depicted by the yellow angle.*

the object. Using a function with a fixed number of coefficients to represent the spherical parameters for one direction instead of explicitly storing sampled values reduces the storage cost. At run-time, the parameters for the spherical cap approximation as seen from a receiver point p are retrieved by evaluating the radial function that it stored for an object in direction to p. The ambient occlusion term for a given spherical cap is then looked up in a precomputed table.

To combine the ambient occlusion effects of multiple occluders at a receiver point p, the accessibility values $1 - A(p)$ of all objects are multiplied. Kontkanen and Laine show how this multiplicative blending is statistically justified.

Malmer et al. [25] use a similar method to compute ambient occlusion shadows from dynamic objects. Instead of using functions to store the spherical cap parameters around an object, they propose storing them in a 3D grid around the object. This increases the memory cost, but allows faster lookups.

Malmer et al. combine the effect of multiple occluders by blending size and direction of the spherical caps in a way similar to Kontkanen and Laine. The blended values for each receiver point are accumulated in an *occlusion buffer*.

Lighting from environment maps is handled by first computing the unoccluded incident environment map lighting at a receiver point using the surface normal and then subtracting from it the environment map lighting coming from the cone of occluded directions which can be looked up in the occlusion buffer.

Since ambient occlusion methods average over a receiver point hemisphere, they are usually less exact than PRT (precomputed radiance transfer) methods that explicitly store hemispherical information of a receiver point in some function basis. However, ambient occlusion methods are a cheaper alternative to the more exact PRT methods, especially when used in conjunction with traditional real-time rendering techniques like local-illumination rendering and shadow maps.

3.4. All-frequency PRT

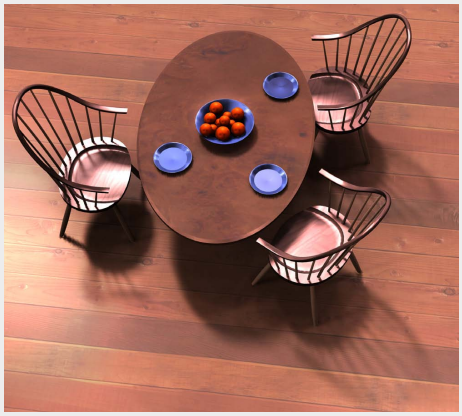


Figure 3.11.: *All-frequency PRT image. Note the relatively sharp shadows and the glossy materials. (Image from [27].)*

The spherical harmonics basis is only suitable for handling low-frequency light transport effects like soft shadows and diffuse reflections. For high frequency effects, like specular highlights and sharp shadows, excessive amounts of SH coefficients are needed and ringing artifacts deteriorate the quality of the approximation. Ng et al. [26, 27] propose using a 2D Haar wavelet basis instead to precompute high frequency light transport effects. The wavelet basis used by Ng et al. is orthonormal and defined as follows (after Stollnitz et al. [38]):

The 2D Haar wavelet basis

There is one scaling basis function defined over the unit square:

$$\Phi(x, y) = 1 \text{ for } (x, y) \in [0, 1]^2 \tag{3.46}$$

All other basis functions are scaled and translated versions of three Haar mother wavelets:

$$\Psi_{01}(x, y) = \begin{cases} 1 & \text{if } x < 0.5 \\ -1 & \text{if } x \geq 0.5. \end{cases} \tag{3.47}$$

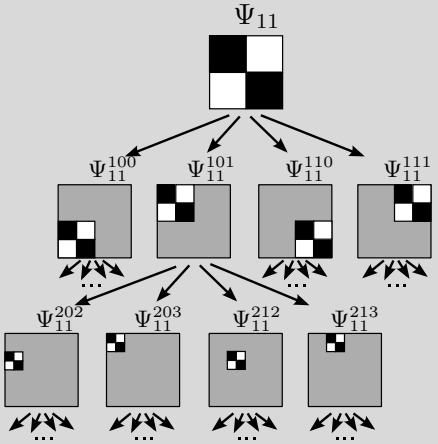
$$\Psi_{10}(x, y) = \begin{cases} 1 & \text{if } y < 0.5 \\ -1 & \text{if } y \geq 0.5. \end{cases} \tag{3.48}$$

$$\Psi_{11}(x, y) = \begin{cases} 1 & \text{if } (x - 0.5)(y - 0.5) > 0 \\ -1 & \text{if } (x - 0.5)(y - 0.5) \leq 0. \end{cases} \tag{3.49}$$

The Haar mother wavelets are defined over the unit square and are implicitly zero outside their domain. Scaling and translating is done quadtree-like:

$$\Psi_M^{lij} = 2^l \Psi_M(2^l x - i, 2^l y - j) \quad i, j \in [0, 2^l) \tag{3.50}$$

where l defines the scale of the wavelet basis function and i and j the position on the unit square. The following figure shows part of the wavelet tree for the Ψ_{11} mother wavelet. Grey areas denote zero values.



Wavelet approximation

In [26, 27], visibility, the BRDF and environment lighting are all approximated in the wavelet basis. Since 2D Haar wavelets are not defined over a sphere like spherical harmonics, Ng et al. first discretize the spherical functions to cubemaps and then project

each cubemap face to the wavelet basis. In contrast to spherical harmonics, high frequencies are accurately represented in the wavelet basis. The wavelet basis coefficients are quantized to 6,7 or 8 bits and only non-zero coefficients are retained. When compared to the cubemap representation, this non-linear wavelet approximation achieves a data compression rate of roughly two to three orders of magnitude.

More coefficients are needed for all-frequency wavelet approximations than for low-frequency spherical harmonics, but much less coefficients are needed than when trying to capture high-frequency effects with spherical harmonics. Ng et al. [26] show that about 100 wavelet coefficients are enough to retain high-frequency lights in an HDR environment map and that spherical harmonics still does much worse than wavelets with as much as 10000 coefficients (partly due to high-frequency ringing).

Factoring the transport function

In their first paper on wavelet PRT [26], Ng et al. computed a single transport function for BRDF and visibility $T(\omega_i, \omega_o, p)$. When using arbitrary BRDFs, this transport function is 6-dimensional: light direction ω_i, view direction ω_o and surface position p. This function's memory requirements are very high, so Ng et al. had to limit their method to either diffuse BRDFs or fixed viewpoint to eliminate the transport function's view-dependency and reduce its dimensionality to a more manageable size.

In their second paper [27], Ng et al. factor the transport operator into BRDF and visibility function. The visibility function is four dimensional, and the BRDF is four dimensional, too, if it could be rotated to a global coordinate frame during run-time.

$$T(\omega_i, \omega_o, p) = \widetilde{\rho}_p(\omega_i, \omega_o) * V(\omega_i, p) \tag{3.51}$$

where $\widetilde{\rho}_p$ is the local BRDF at p rotated to a global coordinate frame. However, wavelet rotation is an expensive operation, so Ng et al. instead use the full 6D BRDF representation (including normal direction) for general, isotropic BRDFs and a reduced representation for the special cases of Lambertian (i.e. perfectly diffuse) and Phong BRDFs. Ng et al. report that when reparametrizing the BRDF to use the reflection vector ω_r instead of the view vector ω_o and sampling the normal direction sparsely, the BRDF representation is compact and the wavelet approximation of the BRDF usually needs less space than the wavelet approximation of the 4D visibility.

Triple Product Wavelet Integrals

At run-time, the Integral of the triple product of visibility V, BRDF ρ and lighting L has to be computed at each receiver point p (the cosine term is included in the BRDF):

$$L_o(p, \omega_o) = \int_\Omega \rho(n, \omega_i, \omega_o) L_{env}(\omega_i) V_p(\omega_i) \tag{3.52}$$

The BRDF ρ is only dependent on the normal n at surface point p and the incoming and outgoing light direction if we assume that the material does not change over the surface of

an object. Ng et al. [27] fetch the precomputed visibility for a given receiver point and the BRDF for a given view direction in each frame from the precomputed tables, removing the dependency on view direction and surface normal. The triple product integral for a specific receiver point and view direction is given by:

$$L_o(p, \omega_o) = \int_\Omega \rho_{n,\omega_o}(\omega_i) L_{env}(\omega_i) V_p(\omega_i) \tag{3.53}$$

Ng et al. developed methods to solve such triple product integrals efficiently in various function bases, including wavelets and the spherical harmonics. Triple Product Integrals also need to be evaluated for products done directly in a function basis, like the SH products for spherical harmonics (see Section 3.1.3). Triple product integrals in the wavelet basis perform best with a linear time complexity $O(n)$ where n is the number of coefficients. Triple product integrals in the SH basis have a time complexity of $O(n^{\frac{5}{2}})$.

Using the techniques described above, Ng et al. can handle all-frequency light transport effects with arbitrary BRDFs in static scenes with changing illumination and viewpoint. They achieve near-interactive rates of 3-5 seconds per frame for typical scenes.

BRDF separation

To decrease the dimensionality of the BRDF, Liu et al. [23] and Wang et al. [42] separate BRDFs into a purely view-dependent part and a purely light-dependent part using singular value decomposition (as in [16]):

$$\rho(\omega_i, \omega_o) = G(\omega_i) F(\omega_o) \tag{3.54}$$

The light-dependent term $G(\omega_i)$ can then be included in the light transport operator without increasing its dimensionality. The view-dependent term is looked up at run-time.

Using BRDF separation, interactive rendering times can be achieved for arbitrary BRDFs in all-frequency lighting environments with changing illumination and viewpoint [23].

Multi-function product integrals

Recently, Sun and Mukherjee [39] developed a general method for product integrals of m functions represented in the 2D Haar wavelet basis (where m may be > 3). The time complexity of the method is $O(nm)$ where n is the number of basis coefficients and m the number of functions.

Sun and Mukherjee use this method for their *Just-in-time Radiance Transfer* (JRT) technique. In JRT, one object at a time can be interactively manipulated (e.g. scaled, translated, etc.). This is achieved by splitting the visibility function into the local visibility of an object that describes self-shadowing and the global occlusions due to other objects in

the scene. All functions are stored in the wavelet basis. The relighting of a scene can then be computed with:

$$L_o(p,\omega_o) = \int_\Omega L_{env}(\omega_i)\rho(p,\omega_i,\omega_o)V(p,\omega_i)O_1(p,\omega_i)O_2(p,\omega_i)\ \ldots\ O_n(p,\omega_i)\ d\omega_i \qquad (3.55)$$

where L_{env} is the environment light, ρ the BRDF, V the local visibility function and O_i the global occlusion due to object i. One of these functions is allowed to change during normal rendering, the product of all other functions is precomputed on the fly in a few seconds from their wavelet representations using Sun and Mukherjee's new method. The product of the fixed functions is the transfer vector T.

For example, when the lighting is allowed to change, the viewpoint and all object occlusions must remain fixed (i.e. the objects must remain static). When the lighting is fixed, the BRDF may change, allowing for a changing viewpoint. When an object is translated (i.e. one of the O_i is changed), lighting and viewpoint must remain fixed.

Rendering is fast, because only a simple dot product between transfer vector and the wavelet representation of the function that is allowed to change has to be computed at each receiver point.

Other function bases

Figure 3.12.: *Images rendered using the method by Green et al. [9]. Note the convincing specular highlights. (Images from [9].)*

Other function bases suited for glossy surfaces have recently been proposed. Tsai and Shih [40] propose using spherical radial basis functions (SRBFs). With SRBFs, usually fewer coefficients than in the wavelet basis are needed to approximate the light transfer function and in contrast to wavelets, SRBFs are naturally defined over the sphere, eliminating the need of an intermediate cubemap representation. There are also some drawbacks when using SRBFs, like having to constrain lighting changes to rotations of the environment map. For more details see [40].

Green et al. [9] propose using a nonlinear Gaussian function basis. Green et al. can render convincing highly glossy materials in real-time with changing lighting and view-

point, although it is unclear whether the method can also handle other all-frequency light transport effects such as sharp shadows. Fore more information see [9].

3.5. PRT for dynamic scenes

PRT works well on static scenes [36, 35] and can handle moderately glossy BRDFs [17, 35] and general all-frequency light transport effects [26, 27, 23, 42, 9, 40, 39]. Dynamic scenes and deformable objects are however more difficult to handle.

Local, deformable PRT

Sloan et al. [37] found a method to handle local deformations on objects by only working with a subset of spherical harmonics where rotations can be implemented with simple rules, in contrast to full spherical harmonics.

The zonal harmonics basis functions are a subset of the spherical harmonics basis functions (see Section 3.1.2). They are symmetric around the z-axis. From spherical convolution, Sloan et al. derive a simple formula to rotate a function represented in the ZH basis:

$$\widehat{\boldsymbol{g}}_l^m = \mathbf{y}_l^m(\omega^*)\sqrt{\frac{4\pi}{2l+1}}\boldsymbol{g}_l \tag{3.56}$$

where $\widehat{\boldsymbol{g}}_l^m$ are the SH coefficients of the rotated ZH vector $\boldsymbol{g}$. ω^* is the axis $\boldsymbol{g}$ has been rotated to and $\mathbf{y}_l^m$ are the basis functions. A single rotated ZH function is called a lobe. An arbitrary SH vector $\boldsymbol{f}_l^m$ can be approximated by the sum of N such lobes:

$$\boldsymbol{f}_l^m \approx \widetilde{\boldsymbol{f}}_l^m = \sum_{i=1}^{N} \mathbf{y}_l^m(\omega_i^*)\sqrt{\frac{4\pi}{2l+1}}\boldsymbol{g}_{l,i} \tag{3.57}$$

The lobe axis ω_i^* and the corresponding ZH vector $\boldsymbol{g}_{l,i}$ are stored for each of the N lobes. These are enough to reconstruct the approximated SH vector $\widetilde{\boldsymbol{f}}_l^m$ at any time. The ZH coefficients $\boldsymbol{g}_l$ of each of the N lobes for approximating an SH vector $\boldsymbol{f}_l^m$ are found iteratively by first using a least squares best match for the first lobe, then subtracting the resulting approximation from $\boldsymbol{f}_l^m$ and approximating the residue with the second lobe and so on. The least squares best match is found with:

$$\sqrt{\frac{4\pi}{2l+1}}\boldsymbol{g}_l = \frac{\sum_{m=-l}^{l} \mathbf{y}_l^m(\omega_i^*)\boldsymbol{f}_l^m}{\sum_{m=-l}^{l}(\mathbf{y}_l^m)^2(\omega_i^*)} \tag{3.58}$$

Sloan et al. use BFGS minimization [30] to improve the approximation.

To rotate a function, the axis of each lobe is rotated. This simple rotation allows rotating the PRT transport vector from a rest coordinate frame to some deformed coordinate frame at run-time. Using this method, the local light transport effects on deformed parts of a model can be approximated with the rotated transport vector.

