Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis: XXIX

Interactive Rendering of Scattering and Refraction Effects in Heterogeneous Media

Daniele Bernabei

14th May 2013

Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis: XXIX

Interactive Rendering of Scattering and Refraction Effects in Heterogeneous Media

Daniele Bernabei

Abstract. In this dissertation we investigate the problem of interactive and real-time visualization of single scattering, multiple scattering and refraction effects in heterogeneous volumes. Our proposed solutions span a variety of use scenarios: from a very fast yet physically-based approximation to a physically accurate simulation of microscopic light transmission. We add to the state of the art by introducing a novel precomputation and sampling strategy, a system for efficiently parallelizing the computation of different volumetric effects, and a new and fast version of the Discrete Ordinates Method. Finally, we also present a collateral work on real-time 3D acquisition devices.

14th May 2013

Contents

Introduction

Realistic visualization of objects is one of the most important goals in Computer Graphics research. Over the last twenty years, rendering engines have become capable of producing images that are nearly indistinguishable from real life photographs. These impressive results, however, often come at the expense of hours of computations. In recent years, since the advent of cheap, powerful, and now ubiquitous, dedicated graphic devices (GPUs), much attention has been devoted on replicating the same effects in real time on end-user machines. Although convincing renderings of objects have been produced using graphic devices, many complex lighting effects are still subject of intense study.

Most real-world materials in fact exhibit sub-surface lighting effects, wherein the appearance of an object composed of such materials is determined by the light-matter interactions that take place under the object's surface. Predicting these interactions with a simple analytical model is often impossible, since physical properties vary non-uniformly through space (heterogeneity) and the microscopic response to light varies with the direction of incoming light itself (anisotropy). These kinds of materials are extremely challenging from a computational point of view. A completely physically accurate simulation, taking into account quantum behavior of light at the atomic level is still an unrealistic proposition. Thankfully, there is a general consensus that such a precise simulation of light behavior is unnecessary in most real world scenarios.

As a convention, these materials are usually divided in two kinds: *translucent objects* and *participating media*. For the first kind it is assumed that the user will not cross the boundary of the object and that such boundary produces meaningful interactions with light that need to be modeled separately. Examples of this kind of datasets are waxy objects, human faces, marble statues, and so forth. The latter class is instead composed of those materials where it is assumed that both the light source and the viewer can be present within the medium itself. Examples of these are fog, smoke, clouds and turbid water.

Despite this classification, the equations that are used to model the volumetric behavior of light are the same for both types of materials: the Radiative Transfer Equation, an integro-differential equation in five dimensions which is very difficult to solve analytically even for extremely simple cases. Like most of the literature on the same topic, we will use this model for light transmission and introduce it accurately in chapter 1.

The aim of the present dissertation is to investigate novel algorithmic approaches to the volumetric rendering problem in order to visualize at real-time, or at near interactive speeds, the effects that light has on translucent objects and participating media. In particular, we concentrate our efforts on *scattering* effects, i.e. the retransmission of light by volume particles due to excitation from a light source (*single* scattering) or from other excited volume particles (*multiple* scattering), and *refraction*, i.e. the apparent change of direction of light when traveling through materials with variable index of refraction. We investigating the effects that different physical approximations produce on the end result, yielding solutions that range from a very fast single scattering method, to a very accurate modeling of the Radiative Transfer Equation.

In the first proposed solution (chapter 3), we present an algorithm for approximating single scattering effects in translucent materials. The solution is fast and is capable of capturing the appearance of objects that feature strong discontinuities below their surface. It converts information about the subsurface structure of the volume to a set of functions centered on points chosen by a sampling algorithm. Essential to the performance of this algorithm is a good sampling of the volumetric data which is performed by our algorithm in a novel way. While this method can produce extremely fast renderings, its need for a precomputation stage can be a limitation in some use-cases, especially if deformable models need to be supported. Therefore we set out to design an algorithm capable of operating without lengthy precomputations (chapter 4). Moreover, the single scattering limitation of the previous method limits its applicability to materials of low albedo. By incorporating a multiple scattering simulation, based on an effective approximation of the Radiative Transfer Equation, coupled with a precise photon marching algorithm for single scattering, we are able to convincingly display materials of albedo in the mid to low range at very fast speeds. Moreover, due to the nature of the marching algorithm, we are capable of incorporating refraction effects, thus greatly adding to the realism of the produced pictures. The strength of this system resides in the way we have managed to separate and parallelize the computation of multiple scattering and the single scattering ray marching algorithm, exploiting the high degree of parallelism of modern GPUs, and yielding a system capable of scaling well with future generations of hardware.

