BSSRDF Explorer: A rendering framework for the BSSRDF

Benedikt Bitterli

Bachelor Thesis
April 2013

Supervisor:
Prof. Markus Gross

Advisors:
Dr. Wojciech Jarosz
Dr. Ralf Habel

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Disney Research

Abstract

This thesis introduces a novel system for the development, display and analysis of bidirectional surface scattering distribution functions (BSSRDFs). The system is capable of interactively rendering analytic BSSRDFs on arbitrary triangle meshes and pointclouds under different lighting conditions. It also facilitates analysis and validation of BSSRDFs by providing comparison plots both for analytic BSSRDFs and measured data in form of radial profiles. Additionally, the system provides reference solutions by the means of path tracing for simple geometries. The system is easily extensible with new analytic BSSRDFs through a simple text file interface. For the BSSRDF, our system is the first of its kind and is targeted both at researchers to assist in development and survey of BSSRDFs as well as artists to allow for interactive feedback and increased control in tweaking BSSRDF parameters. We integrated our system with the popular BRDF Explorer from Walt Disney Animation Studios and release it as an open source application to the computer graphics community.

Zusammenfassung

Diese Arbeit führt ein neues System für die Entwicklung, Darstellung and Analyse von *Bidirectional Surface Scattering Distribution Functions* (BSSRDFs) ein. Das System unterstützt die interaktive Darstellung von BSSRDFs auf beliebigen Dreiecksmodellen und Punktwolken unter verschiedenen Lichtverhältnissen. Es vereinfacht ausserdem die Analyse und Validierung von BSSRDFs durch das Bereitstellen von Vergleichsgraphen für analytische BSSRDFs und gemessene Daten in Form von radialen Profilen. Zusätzlich stellt das System Referenzlösungen auf einfachen Geometrien bereit, die mithilfe von Path Tracing berechnet werden. Das System kann leicht mit neuen analytischen BSSRDFs erweitert werden. Unser System ist das erste vergleichbare System für die BSSRDF und richtet sich sowohl an Forscher, um die Entwicklung und Studie von BSSRDFs zu unterstützen, als auch an Künstler, um das Finden von passenden BSSRDF-Parametern durch interaktives Feedback zu vereinfachen. Unser System basiert auf dem bekannten BRDF Explorer von Walt Disney Animation Studios und wird als Open Source Anwendung veröffentlicht.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Disney Research

Computer Graphics Laboratory ETH Zurich
Prof. Markus Gross

Bachelor's Thesis

BSSRDF Editing in BRDF Explorer

Introduction

The goal of this bachelor thesis is to implement GPU based BSSRDF (Bidirectional Subsurface Scattering Reflection Distribution Function) editing and display capabilities into the open source BRDF Explorer application from Walt Disney Animation Studios. This will allow the assessment of established and novel BSSRDF models and the editing of BSSRDF parameters in a feature production environment.

To achieve this goal, the existing BRDF Explorer application needs to be extended and an interface to GLSL BSSRDFs is to be defined within the BRDF Explorer framework. All standard BSSRDF models such as Dipole and Quantized Diffusion under different boundary conditions are then implemented in this interface.

Tasks

- Orientation, review of related work.
- Reviewing BRDF Explorer code base
- Definition of BSSRDF interface in BRDF Explorer
  - Assessment of effectivity and generality of interface
- Implementation of UI plots of BSSRDF values
- Implementation of Dipole and Quantized Diffusion

  With these tasks fulfilled, the grade is 4.00

- Implementing ray tracing capabilities into BRDF Explorer viewport

  With these tasks fulfilled, the grade is 5.00

- Implementation of a simple GPU based volumetric path tracer for BSSRDF reference.
- Implementation of rapid-hierarchical subsurface scattering on the GPU .

  With these tasks fulfilled, the grade is 6.00

Dates

Start Date: Thursday, November 1, 2012 - Thursday, April 30, 2012

Acknowledgements

I would like to express my gratitude to my advisors, Wojciech Jarosz and Ralf Habel, for devoting their precious time to my questions and their guidance throughout this project. Furthermore, I would like to thank Brent Burley for his feedback and support and enabling this thesis in the first place. Thanks also go to Marios Papas for his interest and insight. I am also deeply grateful to my family for their unconditional support and encouragement.

Contents

List of Figures

List of Algorithms

1

Introduction

Realistic rendering of translucent materials poses a challenging problem in computer graphics. Many materials important for realistic image synthesis, such as plants and fruits, beverages and food, many liquids and even our own skin, exhibit strong subsurface scattering effects. Traditionally, these effects have been approximated using models describing purely reflective light transport, commonly summarized in bidirectional reflectance distribution functions (BRDFs). However, the BRDF only describes scattering at a single point and fails to model light transport within the medium between different points on the surface. To properly account for subsurface light transport, a generalization of the BRDF, the bidirectional surface scattering distribution function (BSSRDF), can be used. The BSSRDF describes light transport between any two rays that hit a surface, enabling the incorporation of subsurface light transport.

Recent developments, such as the introduction of diffusion theory [Sta95] and the diffusion dipole [JMLH01] to computer graphics, have laid the foundation for many practical subsurface light tranport models and spawned numerous novel BSSRDFs. Sophisticated rendering techniques and increase in computational power have also made it feasible to incorporate subsurface scattering effects into movie production, sparking interest in finding and developing BSSRDFs optimal in artist workflow and visual quality.

At this point, a wealth of different BRDF and BSSRDF models exist. For the BRDF, several systems [Rus01, FPBP09, Stu11] address the problem of development, analysis and survey of analytic and measured models. Additionally, some of them [FPBP09, Stu11] provide GPU based rendering capabilities, allowing interactive visual feedback for a wide range of parameter settings and lighting conditions.

So far, no such system exists for the BSSRDF. In addition, the BSSRDF is inherently more expensive to render than the BRDF, making the implementation of accurate interactive rendering methods difficult. However, fast visual feedback when editing the parameters of a BRDF or

BSSRDF is vital to the workflow of an artist.

The contributions of this thesis are as follows:

- Introduction of an extensible system supporting the development, analysis and validation of new and existing BSSRDFs
- Implementation of BSSRDFs widely used in research and production in our system
- GPU implementations of state-of-the-art BSSRDF rendering methods, allowing interactive display and interaction with BSSRDFs

A table of notation used throughout this paper is given in table 1.1.

1.1. Related Work

Several rendering algorithms, such as path tracing [Kaj86] or photon mapping [Jen01] address the problem of numerically approximating volumetric scattering effects. Although these methods are general in that they can handle arbitrary illumination, geometry and materials in an unbiased manner, they are typically computationally expensive and may take long to converge, making them infeasible for practical rendering.

The diffusion approximation introduced to computer graphics by Stam [Sta95] approximates multiple scattering using a diffusion equation and forms the basis for many successful BSSRDFs. Building on work from classical diffusion theory [FPW92], Jensen et al. [JMLH01] introduce a practical diffusion dipole model. Combined with a single scattering term based on work by Hanrahan and Krueger [HK93], they build a BSSRDF and apply it to the rendering of translucent materials. Donner and Jensen [DJ05] extend this work with a multipole and use it to render multi-layered materials. d'Eon and Irving [DI11], among a wealth of improvements upon classic diffusion theory, also introduce a new diffuse multiple scattering term built from a sum of Gaussians.

Several methods attempt to render subsurface scattering effects interactively. d'Eon et al. [dL07] and Jimenez et al. [JWSG10] both approximate subsurface scattering by approximating the diffusion profile by a sum of Gaussians and exploit separability to blur the illumination in texture space. Subsequent work by Jimenez and Gutierrez [JG10] is based on the same observation, but performs the blur in screen-space, increasing performance at the cost of quality. While these techniques work in real-time, they focus on skin rendering and don't allow for general BSSRDFs. Jensen and Buhler [JB02] introduce a two-pass rendering technique to significantly improve performance when evaluating the BSSRDF on general geometry. Although originally intended for use with the diffusion dipole for offline rendering, it is also applicable to general BSSRDFs and is suitable for interactive GPU rendering.

Multiple applications focusing on visualization, analysis and manipulation of BRDFs exist. bv [Rus01] features several analytic BRDFs in lit-sphere and goniometric views. BRDFLab [FPBP09] allows visualization of BRDFs on different geometry and lighting situations and features parameter fitting and BRDF synthesis. The BRDF Explorer by Walt Disney Animation Studios [Stu11] provides an extensible, GPU based system for BRDFs under different lighting

conditions and arbitrary geometry as well as analysis facilities. Our system is based on the open source BRDF Explorer and uses its extensible GPU framework to allow dynamic loading and editing of analytic BSSRDFs.

Symbol	Description
$\vec{x}_i$	Surface location of incident light
$\vec{\omega}_i$	Incident direction
$\vec{x}_o$	Surface location of exitant light
$\vec{\omega}_o$	Exitant direction
$\vec{n}$	Surface normal
f_r	BRDF
f_t	BTDF
S	BSSRDF
$S^{(0)}$	Reduced radiance
$S^{(1)}$	Single scattered radiance
S_d	Multiple scattered radiance
L	Radiance
L_r	Reflected radiance
L_i	Incident radiance
E_i	Irradiance
Φ_i	Incident flux
D	Diffusion constant
ϕ	Fluence
$\vec{E}$	Flux
p	Phase function
g	Mean cosine of scattering angle
σ_a	Absorption coefficient
σ_s	Scattering coefficient
σ'_s	Reduced scattering coefficient: $(1-g)\sigma_s$
σ_t	Extinction coefficient: $\sigma_s + \sigma_a$
σ'_t	Reduced extinction coefficient: $\sigma'_s + \sigma_a$
α	Albedo: σ_s/σ_t
α'	Reduced albedo: σ'_s/σ_t
σ_{tr}	Effective transport coefficient: $\sqrt{\sigma_a/D}$
Q	Source function
η	Relative index of refraction
F_r	Fresnel reflectance
F_t	Fresnel transmittance
F_{dr}	Average Fresnel reflectance
l	Mean free path (mfp): $1/\sigma_t$
l'	Reduced mean free path: $1/\sigma'_t$
l_d	Diffuse mean free path (dmfp): $1/\sigma_{tr}$
R_d	Diffuse reflectance
Ω_+	Hemisphere of outward directions
Ω_-	Hemisphere of inward directions
Ω	Domain of integration

Table 1.1.: *Table of notation*

2

Background

Fundamental to realistic image synthesis is the concept of light transport. It describes the behaviour of light as it is emitted from light sources, interacts with the scene and ultimately arrives at the camera, producing the rendered image. Accurately modeling the light transport is key for synthesizing realistic images. Unfortunately, the physical behaviour of light is complex and not completely understood, necessitating simplifying assumptions to allow for efficient simulation.

2.1. Surface Scattering

Most light phenomena can be described accurately using the model of geometric optics. It is based on the assumption that light propagates with infinite speed along straight lines (rays). Light rays may only change direction at a discrete set of scattering locations, at which their respective energies may be partially absorbed and the remaining fraction scattered into new directions. Importantly, it is assumed that radiance along a ray is constant; the medium through which the ray travels does not influence the light transport.

For most applications, it is a reasonable approximation to restrict the location of scattering events to only the surface of objects. In this model, rays of light travel without scattering until they strike the surface of an object, at which point they are either absorbed completely or scatter and leave the surface at the same location in new directions. This gives rise to the notion of the bidirectional reflectance distribution function (BRDF) introduced by Nicodemus [NRH$^+$77], which, for a ray coming from direction $\vec{\omega}_i$ striking a surface at location x_i, relates the incoming radiance L_i to the reflected radiance L_r:

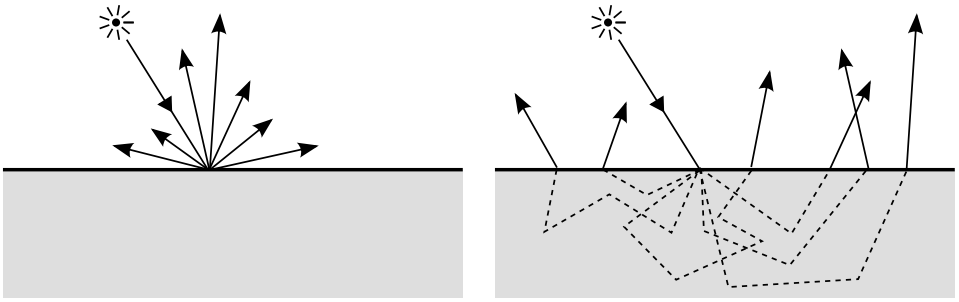


Figure 2.1.: *Light scattering model for (a) BRDF and (b) BSSRDF. Figure reproduced from Jensen et al. [JMLH01]*

$$f_r(\vec{x}, \vec{\omega}_i, \vec{\omega}_o) = \frac{\mathrm{d}L_r(\vec{x}, \vec{\omega}_o)}{\mathrm{d}E_i(\vec{x}, \vec{\omega}_i)} = \frac{\mathrm{d}L_r(\vec{x}, \vec{\omega}_o)}{L_i(\vec{x}, \vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n})\mathrm{d}\vec{\omega}_i}, \tag{2.1}$$

where $\vec{n}$ is the surface normal.

If the incident radiance field L_i is known for all directions, the total reflected radiance L_r in direction $\vec{\omega}_o$ can be obtained by integration:

$$L_r(\vec{x}, \vec{\omega}_o) = \int_\Omega f_r(\vec{x}, \vec{\omega}_i, \vec{\omega}_o)\mathrm{d}E_i(\vec{x}, \vec{\omega}_i) = \int_\Omega f_r(\vec{x}, \vec{\omega}_i, \vec{\omega}_o)L_i(\vec{x}, \vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n})\mathrm{d}\vec{\omega}_i \tag{2.2}$$

Here, Ω represents the hemisphere of incident directions at location $\vec{x}$.

However, for the description of volumetric scattering, these approximations are not sufficient. Several problems arise when dealing with materials that exhibit volumetric scattering - most importantly, the assumption of constant radiance along a ray does no longer hold. The medium in which the ray travels now *does* influence the light transport; it is thus commonly referred to as a *participating* medium. In addition, scattering events happen not only on the surface of objects, but volumetrically within the medium.

2.2. Volumetric Scattering

To properly predict the behaviour of light as it moves through a participating medium, we need an extended model accounting for volumetric scattering. In reality, volumetric scattering is caused by particles suspended in the medium (air molecules, dust, pigments, microorganisms etc.). Light travelling through the medium hits these particles and interacts with them on a microscopic level.

Actually modelling the particles and individual interactions with light is impractical and unnecessary for most rendering applications. However, if we assume the particles to be microscopic (much smaller than the spacing between particles) and randomly positioned, we can instead only model the *probabilistic* behaviour of light moving through the medium, aggregated over many interactions with particles.