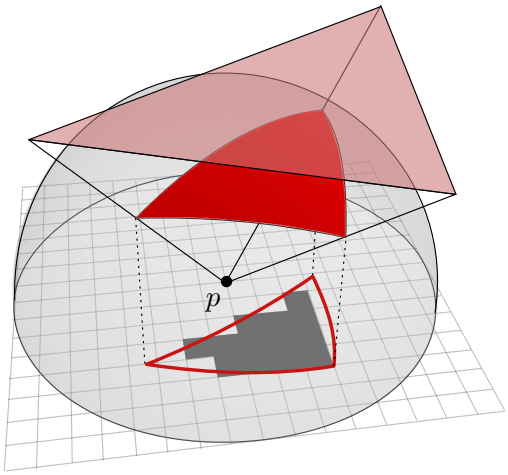


Figure 3.13.: *Illustration of hemispherical rasterization. The projected red triangle is rasterized on a grid tangent to the surface at receiver point p.*

Hemispherical rasterization

Kautz et al. [15] presented a method that can handle shadows in dynamic scenes with deformable objects using the SH basis. They use *hemispherical rasterization* to compute the transfer vector at each vertex on the fly. To compute the SH coefficients of the transfer vector, the following integral has to be evaluated:

$$(T_p)_i(\omega_o) = \int_\Omega \mathbf{y}_i(\omega_i) V_p(\omega_i) \rho(\omega_i, \omega_o) \max(0, \cos\theta)\, d\omega_i \tag{3.59}$$

where $(T_p)_i(\omega_o)$ is the i-th component of the transfer vector at receiver point p for a given view direction ω_o, $\mathbf{y}_i$ is the i-th SH basis function, ρ is the BRDF and V_p is the visibility function at p. θ is the angle between surface normal and light direction ω_i. Note that ρ is not dependent on the receiver point p, since calculations are done in the p's local coordinate frame and the material is assumed to be constant over the surface of an object. Then, the only term of equation 3.59 that is not straightforward to calculate is the visibility function V_p. Hemispherical rasterization is used to approximate this term.

Hemispherical rasterization works as follows: For each vertex, all triangles in the scene are rasterized into a visibility mask. The visibility mask is a regular grid inside the unit disc centered at the vertex. Each cell of the grid contains binary visibility values, which are found by projecting the area of the triangle, as visible from a receiver point, to the grid. Precomputed tables are used to speed up this operation. Multiple triangles are rasterized by using cell-wise OR operations on the visibility of each triangle. The total visibility function for a vertex is then multiplied with the BRDF and the cosine term and projected to the SH basis as described in equation 3.59. The resulting transfer vector can then be used to determine the exit radiance for a vertex.

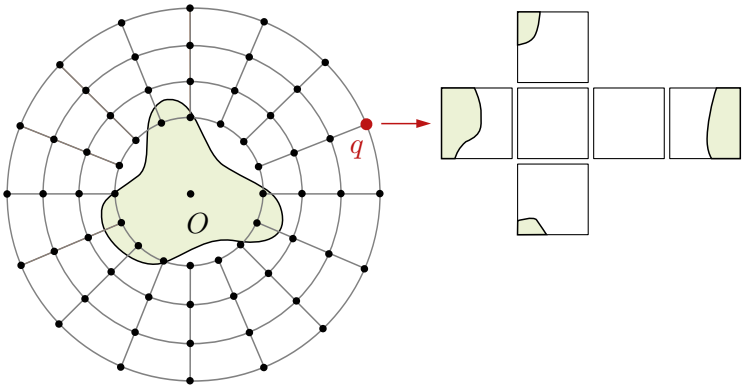


Figure 3.14.: *Illustration of an object occlusion field. The visibility of the green object O is sampled at points in O's sourrounding space. Right: the cubemap sampled for point q.*

Since this method works on the triangles of scene objects at run-time, deformable, dynamic objects can be handled. Kautz et al. report interactive rates for scenes with a few thousand polygons.

Shadow fields

Zhou et al. [44] present a different method for soft shadows in dynamic scenes. They use precomputed *object occlusion fields* (OOF) that describe the shadowing effects of an object on its surrounding space. Local lights can also be handled by using *source radiance fields* (SRF) that describe the incoming light at points sampled around the light source.

At each point of the SRF, the incident radiance is precomputed and stored in a cubemap. OOFs are computed similarly, but visibility values are stored instead of radiance values. In their raw form, the SRF and OOF consume a large amount of storage space. Thus, Zhou et al. compress the SRF and OOF using spherical harmonics for diffuse shadows or wavelets for all-frequency shadows.

At run-time, objects O and local light sources S are sorted by distance from a receiver point p. Only objects closer to the receiver point than a light source may occlude the light source. The occlusion of Object O at receiver point p is fetched from O's OOF using trilinear interpolation of the 8 closest entries to p. Similarly, the lighting from one local light source L incident at p is determined by interpolating the corresponding entries of the L's SRF. The occluded light from L incident at the receiver point p is determined by multiplying the OOF entries of all occluders that are closer to p than L with the SRF entry of L. Light from multiple light sources is summed up at the receiver point.

Using this method, soft shadows from dynamically moving objects can be handled in real-time and all-frequency shadows at near interactive rates. As noted by Zhou et al., the main performance bottleneck of the method is the computation of products of OOF and SRF entries. Since they are represented in the spherical harmonics or wavelet basis, expensive SH or wavelet products have to be calculated at each receiver point for each

object in the scene.

Soft shadows using SH exponentiation

Figure 3.15.: *Images rendered using the method of Ren et al. [32]. The rightmost image shows the sphere approximation of scene objects. (Images from [32, 33].)*

Ren et al. [32] concentrate on soft shadows only, using the low-frequency SH basis. They try to avoid expensive SH products by summing the log of visibility functions instead. Also, Ren et al. compute visibility functions on the fly to keep the storage requirements of their method low.

The outgoing radiance at a receiver point is calculated as usual for PRT[2]:

$$L_o(p) = \int_\Omega L(\omega_i)\rho(n,\omega_i)V_p(\omega_i)\ d\omega_i \tag{3.60}$$

where $L(\omega_i)$ is the environment lighting, ρ the diffuse BRDF, rotated from the surface normal n to the global z-axis and V_p the visibility at receiver point p. The cosine term is included in the BRDF ρ. This triple product integral can be evaluated efficiently in the SH basis with the method described by Ng et al. [27], but double product integrals in the SH basis are still much faster, as they require only a single dot product. For this reason, the SH projection of the product of $L(\omega_i)$ and $\rho(n,\omega_i)$ is precomputed and stored in a table. Finding the outgoing radiance of a point at runtime then only requires a dot product between the precomputed SH projection of $L(\omega_i)\rho(n,\omega_i)$ and the SH projection of the visibility function $V_p(\omega_i)$:

$$\begin{aligned} L_o(p) &\approx \boldsymbol{L}_\rho(n)\cdot\boldsymbol{V}_p \\ \boldsymbol{L}_\rho(n) &= \textstyle\int_\Omega L(\omega_i)\rho(n,\omega_i)\ \mathbf{y}(\omega_i)\ d\omega_i \end{aligned} \tag{3.61}$$

where $\boldsymbol{L}_\rho$ is the SH projection of the product $L(\omega_i)\rho(n,\omega_i)$ and $\boldsymbol{V}_p$ the SH projection of the visibility function. The drawback is that lighting is not allowed to change. Changing

[2]For simplicity, we assume a diffuse BRDF. Glossy BRDFs can also be handled but require more storage space.

lighting can be handled if the SH projection of ρ is multiplied with the product matrix M_L (see Section 3.1.3) of the light in every frame.

The visibility vectors $\boldsymbol{V}_p$ are calculated on the fly in every frame. For this purpose, all scene objects are approximated by sets of spheres in a pre-processing step. Spheres are chosen as approximating geometry, because they are rotationally invariant. The occlusion of a sphere at a receiver point p is described by the blocker visibility function g_p:

$$g_p(\omega_i) = \begin{cases} 1 & \text{if the sphere blocks in direction } \omega_i \text{ as seen from } p \\ 0 & \text{otherwise.} \end{cases}$$

Since the blocker visibility function is axially symmetric, it can be approximated with a single ZH lobe in the direction of the sphere center with the same angular radius the sphere has on p. The SH visibility vector for all possible configurations of one sphere as seen from a point p can then be computed by fetching a precomputed ZH vector of the same angular radius as the sphere and rotating it to the direction of the sphere center.

The combined visibility function of multiple blockers at a receiver point is calculated similar to Zhou et al. [44] as the product of the blocker visibility functions of each sphere:

$$\boldsymbol{V}_p = \boldsymbol{g}_p[1] * \boldsymbol{g}_p[2] * \cdots * \boldsymbol{g}_p[m] \tag{3.62}$$

where $\boldsymbol{g}_p[1]$ is the SH projection of the blocker visibility function of the i-the sphere as seen from point p which can be fetched from the precomputed table.

To avoid expensive SH products, Ren et al. add the log of blocker visibilities:

$$\boldsymbol{V}_p = \exp\left(\log(\boldsymbol{g}_p[1]) + \log(\boldsymbol{g}_p[2]) + \cdots + \log(\boldsymbol{g}_p[m])\right) \tag{3.63}$$

Ren et al. developed new SH exponentiation and SH logarithm techniques to evaluate equation 3.63 efficiently, details are described in Section 3.1.4. For each receiver point p, the combined light and BRDF vector $\boldsymbol{L}_\rho(n)$ can then be retrieved for the surface normal at p. The dot product of the visibility vector $\boldsymbol{V}_p$ and $\boldsymbol{L}_\rho(n)$ approximates the shadowed outgoing radiance at p (see equation 3.61).

This method can handle approximated soft shadows at near real-time rates for scenes consisting of a few ten thousand receiver points (usually vertices) and an average of about 50 processed spheres at each receiver point.

The method described in this diploma thesis is based on the method by Ren et al. Relevant parts of the soft shadow method by Ren et al. will be discussed in more detail in the next chapter.

4. Soft Shadows and Diffuse Reflections in Dynamic Scenes

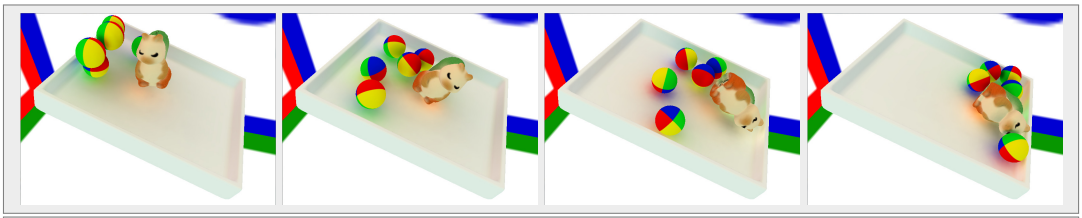


Figure 4.1.: *Images rendered with the new method proposed in this chapter.*

This chapter describes a new method for approximative diffuse reflections and soft shadows in dynamic scenes. It is largely based on the method by Ren et al. [32], described at the end of the previous chapter.

The basic idea is as follows: Soft shadows and diffuse reflections are calculated separately. For soft shadows, the method described in [32] is used. The geometry of every object in the scene is approximated with spheres. Since spheres are rotationally invariant, the visibility of a single sphere can be calculated in the SH basis using simple rules. The total visibility is then computed at each receiver point by summing the log of the single sphere visibilities and using SH exponentiation.

To handle one bounce of diffuse reflections, each sphere also stores radiance values on its surface which represent diffuse reflections from the geometry it is approximating. For each sphere in the scene, the diffuse reflections are updated in each frame using sample points on the geometry surface that have been defined in a precomputation step. At each receiver point, the visibility of a single sphere is calculated as the product of the visibility functions of all closer spheres. The average color of a single sphere as seen from a receiver point is calculated using a form factor on the stored radiance values. The product of visibility value and average color approximates the diffuse reflections from a sphere. The sum of these products from all spheres in the scene approximates the incident radiance from diffuse reflections.

This chapter is organized as follows: First, we describe how to precompute the approximating sphere set of a scene object (Section 4.1). Then, we briefly review the method for soft shadows proposed by Ren et al. [32] which forms the basis for our method and is used for calculating soft shadows (Section 4.2). Finally, we describe how to use the

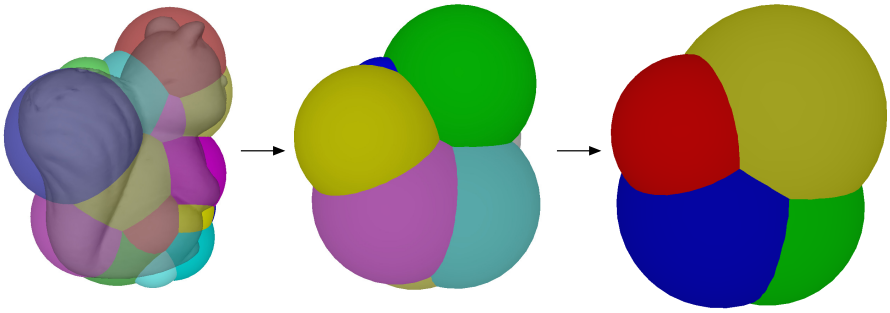


Figure 4.2.: *The sphere hierarchy of a squirrel model. The leftmost image shows the leaf spheres.*

sphere set approximation in precomputation and at run-time to handle one bounce of diffuse reflection (Section 4.3).

4.1. Sphere Hierarchy precomputation

The influence of geometry on diffuse reflections and soft shadows decreases quickly with increasing distance from a receiver point. Thus, at larger distances from a receiver point, coarser sphere approximations provide sufficient accuracy. As in [32], a sphere hierarchy is used to adapt the resolution of the sphere approximation to the distance from a receiver point.

To exploit spatial coherency, receiver point on an object's surface are clustered. At run-time, a sphere cut, i.e. spheres from the appropriate levels in the sphere hierarchy, is found for each receiver point cluster.

Leaf sphere construction

Leaf spheres are constructed with the method described in [43]. The method approximates solid objects with sphere sets and will be reviewed here briefly.

All geometry in a scene is bounded within sphere sets. The algorithm we will describe aims at minimizing the outside volume of a bounding sphere set while maintaining its bounding property. The outside volume of a sphere S_i with respect to an object O is defined as the volume inside the sphere, but outside the object, i.e. how much the sphere "sticks out" of the object. The outside volume of a sphere set $\{S_i\}$ is defined as the sum of the outside volumes of all spheres in the set.

$$E(\{S_i\}, O) = \sum_{i=0}^{n_s} V(S_i/O) \tag{4.1}$$

where n_s is the number of spheres in the sphere set $\{S_i\}$ and $V(S_i/O)$ the volume inside S_i but outside O. The outside volume is a measure of how well the contour of an object as viewed from any direction is approximated by the sphere set.