This second method however exploits a physical approximation that reduces multiple scattering to a diffusion phenomenon. This is valid in general except for strongly forward (or backward) scattering. To overcome these limitations we propose a third method wherein we adopt the Discrete Ordinates Method (DOM) formulation to accurately deal with any kind of anisotropy in addition to the heterogeneity of the previous two methods, computing accurately multiple scattering effects in general participating media and translucent objects (chapter 5). The resulting system is furthermore capable of dealing with volumes much bigger than the available GPU memory thanks to a customized paging system.

This dissertation is concluded with a detour on an algorithm dealing with noisy 3D data produced by a Kinect©acquisition device. We present a system which aids visually-impaired people in navigating indoors environments, introducing an algorithm which processes data at critical speeds and converts it to an auditory stimulus.

To summarize, the present thesis is structured as follows. In chapter 1 we review the physical background, terminology and notation used throughout the dissertation, then in chapter 2 we review the most important papers published in the last thirty years dealing with volumetric rendering, with a particular focus on those works that follow similar assumptions as the ones used throughout this thesis. The first algorithm we propose is presented in chapter 3 in which we tackle single scattering in translucent objects; while the second algorithm for multiple scattering and refraction is presented in chapter4, followed by our third DOM-based proposal in chapter 5. We conclude this dissertation in chapter 6 with our excursus on the navigational system for visually impaired people.

Chapter 1

Physics of Light Transfer

Abstract

We review the basic units of measure employed in Radiometry and the laws governing Light-Matter interaction as used in this dissertation.

Light is a specific type of electromagnetic (EM) radiation which is known to be transported by discrete elements, i.e. photons. However, even if photons at the moment of their emission and absorption exhibit particle behavior, their propagation through space is wave-like. Indeed, they are characterized by a frequency $\nu : \left[\frac{1}{\text{seconds}}\right]$, and the energy that each photon carries is

$$E = h\nu \; : \; [\text{Joules}]$$

where h is the Planck constant[1].

Visible light is simply EM radiation at the frequency range from approximately 380nm and 740nm.