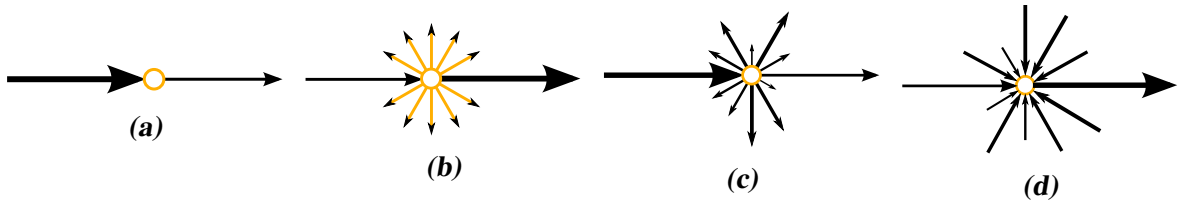


Figure 2.2.: *The behaviour of light in a participating medium is characterized by four different interactions: (a) absorption, (b) emission, (c) outscattering and (d) inscattering. Figure reproduced from Jarosz [Jar08]*

Specifically, we now consider the radiance along an infinitesimal ray of light moving through the medium by describing the change in radiance after an infinitesimal step along the ray. The change is characterized by the current position along the ray $\vec{x}$, the direction of the ray $\vec{\omega}$ and the radiance $L(\vec{x}, \vec{\omega})$. Since we are assuming infinitesimal steps, we consider the differential change $(\vec{\omega} \cdot \nabla)L(\vec{x}, \vec{\omega})$.

For each such infinitesimal step, we model four possible interactions with the medium, illustrated in Figure 2.2:

Absorption Part of the radiance is absorbed by the medium. Since all interactions conserve energy, the radiance doesn't vanish; instead, it is converted into different forms of energy, such as heat. For our purposes, we can simply consider the radiance to have vanished with absorption, since other forms of energy are invisible to the human eye. The absorbed fraction of the radiance is described by the absorption coefficient σ_a. Putting it in differential form describes the change in radiance as:

$$(\vec{\omega} \cdot \nabla)L(\vec{x}, \vec{\omega}) = -\sigma_a L(\vec{x}, \vec{\omega}) \tag{2.3}$$

Emission In some cases, the medium itself may produce light. Chemical reactions, found in burning gases for example, may cause volumetric emission within the medium; in fact, the sunlight that we see daily is produced by self-emission within the sun, caused by fission reaction. Although not necessarily physically plausible, volumetric emission is also of interest in computer animation, leaving many possibilities to the artist for visual effects. In general, we can model emission within the medium using a source term $Q(\vec{x}, \vec{\omega})$, dependent on location and direction. In differential form:

$$(\vec{\omega} \cdot \nabla)L(\vec{x}, \vec{\omega}) = Q(\vec{x}, \vec{\omega}) \tag{2.4}$$

Outscattering At each step, the radiance may also be reduced due to light being scattered into other directions than $\vec{\omega}$. The fraction of radiance being scattered is described by the scattering coefficient σ_s. Similar to the absorption, we can formulate it in differential form as:

$$(\vec{\omega} \cdot \nabla)L(\vec{x}, \vec{\omega}) = -\sigma_s L(\vec{x}, \vec{\omega}) \tag{2.5}$$

Inscattering Finally, we also have to take into account the radiance arriving at $\vec{x}$ from other directions and being scattered towards direction $\vec{\omega}$, increasing the radiance along the ray. Again, we use the scattering coefficient σ_s to describe the fraction of radiance being scattered. Additionally, capturing all incident directions means integrating the incident radiance over the whole sphere of directions Ω:

$$(\vec{\omega} \cdot \nabla) L(\vec{x}, \vec{\omega}) = \int_{\Omega} p(\vec{\omega}', \vec{\omega}) L(\vec{x}, \vec{\omega}') \mathrm{d}\vec{\omega}' \tag{2.6}$$

Here, we used the *phase function* p. The phase function describes the angular distribution of light intensity being scattered, similar to the BRDF. For isotropic scattering, light scatters uniformly in all directions and the phase function is constant. Most real materials exhibit dominant scattering directions however, leading to anisotropic scattering. A measure of the scattering anisotropy is the *mean cosine* g, computed as:

$$g = \int_{\Omega} (\vec{\omega} \cdot \vec{\omega}') p(\vec{\omega}', \vec{\omega}) \mathrm{d}\vec{\omega}' \tag{2.7}$$

If g is negative, the phase function is predominantly backward scattering; vice-versa, positive g describes dominant forward scattering. Finally, $g = 0$ stands for isotropic scattering. Usually, p only depends on the phase angle, $p(\vec{\omega}', \vec{\omega}) = p(\vec{\omega}' \cdot \vec{\omega})$ and is normalized, such that:

$$\int_{\Omega} p(\vec{\omega} \cdot \vec{\omega}') \mathrm{d}\vec{\omega}' = 1 \tag{2.8}$$

Combining the four terms yields the *radiative transfer equation* (RTE):

$$\begin{aligned} (\vec{\omega} \cdot \vec{\nabla}) L(\vec{x}, \vec{\omega}) = & -(\sigma_a + \sigma_s) L(\vec{x}, \vec{\omega}) && (2.9) \\ & + Q(\vec{x}, \vec{\omega}) && (2.10) \\ & + \sigma_s \int_{\Omega} L(\vec{x}, \vec{\omega}) \rho(\vec{\omega}, \vec{\omega}') \mathrm{d}\vec{\omega}' && (2.11) \end{aligned}$$

(2.9) is commonly reformulated in terms of the extinction coefficient $\sigma_t = \sigma_a + \sigma_s$.

The solution of the RTE provides the radiance $L(\vec{x}, \vec{\omega})$ at each location and direction. However, analytical solutions of the RTE are prohibitively rare for general domains and scattering materials. Numerical solutions can be obtained using finite element methods (FEM) or Monte Carlo integration, but are typically computationally expensive.

For the case of subsurface scattering, normally only the radiant exitance on the surface is important for rendering, whereas radiance within the medium is of little importance. This makes it convenient to reformulate the problem in terms of a bidirectional surface scattering reflection distribution function (BSSRDF), which, for any two rays hitting the surface of the medium, relates the incident flux Φ_i at surface location $\vec{x}_i$ coming from direction $\vec{w}_i$ to the reflected radiance L_r at surface location $\vec{x}_o$ in direction $\vec{w}_o$:

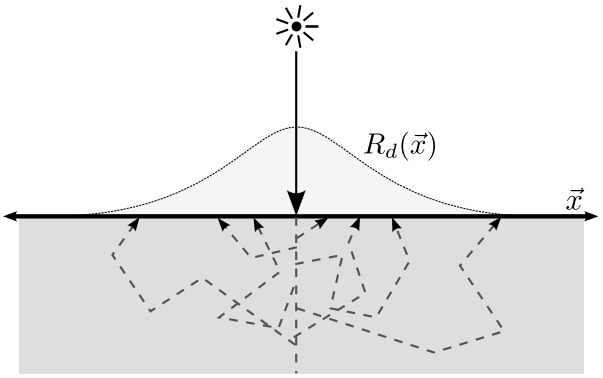


Figure 2.3.: *Outline of the searchlight problem*

$$S(\vec{x}_i, \vec{\omega}_i; \vec{x}_o, \vec{\omega}_o) = \frac{\mathrm{d}L_r(\vec{x}_o, \vec{\omega}_o)}{\mathrm{d}\Phi_i(\vec{x}_i, \vec{\omega}_i)} \tag{2.12}$$

Usually, S is further decomposed into three terms: The reduced radiance term $S^{(0)}$, representing unscattered radiance only affected by extinction; a single scattering term $S^{(1)}$, formed by radiance scattered exactly once; and a multiple scattering term S_d of radiance scattered more than once:

$$S = S^{(0)} + S^{(1)} + S_d \tag{2.13}$$

This splitting is beneficial, since specialized algorithms exist for the individual terms.

In the same spirit as Equation (2.2), we can now obtain the total reflected radiance L_r at surface point $\vec{x}_o$ in direction $\vec{\omega}_o$ due to the BSSRDF through integration:

$$L_r(\vec{x}_o, \vec{\omega}_o) = \int_A \int_\Omega S(\vec{x}_i, \vec{\omega}_i; \vec{x}_o, \vec{\omega}_o) L_i(\vec{x}_i, \vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n}) \mathrm{d}\vec{\omega}_i \mathrm{d}A(\vec{x}_i) \tag{2.14}$$

Note that, unlike the BRDF, we now have to integrate the product of the BSSRDF and the incident radiance for all incident directions *and surface locations*. This makes evaluation of L_r much more complex and numerically expensive compared to the BRDF. Intrinsically, the BSSRDF spans eight dimensions (four spatial, four angular), whereas the BRDF is only six dimensional.

2.3. The Searchlight Problem

Exact BSSRDFs follow trivially from exact solutions to the RTE. Unfortunately, closed form solutions of the RTE for general 3D geometry, illumination and materials are rare, which motivates analysis on simplified, lower dimensional domains. Particularly useful are settings that produce symmetries in the radiance, collapsing the solution space and allowing for detailed investigation.

A problem setting often considered in computer graphics is known as the *searchlight problem*, originally posed in astrophysics [Cha58]. It consists of a focused, infinitesimally thin pencil beam in the origin striking the surface of a half-infinite slab at normal incident. Photons originating in the beam travel along the refracted ray until they are scattered by the medium, leading to a series of scattering events until they are ultimately absorbed or escape through the surface. The setting is illustrated in Figure 2.3.

The distribution of photons exiting through the surface gives rise to a radially symmetric reflectance distribution profile $R_d(\vec{x}) = R_d(||\vec{x}||)$, called the *reduced reflectance*. Note that R_d does *not* depend on the exitant direction $\vec{\omega}_o$, since it represents an accumulation of photons from all exitant directions Ω:

$$R_d(\vec{x}) = \int_\Omega L_r(\vec{x}, \vec{\omega}') f_t(\vec{x}, \vec{\omega}') \mathrm{d}\vec{\omega}' \tag{2.15}$$

Here, f_t is the bidirectional transmittance distribution function (BTDF).

Although normally S_d cannot be constructed exactly from R_d, many methods in graphics approximate S_d using the reflectance distribution profile and a directionally dependent Fresnel transmission term:

$$S_d(\vec{x}_i, \vec{\omega}_i; \vec{x}_o, \vec{\omega}_o) = \frac{1}{\pi} F_t(\vec{x}_i, \vec{\omega}_i) R_d(\vec{x}_o - \vec{x}_i) \frac{F_t(\vec{x}_o, \vec{\omega}_o)}{4C_\phi(1/\eta)} \tag{2.16}$$

Here, F_t represents the Fresnel transmission term, and $4C_\phi$ is an approximate normalization factor.

Equation (2.16) builds the basis for most successful BSSRDF models. The reduced radiance and single scattering terms $S^{(0)}$ and $S^{(1)}$ are usually cheap to compute using standard Monte Carlo methods [JMLH01] compared to the complex diffuse scattering term S_d. By approximating S_d with (2.16), the only remaining problem is to obtain R_d.

Unfortunately, even for the simplified 2D searchlight problem, exact closed form expressions for R_d are not readily available. Of course, R_d can be computed numerically using Monte Carlo integration, but at high computational cost. This motivates finding closed-form approximations for R_d.

2.4. Diffusion Theory

One of the most successful frameworks for approximating the diffuse reflectance profile R_d is the diffusion approximation. It is based on the observation that in highly scattering media, the light distribution tends to become isotropic even if the incident light and phase function are anisotropic. Intuitively speaking, each scattering event "blurs" the light distribution and it becomes more and more uniform as more scattering events are considered.

In this setting, a useful simplification is to only consider angular integrals (nth moments) of the radiance. The first two moments of the radiance are denoted as fluence ϕ and flux $\vec{E}$:

$$\phi(\vec{x}) = \int_{\Omega} L(\vec{x}, \vec{\omega}) \mathrm{d}\vec{\omega} \tag{2.17}$$

$$\vec{E}(\vec{x}) = \int_{\Omega} L(\vec{x}, \vec{\omega}) \vec{\omega} \mathrm{d}\vec{\omega} \tag{2.18}$$

Here, Ω is the sphere of directions.

To arrive at the diffusion approximaton, the radiance is first expanded in spherical harmonics. The expansion is then truncated at 1st order and renormalized to conserve energy. This gives an approximation of the radiance in terms of ϕ and $\vec{E}$ [WWW58]:

$$L(\vec{x}, \vec{\omega}) \approx \frac{1}{4\pi}\phi(\vec{x}) + \frac{3}{4\pi}\vec{E}(\vec{x}) \cdot \vec{\omega} \tag{2.19}$$

By substituting this term into the radiative transport equation and integrating over all directions, we can derive the classic diffusion equation:

$$-D\nabla^2\phi(\vec{x}) + \sigma_a\phi(\vec{x}) = Q(\vec{x}) \tag{2.20}$$

Where Q is a (scalar) isotropic source term.

Additionally, we can formulate the diffuse reflectance profile R_d in terms of the fluence ϕ. R_d is defined as the radiant exitance divided by the incident flux. The radiant exitance is $\vec{n} \cdot \vec{E}(x)$ and, using Fick's law ($\vec{E}(\vec{x}) = -D\vec{\nabla}\phi(\vec{x})$), can be expressed in terms of the fluence:

$$R_d(||x_o - x_i||) = -D\frac{(\vec{n} \cdot \vec{\nabla}\phi)(\vec{x_o})}{\mathrm{d}\Phi_i(\vec{x}_i)} \tag{2.21}$$

Here, D is the diffusion constant:

$$D = \frac{1}{3\sigma_t'} \tag{2.22}$$

This equation makes use of the reduced extinction coefficient, σ_t'. For isotropic scattering ($g = 0$), the reduced extinction coefficient is equal to the extinction coefficient σ_t. For anisotropic scattering ($g \neq 0$), the reduced extinction coefficient is:

$$\sigma_t' = \sigma_s' + \sigma_a \tag{2.23}$$

$$\sigma_s' = \sigma_s(1 - g) \tag{2.24}$$

Essentially, a similarity relation is used to reduce a problem with anisotropic scattering to an approximately equivalent, but much simpler problem with isotropic scattering. The new problem uses modified reduced scattering and transport coefficients.

For an infinite medium with a unit power, isotropic point light source, the diffusion equation has a simple solution, the diffusion Green's function:

$$\phi(\vec{x}) = \frac{1}{4\pi D}\frac{e^{-\sigma_{tr} r(\vec{x})}}{r(\vec{x})}, \tag{2.25}$$

where r is the distance to the location of the light source and $\sigma_{tr} = \sqrt{\sigma_a / D}$ is the effective transport coefficient.