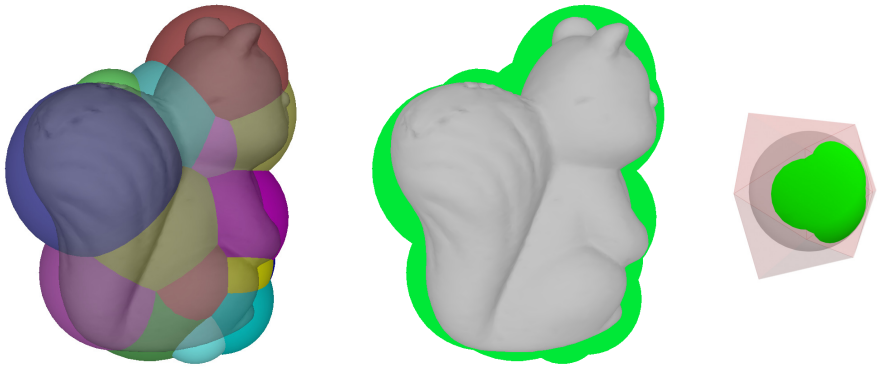


Figure 4.3.: *The outside volume of the squirrel model (as seen from one direction). The image on the right shows a closeup on one of the spheres.*

A variant of Lloyd clustering [24] is used to minimize the outside volume of a sphere set with a given number of spheres. The geometry to be approximated is discretized into a set of surface points and a set of inner points. The points should be approximately equally spaced to keep the number of points low. For the inner points, a regular grid and inside/outside tests define the point positions. For the surface points, a slight variation of the method for even point distribution by Turk [41] is used[1]. A given number of spheres S_i are initialized at random positions inside the Object O. These spheres are then iteratively updated using three steps: point assignment, sphere optimization and sphere teleportation.

In the point assignment step, each point is assigned to exactly one sphere, and the sphere radii are adjusted to bound all points assigned to them. Points are assigned in a flood-fill order, starting from the sphere centers. Points are always assigned to the sphere, whose outside volume increases least.

Sphere optimization is done after each point has been assigned to a sphere. The center of each sphere is optimized for minimum outside volume, while constraining the radius to bound all points that have been assigned to the sphere. Powell's multidimensional minimization [31] is used to find the sphere center with minimum outside volume.

After optimization, all sphere radii and point assignments are reset and point assignment is started again with the new sphere centers. This process is iterated until the outside volume improvement of an iteration falls below a given threshold. At this point, sphere teleportation [5] is used to escape local minima. The most redundant sphere is warped to the location where it is needed most: The sphere S_O with largest overlap is removed and instead, the sphere S_E with largest outside volume is split into two spheres, the new centers being the two points with largest distance in the set of points assigned to

[1]The repelling force in Turk's method is calculated using the geodesic distance after projection onto a plane. Here, the 3D euclidean distance is used instead. This avoids some problems with concave surfaces that are close together in geodesic distance after projecting, but relatively far away in 3D space.

Figure 4.4.: *Receiever point clusters on the squirrel model.*

S_E. Sphere teleportation is followed by another iteration of point assignment and sphere optimization. If the outside volume improved, the teleportation is accepted, otherwise the sphere set is returned to its previous state.

For more details on this method see [43].

Parent sphere construction

Parent spheres are constructed in a similar manner, as described in [32]. Child spheres are assigned to parent spheres in the next higher hierarchy level using Lloyd clustering [24]. A given number of parent spheres P_i are initialized at random locations and child spheres S_i are assigned to the closest parent spheres based on sphere center distance $||c_{S_i} - c_{P_i}||$.

After all child spheres have been assigned, the radius of each sphere is updated to bound all assigned child spheres. Powell's method [31] is then used to optimize the parent sphere's radius by adjusting its center and constraining the radius to continue bounding all assigned child spheres. The parent spheres then become the child spheres for the next higher hierarchy level. This process is repeated until a maximum depth has been reached. The number of parent spheres is usually determined from the number of child spheres by using a given branching factor.

Note that every hierarchy level continues to bound the original geometry, although with a larger outside volume.

Receiver point clusters

Receiver points are clustered using Lloyd clustering [24], as in [32]. A given number of clusters is initialized with random center positions. Each receiver point is assigned to the closest cluster. After all points have been assigned to clusters, each cluster bounding sphere radius is minimized by optimizing its center. Point assignment and cluster optimization is repeated until there is no more significant cluster radius improvement. Teleportation is not used, as it is usually not necessary to achieve good results.

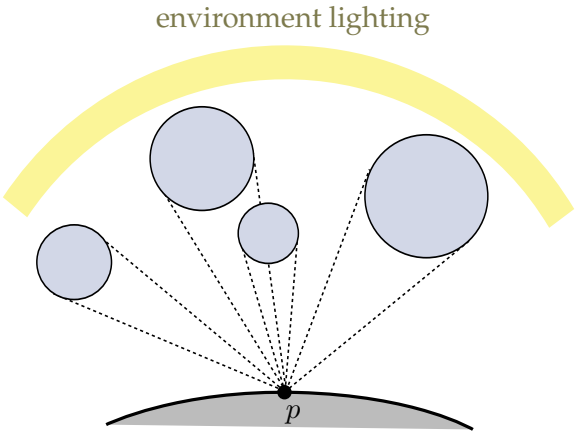


Figure 4.5.: *Shadowing from multiple blocker spheres at the receiver point p.*

4.2. Soft Shadows

Soft shadows are calculated as in Ren at al. [32]. This section will briefly review this method.

In precomputed radiance transfer, soft shadows are calculated using the visibility, incoming light and BRDF at each receiver point. Assuming we only use diffuse BRDFs, the shadowed outgoing radiance at a receiver point can be formulated in this way:

$$L_p = \int_\Omega V_p(\omega_i)\rho_n(\omega_i)L_{env}(\omega_i)\ d\omega_i \tag{4.2}$$

where ω_i is the incoming radiance direction, V_p the visibility function at receiver point p, ρ_n the BRDF for normal n and L_{env} the incoming radiance from the environment.

The main problem in dynamic scenes is finding the visibility function V_p for each receiver point at run-time, since V_p may change in every frame, depending on the scene configuration. Ren et al. try to solve this problem by approximating every object with a set of spheres. Sphere sets have the advantage, that they can approximate the 2D shape of an object as seen from any viewpoint, while the shape of the single parts composing the shape of the object (i.e. the spheres) remain relatively constant from any viewpoint. This allows us to compose the shape of any object from any viewpoint, and thus the visibility from that viewpoint, at run-time using a relatively small set of simple primitives that can be precomputed and stored in a table.

The blocker function of a blocker object B at a receiver point p is defined as:

$$B_{p,B}(\omega_i) = \begin{cases} 0 & \text{if the object } B \text{ blocks in direction } \omega_i \\ 1 & \text{otherwise.} \end{cases} \tag{4.3}$$

$B_{p,B}$ can be approximated by the product of a set of m spherical blocker functions B_{p,S_i}:

$$B_{p,B}(\omega_i) \approx B_{p,S_1}(\omega_i)B_{p,S_2}(\omega_i)\ \ldots\ B_{p,S_m}(\omega_i) \tag{4.4}$$

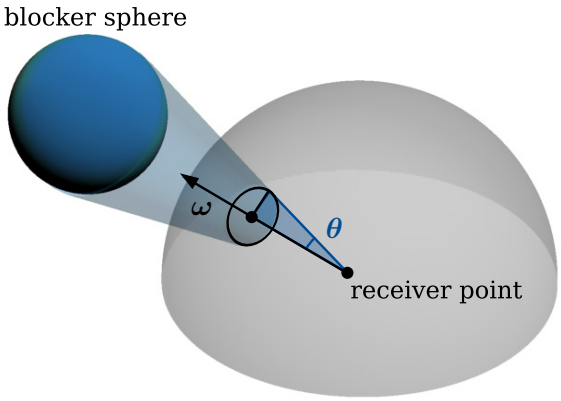


Figure 4.6.: *Parameters for the blocker SH vector of a single blocker sphere.*

The B_{p,S_i} correspond to the m spheres in the sphere set approximation of the blocker B. The total visibility function V_p at a receiver point p is the product of all blocker functions and can be approximated by the product of the blocker functions of all n spheres in the scene:

$$V_p(\omega_i) \approx B_{p,S_1}(\omega_i) B_{p,S_2}(\omega_i) \ \ldots \ B_{p,S_n}(\omega_i) \tag{4.5}$$

Visibility and blocker functions are represented in the SH basis (see section 3.1), so equation 4.5 becomes:

$$\boldsymbol{V}_p \approx \boldsymbol{V}'_p = \boldsymbol{B}_{p,S_1} * \boldsymbol{B}_{p,S_2} * \ \ldots \ * \boldsymbol{B}_{p,S_n} \tag{4.6}$$

where $\boldsymbol{B}_p$ is the SH projection of B_p and $*$ denotes the SH product. SH products are expensive, so instead of using SH products, we can sum the log of the blocker functions and use the SH exponential on the result to get the visibility SH vector:

$$\boldsymbol{V}_p \approx \boldsymbol{V}'_p = \exp\left(\log\left(\boldsymbol{B}_{p,S_1}\right) + \log(\boldsymbol{B}_{p,S_2}) + \ \ldots \ + \log(\boldsymbol{B}_{p,S_n})\right) \tag{4.7}$$

where $\exp$ denotes the SH exponential and $\log$ the SH logarithm. For more information on SH exponentials and SH logarithms see Section 3.1.4. The SH logarithm es even more expensive than the SH product, but since the possible shapes of the sphere blocker functions are limited, the SH logarithm $\boldsymbol{G}_{p,S_i}$ of the $\boldsymbol{B}_{p,S_i}$ can be precomputed.

The blocker SH vector $\boldsymbol{B}_{p,S_i}$ of a sphere S_i at receiver point p can be explicitly defined by two parameters, the direction ω from p to the center of the blocker sphere S_i and the angular radius θ of S_i as seen from p (see Figure 4.6). Since B_{p,S_i} is circularly symmetric, it can be represented as a zonal harmonics vector $\boldsymbol{B}_\theta$ of given angular radius θ rotated to direction ω. The rotation to ω can be done at run-time using the simple rotation rules for ZH vectors (see Section 3.1.2). The log $\boldsymbol{G}_{p,S_i}$ of $\boldsymbol{B}_{p,S_i}$ is also circularly symmetric, so it obeys the same rotation rules. This allows us to store the log blocker vectors of spheres in a 1D table of angular radius, instead of a 3D table of angular radius and center direction.

Precomputation

During precomputation, we tabulate the log $\boldsymbol{G}_\theta$ of the ZH blocker vectors $\boldsymbol{B}_\theta$ by angular radius θ. The table is relatively compact, since ZH vectors only have n_l components, where n_l is the number of SH bands.

To rotate a ZH vector with n_l bands to a direction ω, we have to know the value of each SH basis function $\mathbf{y}_l^m$ at ω, up to band $n_l - 1$. These values are also precomputed and tabulated for directions ω.

Efficient SH exponentiation also requires precomputed a,b coefficients as described in Section 3.1.4. The a,b coefficients are precomputed using the $\boldsymbol{G}_\theta$ and corresponding $\boldsymbol{B}_\theta$ and tabulated by the magnitude of the ZH vector $\boldsymbol{G}_\theta$.

In total, we need three tables to compute the visibility at a receiver point: the 1D table of log ZH vectors $\boldsymbol{G}_\theta$, the 2D table of SH basis function values and the 1D table of a,b coefficients.

Runtime

At runtime, the tabulated $\boldsymbol{G}_\theta$ and $\mathbf{y}_l^m$ are interpolated at each receiver point for the specific angular radius θ and center direction ω of each sphere S_i. At each receiver point, the log of the blocker vectors of all spheres in the scene is then summed up and the SH exponential is used on the sum to approximate the total visibility vector $\boldsymbol{V}_p$, as in equation 4.7. For efficient evaluation of the SH exponential, the precomputed a,b coefficient tables are consulted with the magnitude of the input SH vector.

With a given visibility function $\boldsymbol{V}_p$, expressed in the SH basis, evaluating the integral in equation 4.2 only requires a triple product integral [27] of the three terms visibility, BRDF and incoming radiance, each expressed in the SH basis. Another possibility is to precompute the product of BRDF ρ_n and incoming radiance L_{env} in the SH basis. Evaluation of equation 4.2 at each receiver point given the visibility then reduces to a simple dot product between the visibility SH vector and the precomputed product of BRDF and incoming radiance, although the lighting L_{env} must then remain fixed.

Avoiding problems with receiver points inside spheres

All receiver points on the surface of an object O are inside at least one of the spheres of the sphere set approximation $\{S_i\}_O$ of O, since $\{S_i\}_O$ bounds the object O. Receiver points should not be shadowed by these spheres, so the same rules as in Ren et al. [32] are used to adjust the spheres at each receiver point. A sphere S with center x_s and radius r_s is adjusted at a receiver point p with surface normal n and tangent plane T_p using the following rules:

- If S is completely in front of T_p, do not adjust S.
- If S is completely behind T_p, remove S. We do not handle material transmittance, so these spheres have no influence on the receiver point p.

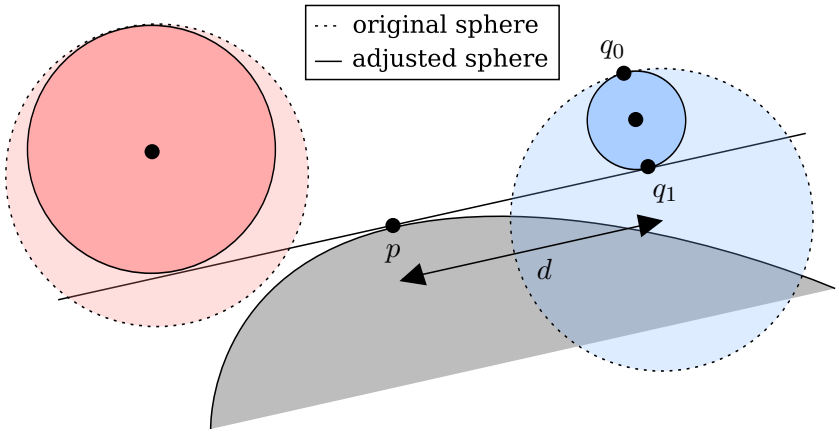


Figure 4.7.: *Blocker sphere adjustments for spheres containing the receiver point.*

- If S intersects T_p and the center x_s is in front of T_p, reduce the sphere radius r_s and adjust the center x_s so that S becomes tangent to T_p (see Figure 4.7, red sphere).
- If S intersects T_p and the center x_s is in behind T_p, remove S if the receiver point p is contained in S. If p is not contained in S, reduce the radius of S and move it along the direction of the normal n until it is in front of and tangent to T_p (see Figure 4.7, blue sphere). To avoid discontinuities when p moves from inside S to the outside, S is scaled as a function of p's distance to S using the following factor α:

$$\alpha = \min(1, \frac{||p - q_1|| - d}{d}) \quad d = \sqrt{r_s^2 - (r_s - ||q_1 - q_0||)^2}$$

These rules ensure, that p is not contained in any sphere that is used for calculating the visibility at p.