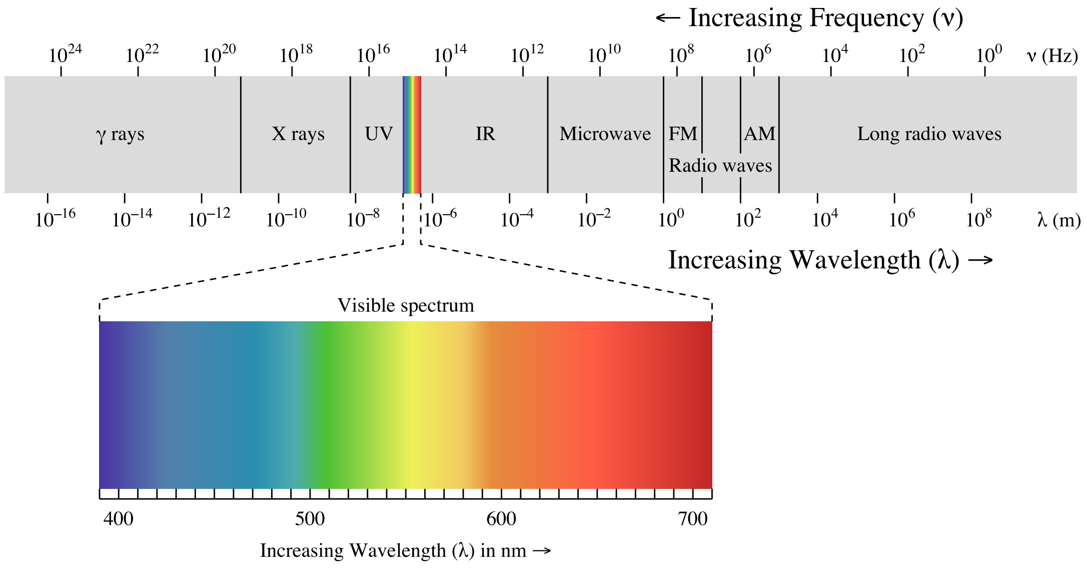


Figure 1.1: **The range of EM radiation** (source: wikipedia)

[1] $h = 6.626068 \times 10^{-34} \; : \; \left[\frac{\text{Joules}}{\text{seconds}}\right]$

Since a complete wave description of light introduces a considerable overhead both in mathematical derivations and in computations, we will follow the standard approach in Computer Graphics literature of Geometric Optics. According to Geometric Optics light propagation can be modeled as "rays". These rays can be imagined to be the paths taken by single photons, or infinitesimally small beams of light; however, it is more physically correct to define them as those geometric lines that are perpendicular to the equiphase surfaces of the EM waves, i.e. the "wavefronts"[2]. We will describe rays parametrically with equations of the type:

$$\boldsymbol{x}(t) = \boldsymbol{x}_0 + t\boldsymbol{\omega}$$

Throughout this dissertation we will indicate points $\boldsymbol{x} \in \mathbb{R}^3$ with boldface Roman letters, and we will use a round bracket notation for its components: $\boldsymbol{x} = (x, y, z)$. We will instead indicate vectors with boldface Greek letters like $\boldsymbol{\omega}$ and component notation with angled brackets: $\boldsymbol{\omega} = \langle x, y, z\rangle$. Subscripts will be used to indicate a specific component, e.g. $\boldsymbol{x}_y$. All directions indicated in this text will always be normalized (i.e. $\boldsymbol{\omega} \cdot \boldsymbol{\omega} = 1$), therefore we will sometimes use Cartesian angles ϑ and φ to indicate a direction, with the convention that $\boldsymbol{\omega} = \langle \sin\vartheta\cos\varphi,\, \sin\vartheta\sin\varphi,\, \cos\vartheta\rangle$

When it is clear from the context, and with a slight abuse of notation, a ray $\boldsymbol{x}(t)$ will be simply denoted as $\boldsymbol{x}$.

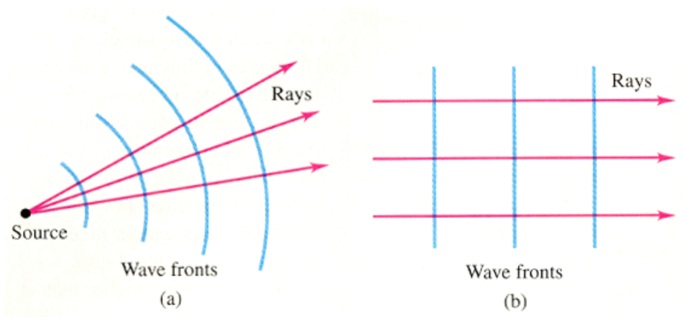


Figure 1.2: **Rays** are always perpendicular to Wave fronts

1.1 Radiometry

In order to introduce the notation used in this dissertation, we will briefly review the definitions and unit of measures of radiometry. However; we will follow the intuitive operative definitions given by Preisendorfer [106], whose rigorous axiomatic description of light is based on measure theory and sets Radiant Flux as the fundamental quantity[3].