If the medium is finite, the diffusion equation has to be solved subject to boundary conditions at the surface. One important observation is that the incoming radiance at the surface of the medium can be treated as a volumetric source inside the medium by considering all first-order scattering events:

$$Q(\vec{x}, \vec{\omega}) = \sigma_s \int_\Omega p(\vec{\omega}', \vec{\omega}) L_{rr}(\vec{x}, \vec{\omega}') \mathrm{d}\vec{\omega}' \tag{2.26}$$

Here, L_{rr} is the reduced radiance. For an infinitesimal ray entering the medium:

$$L_{rr}(\vec{x}_i + s\vec{\omega}_i, \vec{\omega}_i) = e^{-\sigma_t s} L_i(\vec{x}_i, \vec{\omega}_i) \tag{2.27}$$

By capturing the incident radiance in the source term Q, we can easily formulate an appropriate boundary condition by setting the inward flux to zero on each point $\vec{x}_s$ on the surface:

$$\int_{\Omega_-} L(\vec{x}_s, \vec{\omega})(\vec{\omega} \cdot \vec{n}(\vec{x}_s)) \mathrm{d}\vec{w} = 0 \tag{2.28}$$

Where $\vec{n}(\vec{x}_s)$ is the surface normal at $\vec{x}_s$ and Ω_- denotes the hemisphere of inward directions.

For media with non-matching refractive indices, the boundary condition must also respect the reflection at the interface. If F_r is the Fresnel reflectance, we can compute the average diffuse Fresnel reflectance F_{dr} as:

$$F_{dr} = \int_{\Omega_+} F_r(\eta, \vec{n} \cdot \vec{\omega}') \mathrm{d}\vec{\omega}' \tag{2.29}$$

While F_{dr} can be computed analytically [Kor69], typically an approximate rational term [EH79] is used instead:

$$F_{dr} = -\frac{1.440}{\eta^2} + \frac{0.710}{\eta} + 0.668 + 0.0636\eta \tag{2.30}$$

The resulting boundary condition is then:

$$-\int_{\Omega-} L(\vec{x}_s, \vec{\omega})(\vec{\omega} \cdot \vec{n}(\vec{x}_s)) \mathrm{d}\vec{w} = F_{dr} \int_{\Omega+} L(\vec{x}_s, \vec{\omega})(\vec{\omega} \cdot \vec{n}(\vec{x}_s)) \mathrm{d}\vec{w} \tag{2.31}$$

The negative sign on the left hand side stems from the convention that the surface normal points outward, whereas the integral is over inward directions.

Using the two-term expansion, the boundary condition becomes:

$$\phi(\vec{x}_s) - 2AD(\vec{n} \cdot \vec{\nabla})\phi(\vec{x}_s) = 0 \tag{2.32}$$

2.5. The Diffusion Dipole

Several methods for solving diffusion boundary conditions are available. For the searchlight problem, one particularly popular choice is the method of images [Bry91]. The method of images casts a new, analogue problem, where the domain of the solution function is extended by its negative mirror image with respect to a mirror plane. This extended function is zero on the mirror plane.

Moulton et al. [MoEP90] show that the boundary condition given in (2.32) can be satisfied by setting the fluence ϕ to zero on an *extrapolated boundary* at a height $z_b = 2AD$ above the surface. That is, given a solution ϕ' for the fluence in an infinite medium, we can easily produce a solution for a semi-infinite medium satisfying the boundary condition using the method of images, simply by adding a negated image of ϕ' mirrored about the mirror plane at height z_b.

Recalling that for a unit power point light in an infinite medium, the fluence is given by the diffusion Green's function (2.25), we can now derive the fluence for a semi-infinite medium. If the point light source is at a distance z_r below the surface, we place the image source at a distance $z_v = z_r + 4AD$ above the surface. The resulting *dipole fluence* $\phi(\vec{x})$ is then:

$$\phi(\vec{x}) = \frac{1}{4\pi D}\left(\frac{e^{-\sigma_{tr}d_r}}{d_r} - \frac{e^{-\sigma_{tr}d_v}}{d_v}\right) \tag{2.33}$$

Here, $d_v = ||\vec{x} - \vec{x_v}||$ and $d_r = ||\vec{x} - \vec{x_r}||$ is the distance of $\vec{x}$ to the image source and the point light source, respectively. The dipole setup is illustrated in Figure 2.4.

To model an arbitrary incident light distribution with (2.33), a point light representation of all first scatter events is required. In the most general case, this is a convolution of the diffusion Green's function and Q. Even for the simple case of the 2D searchlight problem, where Q is only nonzero along a ray in the origin,

$$Q(z) = \sigma'_s e^{-\sigma'_t z}, \tag{2.34}$$

this results in an integral with no closed form solution.

Farell et al. [FPW92] and Jensen et al. [JMLH01] instead approximate the incident source distribution with a *single* isotropic point light source. It is placed at the average first scatter depth, the reduced mean free path l':

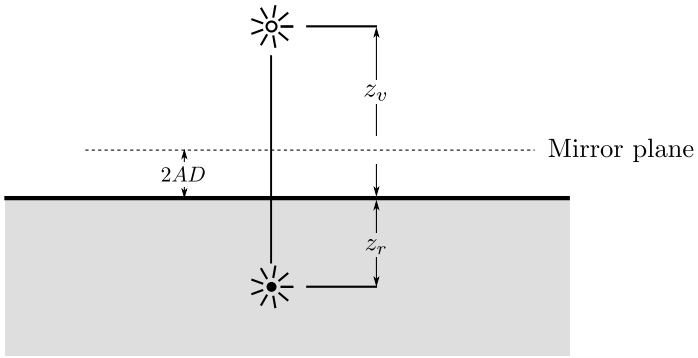


Figure 2.4.: *The dipole setup*

$$l' = \int_0^\infty x\sigma_t' e^{-\sigma_t' x} \mathrm{d}x = \frac{1}{\sigma_t'} \tag{2.35}$$

below the surface, with intensity equal to the average intensity of the refracted ray, formulated in terms of the reduced albedo α':

$$\sigma_s' \int_0^\infty L_{rr}(x)\mathrm{d}x = \sigma_s' \int_0^\infty L_i e^{-\sigma_t' x}\mathrm{d}x = \frac{\sigma_s'}{\sigma_t'} L_i = \alpha' L_i \tag{2.36}$$

Finally, substituting the resulting dipole fluence into (2.21) yields a closed form approximation for the diffuse reflectance:

$$R_d(||x_o - x_i||) = \frac{\alpha'}{4\pi}\left((\sigma_{tr}d_r + 1)\frac{e^{-\sigma_{tr}d_r}}{\sigma_t' d_r^3} + z_v(\sigma_{tr}d_v + 1)\frac{e^{-\sigma_{tr}d_v}}{\sigma_t' d_v^3}\right) \tag{2.37}$$

Using (2.37) in the Fresnel reshaping term (2.16) results in the diffusion dipole BSSRDF model, which has found successful adaption in computer graphics applications.

2.6. Rendering with the BSSRDF

Provided with a BSSRDF such as the diffusion dipole, we now stand before the task of efficiently rendering it on arbitrary geometry. Given the complexity of the double integral (2.14) necessary to compute the radiant exitance, it is apparent that this task is non-trivial.

In the following, we consider the three subterms of the BSSRDF (reduced radiance, single scattering and multiple scattering) separately, since they show different distributions and benefit from specialized evaluation strategies.

2.6.1. Reduced Radiance

Computing the reduced radiance exactly is fairly straightforward. It is only nonzero if the exitant position lies on the incident ray, in which case it is:

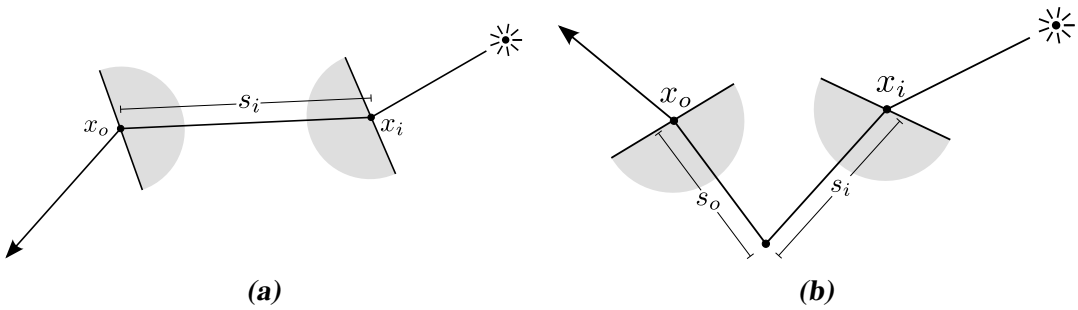


Figure 2.5.: *Evaluation of (a) reduced radiance and (b) single scattering*

$$L_o^{(0)} = f_t(\vec{x_o}, \vec{\omega_o}) f_t(\vec{x_i}, \vec{\omega_i}) e^{-s_i \sigma_t} L_i(\vec{x_i}, \vec{\omega_i}) \tag{2.38}$$

Since the reduced radiance is formed only by photons that did not scatter in the medium, the radiance is only affected by surface transmission and extinction inside the medium. s_i denotes the distance travelled along the ray inside the medium, in this case $s_i = ||x_o - x_i||$. The geometry of the problem is illustrated in Figure 2.5 (a).

For optically thick media, the contribution of the reduced radiance is insignificant, and the term is usually omitted in practical applications.

2.6.2. Single Scattering

Evaluation of the single scattering term is commonly performed using Monte Carlo integration with importance sampling. Since the integrand is strongly peaked, the Monte Carlo method converges quickly.

Constructing a Monte Carlo sample of the single scattering term consists of two steps. First, the location of the scattering event is fixed by choosing a distance s_o along the refracted ray. The distance a ray travels before interacting with the material is described by the PDF $\sigma_t' e^{-\sigma_t' x}$. Given a uniformly distributed random number $\xi \in]0, 1]$, the distance on the ray is given by importance sampling this PDF, $s_o = -log(\xi)/\sigma_t$.

Then, an incident direction $\vec{\omega_i}$ has to be chosen by importance sampling the phase function p. If $g = 0$, the phase function is isotropic and the incident direction can be chosen uniformly. Finally, the intersection $\vec{x_i}$ of the incident ray starting at the scattering location with the surface is found and the outscattered radiance computed as follows:

$$L_o^{(1)}(\vec{x_o}, \vec{\omega_o}) = f_t(\vec{x_o}, \vec{\omega_o}) f_t(\vec{x_i}, \vec{\omega_i}) e^{-s_i \sigma_t} e^{-s_o \sigma_t} \alpha p(\vec{\omega_i} \cdot \vec{\omega_o}) L_i(\vec{x_i}, \vec{\omega_i}) \tag{2.39}$$

The equation computes the outscattered radiance from the irradiance as the fraction of radiance transmitted through the surface, subject to extinction along the incident- and exitant ray as well as partial scattering at the single scattering location. Here, s_i is the distance from scattering location to incident position, $s_i = ||\vec{x_i} - (\vec{x_o} - s_o \vec{\omega_o})||$. This is illustrated in Figure 2.5 (b).

If the illumination is not uniform, importance sampling only the phase function might lead to

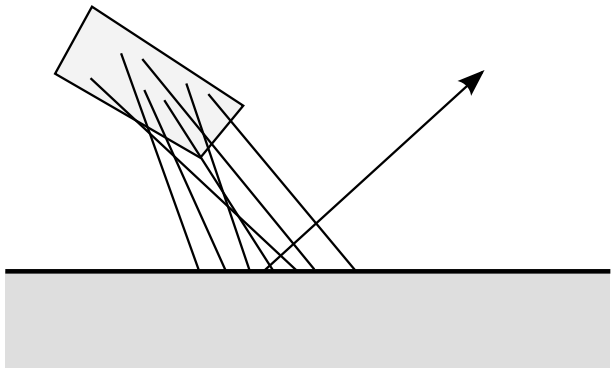


Figure 2.6.: *Monte Carlo integration of the multiple scattering term requires sampling both surface area and light sources. Figure reproduced from [JMLH01]*

high variance. This is especially true if the incident light distribution consists of a delta function (point lights or directional lights), in which case the incident rays will never hit the light source.

If no refraction occurs at the interface, standard multiple importance sampling techniques may be used to combine sampling strategies for the phase function and the illumination. However, for refractive media, choosing incident directions such that the incident rays optimally sample the illumination *after refracting at the surface* is difficult for arbitrary geometry. Walter et al. [WZHB09] introduce an accurate method to find all paths connecting two points inside and outside a refractive medium bounded by a triangular mesh and apply it to single scattering. Although efficient, it is unsuitable for interactive rendering. Jensen et al. [JMLH01] instead use an approximative solution by not refracting the incident ray at the surface. To account for refraction, the distance s_i along the incident ray is adjusted to approximate the true refracted distance. Although this method is unable to produce single scattering effects such as volumetric caustics, it is much faster to compute than the exact solution.

2.6.3. Monte Carlo Multiple Scattering Evaluation

Similar to the single scattering term $S^{(1)}$, it is possible to evaluate the multi scattering term S_d with Monte Carlo integration. Unlike the single scattering term however, the integrand is not as strongly peaked and may have wide support over the surface for materials with low absorption. In addition, evaluating the double integral (2.14) with a Monte Carlo method requires sampling both surface area and light sources, illustrated in Figure 2.6.

Jensen et al. [JMLH01] exploit the exponential falloff with distance in the diffusion dipole and, for an evaluation point $\vec{x}_o$, sample the surface with density $\sigma_{tr}e^{-\sigma_{tr}d}$ at a distance d from $\vec{x}_o$. However, it is not clear how to uniformly pick sample points on arbitrary meshes at a distance d from a point on the surface. Additionally, for low absorption and large scattering coefficients, sample densities may be large even far away from the evaluation point, requiring large areas to be sampled at high computational cost.

Several image-space methods have been proposed [SKP09, JG10], but, while cheaper to compute, impose strong restrictions on the lighting and BSSRDF and only handle a subset of material parameters well.

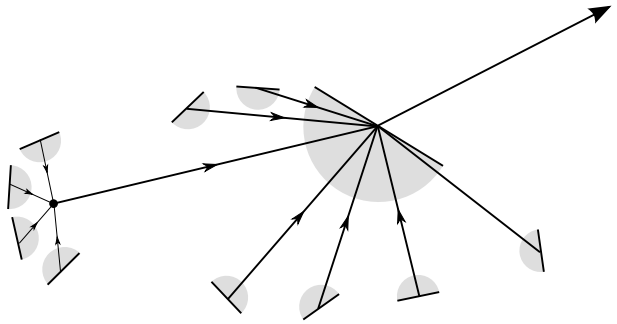


Figure 2.7.: *Hierarchical pointcloud evaluation uses irradiance values at precomputed surface locations and may cluster distant samples*

2.6.4. Hierarchical Multiple Scattering Evaluation

To mitigate the problems with Monte Carlo integration of the S_d term, Jensen and Buhler [JB02] introduce a two-pass technique for computing the contribution of the S_d term.

The central idea of the technique is to decouple the irradiance computation from the evaluation of the S_d term by caching the irradiance at a set of surface locations. This allows reuse of irradiance samples for S_d evaluations close to each other. Additionally, the choice of sample representation allows for the clustering of distant samples to greatly improve evaluation performance; this is illustrated in Figure 2.7.

Although originally developed for rendering of the diffusion dipole [JMLH01], the two-pass method is applicable to any multi scattering model S_d and is suitable for interactive rendering.