Using sphere hierarchies

So far we have only considered the leaf spheres of sphere set approximations. Sphere hierarchies are used in a way similar to Ren et al. [32], although with small changes that are necessary due to diffuse reflections. The Section 4.3 will explain why these changes are necessary. In contrast to Ren et al., we choose only spheres from a single level in the hierarchy. Which level in the hierarchy is used is determined once per receiver point cluster and the same hierarchy level is used for all points in the cluster.

Given a receiver point cluster with a bounding sphere of radius r_c and center x_c, the maximum angular radius subtended by a given sphere S_i at any point in the cluster is given by

$$\theta_{max} = \sin^{-1}\left(\frac{r_s}{||x_s - x_c|| - r_c}\right) \tag{4.8}$$

where r_s is the radius and x_s the center of S_i. The level in the sphere hierarchy is chosen so that the angular radius θ_{max} of any sphere in the hierarchy level is not larger than a given threshold.

To avoid discontinuities between clusters that use different hierarchy levels, at each receiver point in the cluster, a ratio is applied to the log visibility SH vectors $\boldsymbol{F}_{p,S_i}$ of all spheres at the chosen hierarchy level. The ratio is calculated once in every cluster by computing two visibility vectors for each sphere at the cluster center x_c: the *bounding* log visibility vector $\boldsymbol{F}_b$ of a parent sphere S_i and the *detailed* log visibility vector $\boldsymbol{F}_d$. $\boldsymbol{F}_d$ is the sum of the log visibility vectors of S_i's child spheres. The ratio vector then scales each band of $\boldsymbol{F}_b$ to match $\boldsymbol{F}_d$:

$$(\boldsymbol{W}_{S_i})_l = \frac{\sum_{m=-l}^{l}(\boldsymbol{F}_b)_{lm}(\boldsymbol{F}_d)_{lm}}{\sum_{m=-l}^{l}(\boldsymbol{F}_b)_{lm}^2} \tag{4.9}$$

At run-time, the ratio vector is used to scale each band of the log sphere visibility functions $\boldsymbol{F}_{p,S_i}$ at each receiver point. A similar ratio is calculated for diffuse reflections which will be described in the next section.

4.3. Diffuse Reflections

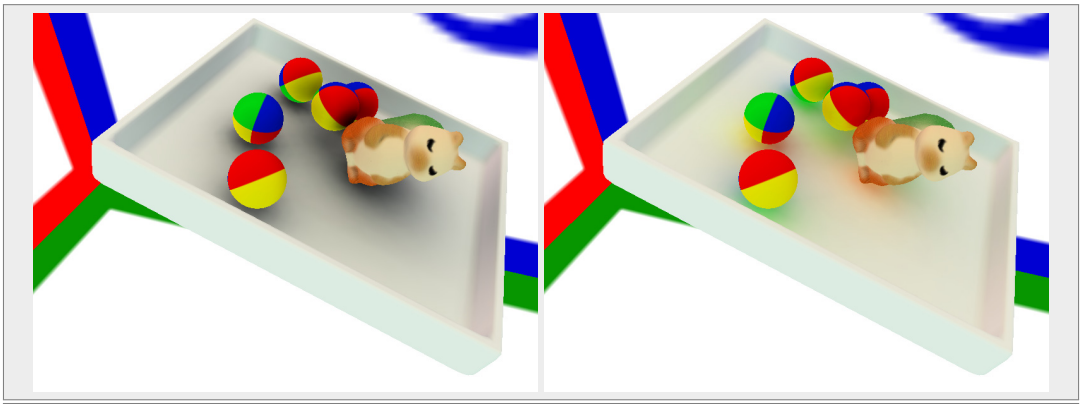


Figure 4.8.: *Left: image with soft shadows, rendered using the method of Ren et al. [32]. Right: The same image with one bounce of diffuse reflections.*

This section describes a possible method to extend the soft shadows method by Ren et al. [32] to handle approximate diffuse reflections.

To complement soft shadows, diffuse reflections also have to originate from the sphere set approximations of objects in the scene. If different approximating geometry was used, soft shadows and diffuse reflections cast on a surface would not match.

In a precomputation step, the surface properties of an object that are relevant for diffuse reflections are approximated in the sphere set approximation of that object. Each sphere

in the set stores diffuse surface color (i.e. the albedo for each r,g,b frequency band) and surface position and normal samples of the object's surface in the sphere's vicinity.

In a first pass, this information is used to approximate the diffuse reflections coming from an object as seen from any direction. The direct illumination of the object's surface, including soft shadows, is projected to the sphere set approximation of the object. Each sphere approximates the illumination of a part of the object's surface. The approximated illumination is stored in an SH vector on each sphere, called the *radiance vector*.

In a second pass, each receiver point determines one unclamped color for each sphere by multiplying the radiance vector with a form factor vector. The resulting color approximates the average diffuse reflections coming from the direction of the sphere. This color is then multiplied with the visibility of the sphere and the BRDF to determine the contribution of the sphere to diffuse reflections at the receiver point. The sum of the diffuse reflection contributions of all spheres approximates the total incident radiance from diffuse reflections at the receiver point.

4.3.1. Precomputation for diffuse reflections

At run-time, diffuse reflections are calculated from information stored in the sphere approximation, rather than using the actual geometry. Surface properties of an object are projected to its sphere set approximation in a precomputation step. For each sphere in the sphere set approximation, *surface color environments* and *surface sample points* are precomputed. *Surface color environments* store the diffuse surface color, i.e. the albedo, of the object's surface. *Surface sample points* store the surface position and normal for a predefined set of sample directions.

Surface color environments

The surface color environment describes the diffuse color of the surface of an object in the vicinity of a sphere. For an object O and a sphere S_i, the surface color environment c_{S_i} is obtained by projecting the diffuse color of O to the surface of S_i, i.e. each direction ω on the sphere represents the diffuse color of O at the intersection point of a ray starting at the center of the sphere S_i in direction ω. The surface color environment c_{S_i} is stored as three SH vectors $(\boldsymbol{c}_{S_i})_r$, $(\boldsymbol{c}_{S_i})_g$ and $(\boldsymbol{c}_{S_i})_b$ over the surface of sphere S_i, one for each of the r,g,b color components.

First, a segment of the surface of object O is assigned to each sphere. A sphere S_i gets the set of surface points P_{S_i} which have the smallest distance to S_i. We use a distance measure d that takes into account the radius of S_i:

$$d(p, S_i) = \frac{||p - x_{S_i}||}{r_{S_i}} \tag{4.10}$$

where p is a surface point on O, x_{S_i} the center of sphere S_i and r_{S_i} the radius. This ensures that each part of the surface is only approximated by a single sphere.

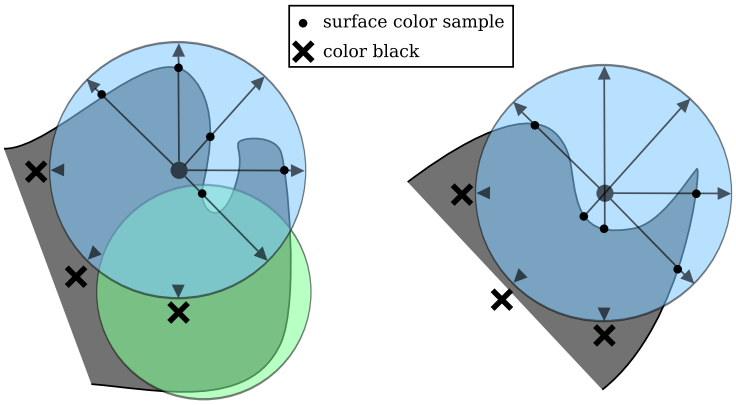


Figure 4.9.: *Surface color sampling. The left figure shows the first two cases, for a sphere center is inside the model. The right figure shows the third case for a sphere center outside the model.*

Now, we shoot rays from the center of each sphere S_i in all directions ω and record the intersection points with the surface of object O.

$$c_{S_i}(\omega) = c_o(h(x_{S_i}, \omega)) \tag{4.11}$$

where c_o is the diffuse surface color of object O and $h(x_{S_i}, \omega)$ is the intersection point of a ray, cast from the center x_{S_i} of sphere S_i in direction ω, with the surface of object O. We want to capture the color of the object surface as seen from a direction opposite to the ray's direction, so we ignore intersections with the front side of the surface and only handle intersections with the back side.

When shooting rays from S_i, the surface of O may be hit more than once in any given direction ω. In general, there are three possible cases (see Figure 4.9):

- There are one or more ray intersection points with P_{S_i} . In this case, the farthest intersection point contained in P_{S_i} is used. More than one intersection with P_{S_i} may happen if the sphere approximation is very coarse (e.g. on higher levels of the hierarchy). In such a case, we can assume that the viewpoint is relatively far away from the sphere (otherwise a finer sphere approximation would have been used) and the outermost surface is visible in direction ω.

- There are only intersection points outside P_{S_i}. Since the sphere set bounds the object O, there must be another sphere S_o of the sphere set in direction ω that approximates the surface color in that direction. Using black (i.e. no diffuse reflections) for that direction is perfectly valid, because the sphere S_i will always be occluded by S_o in direction ω. By using black for these directions, we include occlusions from intersecting spheres in the surface color environment. These precomputed occlusions will be useful at runtime when calculating the visibility of a sphere (see Section 4.3.3).

- There are no intersections. Since geometry for sphere approximation has to be closed, this means that the sphere center is outside the object. In this case, we shoot a ray in the opposite direction ω', this time allowing only the front sides of surfaces to be hit. If this ray does not hit a surface either, use black as the color (i.e. zero albedo, no diffuse reflections at run-time). This case is very rare for usual geometry as the sphere centers are usually inside the object.

Sampling of the object's surface has to be dense in order not to miss any textural detail. Sampling the surface with rays would require to shoot a large amount of rays, making this approach impractical. Instead, the surface color environment is determined by rendering the object to a cubemap using OpenGL.

For every sphere S_i in the sphere set $\{S_i\}_O$, we position the camera at the center of the sphere. We only use the set of triangles that are closest to S_i, using the distance measure d defined above. If parts of a triangle are closer to another sphere, we do not split the triangle, but instead assign it to both spheres. Objects are usually tessellated finely enough so that this approximation is not noticeable in the final SH vectors $\boldsymbol{c}_{S_i}$. For each cubemap face, the camera is rotated to look in the direction of the cubemap face. We then render the object without clipping planes such that triangles behind the camera are rendered, too, but have a lower depth. Triangles are rendered back facing with inverted depth function. The resulting cubemap is then projected to the SH basis for each color component to get the surface color environment SH vectors $\boldsymbol{c}_{S_i}$

Surface sample points

Surface sample points describe the surface position and normal of an object in the vicinity of a sphere. In contrast to surface color environments, the surface positions and normals are stored as sample points for a set of predefined, evenly spaced directions $\{\omega_j\}$ on the sphere. Given a sphere S_i and an object O, for each direction ω_j a ray is cast from the center of S_i in direction ω_j and intersections with O are recorded (see Figure 4.9):

$$n_{S_i}[j] = n(h(x_{S_i}, \omega_j)) \tag{4.12}$$

$$p_{S_i}[j] = h(x_{S_i}, \omega_j) \tag{4.13}$$

For sphere S_i, $n_{S_i}[j]$ denotes the surface normal at the sample point for direction ω_j and $p_{S_i}[j]$ the surface position. $n(x)$ is the normal of the surface at point x and $h(x_{S_i}, \omega_j)$ is the intersection point of a ray, cast from the center x_{S_i} of the sphere S_i in direction ω_j, with the object O. The same surface segmentation defined for the surface environments is used and the same three rules are applied to intersection points with O, only this time, directions ω_j are marked as invalid by using a zero-length normal instead of the color black.

The directions $\{\omega_j\}$ are derived from the vertices of a geodesic sphere (see Figure 4.10). This set of directions is chosen, because it is approximately evenly spaced over the sphere.

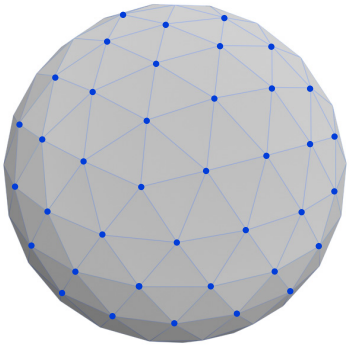


Figure 4.10.: *The vertices of an icosahedral geodesic sphere.*

We assume that object surfaces are relatively smooth, so we find the positions and normals by shooting a single ray per direction ω_j. If the surface of an object was not smooth, the average of multiple rays would have to be used for each direction ω_j.

4.3.2. First Pass: Approximate diffuse reflections in sphere sets

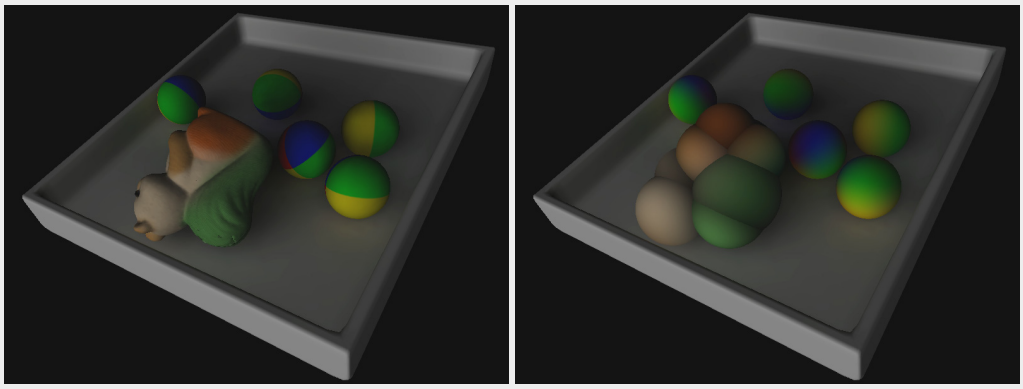


Figure 4.11.: *Radiance SH vectors displayed on the surface of each sphere. The image on the left shows the actual geometry, the image on the right shows the radiance SH vectors of all spheres.*

In the first pass, the surface sample points are used to approximate the diffuse reflections of an object in the vicinity of a sphere. The shadowed exit radiance from direct illumination is calculated at each surface sample point of the sphere using the color white (i.e. no absorption) instead of the actual surface color. The resulting sample radiance values are projected to an SH vector and multiplied with the surface color environment SH vector using SH products. The result is the *radiance SH vector* $\boldsymbol{r}_{S_i}$, defined over the surface of a sphere. A radiance SH vector is computed for each sphere in the sphere set approxima-

tion of an object. It approximates the diffuse reflections of that object in the vicinity of the sphere. Using only the radiance SH vectors, we can approximate the diffuse reflections coming from that object in the second pass. Figure 4.3.2 shows the radiance SH vectors of all spheres in a sample scene.