Generating Sample Locations

One important input of the algorithm is a set P of samples p on the surface, referred to as a point cloud. Each sample location has an associated normal $\vec{p_n}$ and surface area p_a. If A is the total surface area of the sampled geometry, $\sum_{p \in P} p_a = A$ must hold.

The sampling of the surface is not required to be uniform, which allows for optimizations on the point cloud such as increased sample density around discontinuities in geometry or irradiance and decreased density on flat, uniformly lit areas.

In practice, however, non-uniform sampling easily leads to artifacts. The main problem is the sensitivity of the second pass of the algorithm to inaccuracies in the computed sample area p_a. Even small inaccuracies lead to local under- or overestimation of the radiant exitance, manifesting as distracting low frequency noise in the rendered image. Although non-uniform sampling does not strictly lead to artifacts, the complexity of computing accurate sample areas, especially in transitions from high- to low sample density domains, makes it unattractive for rendering.

To produce accurate estimates for the sample area, Jensen and Buhler [JB02] use a uniform sampling of the surface. The sample area can then be computed trivially as the total surface area divided by the number of samples, $p_a = A/|P|$. To produce a uniform sample distribution,

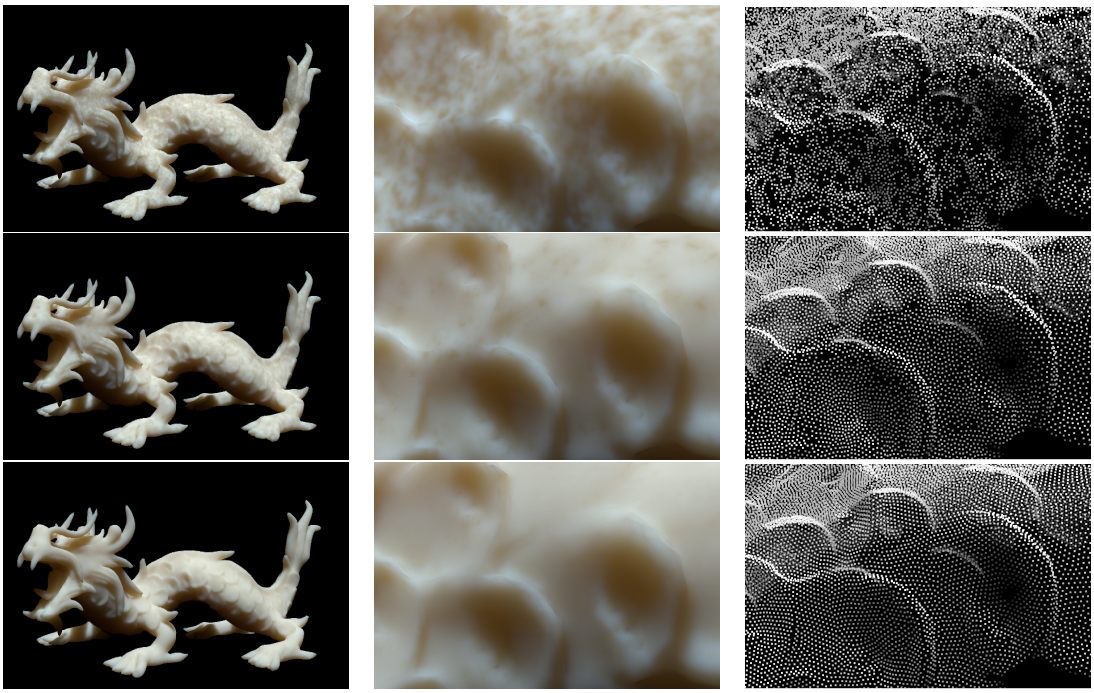


Figure 2.8.: *Results for 10 iterations (top), 100 iterations (middle) and 1000 iterations (bottom) of the point relaxation algorithm, rendered with the dipole BSSRDF (left, middle) and visualized directly (right). Model courtesy of XYZRGB.*

the point repulsion algorithm introduced by Turk [Tur92] is used.

The algorithm works as follows: Starting with an initial guess of uniformly random distributed samples on the surface, the algorithm performs iterative relaxation of the sample set by computing a repulsion force for each sample point based on the proximity to its neighbours. Each point is then displaced by the repulsion force. To ensure that each sample still resides on the surface after displacement, sample points that are pushed off the polygonal face they reside on are projected onto their neighbouring polygon if it exists, or clamped to the closest edge otherwise.

The number of relaxation iterations required for a high quality point cloud depends on the complexity of the input geometry and the number of sample points. For dense meshes and high number of samples, the method may take several hundred or even thousand iterations to converge to an acceptable result. This is illustrated in Figure 2.8.

Quasi-uniform sampling can be obtained much faster using approximate blue noise methods [KS12], although they introduce low-frequency noise in the rendered image. Since quality is more important than speed in our case, we use the point repulsion algorithm.

First Pass: Irradiance Sampling

Given the set P of surface samples, the irradiance for each sample has to be computed. Since all the samples are placed at the surface of geometry, virtually any standard rendering technique may be used.

For offline rendering in a complex setting with occluders, global illumination systems such as

(a) *(b)* *(c)*

Figure 2.9.: *Benefits of BSSRDF evaluation with sample clustering compared to evaluation without sample clustering, illustrated on a pointcloud with* $100'000$ *samples, rendered at* 1680×1050 *pixels. With sample clustering (a), rendered in 140ms; without sample clustering and equal number of samples (b) rendered in 8.5 seconds; no sample clustering and equal render time (140ms) (c) allows only 400 samples. Note that (a) is visually equivalent to (b), despite being 60 times faster. Model courtesy of the Stanford Computer Graphics Laboratory.*

photon mapping [Jen96] or bi-directional path tracing [LW93] can be employed to compute the irradiance.

For interactive rendering, global illumination techniques are usually too expensive, and methods such as shadow mapping [Wil78], which deal only with direct illumination, may be used instead.

Second Pass: BSSRDF Evaluation

Given the set of irradiance samples and an evaluation point $\vec{x}_o$, we now need an efficient method for evaluating the radiant exitance at $\vec{x}_o$. Evaluating the sample contributions directly by computing the BSSRDF for all irradiance samples is too expensive, since typical point clouds may contain hundreds of thousands of samples. A better choice is to evaluate the BSSRDF using a hierarchical data structure instead, as illustrated in Figure 2.9. Any hierachical structure can be used that allows efficient traversal and clustering of nearby points.

Jensen and Buhler [JB02] use an octree in their implementation. Each node in the octree contains the total surface area A_v of all the samples it contains, the total irradiance E_v of all its children and an irradiance weighted average location $\vec{P}_v$. To compute the total radiant exitance at $\vec{x}_o$, the octree is traversed from the root and each child node is either recursively traversed or evaluated directly, depending on an error heuristic.

Finding an accurate estimation of the error is non-trivial. Jensen and Buhler [JB02] instead propose an approximate criterion based on the estimated solid angle $\Delta\omega$ spanned by a node:

$$\Delta\omega = \frac{A_v}{||\vec{x_o} - \vec{P}_v||^2} \tag{2.40}$$

A node is then recursively traversed if the estimated solid angle is smaller than some user-defined maximum error ϵ.

Evaluating the radiant exitance for a leaf node is equivalent to summing the contributions of the samples contained in the node. For nodes higher in the hierarchy, the contribution may be computed by treating the cluster as a single surface point and evaluating the BSSRDF using the average location, total irradiance and total area of the node.

It should be noted that the expression in (2.40) is problematic, since the average sample location is not necessarily an accurate representation of the point distribution. The denominator $||\vec{x_o} - \vec{P}_x||^2$ could thus significantly overestimate the distance to the sample set, leading to an erroneously small estimate for the solid angle.

For example, a node with a majority of the samples distant from the evaluation point, but a few outliers close to it may not be recursed, since the *average* location is far from the evaluation point. If the node is evaluated directly instead, this may cause significant error, since outliers arbitrarily close to the evaluation point are ignored.

A more conservative estimate is to use the distance to the enclosing bounding box of the node instead. This has the advantage of never overestimating the distance to the sample set, which avoids errors in the estimated radiant exitance at a higher computational cost.

3

Implementation

This chapter will describe implementation details of our system. A first section will provide a general overview of the system and describe the individual components. The subsequent sections will cover the individual application features in detail.

3.1. System Overview

Our system is built from a set of mostly independent modules, each performing specific tasks dealing with the BSSRDF and tying in with a portion of the user interface. A central control module summarizes parameters and loaded BSSRDFs and ties the rest of the modules together. In the following, the list of the implemented modules is given and described. Details about algorithm implementations are given in sections 3.2 through 3.5. Section 3.6 will cover the user interface.

Our application has been implemented in C++ and uses the open source, cross-platform UI framework Qt as well as the multi-platform graphics API OpenGL.

3.1.1. Parameter Module

The parameter module summarizes the list of parameters frequently used throughout the application. These include plot settings, lighting conditions and, most importantly, material parameters.

Material parameters consist of the relative index of refraction η, the scattering mean cosine g, the basic unit scale and the absorption and scattering coefficients, σ_a and σ_s. The module also

supports an alternative parametrization introduced by Jensen and Buhler [JB02] using the total diffuse reflectance R_{dr} and diffuse mean free path l_d instead of σ_a and σ_t. Internally, the system only works with the physically based parameters σ_a and σ_s; parameters entered in the R_{dr}, l_d format are converted before use. Section 3.4 covers the details of the inversion procedure.

In addition, the parameter module keeps the list of loaded BSSRDFs, both analytic and measured. Analytic BSSRDFs are provided with the material parameters, but may define an additional set of parameters if so required by the model. These are defined in the BSSRDF file interface and are exposed in the user interface. See the appendix A.2 for details about the file interface.

3.1.2. Point Cloud Module

The point cloud module provides an implementation of the hierarchical point cloud evaluation described in Section 2.6.4. It allows interactive rendering of a selected loaded BSSRDF (analytic or measured) on an arbitrary triangle mesh (OBJ format) or point cloud (PTC, PDA, PDB, GEO or BGEO format). For triangle meshes, the module generates a point cloud automatically. Generated point clouds are saved to disk in PTC format to speed up future loading times.

The point cloud module supports various different lighting conditions. These include directional lighting, filtered directional lighting and image based lighting (IBL). The filtered directional lighting consists of an ordinary directional light source multiplied with a user specified texture, akin to light falling through stained glass, for example. Its purpose is to allow the user to introduce discontinuities and different colors into the illumination. The IBL mode computes lighting due to an environment map and optionally supports self-occlusion on the model, which is more expensive, but much more accurate.

A visual overview of the different lighting modes is given in Figure 3.1.

3.1.3. Lit Sphere Modules

The lit sphere modules consist of two separate modules modelling subsurface scattering on the sphere under a directional light source. The first module computes the subsurface scattering using a volumetric path tracer, descibed in Section 3.2. The second module evaluates a selected analytic BSSRDF on the sphere using Monte Carlo integration, covered in Section 3.5.

3.1.4. Pencil Beam Module

The pencil beam module computes cross section plots for all loaded BSSRDFs on a half infinite slab lit by a pencil beam. Plotting modes set in the parameter module control the axes (linear, logarithmic or square root) and color modes (red, green, blue color channels or luminance). Additionally, a reference solution computed by means of a volumetric path tracer, described in Section 3.2, is plotted in the same coordinate system.

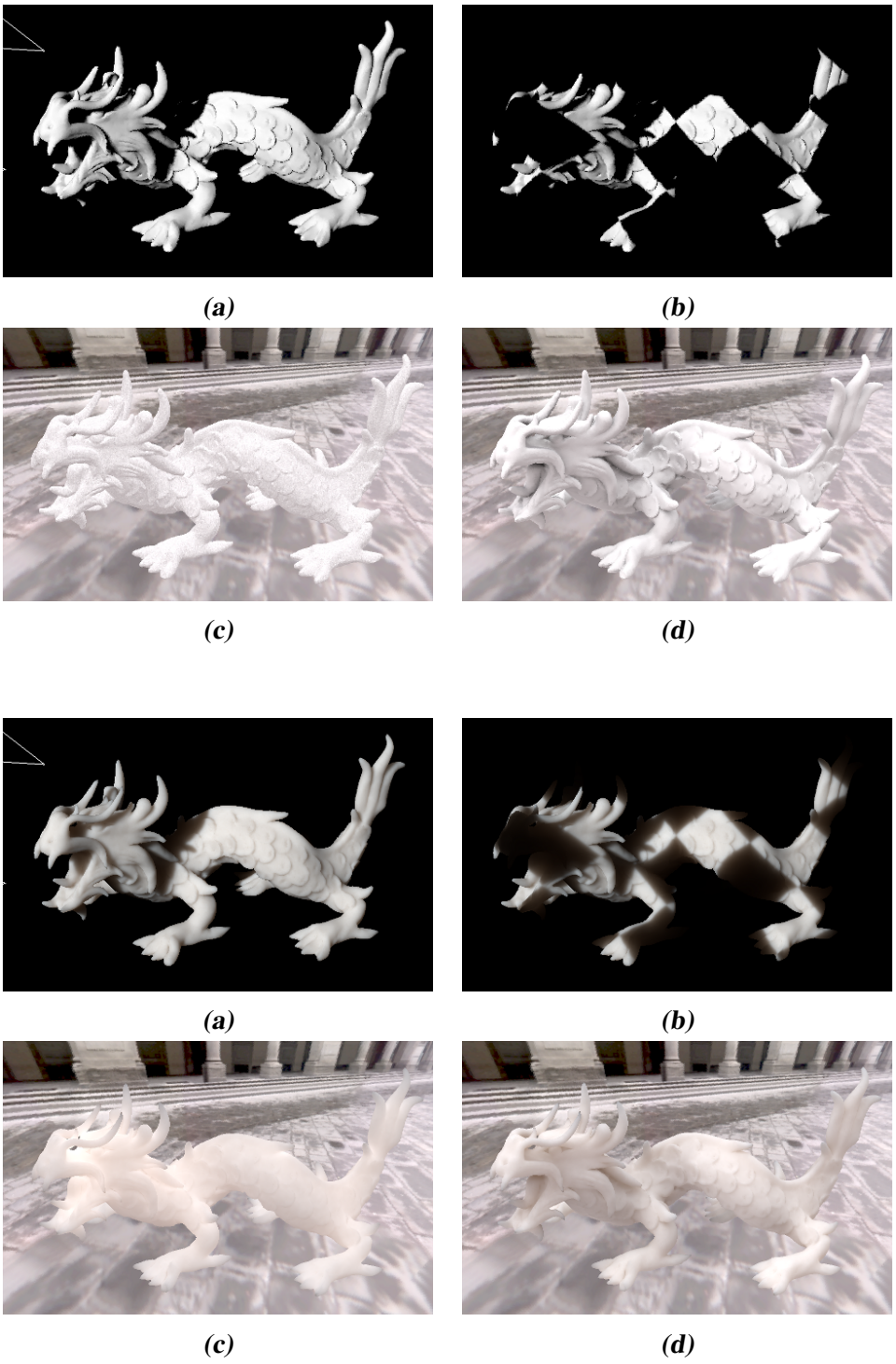


Figure 3.1.: *Summary of different lighting modes supported by the point cloud module: directional (a), filtered directional (here with checker texture) (b), IBL (c) and occluded IBL (d). Top: Irradiance samples. Bottom: Rendered with dipole BSSRDF. Model courtesy of XYZRGB. Environment map courtesy of Paul Debevec.*

3.2. Volumetric Path Tracing

To support the verification of analytic BSSRDFs, our application provides reference solutions for two kinds of geometry: A half-infinite slab lit by a pencil beam, and a sphere under a directional light source. Both rely on unbiased volumetric path tracing to compute a numerical solution for the radiance given the material parameters.