Consider a sphere S_i of the sphere set approximation $\{S_i\}_O$ of object O. In the precomputation step, a surface color environment SH vector $\boldsymbol{c}_{S_i}$ and surface sample points $p_{S_i}[\omega_j]$ and $n_{S_i}[\omega_j]$ have been assigned to the sphere.

The shadowed outgoing radiance is calculated at each surface sample point using the method for soft shadows by Ren et al. (see Section 4.2):

$$L_p = \int_\Omega L_{env}(\omega_i)\rho_n(\omega_i)V_p(\omega_i)\ d\omega_i \tag{4.14}$$

We make some assumptions about the BRDF that are common in real-time rendering: We assume that the BRDF of a diffuse white surface (i.e. no absorption) is constant over all wavelengths and that the diffuse surface color r,g,b values uniformly scale the BRDF for their respective frequency bands.

$$\rho_{n,cc}(\omega_i) = i_{cc}\ \rho_{n,1}(\omega_i) \tag{4.15}$$

where cc is one of r,g,b and i_{cc} is the reflection intensity (i.e. albedo) for color component cc. $\rho_{n,1}$ is the BRDF of a diffuse white surface and $\rho_{n,cc}$ is the BRDF for one of the r,g,b frequency bands.

Under these assumptions, we calculate the shadowed outgoing radiance for each of the r,g,b frequency bands at each surface sample point p_j using a white diffuse surface color and multiply with the reflection intensity i_{cc} of each of the r,g,b later. Equation 4.14 can then be rewritten for each frequency band as:

$$\begin{aligned} L_{p,cc} &= i_{cc}\ M_{p,cc}(\omega_i) \\ M_{p,cc}(\omega_i) &= \textstyle\int_\Omega L_{env,cc}(\omega_i)\rho_{n,1}(\omega_i)V_p(\omega_i)\ d\omega_i \end{aligned} \tag{4.16}$$

The $M_{p,cc}$ are unclamped r,g,b color values representing the maximum outgoing diffuse radiance (i.e. no absorption) at each of the surface sample points p_j of a sphere S_i. These radiance samples are used to construct an SH vector over the sphere S_i. Because the directions ω_j of the surface sample points are evenly spaced on their sphere S_i, we can construct the SH vector on S_i via (see Section 3.1.1):

$$\boldsymbol{M}_{cc} = \frac{4\pi}{m}\sum_{j=1}^{m} M_{p_j,cc}\ \mathbf{y}(\omega_j) \tag{4.17}$$

where m is the number of sample directions ω_j and $M_{p_j,cc}$ is the maximum outgoing diffuse radiance for color component cc at the surface sample point p_j with direction ω_j.

If a large number of SH bands is used for this projection, there might be too few sample directions ω_j, resulting in undersampling of the SH basis functions. To avoid this, we increase the sample directions and use nearest-neighbour interpolation between the sample

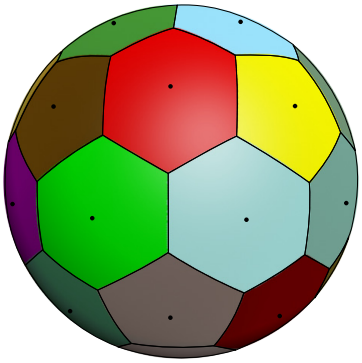


Figure 4.12.: *Voronoi cells of the geodesic samples p_j (black dots) on the sphere surface. The value sampled at p_j is used for all samples inside p_j's Voronoi cell.*

values $M_{p_j,cc}$ of the original set of directions, effectively projecting a function consisting of the Voronoi cells of the original geodesic samples p_j (see Figure 4.12):

$$\boldsymbol{M}_{cc} = \frac{4\pi}{N} \sum_{k=1}^{N} M_{V(\hat{p}_k),cc}\, \mathbf{y}(\hat{\omega}_k) \tag{4.18}$$

where N is the new increased number of sample directions $\hat{\omega}_k$, $N \gg m$. $V(\hat{p}_k)$ is the center of the Voronoi cell containing the sample point $\hat{p}_k$ of direction $\hat{\omega}_k$, i.e. the nearest neighbour of $\hat{p}_k$ in the old set of samples. The set of new sample directions can then be grouped into m disjoint sets $\hat{\omega}_k^v$, each set containing the directions of the sample points in a single Voronoi cell. Equation 4.18 can then be rewritten as:

$$\boldsymbol{M}_{cc} = \frac{4\pi}{N} \left(M_{p_1,cc} \sum_{k=1}^{N_1} \mathbf{y}(\hat{\omega}_k^1) + M_{p_2,cc} \sum_{k=1}^{N_2} \mathbf{y}(\hat{\omega}_k^2) + \cdots + M_{p_m,cc} \sum_{k=1}^{N_m} \mathbf{y}(\hat{\omega}_k^m) \right) \tag{4.19}$$

where N_v is the number of sample points in Voronoi cell v and $\sum_{v=0}^{m} N_v = N$. The new set of directions is chosen randomly as described in section 3.1.1 and each sample point $\hat{p}_k$ is assigned to the Voronoi cell containing $\hat{p}_k$ (i.e. to the closest sample point in the old set of samples). The sums of the basis function samples over each Voronoi cell in equation 4.19 can be precomputed. Thus, the computational complexity at run-time remains $O(m)$ instead of $O(N)$.

Given the SH projection of the maximum diffuse radiance $\boldsymbol{M}_{S_i,cc}$, the radiance SH vector $\boldsymbol{r}_{S_i}$ of a sphere S_i is calculated by using SH products to multiply $\boldsymbol{M}_{S_i,cc}$ with the surface color SH vector $\boldsymbol{c}_{S_i,cc}$ for each color component cc:

$$\boldsymbol{r}_{S_i,cc} = \boldsymbol{M}_{S_i,cc} * \boldsymbol{c}_{S_i,cc} \tag{4.20}$$

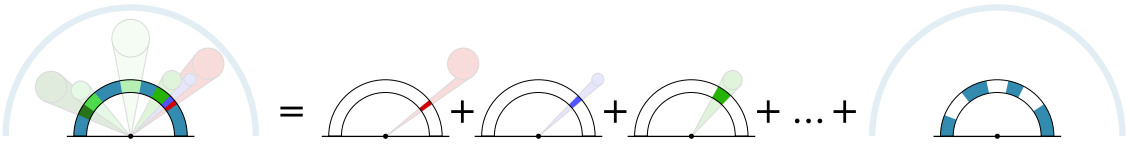


Figure 4.13.: *TODO: write caption*

4.3.3. Second Pass: Calculate exit radiance at receiver points

In the second pass, diffuse reflections from all spheres in the scene are accumulated at each receiver point. The sum of the occluded diffuse reflections of all spheres in the approximating sphere set $\{S_i\}_O$ of object O approximate the diffuse reflections coming from O. The exit radiance at a receiver point p due to diffuse reflections is then given by:

$$L_p^R = \int_\Omega L_{p,S}(\omega_i)\rho_n(\omega_i)\, d\omega_i = \sum_{i=0}^{n_s} \int_\Omega L_{p,S_i}(\omega_i)\rho_n(\omega_i)V_{p,S_i}(\omega_i)\, d\omega_i \tag{4.21}$$

where $L_{p,S}$ is the incident radiance due to diffuse reflections coming from all spheres in the scene and L_{p,S_i} is the unoccluded incoming radiance from sphere S_i. ρ_n is the BRDF for normal n. Since the visibility functions V_{p,S_i} of any two spheres don't overlap, the integral over the hemisphere can be replaced by a sum of n_s integrals, one for each sphere in the scene (see Figure 4.13).

For each sphere S_i, an average of the sphere radiance SH vector $\boldsymbol{r}_{S_i}$ in direction of a receiver point is computed to get the average radiance $\widetilde{L}_{p,S_i}$ from S_i to the receiver point p. This average $\widetilde{L}_{p,S_i}$ is used instead of the function $L_{p,S_i}(\omega_i)$ to approximate equation 4.21:

$$L_p^R \approx \sum_{i=0}^{n_s} \widetilde{L}_{p,S_i} \int_\Omega \rho_n(\omega_i)V_{p,S_i}(\omega_i)\, d\omega_i = \sum_{i=0}^{n_s} \widetilde{L}_{p,S_i}\widetilde{V}_{p,S_i} \tag{4.22}$$

The remaining visibility value $\widetilde{V}_{p,S_i}$ can be determined with a simple dot product of the BRDF SH vector $\boldsymbol{\rho}_n$ and the visibility SH vector $\boldsymbol{V}_{p,S_i}$ of sphere S_i. Equation 4.22 can be evaluated efficiently at run-time. This is critical, since the equation has to be evaluated at each receiver point.

After L_p^R is found, it is added to the radiance value calculated by the soft shadows method L_p^S to get the final shadowed exit radiance value including diffuse reflections at a receiver point p.

$$L_p = L_p^R + L_p^S \tag{4.23}$$

The next two sections describe how to find the average sphere radiance $\widetilde{L}_{p,S_i}$ and the sphere visibility value $\widetilde{V}_{p,S_i}$.

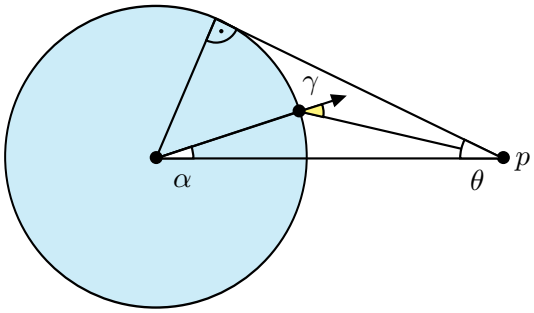


Figure 4.14.: *The form factor geometry. The form factor function is the cosine of angle γ for the angular radius θ of the sphere at the receiver point p.*

Average sphere radiance

To determine the average diffuse radiance coming from a sphere incident at a receiver point (i.e. the average sphere color as seen from the receiver point), we dot the radiance SH vector of the sphere with a form factor SH vector $\boldsymbol{f}$ resulting in a correctly weighted average of the radiance SH vector.

The form factor SH vector $\boldsymbol{f}$ for a sphere S_i at receiver point p depends on the center direction ω and the angular radius θ of S_i at p. $\boldsymbol{f}$ is always circularly symmetric, so it can be represented with a ZH vector and rotated to the direction ω at runtime using the simple rotation rules for ZH vectors (see Section 3.1.2). This allows us to precompute and tabulate $\boldsymbol{f}$ only by angular radius θ. The form factor function f_θ is defined on the sphere S_i and weighs each point on the sphere by the cosine of the angle between surface normal at that point and receiver point direction, similar to the cosine term in the rendering equation (see Figure 4.14). It is a function of angular distance α from the direction $-\omega$ from sphere center to receiver point and given by:

$$f_\theta(\alpha) = \max\left[0, \sin\left(\tan^{-1}\left(\frac{1-\sin\theta\,\cos\alpha}{\sin\theta\,\sin\alpha}\right)-\alpha\right)\right] \tag{4.24}$$

In a precomputation step, this $f_\theta(\alpha)$ is normalized, projected to the ZH basis and tabulated by θ. At run-time, given a sphere S_i, we look up the ZH vector $\boldsymbol{f}_\theta$ in the precomputed table using the angular radius of S_i at the receiver point p. Then we rotate $\boldsymbol{f}_\theta$ to the direction of p and dot the resulting SH vector with the sphere radiance SH vector $\boldsymbol{r}_{S_i}$ to get the average radiance $\widetilde{L}_{p,S_i}$ from S_i to p.

$$\widetilde{L}_{p,S_i} = \boldsymbol{r}_{S_i} \cdot R_p(\boldsymbol{f}_\theta) \tag{4.25}$$

where $\boldsymbol{f}_\theta$ is the form factor SH vector for the angular radius θ and R_p denotes the ZH rotation of $\boldsymbol{f}_\theta$ to the direction of p.

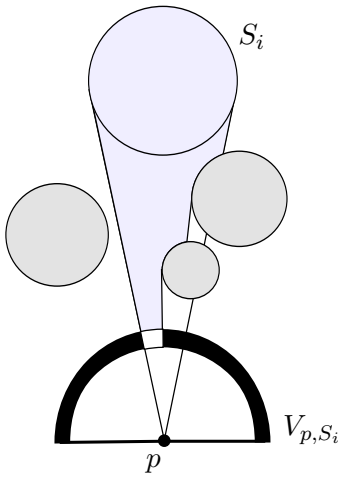


Figure 4.15.: *The visibility function V_{p,S_i} of sphere S_i at the receiver point p with multiple occluding spheres. Zero values are black, values of 1 are white.*

Sphere visibility value

The visibility function of a single sphere at a receiver point is computed similar to Ren et al. [32].

Consider a receiver point p and a sphere S_i. The sphere visibility function is defined as follows:

$$V_{p,S_i}(\omega) = \begin{cases} 1 & \text{if } S_i \text{ is visible in direction } \omega \text{ at receiver point } p \\ 0 & \text{otherwise.} \end{cases} \tag{4.26}$$

The sphere blocker function (see Section 4.2) is defined as:

$$B_{p,S_i}(\omega) = \begin{cases} 0 & \text{if } S_i \text{ blocks in direction } \omega \text{ at receiver point } p \\ 1 & \text{otherwise.} \end{cases} \tag{4.27}$$

Note that $V_{p,S_i}(\omega) \neq 1 - B_{p,S_i}(\omega)$, since sphere blocker functions do not depend on other blockers in the scene, in contrast to sphere visibility functions, which can, in principle, have any shape. For this reason, only sphere blocker functions B_{p,S_i} can be precomputed and tabulated.