Only media with homogeneous scattering properties are considered, greatly simplifying the implementation and improving the convergence rate. Still, the algorithm remains expensive and care must be taken to ideally use the available budget of samples.

The path tracing implementation is characterized by a *trace* function. It traces a random path through the medium and returns the exit location as well as the weight of the path. To compute the total radiant exitance at a surface point, the algorithm then only has to repeatedly invoke the trace function and for each path compute the irradiance at the exit location. The weighted average of the irradiances multiplied by the transmission term then gives the total radiant exitance.

A simple, but naive and inefficient implementation of the trace function is given as follows. It is a straightforward application of Monte Carlo to the RTE:

1. Start with a position $\vec{x}$ and direction $\vec{\omega}$
2. Pick a random distance s to travel before the next scattering event
3. Advance by s along $\vec{\omega}$ and pick a new random direction $\vec{\omega}'$ on the sphere
4. Modify the path weight by extinction $e^{-\sigma_t s}$, scattering coefficient σ_s and phase function $p(\vec{\omega} \cdot \vec{\omega}')$
5. Repeat steps 2-4 until the surface is hit
6. Modify the path weight by the transmission term f_t

While simple to implement, this version of the trace function is highly inefficient in that it takes a large number of samples to converge, especially for materials with high absorption. The problem is that path weights rapidly go to zero with the number of scattering events and the amount of distance travelled. The algorithm is completely agnostic of the mean free path, the absorption coefficient and the scatter anisotropy g.

An improved algorithm distributes samples non-uniformly such that the path weights are constant. Intuitively, we still want to sample all paths - to remain unbiased - but would like to more frequently sample paths that contribute a lot to the final image. By distributing sampling probabilities corresponding exactly to the path weights, the new path weights are the same for all paths. To achieve this, the naive algorithm has to be modified in four places:

- Don't pick a uniformly random distance s, but pick it according to the probability density function $e^{-\sigma_t s}$
- Instead of choosing the new direction uniformly, choose it with probability according to the phase function p
- Don't modify the path weight by the scatter coefficient σ_s, but perform a Russian roulette,

terminating the path with probability $1 - \alpha$ and continuing with probability α.

- Instead of modifying the weight by the transmission term f_t, run a Russian roulette, exiting the surface with probability f_t and performing internal reflection with probability $1 - f_t$.

The resulting algorithm is given in Algorithm 1.

Algorithm 1 Volumetric path tracing

```
1:  function TRACE(α, σ_t, x⃗, ω⃗)
2:      loop
3:          t ← −log(ξ)/σ_t
4:          t_max ← intersect(x⃗, ω⃗)
5:          if t > t_max then                          ▷ Surface intersection
6:              if ξ > f_t(x⃗, ω⃗) then
7:                  ω⃗ ← reflect(ω⃗)                    ▷ Internal reflection
8:                  x⃗ ← x⃗ + (t − t_max)ω⃗
9:              else
10:                 return exited                      ▷ Exit
11:             end if
12:         else
13:             x⃗ ← x⃗ + tω⃗                           ▷ Advance
14:         end if
15:         if ξ > α then return absorbed              ▷ Absorb
16:         ω⃗ ← scatter(ω⃗)                            ▷ Scatter
17:     end loop
18: end function
```

When dealing with interactivity constraints, one may encounter a peculiar problem with this method. On the half-infinite slab, although unlikely, the trace method may follow a long series of scattering events deep through the medium and take an unpredictable amount of bounces before it finally hits the surface. Although these paths are in theory correct contributions, they carry little weight since they only occur rarely. However, they pose a serious problem, since they might trap the path tracing algorithm following a single path for a long time and can interfere with user interactivity.

To alleviate the issue, we cannot simply terminate paths after a maximum number of scattering events lest we introduce bias. Instead, we can simply make such paths *less likely* at the cost of higher variance: After a path exceeds a certain number of scattering events, we run an additional Russian roulette after each scatter event that follows. With a 90% chance, the path is immediately terminated; with a 10% chance, it survives another turn and instead its weight is increased by a factor of 10. This results in a path tracer that is still unbiased, but almost never exceeds a certain number of bounces. Of course, should any of these paths reach the surface after surviving a few russian roulettes, they will introduce significant variance. However, these events are increasingly rare and can be considered a reasonable sacrifice compared to the benefits.

In our application, we provide both CPU and GPU implementations of the path tracing algorithm for the user to choose from at build-time, depending on the target machine. Machines with

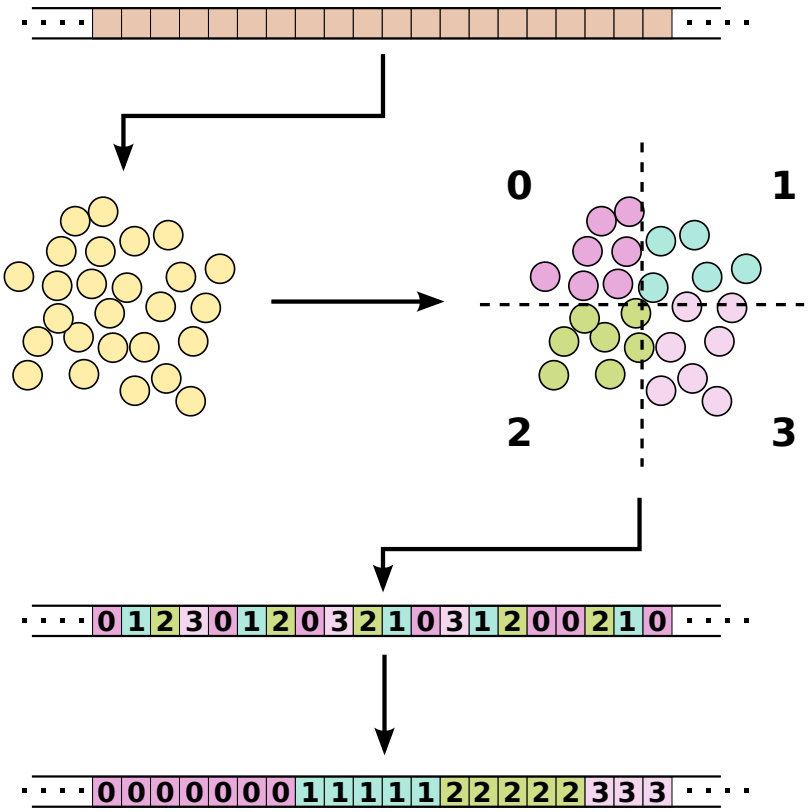


Figure 3.2.: *Sample set subdivision for octree construction*

slower GPUs greatly benefit from the CPU implementation, since long running GPU threads may interfere with interactivity of the entire operating system. The CPU implementation on the other hand can run concurrently in background threads, leaving the user interface responsive even in the case of long scatter paths.

For the half-infinite slab, our implementation has been verified with the well-tested MCML benchmarking tool [WJZ95]. For the lit sphere, the same code is used, with only the primitive intersection function modified.

3.3. Rendering Translucency

Our system is capable of rendering the BSSRDF on arbitrary meshes and pointclouds under various lighting conditions at interactive framerates. To achieve this, we take advantage of the large processing powers present in modern programmable GPUs by implementing the two-pass pointcloud technique in the OpenGL shading language, GLSL. Timings of our implementation for various models and point cloud sizes are given in Table 3.1.

Model	Size of point cloud	Render time
Armadillo	339'679	138ms
Happy Buddha	301'986	149ms
Bunny	489'088	143ms
Cow	203'355	45ms
Stanford Dragon	88'551	121ms
Lucy	202'801	84ms
XYZRGB Dragon	199'174	102ms

Table 3.1.: *Render times of our GPU pointcloud implementation at a resolution of* 1280×720 *pixels on a GTX 480*

3.3.1. Point Cloud Generation

If generating a new point cloud is necessary, the algorithm detailed in Section 2.6.4 is employed to generate a sample set uniformly covering the surface. Unfortunately, the algorithm is not easily portable to the GPU, and a CPU implementation is used instead. Since the point cloud generation represents the most expensive step of the mesh preprocessing pipeline, our implementation takes advantage of multiple cores, using OpenMP for parallelization.

Once the point cloud is available, a matching octree is constructed. The octree building algorithm starts with the entire sample set, stored in an array. Then, the center location of the sample set is computed; this partitions the space into eight octants, depending on relative position to the center. All points in the set are labelled with an index in $[0, 7]$, marking the octant in which they are located. The sample array is sorted with a linear time counting sort [Knu98], using the octant index as key. This results in 8 or less new sets, formed by all samples with the same index. Each of these sets is then recursively subdivided using the same algorithm, until the sets are smaller than some threshold (8 samples in our implementation). The hierarchy of sample sets then forms the octree.

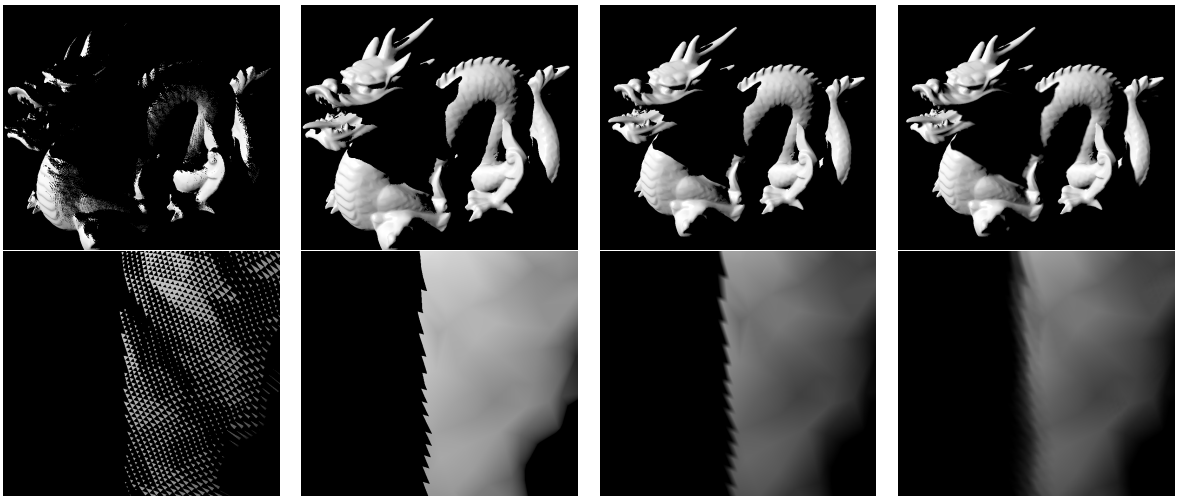
A single subdivision step is illustrated on a 2D example in Figure 3.2.

3.3.2. Irradiance Computation

After generating the point cloud on the CPU, the point samples are transferred to an OpenGL *vertex buffer object* (VBO). VBOs represent read/writeable regions of GPU memory dedicated to vertex data and allow the application to store vertices in GPU memory instead of retransmitting them at every rendering step. Each vertex stores attributes for position, normal and irradiance.

For the first rendering pass, the application makes use of a vertex shader to compute the irradiance. Vertex shaders are executed once for each vertex arriving at the vertex transformation stage. In our case, the vertices represent the point samples for which the irradiance is to be computed.

Normally, the results of the vertex transformation stage are only used for rasterization and then



(a) *(b)* *(c)* *(d)*

Figure 3.3.: *Results for different shadow mapping methods: naive (a), with offset (b), with offset and hardware PCF (c) and with offset, hardware PCF and multiple jittered lookups (d). Model courtesy of the Stanford Computer Graphics Laboratory.*

discarded. However, this behaviour can be modified using *transform feedback*. Transform feedback enables the application to stream the results of the vertex transformation stage back into a VBO. In our case, we disable the rasterization stage completely, compute the sample irradiance with the vertex shader and stream the results into a second VBO, which is then used for rendering.

Some of the implemented lighting techniques perform an incremental computation of the irradiance. In these cases, the application invokes the vertex shader multiple times and ping-pongs between two VBOs used as source and target in turn.

Directional Lighting

Computing irradiance due to a directional light source is trivial except for evaluating the visibility of the light source. In our system, we use standard shadow mapping techniques [Wil78] to compute occlusion. In an initial pass, the model is rendered as a depth map from the perspective of the light source into an off-screen buffer. In the irradiance pass, the vertex shader then transforms each point sample into the coordinate space of the light source and projects it onto the depth map. If the depth of the point sample in the light coordinate system is larger than the value read at at the corresponding cell in the depth map, the sample is assumed to lie in shadow.

A naive implementation of this algorithm easily leads to artifacts due to quantization errors and limited resolution of the depth buffer, seen in Figure 3.3 (a). Adding a small offset to the samples read from the depth map removes the majority of artifacts. However, the binary classification scheme and the limited resolution of the buffer lead to a blocky appearance of shadow borders, Figure 3.3 (b). A more visually appealing result can be achieved using percentage closer filtering (PCF) [RSC87], which averages over multiple shadow lookups. Most GPU hardware provide built-in PCF, performing the depth test on the four closest texels on the

depth buffer and bilinearly interpolating between the four results. Although this improves the appearance of the shadow border, aliasing due to depth map resolution is still clearly visible, Figure 3.3 (c). Performing multiple, jittered lookups, each with hardware PCF, further reduces aliasing artifacts and helps smooth out the shadow border, Figure 3.3 (d).

Image Based Lighting

Computing the irradiance due to an environment map is a lot more costly than for the purely directional case, since we now have to compute the integral of the incident radiance over the hemisphere of incident directions. Implicitly, this was already the case for the directional light, but because the incident light was represented by a delta function, the integral collapses and results in a very simple expression. For the environment map, now all incident directions can potentially carry radiance, requiring full evaluation of the integral.

We can build an approximation of the true irradiance by noting that if occlusion is ignored, the irradiance no longer depends on the location of the sample point, but only on the surface normal. This allows precomputation of the integral for a set of possible surface normals, such that at runtime, only a lookup of the precomputed integral has to be performed. This allows for fast estimation of the irradiance even when the model or the environment map are rotated. In our system, precomputation of the integral is performed when an environment map is loaded by the user. The integral is approximated by a Monte Carlo method, computed for a large set of possible surface normals and then stored in a cubemap. At runtime, the vertex shader only has to perform a lookup in the cubemap texture using the surface normal.

Although this method is fast, disregarding self-occlusion by the mesh results in overestimation of the irradiance at most locations, seen in Figure 3.1 (c). These artifacts can only be removed by evaluating the integral while taking occlusion into account. Not only does this require a per-point sample evaluation of the integral each time the lighting is changed, thwarting precomputation - it also necessitates a method to compute the visibility of a point on the environment map seen from a surface point. In offline rendering, typically geometric ray-surface intersections are used to compute visibility, which makes the process unfeasibly slow for interactive rendering, especially for large point clouds and complex meshes.

However, we can speed up this process by taking advantage of shadow mapping techniques. The integral is still computed with a Monte Carlo method as before, but instead of choosing samples individually for each surface point, we choose the same samples for all points instead. This way, we can pick a sample on the environment map and treat it as a directional light source, allowing the use of a shadow map to compute visibility for each surface point. Although a full shadow map has to be computed for each sample, it contributes to all points at the same time, greatly improving performance [PH10]. It should be noted that choosing the same samples for all surface points leads to banding artifacts if few samples are used.

3.3.3. Radiance Evaluation

For the second rendering pass, a representation of the octree in GPU memory is required. Since GLSL does not support full access to GPU memory, the data structure has to be stored in

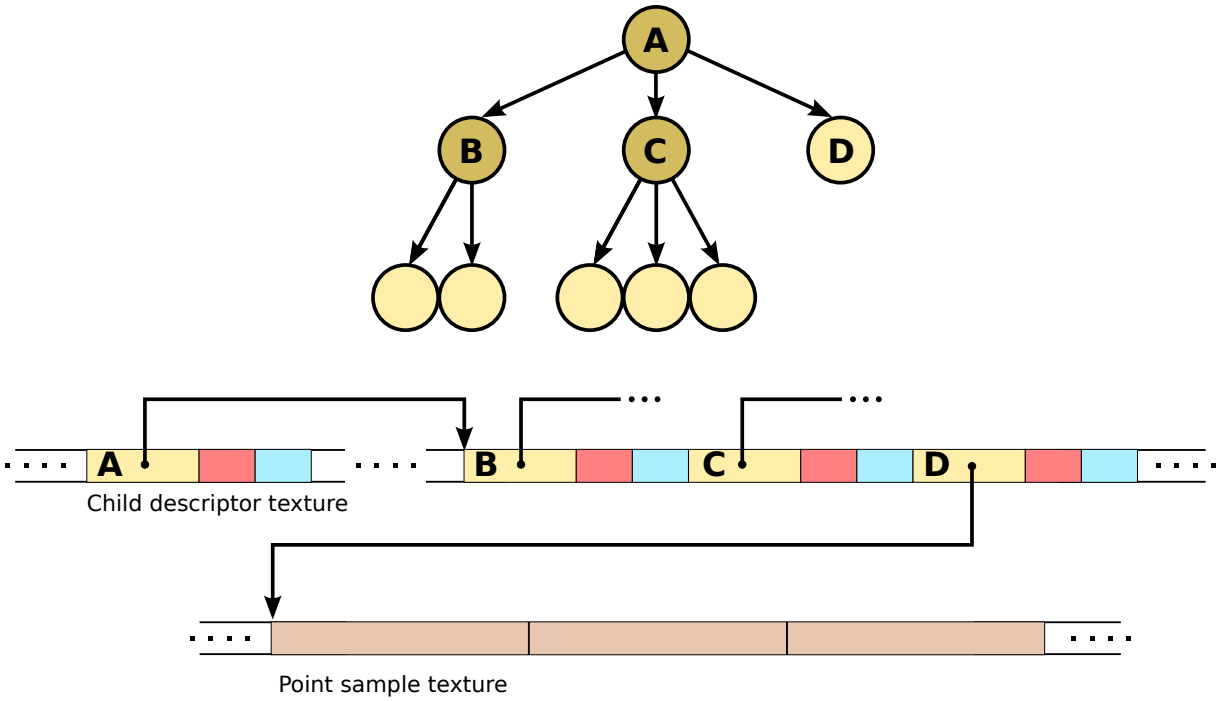


Figure 3.4.: *Layout of the descriptor arrays. Top: Example octree. Middle: Child descriptor array. Bottom: Point sample array. Not shown: Node attribute arrays*

OpenGL primitives. For optimal performance, we choose to store our data structure in OpenGL textures. To achieve a compact representation in memory, we store a sparse octree representation similar to sparse voxel octrees [LK10]. The nodes and sample points are laid out linearly in memory in a tightly packed array and the data necessary for traversal can be represented with only one 32bit child descriptor per octree node.

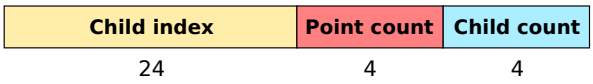


Figure 3.5.: *32bit child descriptor*

Each of these 32bit descriptors is divided into three different fields. The lowest four bits store the number of children contained in the node, which is zero for leaf nodes. The next 4 bits store the number of point samples contained in the node and are only non-zero for leaf nodes. Using only 4 bits restricts the number of samples in a leaf node to at most 15, but since we limit the number of points in a leaf node to at most 8 in the octree construction, this does not represent a problem. It should be noted that, since one of the two lower fields is always zero, a more efficient data packing can be achieved by using one bit to identify a leaf node and storing either the number of points or children in the remaining 7 bits. Since we do not require more than 4 bits for either fields, we choose the redundant packing instead, since it allows for slightly faster traversal.

Finally, the highest 24 bits store the child index. For interior nodes, this is an absolute index in the child descriptor array pointing to the location of the first child. All children of a node lie at consecutive slots in the array, removing the need for storing a separate index for each child. For a leaf node, the child index instead represents an absolute index into the array containing the

point samples; again, all points of a leaf node are laid out consecutively in memory, requiring only the array index to address all points of a leaf node. The layout of the descriptors in memory is illustrated in Figure 3.4.

The internal nodes also carry information about the total irradiance, the total area, the irradiance weighted average location and the bounding box of the point samples represented by the node. The irradiance (RGB) and total area (1 float) are merged and stored in a single, 4 channel float texture. Lower- and upper bound of the bounding box as well as the average location are all vectors of 3 floats, but unfortunately, 3 channel textures are not widely supported in OpenGL. Subsequently, these attributes, too, are stored in 4-channel float textures, leaving one color channel per attribute unused. Although 1D textures most closely mirror the nature of arrays, most OpenGL implementations limit the size of 1D textures to a size much smaller than required for most octrees. Instead, we store the arrays in 2D textures using row-major order.

Traversal of the octree to compute the radiant exitance closely follows Jensen and Buhler [JB02]. Starting with the root node, the shader recursively traverses its child nodes as long as the error according to the solid angle criterion is too big and evaluates the BSSRDF otherwise. One problem with this formulation is that GPUs do not have a stack, making "recursive traversal" in the traditional sense impossible. For this reason, the shader uses a loop instead of recursion and keeps track of previous iterations manually using an array. It should be noted that arrays in GLSL internally incur a high cost in terms of shader complexity. The array size also stands in direct relation to usage of limited GPU resources by the shader, thus limiting the amount of threads that can be executed concurrently.

For this reason, reducing the size of the stack (and therefore the array) to a minimum is important for shader performance. We note that for traversal, each level of the stack needs to store the array index of the node currently being processed and the number of child nodes left to visit. By storing the array index in the upper 24 bits and the child count in the lower 8 bits of an integer, we can reduce the storage requirement for each recursion to a single 32bit integer. In the application, we limit the maximum depth of the octree to 10 levels (allowing for up to 8.5×10^9 sample points), and can thus safely compact the stack to an array of 10 32bit integers.

Pseudocode of our traversal routine is found in Algorithm 2.

3.4. Parameter Inversion

Following Jensen et al. [JMLH01], we support an alternative parametrization of the BSSRDF using the total diffuse reflectance R_{dr} and diffuse mean free path l_d in place of absorption and scattering coefficient σ_a and σ_s. This is motivated by the fact that the effects of σ_a and σ_s on the rendered image are highly non-linear, making it difficult to predict the appearance of a model with subsurface scattering for a given set of parameters.

The alternative parametrization makes it easier to control the rendered image. Intuitively, the total diffuse reflectance R_{dr} controls the color of surfaces under direct light, whereas the diffuse mean free path l_d controls the color of the scattered light on surfaces that lie in shadow. Additionally, the magnitude of l_d controls the translucency of the material. Large values for l_d result in more translucent materials, and vice-versa.

Algorithm 2 Octree traversal

```
1: radiance ← 0
2: node ← 0
3: childnum ← 0
4: loop
5:     if subdivide(node) then
6:         recurse(node, childnum)
7:     else
8:         if is leaf(node) then
9:             radiance ← radiance + point contributions(node)
10:        else
11:            radiance ← radiance + node contribution(node)
12:        end if
13:        next node(node, childnum)
14:    end if
15: end loop
16:
17: function RECURSE(node, childnum)
18:    if node ≠ 0 then push(node, childnum)
19:    node ← last child(node)
20:    childnum ← child count(node) − 1
21: end function
22:
23: function NEXT NODE(node, childnum)
24:    if childnum = 0 then
25:        unrecurse()
26:    else
27:        node ← node − 1
28:        childnum ← childnum − 1
29:    end if
30: end function
31:
32: function UNRECURSE
33:    if stack empty then
34:        set result(radiance)
35:        end shader()
36:    else
37:        node, childnum ← pop()
38:    end if
39: end function
```

To render an image with the alternative parametrization, the parameters have to be inverted to give the corresponding values of σ_a and σ_s. The total diffuse reflectance is defined as the integral of the diffuse reflectance profile:

$$R_{dr} = 2\pi \int_0^\infty R_d(r) r \mathrm{d}r \tag{3.1}$$

For the diffusion dipole, this yields [JB02]:

$$R_d = \frac{\alpha'}{2}\left(1 + e^{-\frac{4}{3}A\sqrt{3(1-\alpha')}}\right) e^{-\sqrt{3(1-\alpha')}} \tag{3.2}$$

In general, we can formulate R_{dr} as a function of the reduced albedo α' and the internal reflection parameter A. Solving this function for α' yields a term depending on A and R_{dr}, which are given. Then, the rest of the parameters can be computed using α' and l_d:

$$\sigma_t' = \frac{1}{l_d\sqrt{3(1-\alpha')}} \tag{3.3}$$

$$\sigma_s' = \alpha'\sigma_t' \tag{3.4}$$

$$\sigma_a = \sigma_t' - \sigma_s' \tag{3.5}$$

In our implementation, we cannot analytically solve for α', since we allow any BSSRDF to be defined and therefore can't assume any knowledge about R_d or R_{dr}. Instead, we define a function stub to evaluate R_{dr} given α' in our file interface, which can be implemented by the BSSRDF. A standard bisection root solver is then employed to solve for α' numerically. Although more sophisticated numerical root solvers can show superior convergence to the bisection method, we found them to become unstable for some R_{dr} functions. Additionally, the bisection method is easily parallelizable, allowing us to take advantage of the GPU.

In our implementation, we start with the interval $[0, 1]$ and divide it into 512 equally spaced subintervals. We then run a shader computing the value of R_{dr} for the end points of each of the subintervals in parallel. Then, the subinterval containing the user-specified R_{dr} is selected and the process is repeated. The selected subinterval gives the bounds for α'. We repeat this procedure three times, guaranteeing a maximum error of 2^{-27} for α', which is beyond floating point accuracy for the majority of input values.

3.5. BSSRDF Integration on the Sphere

Complementary to the hierarchical pointcloud renderer, we also provide a module rendering the BSSRDF on a sphere with Monte Carlo integration. Motivation for this module is to allow comparison with a volumetric path tracer with the same geometry and lighting conditions. The point cloud, although fast, introduces discretization error; Monte Carlo integration however converges to the correct value of the double integral (2.14).

Performing the Monte Carlo integration is straightforward. First, a uniformly random position on the sphere is picked. Then the BSSRDF is evaluated. The sum of the contributions divided by the individual sample probabilities yields an estimate for the radiant exitance.

Note that this formulation is inefficient if the integrand is strongly peaked and may take many samples to converge. Ideally, the method would apply importance sampling to pick samples on the sphere. Unfortunately, we cannot make any assumptions about the integrand, since we allow the user to define any function for the BSSRDF. Providing an importance sampling interface general enough to handle all BSSRDFs exceeds the scope of this thesis and is left for future work.

3.6. User Interface

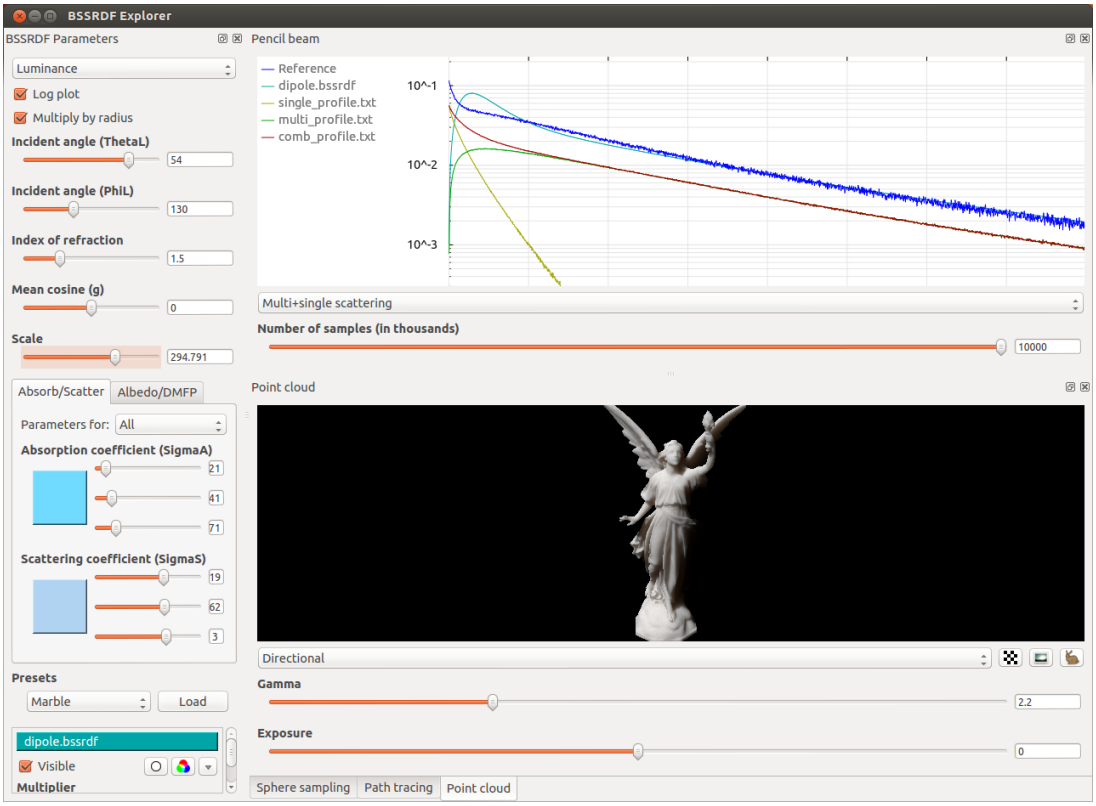


Figure 3.6.: *User interface. Model courtesy of the Stanford Computer Graphics Laboratory.*

This section will provide details about the user interface of our system. The modules and algorithms presented in the previous sections are all represented within the UI by a set of widgets. Initially, all widgets are laid out in tabbed views within a single window; however, widgets can be individually popped out from the main window, resized and moved, depending on user preference.