To compute the sphere visibility functions at a receiver point p, all spheres in the scene are sorted by distance from p. The sphere visibility function of a sphere S_i can then be computed by multiplying the product of the blocker functions of all sphere closer to p than S_i with the unoccluded sphere visibility function $1 - B_{p,S_i}$ of S_i (see Figure 4.15):

$$V_{p,S_i}(\omega) = (1 - B_{p,S_i}(\omega))\, B_{p,S_i-1}(\omega)\ \ldots\ B_{p,S_1}(\omega) \tag{4.28}$$

where all sphere S_j with $j < i$ are closer to the receiver point p than S_i. The visibility of a sphere is computed in the SH basis:

$$\boldsymbol{V}_{p,S_i} = \widehat{\boldsymbol{B}}_{p,S_i} * \boldsymbol{B}_{p,S_i-1} * \ \ldots\ * \boldsymbol{B}_{p,S_1} \tag{4.29}$$

where $*$ denotes the SH product and $\widehat{\boldsymbol{B}}_{p,S_i}$ the SH projection of $1 - B_{p,S_i}(\omega)$. As explained by Ren et al., it is more efficient to sum the log of the blocker functions:

$$\boldsymbol{V}_{p,S_i} = \exp\left(\log(\widehat{\boldsymbol{B}}_{p,S_i}) + \log(\boldsymbol{B}_{p,S_i-1}) + \ \ldots\ + \log(\boldsymbol{B}_{p,S_1})\right) \tag{4.30}$$

The log of the $\boldsymbol{B}_{p,S_i}$ is precomputed and tabulated as described in Section 4.2. $\widehat{\boldsymbol{B}}_{p,S_i}$ however is not well suited for the log operation, since the value of the corresponding function $1 - B_{p,S_i}(\omega)$ is zero in most parts of its domain for typical spheres S_i. $\widehat{\boldsymbol{B}}_{p,S_i}$ would have to be clamped in large parts of its domain, increasing the approximation error. Also, $\log(\widehat{\boldsymbol{B}}_{p,S_i})$ results in an SH vector with large magnitude which is difficult to handle with SH exponentiation. Instead, we use the difference between sphere blocker function products to determine the sphere visibility function SH vector $\boldsymbol{V}_{p,S_i}$. Equation 4.28 is reformulated as:

$$\begin{aligned} V_{p,S_i}(\omega) &= B_{p,S_i-1}(\omega)\ B_{p,S_i-2}(\omega)\ \ldots\ B_{p,S_1}(\omega) \\ &\quad - B_{p,S_i}(\omega)\ B_{p,S_i-1}(\omega)\ \ldots\ B_{p,S_1}(\omega) \qquad (4.31) \\ \boldsymbol{V}_{p,S_i} &= \exp\left(\ \log(\boldsymbol{B}_{p,S_i-1}) + \log(\boldsymbol{B}_{p,S_i-2}) + \ \ldots\ + \log(\boldsymbol{B}_{p,S_1})\ \right) \\ &\quad - \exp\left(\ \log(\boldsymbol{B}_{p,S_i}) + \log(\boldsymbol{B}_{p,S_i-1}) + \ \ldots\ + \log(\boldsymbol{B}_{p,S_1})\ \right) \qquad (4.32) \end{aligned}$$

Using this formulation, we avoid using $\widehat{\boldsymbol{B}}_{p,S_i}$, although at the cost of an additional SH exponential.

At each receiver point, the sphere visibility SH vector $\boldsymbol{V}_{p,S_i}$ is then dotted with the precomputed BRDF SH vector $\boldsymbol{\rho}_n$ for normal n to get the sphere visibility value $\widetilde{V}_{p,S_i}$:

$$\widetilde{V}_{p,S_i} = \boldsymbol{V}_{p,S_i} \cdot \boldsymbol{\rho}_n \tag{4.33}$$

Sphere intersections are difficult to handle when calculating the visibility of a sphere S_i. The region of V_{p,S_i} blocked by a sphere that intersects S_i may not be circularly symmetric (see Figue 4.16) and is therefore difficult to precompute. We distinguish two cases for intersecting spheres. In the first case, spheres in the sphere set approximations of different objects intersect. To avoid this case, we do not allow the sphere set approximations of different objects to intersect. In the second case, spheres in the same sphere set approximation intersect. Since we assume that objects are not deformable, these intersections are already known at precomputation. We approximately handle these intersections by using the rules for precomputing surface color environments (see Section 4.3.1) and ignoring spheres that intersect S_i in the computation of the visibility $\boldsymbol{V}_{p,S_i}$. The rules for precomputing surface color environments already account for these intersecting spheres and approximate their blocking effect on $\boldsymbol{V}_{p,S_i}$ by integrating their occlusions in the sphere color environment.

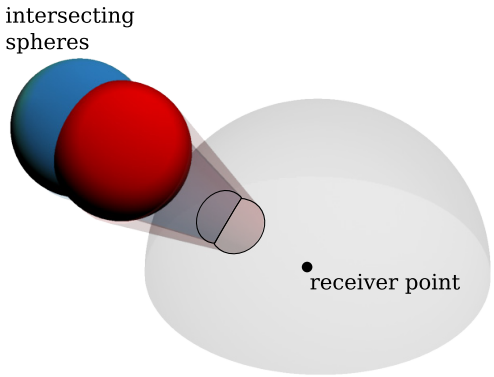


Figure 4.16.: *The visibility for two intersecting spheres. Their visibility functions at the receiver point (red and blue regions on the hemisphere) are not circularly symmetric.*

Using sphere hierarchies with diffuse reflections

How to use sphere hierarchies at run-time is described primarily in Section 4.2. As pointed out in that section, only spheres of a single level in the sphere hierarchy are used. The reason for this is, that the precomputed blocking effects of intersecting spheres that are included the sphere color environments are only valid for spheres in the same level of the hierarchy. Sphere color environments of spheres in different levels of the hierarchy would therefore not match.

To avoid discontinuities at cluster borders, ratios similar to those in Section 4.2 are also calculated for diffuse reflections coming from spheres. These ratios are calculated once in every frame at each cluster center p_c. For this purpose, the occluded diffuse reflection value $(\widetilde{L}_{p_c,S_i}\widetilde{V}_{p_c,S_i})$ of each sphere S_i at the chosen hierarchy level is computed at the cluster center p_c. This value is compared to the sum of the occluded diffuse reflection values of all child spheres of S_i and the ratio is calculated via:

$$w_{S_i} = \frac{\widetilde{L}_{p_c,S_i}\widetilde{V}_{p_c,S_i}}{\sum_{j=0}^{n}\widetilde{L}_{p_c,S_j}\widetilde{V}_{p_c,S_j}} \tag{4.34}$$

where the S_j are the n child spheres of S_i.

At run-time, all receiver points in a cluster multiply the occluded diffuse reflection value of each sphere S_i with the ratio w_{S_i}, thus equation 4.22 becomes:

$$L_p^R \approx \sum_{i=0}^{n_s}\widetilde{L}_{p,S_i}\widetilde{V}_{p,S_i}w_{S_i} \tag{4.35}$$

Object Transformations

Adjusting the sphere set approximation for transformed objects is only possible without major effort for some types of transformations. Translation and uniform scaling are

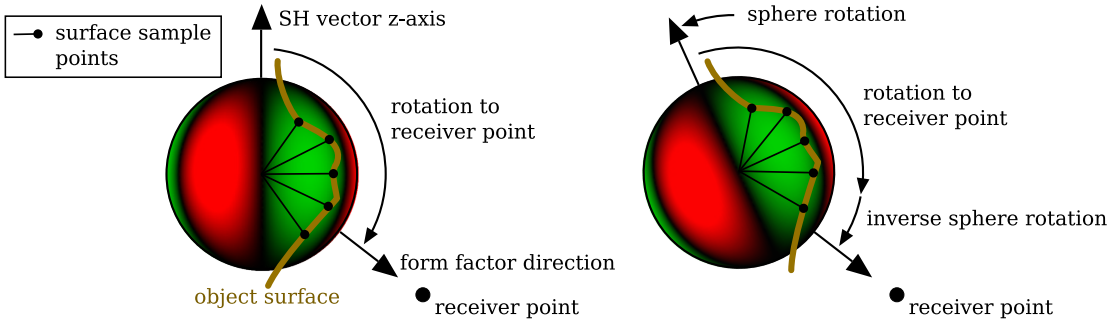


Figure 4.17.: *Rotations of diffuse reflecting objects. To handle object rotations, the form factor direction is rotated with the inverse object rotation. Surface sample points and the surface color environment are kept in the local coordinate frame of each sphere.*

straight-forward. Surface sample points have to be translated/scaled along with the object and for uniform scaling, the sphere center and radius has to be scaled accordingly.

For object rotations, the surface sample points, as well as the surface color environments of each sphere would have to be rotated to global coordinates along with the spheres. However, rotating an SH vector like the surface color environment is an expensive operation. Additionally, when using global coordinates for the SH vectors on each sphere, the precomputed SH basis function sums used for projecting the surface sample radiance values to an SH vector (see Section 4.3.2) would not match the rotated surface sample points. Instead, the surface color environment and the projected radiance value vector are kept in local coordinates so that they do not need to be rotated. When calculating the average sphere colors at each receiver point, the form factor ZH vector (see Section 4.3.3) is rotated with the inverse rotation of the sphere, resulting in a correct average sphere color for the rotated sphere (see Figure 4.17).

Object deformations are more difficult to handle. For typical articulated motion, we need to update the surface environments and the normal and position for the surface sample points, i.e. the surface of the object would have to be re-sampled at each sphere. Finding methods to re-sample and object's surface that are suitable for computation at run-time is left for future work (see Section 7).

5. GPU Implementation

Since we only handle low-frequency effects like soft shadows and diffuse reflections, dense sampling on an object's surface is usually not required. In our implementation, we sample at each vertex and increase the tessellation of an object if necessary. We work with $n = 4$ SH bands (i.e. 16 component vectors). This number of bands usually gives good results for low-frequency effects and fits well into the 4-float architecture of older GPUs. Our implementation needs a relatively large amount of temporary GPU registers, as will be explained later in this section. At the time of this writing, the GeForce 8800 has by far the most temporary registers and we assume that future hardware will continue this trend.

Before using an object for rendering in our implementation, it is first transformed into a suitable format. This format contains vertex clusters and a sphere set approximation of the object, including surface color environments and surface sample points (as described in Section 4.1).

In a precomputation step, we generate all needed tables and store them in textures (see Figure 5.1): The product of environment lighting and BRDF $\rho_n * L_{env}$ is stored as SH vectors on a cubemap of normal directions n, i.e. one SH vector per cubemap pixel. This would require four cubemaps for 16-component SH vectors. To save texture units, we place the cubemap faces of all four cubemaps on a standard 2D texture, using the layout illustrated in Figure 5.1. Erroneous interpolation between neighbouring cubemap faces on the 2D texture is avoided by restricting the texture coordinates to be outside the zones of erroneous interpolation. Texture coordinates inside these zones are clamped to the nearest valid texture coordinate (see Figure 5.1). The SH basis function samples $y_l^m(\omega_j)$ needed for ZH vector rotation (see Section 3.1.2) are stored similar to the product of lighting and BRDF, as SH vectors on a cubemap of sample directions ω_j. The precomputed sphere blocker functions $B(\theta)$ for angular radius θ, the a, b SH exponentiation coefficients for a given vector magnitude and the form factor SH vectors $\boldsymbol{f}(\theta)$ for angular radius θ are stored as 1D tables in a texture, as well as the sum of SH function samples for the Voronoi cells (see Section 4.3.2).

At run-time, two passes are performed in every frame. In each pass, scene information is first prepared on the CPU and then sent to the GPU for processing at each vertex.

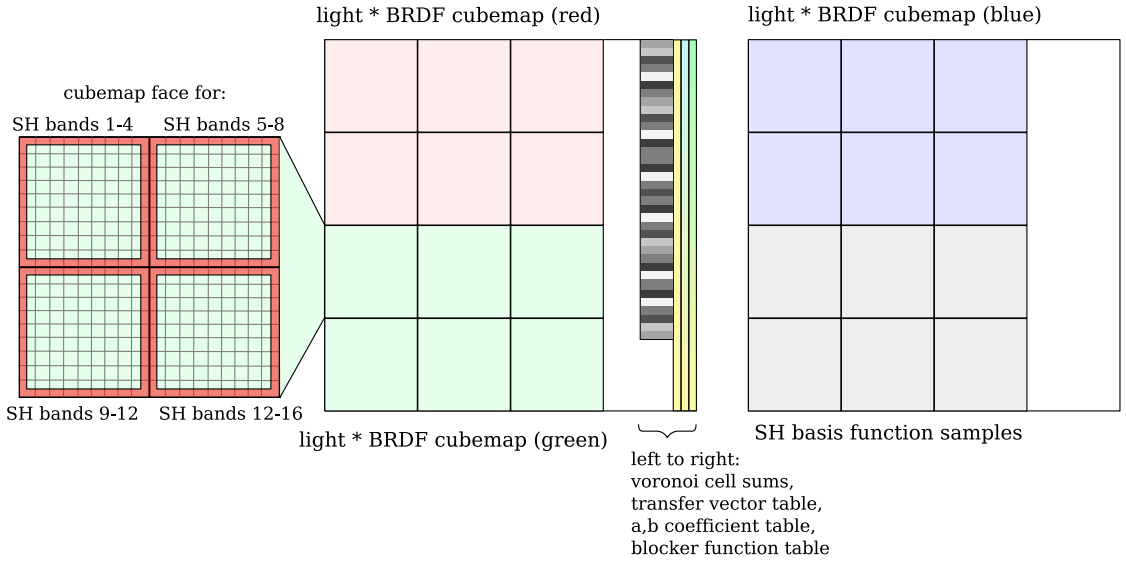


Figure 5.1.: *Two textures that store the tables generated during precomputation. The left image shows a closeup of one of the cubemap faces. Texture coordinates in the red areas are clamped to the closest value outside the red areas to avoid erroneus interpolation.*

5.1. First Pass

In the first pass, the diffuse reflection approximation from each sphere is found by first calculating the shadowed diffuse exit radiance at each surface sample point and then projecting the resulting unclamped color values to an SH vector defined over the sphere and forming the SH product with the surface color environment (see Section 4.3.3). Figure 5.2 gives an overview of these steps.

Three textures are prepared on the CPU in each frame. The *sphere info map* [32] contains one sphere cut (i.e. the spheres of one hierarchy level of each object in the scene) in each row of the texture. One sphere cut is computed for each sphere in the scene and used by all surface sample points on the sphere. Each sphere cut stores radius, center and ratio vector of the spheres in the cut. Thus, the texture size limits the maximum number of spheres in a sphere cut. We use a texture size of 256×256 pixels for a maximum of 128 spheres per cut. The *surface sample info map* contains position, normal and sphere id of the surface sample points of all sphere in the scene. The sphere id determines the row (i.e. the sphere cut) in the sphere info map that is used for a specific surface sample point. Finally, the *sphere color environments texture* stores the surface color environments of each sphere.