The complete user interface is shown in Figure 3.6.

3.6.1. Parameter Window

The parameter window exposes the internal set of parameters in the user interface. Figure 3.7 shows an overview of the interface.

All material parameters are accessible through sliders. To account for the wide value range of some parameters, the sliders are in logarithmic scale where appropriate. As described in earlier sections, scatter parameters can be entered both in the traditional σ_s, σ_a format as well as the alternative R_{dr}, l_d format. Additionally, scatter parameters can be set for all BSSRDFs at the same time, or tweaked individually for each BSSRDF.

The scatter parameters are displayed as an approximate color representation to give an intuition about the appearance of the material. This is achieved by interpreting the scatter parameters

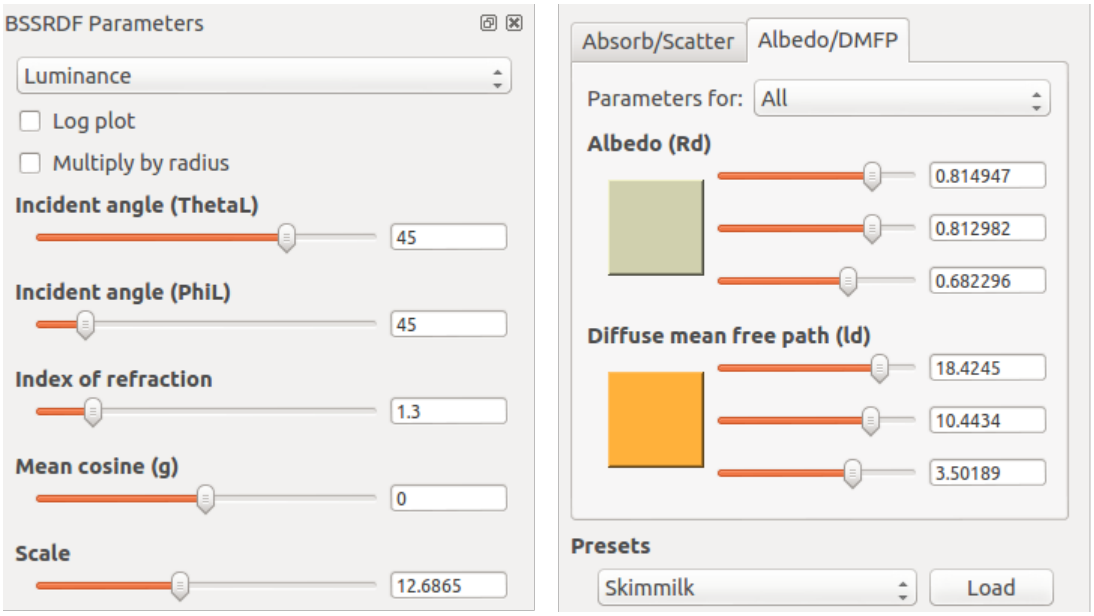


Figure 3.7.: *Overview of the parameter window (split up into columns)*

as RGB values and rescaling them to result in a constant luminance of 0.75 to handle the high dynamic range of the parameters.

Similar to many rendering packages, our system offers a list of material presets, taken from Jensen et al. [JMLH01].

In addition to the scatter parameters, the parameter window also provides a scale parameter. Widgets working with geometry automatically normalize the models to occupy one basic unit along their longest axis. The scale parameter then allows the user to specify the scale of one basic unit in relation to the scatter parameter units. For example, the table of measured parameters in Jensen et al. [JMLH01] assumes units of millimeters; entering a scale of 100 would then treat the model as if it occupied 10cm along its largest axis. Although one could of course manually scale the scatter parameters to give the same result, changing the scale value instead is much quicker and more convenient.

Figure 3.8 demonstrates the effect of different scale parameters.

3.6.2. Lit Sphere Widgets

The lit sphere widgets display subsurface scattering effects rendered on a sphere under directional light, shown in Figure 3.9. The sphere sampling widget integrates the currently selected BSSRDF over the surface area of the sphere, described in Section 3.5. The path tracing widget instead computes the radiant exitance using volumetric path tracing, described in Section 3.2.

Depending on the material parameters, images may take different numbers of samples to converge. To account for this, we allow the user to adjust the number of samples computed per pixel. Note that these are typically much higher for the sphere sampling widget than the path tracing widget, caused by the lack of importance sampling. In practice, this is not a problem,

(a) *(b)* *(c)* *(d)*

Figure 3.8.: *Influence of the scale parameter on the rendered image: Values of 10 (a), 50 (b), 200 (c) and 500 (d) for the dipole BSSRDF, from the front (top) and back (bottom). Model courtesy of the Stanford Computer Graphics Laboratory.*

since computing a BSSRDF sample is many magnitudes cheaper in terms of computation time compared to tracing a full scatter path through the medium.

Initially, the widgets are grouped into a tabbed view. This allows quickly switching between the two rendered images, making it easier to spot differences in the results. Like all widgets, they may also be popped out from the main window and rescaled to allow for a side by side comparison, such as shown in Figure 3.9.

Both widgets perform rendering in high dynamic range (HDR) and support post-process gamma correction and exposure adjustment using sliders, as seen in Figure 3.9.

3.6.3. Pencil Plot Widget

The pencil plot widget shows cross section plots of all loaded BSSRDFs, evaluated on a semi-infinite slab illuminated by a pencil beam in the origin. Additionally, a reference solution is plotted, computed with volumetric path tracing.

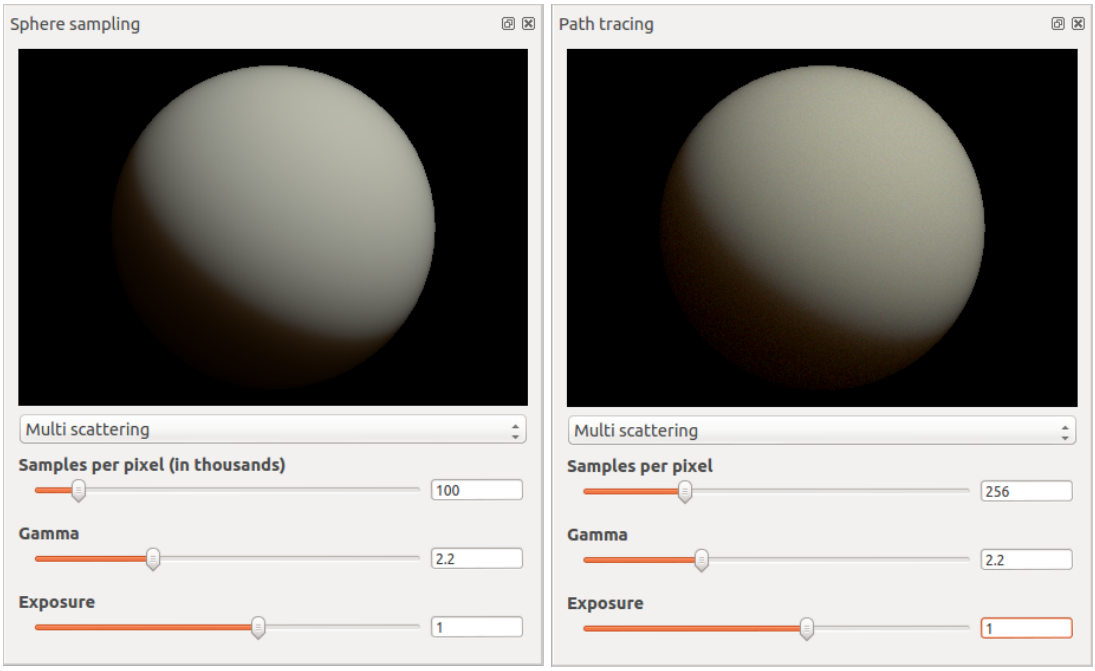


Figure 3.9.: *Overview of the lit sphere widgets*

Comparing BSSRDFs in a linear plot can be problematic, since they typically cover large value ranges. The plot thus supports both linear and logarithmic scales on the y axis. Logarithmic plots are common in fields other than image synthesis, because they allow accurate comparison for regions far away from the source where the function values are small. For image synthesis however, regions close to the source, where the function values are large, are more important, since they influence the light transport the most. Logarithmic plots can be problematic here, because they might downplay large errors near the source - for example, the values 10 and 20 appear to have the same distance in the log plot as 0.01 and 0.02.

For this reason, the widget also supports a square root scale on the y axis as a useful addition to logarithmic plots, allowing a more accurate visual comparison near the source. Complementary to different axis scales, the widget also supports plotting the product of radius from the origin and BSSRDF, which is a useful modification to damp the strong peak near the source. All plotting features may be enabled or disabled individually from the parameter window.

It should be noted that all BSSRDFs compute results in RGB, whereas the plot only displays a single value per BSSRDF. To account for this, the plot may be adjusted to either display the luminance or the red, green or blue color channel. In addition, a single BSSRDF can be selected to be “soloed”, hiding all other BSSRDFs and displaying the color channels of the BSSRDF as separate graphs in the plot.

Similar to the path trace widget, the number of samples computed for the path traced solution may be adjusted to account for different materials.

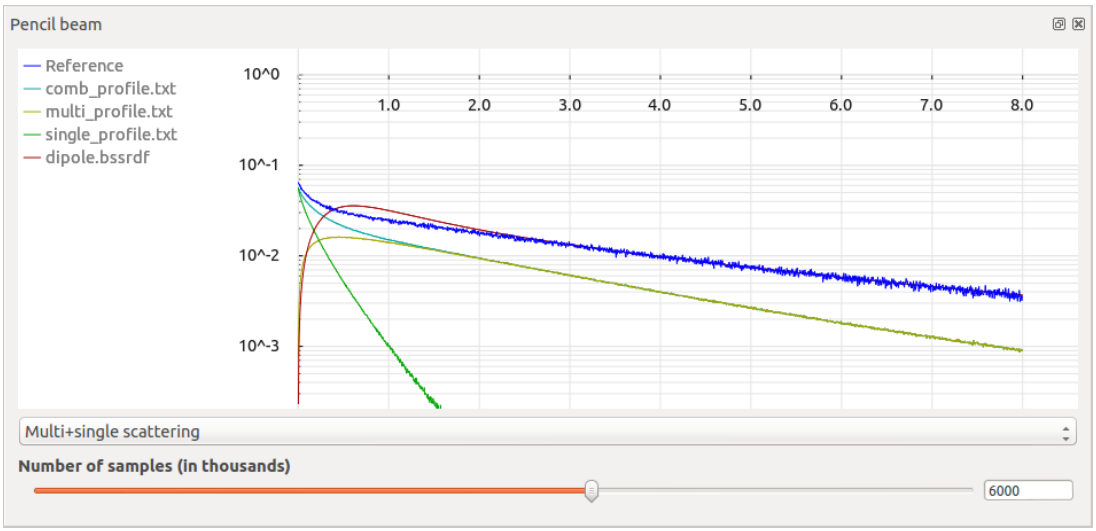


Figure 3.10.: *Overview of the pencil plot widget*

3.6.4. Point Cloud Widget

The point cloud widget is the most useful widget to evaluate the visual performance of a BSSRDF. It allows rendering any BSSRDF on arbitrary triangular meshes or point clouds and supports various lighting modes.

When loading a model, the widget makes an attempt at finding both a polygonal as well as a point cloud representation of the model on the file system. If both are available, the polygonal mesh is used for rendering and the point cloud to sample the irradiance. If only a polygonal mesh is available, a point cloud representation is generated and the result is cached to disk. Vice-versa, if only the point cloud is available, an approximate polygonal reconstruction is generated and used for rendering. Internally, the system uses the Partio library [Stu10] for efficient point cloud input/output, supporting most popular point cloud formats.

A visual summary of the supported lighting modes is given in Figure 3.1. The occluded IBL mode is comparatively expensive and would take too much time if computed directly. Instead, the irradiance is constructed incrementally, computing only a few samples every frame, allowing for visual feedback on the construction as well as intermediate user input. We can also take advantage of the fact that camera rotation does not influence the illumination, making recomputing the irradiance unnecessary when only the camera is changed.

Although the rendering step generally performs well, the framerate may drop too low for complex models and large resolutions. For this reason, the image is only rendered at half the regular resolution when the environment is changing, for example due to material or camera changes. Once the environment is at rest, the full resolution image is computed.

Similar to the lit sphere widgets, the point cloud widget renders in high dynamic range and supports post-process gamma correction and exposure adjustment.

4

Conclusion and Future Work

In this thesis, we presented an extensible rendering and analysis framework for the BSSRDF. We summarized existing research in the field and implemented the presented rendering techniques on graphics hardware, allowing for rendering at interactive framerates. We implemented unbiased rendering methods, both on CPU and GPU, to provide reference solutions for simple geometries. We implemented the classic diffusion dipole [JMLH01] and the better dipole [d'E12] in our system. We implemented plotting tools to study the cross-section behaviour of BSSRDFs. We implemented our work in a multi-platform GUI application, building on the popular BRDF Explorer by Walt Disney Animation Studios. We hope that our system becomes useful to researchers and artists alike and, by releasing the application as an open source project, hope that the system can be improved even after this thesis.

Future improvements include extending the BSSRDF interface with an importance sampling framework to properly handle strongly peaked BSSRDFs. It would also be useful to add a volumetric path tracer for arbitrary geometry to compare the appearance of the BSSRDF on arbitrary meshes to a reference solution. Another interesting feature to consider is integration of surface BRDFs into the application to increase the visual fidelity of the rendered images.

A

Appendix

A.1. Efficient Pseudorandom Float Generation

One of the core ingredients of a fast and accurate path tracer is a good *pseudorandom* number generator (PRNG). A PRNG is only pseudorandom, because its output sequence is deterministic, periodic and reproducible given the same seed value. Truly random number sequences are hard to produce on a deterministic machine, but "good enough" pseudorandomness suffices for most purposes. A PRNG suitable for Monte Carlo integration has to fulfill the following properties:

- Guaranteed long sequence periods
- Uniform distribution of the generated numbers within the PRNG value range
- No correlation between successive values

Several PRNGs fulfill these properties and are fast enough for Monte Carlo integration, such as linear congruental generators (LCG) [Knu81], lagged fibonacci generators (LFG) [Bre92] or the Mersenne Twister [MN98].

LCGs are attractive for GPUs, because they require minimal storage and are fast to compute. An LCG is defined by the recurrence relation

$$X_{n+1} \equiv (aX_n + c) \pmod{N} \tag{A.1}$$

Where N is the value range, X_n is the current state, X_0 is the start value ("seed") and a and c are the parameters of the LCG. Care must be taken when choosing a and c, since bad choices may lead to disastrously short periods, greatly distorting results gained through Monte Carlo

methods. For our implementation, we use $m = 2^{31}$, $a = 1103515245$, $c = 12345$, corresponding to ANSI C `rand`[Ent97]. This generator has a full period length of 2^{31}, although it has been noted that "the low bits of the numbers generated are not very random" [Ent97]. This is fine for our purposes, since we only use the upper 23 bits (see below).

Typically, path tracing implementations require uniform floating point numbers in $[0, 1)$ or $[-1, 1)$ range and not integers that most PRNGs produce. A straightforward way to construct a floating point PRNG is to simply use an integer PRNG and divide the returned number sequence by N. While this method achieves the desired result, it is comparably expensive, requiring both an integer-to-floating point conversion as well as a floating point division. For a volumetric path tracer, large amounts of random floats are consumed for little computation, easily making the PRNG the bottleneck of the algorithm.

Thus, a fast conversion between integer PRNGs and floating points in the desired value range is of high interest for fast Monte Carlo integrators. In our implementation, we exploit the IEEE 754 float representation [otICS08] and GPU intrinsics.

Figure A.1.: *IEEE Float representation*

The layout of the IEEE float is given in Figure A.1. A float is represented as a 32 bit string, partitioned into three fields. Here, we denote s as the sign bit, m as the mantissa and q as the biased exponent. Then we can define the significand $c = 1.m$ and the unbiased exponent $e = q - 127$. This yields the *normalized* float representation

$$(-1)^s \times c \times 2^e \tag{A.2}$$

It should be noted that, for increased precision, the IEEE float format always adds an implied leading 1 in front of the mantissa m, leading to the perhaps odd notation of $1.m$

We can now use this representation to our advantage to efficiently generate a random floating point number in $[0, 1)$ using a two-step construction. First, a random float in $[1, 2)$ is produced. By setting the bits corresponding to the mantissa to a random 23 bit integer k and letting $s = 0$, $q = 127$, (A.2) yields

$$(-1)^0 \times 1.k \times 2^{127-127} = 1.k \tag{A.3}$$

Which is a floating point number in range $[1, 2)$. Simply subtracting 1 from this number yields the desired random floating point in $[0, 1)$.

Similarly, we can generate a random floating point in range $[-1, 1)$ following the previous construction, but setting $q = 128$ instead. This gives us

$$(-1)^0 \times 1.k \times 2^{128-127} = 2 \times 1.k \tag{A.4}$$

Which is a floating point number in range $[2, 4)$. Simply subtracting 3 gives us the desired $[-1, 1)$ distribution.

Contructing the corresponding floating point bit representation from the result of a PRNG is easily achieved using bit level operators present in most programming languages. Typically, these operators are only available for integers and not for floats, making a direct construction of the desired floating point number difficult. Fortunately, GLSL offers an instruction `uintBitsToFloat`, which takes an unsigned 32 bit integer and reinterprets it as a floating point number without conversion.

Using this instruction, we can first construct the floating point bit representation using bit level operators on integers. The result is then reinterpreted as a floating point number and the floating point subtraction is performed.

GLSL code for generating pseudorandom floating point numbers as described in this section can be found in Listing A.1. Note that on current hardware, `uintBitsToFloat`performs no actual operation and only serves as an instruction to the compiler, making the conversion free in terms of computation.

Listing A.1: *GLSL pseudorandom float generation*

```
/* Has to be initialized to seed value before use */
uint RandSeed;

/* Returns float in range [0, 1). Period length is 2^31 */
float Rand() {
  RandSeed = (1103515245u*RandSeed + 12345u) & 0x7FFFFFFFu;

  uint FloatBits = (RandSeed >> 8u) | 0x3F800000u;

  return uintBitsToFloat(FloatBits) - 1.0;
}

/* Returns float in range [-1, 1). Period length is 2^31 */
float Rand2() {
  RandSeed = (1103515245u*RandSeed + 12345u) & 0x7FFFFFFFu;

  uint FloatBits = (RandSeed >> 8u) | 0x40000000u;

  return uintBitsToFloat(FloatBits) - 3.0;
}
```

A.2. BSSRDF Interface

Instead of implementing analytic BSSRDFs inside the application code, our system makes use of an extensible system based on human-readable text files. These files carry the extension **.bssrdf* and describe attributes of the BSSRDF, expose parameters to the user interface and define analytic functions later invoked by the application.

The first line of a BSSRDF file must contain a list of attributes describing the BSSRDF. The rest of the file consists of a list of sections in arbitrary order. These sections contain either parameter

declarations parsed by the application or segments of GLSL code, which are pieced together with code from our system to produce various shaders responsible for rendering. Sections are enclosed within a `::begin ... ::end` block and must be named. If the name of a section is not recognized by the application, it is ignored.

The following paragraphs describe the attributes and sections in detail. An example BSSRDF file for the diffusion dipole is given in listing A.2.

Attributes The list of attributes currently recognized by the application are as follows:

1. *analytic*: This BSSRDF is in analytic form. This attribute is mandatory for all BSSRDF files.
2. *radial*: The BSSRDF only depends on $||x_o - x_i||$, not the relative position of x_i and x_o
3. *no_wi*: BSSRDF does not depend on incident direction
4. *no_wo*: BSSRDF does not depend on exitant direction

The *radial* attribute provides an important hint to the point cloud module, in that the BSSRDF can be represented by a radial profile. This allows the module to precompute the profile into a 1D texture and only perform texture lookups when rendering, greatly improving performance.

Parameters Section A BSSRDF may define a list of parameters exposed in the user interface. The application allows float, boolean and colour parameters. Parameter declarations are of the form

```
float [name] [minimum value] [maximum value] [default value]
bool [name] [default value]
color [name] [default red] [default green] [default blue]
```

Parameters defined this way are available as uniforms to the shader and can be referred to like any other variable in the GLSL code.

Initializer Section The *intializer* section allows the BSSRDF to define an `init()` function, which is invoked once before the first evaluation of the BSSRDF. It is useful to precompute values frequently used in the evaluation of the BSSRDF to improve performance.

Diffuse Reflection Section To support inversion of parameters from the R_{dr}, l_d format to σ_a, σ_s format, a BSSRDF must provide a `DiffuseReflectance` function, which computes the value of R_{dr} given α'. This section can be omitted, in which case the R_{dr}, l_d parametrization is disabled for this BSSRDF.

Shader Section The *shader* section contains the rest of the GLSL code and defines the analytic BSSRDF function, which computes the value of the BSSRDF given incident position/direction and exitant position/direction.

Listing A.2: *Diffusion dipole *.bssrdf*

```
 analytic radial no_wi no_wo

::begin parameters
float Multiplier 0.0 10.0 1.0
::end parameters

::begin initializer

vec3 Lu;
vec3 AlphaPrime;
vec3 SigmaTr;
float A;

void init() {
  float Fdr = -1.44/(IOR*IOR) + 0.71/IOR + 0.668 + 0.0636*IOR;
  A = (1.0 + Fdr)/(1.0 - Fdr);

  vec3 SigmaSPrime = SigmaS*(1.0 - G);
  vec3 SigmaTPrime = SigmaSPrime + SigmaA;
  SigmaTr = sqrt(3.0*SigmaA*SigmaTPrime);
  AlphaPrime = SigmaSPrime/SigmaTPrime;
  Lu = 1.0/SigmaTPrime;
}
::end initializer

::begin diffuseReflectance

vec3 DiffuseReflectance(vec3 AlphaPrime) {
  vec3 T = sqrt(3.0*(1.0 - AlphaPrime));
  return AlphaPrime*0.5*(1.0 + exp(-4.0/3.0*A*T))*exp(-T);
}

::end diffuseReflectance

::begin shader

vec3 Bssrdf(vec3 Xi, vec3 Wi, vec3 Xo, vec3 Wo) {
  vec3 R = vec3(dot(Xi - Xo, Xi - Xo));
  vec3 Zr = Lu;
  vec3 Zv = Lu*(1.0 + 4.0/3.0*A);
  vec3 Dr = sqrt(R + Zr*Zr);
  vec3 Dv = sqrt(R + Zv*Zv);
  vec3 C1 = Zr*(SigmaTr + 1.0/Dr);
  vec3 C2 = Zv*(SigmaTr + 1.0/Dv);
  vec3 FluenceR = C1*exp(-SigmaTr*Dr)/(Dr*Dr);
  vec3 FluenceV = C2*exp(-SigmaTr*Dv)/(Dv*Dv);

  return Multiplier/(4.0*PI)*AlphaPrime*(FluenceR + FluenceV);
}

::end shader
```

Bibliography

[Bre92] Richard P. Brent. Uniform random number generators for supercomputers. In *Proc. Fifth Australian Supercomputer Conference*, pages 95–104, 1992.

[Bry91] G. H. Bryan. An application of the method of images to the conduction of heat. In *Proceedings of the London Mathematical Society*, pages 424–430, 1891.

[Cha58] Subrahmanyan Chandrasekhar. *Proceedings of the National Academy of Sciences of the United States of America*, chapter 9, pages 933–940. Number v. 44. National Academy of Sciences, 1958.

[d'E12] Eugene d'Eon. A better dipole, 2012.

[DI11] Eugene D'Eon and Geoffrey Irving. A quantized-diffusion model for rendering translucent materials. *ACM Trans. Graph.*, 30(4):56:1–56:14, July 2011.

[DJ05] Craig Donner and Henrik Wann Jensen. Light diffusion in multi-layered translucent materials. *ACM Trans. Graph.*, 24(3):1032–1039, July 2005.

[dL07] Eugene d'Eon and David Luebke. Advanced techniques for realistic real-time skin rendering. *GPU Gems*, 3, 2007.

[EH79] W.G. Egan and T.W. Hilgeman. *Optical properties of inhomogeneous materials: applications to geology, astronomy, chemistry, and engineering*. Academic Press, 1979.

[Ent97] Karl Entacher. A collection of selected pseudorandom number generators with linear structures, 1997.

[FPBP09] Adrià Forés, Sumanta N. Pattanaik, Carles Bosch, and Xavier Pueyo. BRDFLab: A general system for designing BRDFs. In *Proceedings CEIG'09*. Eurographics,

2009.

[FPW92] T. J. Farrell, M. S. Patterson, and B. Wilson. A diffusion theory model of spatially resolved, steady-state diffuse reflectance for the noninvasive determination of tissue optical properties in vivo. *Medical Physics*, 19(4):879–888, July 1992.

[HK93] P. Hanrahan and W. Krueger. Reflection from layered surfaces due to subsurface scattering. In *Computer Graphics*, SIGGRAPH 93 Proceedings, pages 165–174, Anaheim, CA, aug 1993.

[Jar08] Wojciech Jarosz. *Efficient Monte Carlo Methods for Light Transport in Scattering Media*. PhD thesis, UC San Diego, September 2008.

[JB02] H. W. Jensen and J. Buhler. A rapid hierarchical rendering technique for translucent materials. In *Computer Graphics*, SIGGRAPH 2002 Proceedings, pages 576–581, Los Angeles, CA, jul 2002.

[Jen96] Henrik Wann Jensen. Global illumination using photon maps. *Rendering Techniques '96*, pages 21–30, 1996.

[Jen01] H. W. Jensen. *Realistic Image Synthesis Using Photon Mapping*. AK Peters, 2001. ISBN: 1568811470.

[JG10] Jorge Jimenez and Diego Gutierrez. *GPU Pro: Advanced Rendering Techniques*, chapter Screen-Space Subsurface Scattering, pages 335–351. AK Peters Ltd., 2010.

[JMLH01] H. W. Jensen, S. R. Marschner, M. Levoy, and P. Hanrahan. A practical model for subsurface light transport. In *Computer Graphics*, SIGGRAPH 2001 Proceedings, pages 511–518, Los Angeles, CA, aug 2001.

[JWSG10] Jorge Jimenez, David Whelan, Veronica Sundstedt, and Diego Gutierrez. Real-time realistic skin translucency. *IEEE Computer Graphics and Applications*, 30(4):32–41, 2010.

[Kaj86] James T. Kajiya. The rendering equation. In *SIGGRAPH*, pages 143–150, 1986.

[Knu81] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition*. Addison-Wesley, 1981.

[Knu98] Donald E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[Kor69] G. Kortüm. *Reflectance spectroscopy: Principles, methods, applications*. Springer, 1969.

[KS12] Nima Khademi Kalantari and Pradeep Sen. Fast generation of approximate blue noise point sets. *Computer Graphics Forum*, 31(4):1529–1535, 2012.

[LK10] Samuli Laine and Tero Karras. Efficient sparse voxel octrees. In *Proceedings of ACM SIGGRAPH 2010 Symposium on Interactive 3D Graphics and Games*, pages 55–63. ACM Press, 2010.

[LW93] Eric P. Lafortune and Yves Willems. Bi-directional path tracing. In *Compugraphics '93,*, pages 145–153, Dec 1993.

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, January 1998.

[MoEP90] J.D. Moulton and McMaster University. Dept. of Engineering Physics. *Diffusion Modelling of Picosecond Laser Pulse Propagation in Turbid Media*. McMaster University, 1990.

[NRH+77] F. Nicodemus, J. Richmond, J. Hsia, I. Ginsberg, and T. Limperis. Geometric considerations and nomenclature for reflectance. Monograph 160, National Bureau of Standards (US), October 1977.

[otICS08] Microprocessor Standards Committee of the IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. Technical report, Microprocessor Standards Committee of the IEEE Computer Society, 3 Park Avenue, New York, NY 10016-5997, USA, August 2008.

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010.

[RSC87] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, volume 21, pages 283–291, July 1987.

[Rus01] Szymon Rusinkiewicz. bv - a brdf browser. `http://www-graphics.stanford.edu/~smr/brdf/bv/`, January 2001.

[SKP09] Musawir A. Shah, Jaakko Konttinen, and Sumanta Pattanaik. Image-space subsurface scattering for interactive rendering of deformable translucent objects, 2009.

[Sta95] J. Stam. Multiple scattering as a diffusion process. In *Proceedings of the 6th Eurographics Workshop on Rendering*, pages 51–58, Dublin, Ireland, jun 1995.

[Stu10] Walt Disney Animation Studios. Partio. `http://www.disneyanimation.com/technology/partio.html`, October 2010.

[Stu11] Walt Disney Animation Studios. Brdf explorer. `http://www.disneyanimation.com/technology/brdf.html`, August 2011.

[Tur92] G. Turk. Re-tiling polygonal surfaces. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 55–64. ACM Press, 1992.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 78)*, volume 12, pages 270–274, aug 1978.

[WJZ95] Lihong Wang, Steven L. Jacques, and Liqiong Zheng. Mcml?monte carlo modeling of light transport in multi-layered tissues. *Computer Methods and Programs in Biomedicine*, 47:131–146, 1995.

[WWW58] A.M. Weinberg, P. Wigner, and E.K. Wigner. *The physical theory of neutron chain reactors*. University of Chicago Press, 1958.

[WZHB09] Bruce Walter, Shuang Zhao, Nicolas Holzschuch, and Kavita Bala. Single scattering in refractive media with triangle mesh boundaries. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 92:1–92:8, New York, NY, USA, 2009. ACM.