These textures are then used in two steps on the GPU. The results of each step are rendered directly to textures using OpenGL frame buffer objects, to avoid slow transfers between video memory and main memory. In the first pass, the surface sample info map and the sphere info map are used to calculate the shadowed outgoing radiance at the

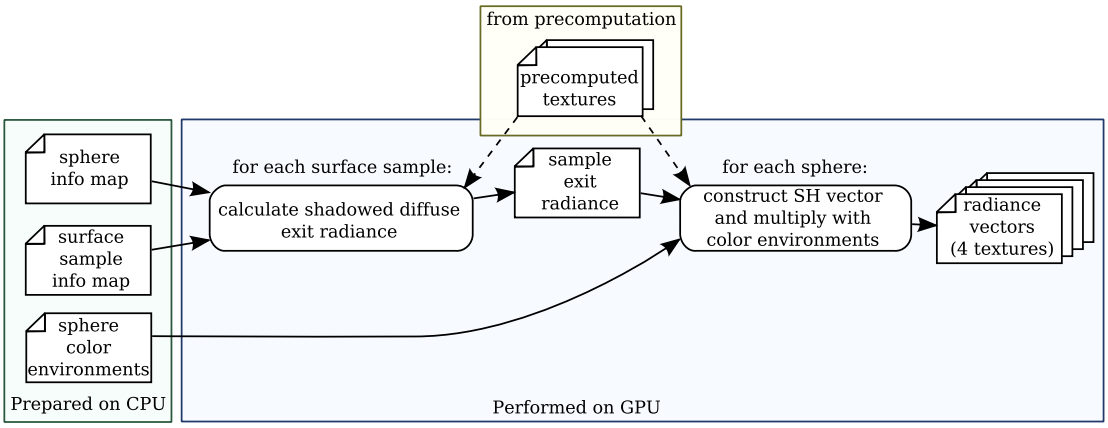


Figure 5.2.: *The computation steps in the first pass. Round boxes correspond to computation steps and square boxes correspond to textures. For details on the precomputed textures see the beginning of Section 5.*

surface sample points of each sphere. The resulting color values are stored in a texture. The second step operates on spheres. The result from the first pass and the sphere color environments texture are used at each sphere to construct an SH vector defined over the sphere and multiply it with the surface color environment of the sphere using SH products. The resulting radiance vectors $\boldsymbol{r}_{S_i}$ are written to four textures of the same size. Four corresponding pixels in the textures contain the SH vector for one color component of the radiance vector $\boldsymbol{r}_{S_i}$. Four textures have to be used, because we can only write to one pixel in each texture and the SH vector resulting from this step needs 16 floats (when using $n = 4$ bands), i.e. 4 pixels. Most modern video cards have enough texture units to handle these four textures directly in the following steps, but if necessary, we could use a third step to merge the four textures into a single texture.

5.2. Second Pass

In the second pass, we use the sphere radiance vector maps from the first pass to calculate the shadowed exit radiance value including diffuse reflections at each receiver point, as described in Section 4.3.3. Figure 5.3 gives an overview of the steps in the second pass.

First, the *vertex info map* and the *sphere info map* are prepared on the CPU, as in [32]. The *vertex info map* contains the position, normal and cluster id of each vertex in the scene. The *sphere info map* is similar to the sphere info map used in the first pass. Each row contains a sphere cut for one vertex cluster. For each sphere, center, normal, ratios are stored. Additionally, a pointer to the sphere radiance vector maps from the first pass is stored for each sphere S_i, i.e. the texture coordinates of the radiance vector $\boldsymbol{r}_{S_i}$ of sphere S_i in the radiance vector maps (the texture coordinate is the same for all four textures,

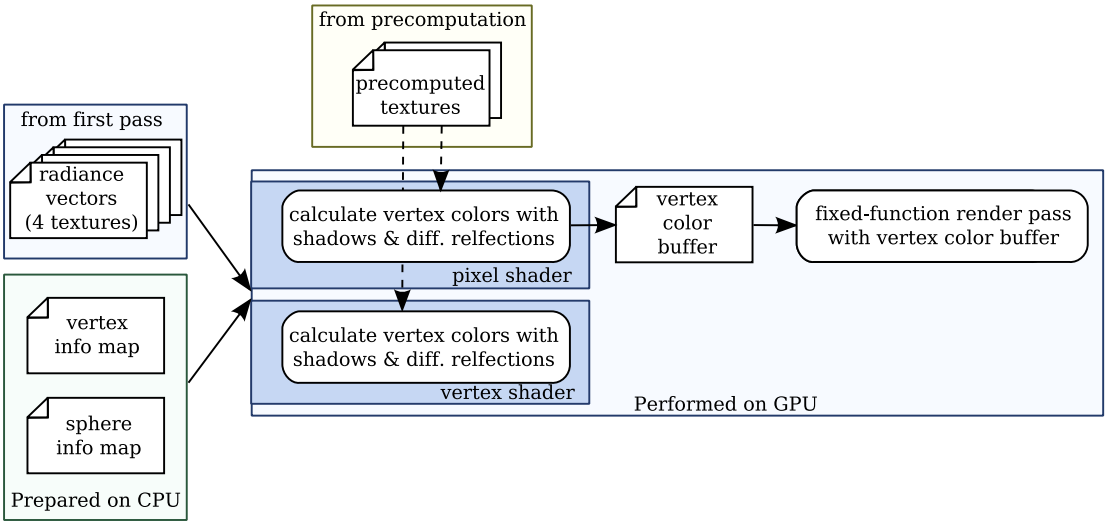


Figure 5.3.: *The computation steps in the second pass. Either vertex or pixel shaders can be used to compute vertex colors.*

only one texture coordinate has to be stored per sphere). The cluster id in the vertex info map is used at each vertex to identify the row in the sphere info map corresponding to the cluster of the vertex. The spheres in each row of the sphere info map are pre-sorted by distance from the cluster center using the CPU.

The remainder of the method has been implemented both as vertex and pixel shader. The pixel shader implementation runs faster, even on modern GPUs like the GeForce 8800. The vertex shader implementation requires one less video memory to video memory copy, but the implementation runs slower. Also, we do not need a vertex info map for the vertex shader implementation, because the vertex data is passed to the GPU using the standard fixed function pipeline. For the pixel shader implementation, we first render the vertex colors (i.e. the diffuse exit radiance value at each vertex, including diffuse reflections and soft shadows) to the frame buffer, one pixel for each vertex color. Then we read the result to a vertex buffer using the OpenGL function `glReadPixels`, as described in [32]. The vertex colors in the buffer are then used in a standard render pass without lighting to render the final image.

The shaders used for both implementations are largely identical: On the GPU, the sphere cut for a vertex is found by using the row in the sphere info map corresponding to the cluster id of the vertex. The pre-sorted spheres in the cut are now sorted by distance from the vertex. The list of spheres is already nearly sorted, so a simple insertion sort performs well. The sorted sphere information is stored in temporary registers so we do not need to do texture lookups when accessing sphere information. As of this writing, the GeForce 8800 has by far the most temporary registers [1], although we assume that future

[1]The GeForce 8800 has 4096 temporary registers, followed by the Radeon X1 series with 128 temporary

hardware will have a number of temporary registers at least comparable to the GeForce 8800. The sorted spheres and the sphere radiance vector maps are then used to calculate the shadowed exit radiance including diffuse reflections as described in sections 4.2 and 4.3. For the SH exponentiation in equation 4.7, either the optimal linear approximation or the hybrid method is used (see Section 3.1.4 for more details on these methods). Both variants were tested and the results are described in the next chapter.

registers.

6. Results

The method for soft shadows and diffuse reflections described in the last chapters achieves framerates of over 20 fps in dynamic scenes by approximating the relatively expensive calculations for soft shadows and diffuse reflections on the actual geometry with cheaper calculations on simple primitives. It can be thought of as a per-sphere ambient occlusion technique, since a single visibility value is calculated for each sphere which is then multiplied with the average diffuse radiance from the sphere. Thus, this method is more exact than pure ambient occlusion, but less exact than pure precomputed radiance transfer, as described in [36]. In the following, we will describe sources of approximation error in this method.

First, as pointed out by Ren et al. [32], SH products only approximate the product of their reconstructed functions. Thus, when multiplying blockers for soft shadows or when calculating the sphere visibility values for diffuse reflections, the result will contain approximation error. This results in somewhat darker shadows and darker diffuse reflections. Ren et al. identified the approximation with SH products as the largest source of approximation error in the soft shadows method.

Second, the blocker accumulation in log space introduces approximation error due to the optimal linear or hybrid approximations of SH exponentiation. As stated by Ren et al., the approximation error of the hybrid method is typically small when compared to the approximation error of SH products. When using the optimal linear method however, it can become noticeable for SH vectors with large magnitude, e.g. if a blocker sphere is close to a receiver point and fills most of its hemisphere (see Figure 6). Later in this chapter, we will compare the performance of the hybrid and the optimal linear methods for SH exponentiation.

Third, the sphere set approximation causes approximation error. How well the sphere set approximates soft shadows and diffuse reflections depends on the number of spheres used. For diffuse reflections, the number of SH bands for the sphere radiance vectors is also important, because it determines how much surface detail a single sphere can approximate. Apart from the resolution of the sphere set, the precomputed occlusions for intersecting spheres also cause approximation error. These occlusions would normally be view-dependent, but in precomputation, we include the occlusions in the color environment, making them less dependent on the viewpoint. The result is a brightening of diffuse reflections for some configurations of intersecting spheres (see Figure 6.2). In our tests, approximation of diffuse reflections in the sphere sets and the precomputation of occlusions for intersecting spheres were the largest sources of approximation error. The former can be alleviated by increasing the number of spheres in a sphere set approxima-

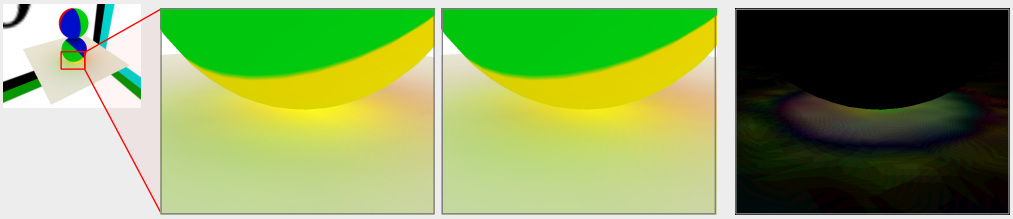


Figure 6.1.: *Comparision of the optimal linear and the hybrid method for SH exponentiation. The images show a closeup of the scene depicted on the left. The left closeup was rendered using the optimal linear method, the right closeup using the hybrid method. The left closeup is slightly too bright, due to the approximation error of the optimal linear method. The rightmost figure shows the difference between the two closeups (the color values were multiplied by a value of 10 to enhance visibility).*

tion at the cost of speed. This is a decision of quality versus performance. The latter could be solved by calculating the visibility of intersecting spheres at run-time. This could possibly be done by precomputing the visibility for possible configurations of intersecting spheres, but this is left for future work (see Section 7).

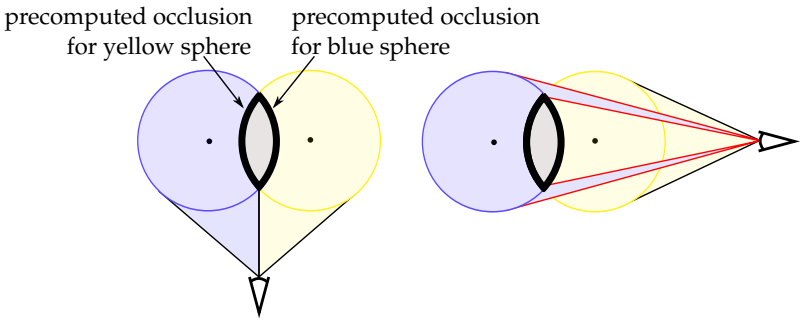


Figure 6.2.: *Approximation error due to precomputed occlusions of intersecting spheres. The left images shows a setup without approximation error. Both spheres have correct visibility. In the right image, too little of the blue sphere is occluded, resulting in diffuse reflections that are too bright.*

In our current implementation, the performance of the algorithm depends quadratically on the number n of spheres in the scene $O(\frac{n^2}{2})$, since every sphere has to be occluded by every sphere closer to the receiver point. This could be reduced to $O(mn)$, where m is the average number of intersections per sphere, if results were re-used during the calculation of the sphere visibility value, i.e. large parts of equation 4.32 are identical for different spheres at the same receiver point and could be calculated only once. The difficulty in re-using results is that only parts of the results from intersecting spheres can be re-used. Finding efficient ways to store and re-use such results on the GPU is left for future work.

Performance tests were done on a PC with an Athlon 64 X2 4200+ processor, 1024 MB ram and a GeForce 8800 GTX. Frames were rendered at a resolution of 800×600 using OpenGL.

The method was tested on three scenes. In each scene, vertices were clustered in a precomputation step and a sphere hierarchy is used for each object causing soft shadows and diffuse reflections (i.e. all objects except the ground box). Soft shadows are always calculated using hybrid SH exponentiation. For diffuse reflections, both hybrid and optimal linear SH exponentiation was tested. Ratios are used for both diffuse reflections and soft shadows (see Section 4.2 and 4.3.3). All scenes are illuminated by the grace cathedral environment lighting [7]. Both the pixel shader and the vertex shader implementations were tested for all scenes.

The squirrel scene is an animated sequence where a squirrel model and a few balls slide down an inclined box. It contains 21k vertices, 14 leaf spheres and 25 vertex clusters. The duck scene contains three rubber ducks in close proximity on a ground box. It contains 33k vertices, 30 leaf spheres and 40 vertex clusters. The vase scene consists of five vases on a ground box. It contains 15k vertices, 20 leaf spheres and 35 vertex clusters. All geometry in the scenes is dynamic. Figure 6.3 shows images from each of the scenes.

The method by Ren et al. for soft shadows achieves real-time frame rates for each scene. Frame rates for the method by Ren et al. are summed up in the following table:

	Pixel Shader	Vertex Shader
squirrel & balls	102 fps	287 fps
ducks	57 fps	107 fps
vases	95 fps	215 fps

The vertex shader implementation outperforms the pixel shader implementation, since one less video memory to video memory copy is needed.

The measured frame rates using the new method for soft shadows and diffuse reflections remain relatively constant in each scene and are summed up in the following table:

	Pixel Shader		Vertex Shader	
	ol	hyb	ol	hyb
squirrel & balls	29 fps	21 fps	17 fps	10 fps
ducks	9 fps	7 fps	5 fps	3 fps
vases	25 fps	21 fps	20 fps	14 fps

The “hyb“ column was measured with hybrid SH exponentiation and the “ol“ column with optimal linear SH exponentiation. The pixel shader implementation consistently

outperforms the vertex shader implementation, even though one less video memory to video memory copy is needed for the vertex shader implementation. When using hybrid SH exponentiation in the pixel shader, the performance only drops by $20\% - 30\%$ compared to the optimal linear method. In the vertex shader, the hybrid method makes performance drop by $30\% - 40\%$. We achieve interactive performance for the ducks scene and relatively smooth framerates of over 20 fps for the squirrel scene and the vase scene using the pixel shader for both optimal linear and hybrid SH exponentiation.

ducks scene (33k vertices, 30 leaf spheres and 40 vertex clusters)

vases scene (15k vertices, 20 leaf spheres and 35 vertex clusters)

animated squirrel scene (21k vertices, 14 leaf spheres and 25 vertex clusters)

Figure 6.3.: *Images from the test scenes. The left column was rendered with soft shadows only, using the method of Ren et al. [32]. The right column was rendered with one bounce of diffuse reflections using the new method. All objects are dynamic.*

7. Conclusion

Precomputed radiance transfer provides a method for rendering realistic images in real-time. Until recently, it has been limited to static scenes, but in the past two or three years, methods for dynamic scenes have been developed, although all of them still have some limitations that make their implementation in general dynamic real-time applications like games difficult. In this thesis, we have tried to contribute to the development of dynamic PRT by presenting a method for approximative diffuse reflections and soft shadows in dynamic scenes, based on the method by Ren et al. [32]. We achieve framreates of over 20 fps for moderately complex geometry and relatively simple sphere approximations and interactive rates for more complex sphere approximations. This performance is achieved by approximating all geometry in the scene with simple primitives and calculating soft shadows and diffuse reflections using only this less complex approximation. We have described how the diffuse reflections of an object can be approximated in a sphere set and how these approximated diffuse reflections can be used at each receiver point.

This method can be extended in several ways. First, we could handle sphere intersections at run-time, e.g. by tabulating the sphere visibility functions for all possible configurations of the intersection of two spheres. Intersections of multiple sphere could be handled by multiplying the visibility functions of the appropriate two-sphere intersections. Handling sphere intersections at run-time would avoid approximation errors caused by the precomputation of occlusions for those intersections.

Also, our method does currently not handle object deformations. To handle these, we would have to change the surface color environment at run-time, possibly by using sample points on an object's surface.

The quadratic dependency of the current implementation on the number of spheres could be made linear if results of the sphere visibility computations at a receiver point could be re-used, as described in Chapter 6.

Currently, our method only simulates one light bounce for diffuse reflections. To extend the method to handle multiple light bounces, we would only need to adjust the way the radiance at the sphere sample points is computed to include diffuse reflections, i.e. use the same algorithm for sphere sample points and receiver points.

7.1. Acknowledgements

I would like to thank Dr. Stefan Jeschke for suggestions and useful tips and the staff at the Institute for Computer Graphics at the Technical University of Vienna for providing

me with the necessary hardware to develop and test the method.

I would also like to thank Paul Debevec for the free lighting environments on his website [7] and the AIM shape repository [1] for providing free models.

A. Formulas for the SH Basis Functions

The spherical harmonics basis functions can be obtained by adapting the 1D associated Lengendre polynomials to the 2D surface of a sphere. Most of the following code and formulas have been adapted from [10].

The associated Legendre polynomials are real-valued functions defined over the range $[-1, 1]$. Like the spherical harmonics basis functions, they have band index l and position inside the band m which is in the range $[0, l]$. The associated Legendre polynomials are orthogonal with respect to a constant for a fixed l (i.e. inside a band) and orthogonal with a different constant for a fixed m (i.e. between bands).

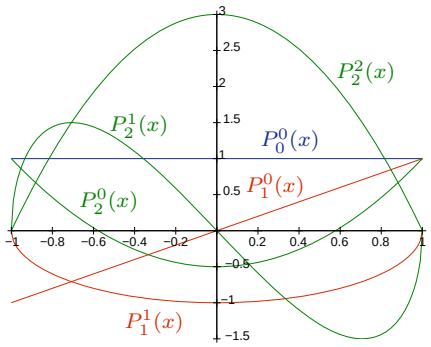


Figure A.1.: *The associated Legendre polynomials up to band 2.*

The polynomials can be constructed from the following three recurrence relations, building polynomials in higher bands from lower-band polynomials:

$$(l-m)P_l^m(x) = x(2l-1)P_{l-1}^m(x) - (l+m-1)P_{l-2}^m(x) \quad \text{(A.1)}$$

$$P_m^m(x) = -1^m(2m-1)!!(1-x^2)^{\frac{m}{2}} \quad \text{(A.2)}$$

$$P_{m+1}^m(x) = x(2m+1)P_m^m(x) \quad \text{(A.3)}$$

In [30], Press et al. describe a strategy to efficiently build higher-order polynomials using these relations. Equation A.2 is a good starting point, since it does not build on lower-order polynomials. It just requires one application for any P_m^m and therefore introduces less roundoff error than iterating any of the relations to get to P_m^m. For the remaining polynomials with $m < l$, the other two relations A.1 and A.3 have to be used. A.1 is

preferable over A.3 since it introduces less roundoff error. Starting at the P_m^m with highest possible m, we have to apply A.3 once and then iterate A.1 until reaching the desired P_l^m. This algorithm is summed up in the following C code:

```
double P(int l, int m, double x)
{
  // calculate P_m^m (second recurrence relation)
  double pmm = 1.0;
  if(m>0) {
    double somx2 = sqrt((1.0-x)*(1.0+x));
    double fact = 1.0;
    for(int i=1; i<=m; i++) {
      pmm *= (-fact) * somx2;
      fact += 2.0;
    }
  }
  if(l==m) return pmm;

  // calculate P_{m+1}^m (third recurrence relation)
  double pmmp1 = x * (2.0*m+1.0) * pmm;
  if(l==m+1) return pmmp1;

  // iterate the first recurrence relation until reaching P_l^m
  double pll = 0.0;
  for(int ll=m+2; ll<=l; ++ll) {
    pll = ( (2.0*ll-1.0)*x*pmmp1-(ll+m-1.0)*pmm ) / (ll-m);
    pmm = pmmp1;
    pmmp1 = pll;
  }
  return pll;
}
```

Adapting the associated Legendre Polynomials to the 2D surface of a sphere using trigonometric functions gives the spherical harmonics basis functions. In general, the spherical harmonics basis functions are defined on imaginary numbers:

$$Y_l^m(\theta, \varphi) = N_l^m \, e^{mi\varphi} \, P_l^m(\cos\theta) \tag{A.4}$$

where N_l^m is a normalization constant:

$$N_l^m = \sqrt{\frac{2l+1}{4\pi} \frac{(l-m)!}{(l+m)!}} \tag{A.5}$$

However, we are only interested in real-valued functions. Spherical harmonics for real functions are defined as follows:

$$y_l^m(\theta,\varphi) = \begin{cases} \sqrt{2}N_l^m \cos(m\varphi)P_l^m(\cos\theta) & \text{if } m > 0 \\ \sqrt{2}N_l^m \sin(-m\varphi)P_l^{-m}(\cos\theta) & \text{if } m > 0 \\ N_l^0 P_l^0(\cos\theta) & \text{if } m = 0. \end{cases} \tag{A.6}$$

where m is in the range $[-l, l]$. The following C code calculates values for the spherical harmonics basis functions using the transformations on the associated legendre polynomials described above:

```
double N(int l, int m)
{
  return sqrt( ((2.0*l+1.0)*factorial(l-m)) /
               (4.0*PI*factorial(l+m)) );
}

double SH(int l, int m, double theta, double phi)
{
  const double sqrt2 = sqrt(2.0);
  if(m==0) return N(l,0)*P(l,m,cos(theta));
  else if(m>0) return sqrt2*N(l,m)*cos(m*phi)*P(l,m,cos(theta));
  else return sqrt2*N(l,-m)*sin(-m*phi)*P(l,-m,cos(theta));
}
```

Bibliography

[1] AIM@SHAPE project - shape repository. http://shapes.aim-at-shape.net/.

[2] szaber.com. http://szaber.com/grafx/index.php?id=2.

[3] BLINN, J. F. Models of light reflection for computer synthesized pictures. *ACM Trans. Graph. 11*, 2 (1977), 192–198.

[4] BUNNELL, M. Dynamic ambient occlusion and indirect lighting. In *GPU Gems2: Programming Techniques for High-Performance Graphics and General Purpose Computation* (2004), Addison-Weseley Professional, pp. 223–233.

[5] COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. Variational shape approximation. *ACM Trans. Graph. 23*, 3 (2004), 905–914.

[6] COOK, R. L., AND TORRANCE, K. E. A reflectance model for computer graphics. *ACM Trans. Graph. 15*, 3 (1981), 307–316.

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

[8] EDMONDS, A. Angular momentum in quantum mechanics. *Princeton University* (1960).

[9] GREEN, P., KAUTZ, J., MATUSIK, W., AND DURAND, F. View-dependent precomputed light transport using nonlinear gaussian function approximations. *Proceedings of the 2006 symposium on Interactive 3D graphics and games* (2006).

[10] GREEN, R. Spherical harmonic lighting: The gritty details. *Game Developers' Conference* (2003).

[11] HECKBERT, P. S. Adaptive radiosity textures for bidirectional ray tracing. *ACM Trans. Gr. 24*, 4 (1990), 145–154.

[12] IVANIC, J., AND RUEDENBERG, K. Rotation matrices for real spherical harmonics. direct determination by recursion. *J. Phys. Chem. 100*, 15 (1996), 6342–6347.

[13] IVANIC, J., AND RUEDENBERG, K. Additions and corrections : Rotation matrices for real spherical harmonics. *J. Phys. Chem. A 102*, 45 (1998), 9099–9100.

[14] KAJIYA, J. T. The rendering equation. *ACM Trans. Gr. 20*, 4 (1986), 143–150.

[15] KAUTZ, J., LEHTINEN, J., AND AILA, T. Hemispherical rasterization for self-shadowing of dynamic objects. In *Proc. of Eurographics Symposium on Rendering 2004* (2004), pp. 179–184.

[16] KAUTZ, J., AND MCCOOL, M. Interactive rendering with arbitrary brdfs using separable approximations. *10th Eurographics Rendering Workshop* (1999), 281–292.

[17] KAUTZ, J., SLOAN, P.-P., AND SNYDER, J. Fast, arbitrary brdf shading for low-frequency lighting using spherical harmonics. *Proc. of the 13th Eurographics Workshop on Rendering* (2002), 291–296.

[18] KONTKANEN, J., AND LAINE, S. Ambient occlusion fields. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (2005), pp. 41–48.

[19] KRIVANEK, J., KONTTINEN, J., PATTANAIK, S., BOUATOUCH, K., AND ZARA, J. Fast approximation to spherical harmonic rotation. *Sketches ACM SIGGRAPH 2006* (2006).

[20] LANDIS, H. Renderman in production. *ACM SIGGRAPH 2002 Course 16* (2002).

[21] LEPAGE, G. P. A new algorithm for adaptive multidimensional integration. *J. Comput. Phys. 27* (1978), 192.

[22] LINDE, Y., BUZO, A., AND GRAY, R. An algorithm for vector quantizer design. *IEEE Transactions on Communication 28* (1980), 84–95.

[23] LIU, X., SLOAN, P.-P., SHUM, H.-Y., AND SNYDER, J. All-frequency precomputed radiance transfer for glossy objects. *Eurographics Symposium on Rendering* (2004).

[24] LLOYD, S. Least squares quantization in pcm. *IEEE Transactions on Information Theory 28* (1982), 129–137.

[25] MALMER, M., MALMER, F., ASSARSON, U., AND HOLZSCHUCH, N. Fast precomputed ambient occlusion for proximity shadows. *Journal of Graphics Tools* (2006).

[26] NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Trans. Graph. 22*, 3 (2003), 376–381.

[27] NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. Triple product wavelet integrals for all-frequency relighting. *ACM Trans. Graph. 23*, 3 (2004), 477–487.

[28] NICODEMUS, F. E., RICHMOND, J. C., HSIA, J. J., GINSBERG, I. W., AND LIMPERIS, T. Geometrical considerations and nomenclature for reflectance. *NBS Monograph 160* (1977).

[29] PHONG, B. T. Illumination for computer generated pictures. *Commun. ACM 18*, 6 (1975), 311–317.

[30] PRESS, W., FLANNERY, B., TEUKOLSKY, S., AND VETTERLING, W. *Numerical Recipes in C, Second Edition. Cambridge University*. Cambridge University Press, 1992.

[31] PRESS, W., FLANNERY, B., TEUKOLSKY, S., AND VETTERLING, W. *Numerical Recipes in C, Second Edition. Cambridge University*. Cambridge University Press, 1992, ch. 10 Minimization and Maximization Functions, pp. 168–185.

[32] REN, Z., WANG, R., SNYDER, J., ZHOU, K., LIU, X., SUN, B., SLOAN, P.-P., BAO, H., PENG, Q., AND GUO, B. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM Trans. Gr. 25*, 3 (2006), 977–986.

[33] REN, Z., WANG, R., SNYDER, J., ZHOU, K., LIU, X., SUN, B., SLOAN, P.-P., BAO, H., PENG, Q., AND GUO, B. Supplemental for real-time soft shadows in dynamic scenes using spherical harmonic exponentiation., 2006.

[34] SCHETZEN, M. The volterra and wiener theories of nonlinear systems., 1980.

[35] SLOAN, P.-P., HALL, J., HART, J., AND SNYDER, J. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph. 22*, 3 (2003), 382–391.

[36] SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph. 21*, 3 (2002), 527–536.

[37] SLOAN, P.-P., LUNA, B., AND SNYDER, J. Local, deformable precomputed radiance transfer. *ACM Trans. Gr. 24*, 3 (2005), 1216–1224.

[38] STOLLNITZ, E. J., DEROSE, T. D., AND SALESIN, D. H. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kauffman, 1996.

[39] SUN, W., AND MUKHERJEE, A. Generalized wavelet product integral for rendering dynamic glossy objects. *ACM Trans. Graph. 25*, 3 (2006), 955–966.

[40] TSAI, Y.-T., AND SHIH, Z.-C. All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. *ACM Trans. Graph. 25*, 3 (2006), 967–976.

[41] TURK, G. *Texturing Surfaces Using Reaction-Diffusion*. 1992, ch. 4.3 Even Distribution of Points over a Polygonal Surface, pp. 50–55.

[42] WANG, R., TRAN, J., AND LUEBKE, D. All-frequency relighting of non-diffuse objects using separable brdf approximation. *Eurographics Symposium on Rendering* (2004).

[43] WANG, R., ZHOU, K., SNYDER, J., LIU, X., BAO, H., PENG, Q., AND GUO, B. Variational sphere set approximation for solid objects. *Vis. Comput. 22*, 9 (2006), 612–621.

[44] ZHOU, K., HU, Y., LIN, S., GUO, B., AND SHUM, H.-Y. Precomputed shadow fields for dynamic scenes. *ACM Trans. Gr. 24*, 3 (2005), 1196–1201.

[45] ZHUKOV, S., INOES, A., AND KRONIN, G. An ambient light illumination model. In *Rendering Techniques '98* (1998), G. Drettakis and N. Max, Eds., Eurographics, Springer-Verlag Wien New York, pp. 45–56.