1.1.1 Radiant Flux

Given any planar surface S (e.g. the small surface of a measurement device) and a set of directions Ξ which are visible from the surface (which can be at most the full hemisphere of visibility), the amount of energy received by the surface per unit time at time t is called Radiant Flux. Since we will be mostly dealing with steady state systems, we will drop the temporal dependence and simply denote this quantity as:

$$\Phi(S, \Xi) \;:\; [\text{Watts}] = \left[\frac{\text{Joules}}{\text{seconds}}\right]$$

[2]This ray-wavefront connection can be made explicit by the eikonal equation

[3]Sometime it is radiance, presented later, which has interesting properties, that is taken as the fundamental quantity. This is operationally clearer for many reasons.

This quantity is directly proportional to the number of photons (and the energy they carry) impinging on the surface per unit time; therefore, a more precise definition should be parametrized on the specific frequency of interest, (e.g. $\Phi(S, \Xi, \nu)$) thus allowing a description of the distribution of power across the full range of EM frequencies (or spectrum). On the other hand, since it is now accepted that almost any color can be reproduced for human consumption by a linear combination of three primary spectrally pure colors, we will follow the usual convention of representing all frequency-dependent phenomena with three mutually independent values, corresponding to pure Red, Green and Blue. This convention forces us to disregard all frequency-shifting phenomena (like fluorescence) but is nonetheless a perceptually valid approximation.

1.1.2 Irradiance

It can be experimentally verified that Radiant Flux has the following property (call it S-property): for any two surfaces S_1 and S_2,

$$\Phi(S_1, \Xi) + \Phi(S_2, \Xi) = \Phi(S_1 \cup S_2, \Xi)$$

and if we define a *measurement* of the area of the surface as $A(S)$ then

$$\text{if } A(S) = 0 \text{ then } \Phi(S, \Xi) = 0$$

We can therefore define the derived quantity Irradiance at a point $\boldsymbol{x}$ in a sound way as:

$$E(x, \Xi) = \lim_{A(s) \to 0} \frac{\Phi(S, \Xi)}{A(S)} : \left[\frac{\text{Watts}}{\text{meters}^2}\right]$$

In most situations we are interested in the entire hemisphere centered around a preferred direction $\boldsymbol{\xi}$, indicated as $\Xi^+(\boldsymbol{\xi}) = \{\boldsymbol{\omega} | \boldsymbol{\omega} \cdot \boldsymbol{\xi} > 0\}$. As a shorthand notation we can therefore write Irradiance as follows:

$$E(\boldsymbol{x}, \boldsymbol{\xi}) = E(\boldsymbol{x}, \Xi^+(\boldsymbol{\xi}))$$

In Computer Graphics textbooks, Irradiance is usually defined on the surface of a generic object. In fact, when the direction is omitted, it is implicitly assumed that $\boldsymbol{\xi} = \boldsymbol{n(x)}$, i.e. the normal to the surface at point $\boldsymbol{x}$.

$$E(\boldsymbol{x}) = E(\boldsymbol{x}, \Xi^+(\boldsymbol{n}))$$

This notation will never be used in this dissertation in order not to generate confusion with a similar notation, which is necessary to rigorously define Irradiance in free space or volumes.

The Cosine Law for Irradiance The definition of Irradiance as the ratio of flow to area has the "cosine law" as an important implication. In fact a surfaceS' placed at an angle α with respect to a surface S, if it covers an area such that $A(S') = \frac{A(S)}{\cos\alpha}$, then it will receive the exact same amount of radiant flux as S from the same fixed set of directions $\Xi(\boldsymbol{\xi})$. Therefore, the Irradiance will be less for S' than for S. More precisely,

$$E(\boldsymbol{x}', \Xi(\boldsymbol{\xi})) = \cos\alpha \cdot E(\boldsymbol{x}, \Xi(\boldsymbol{\xi})) \tag{1.1}$$

for any $\boldsymbol{x}' \in S'$, $\boldsymbol{x} \in S$.

1.1.3 Radiance

At the macroscopic level, Radiant Flux also seems to exhibit the following property (call it D-property): for any disjoint set of directions Ξ_1 and Ξ_2,

$$\Phi(S, \Xi_1) + \Phi(S, \Xi_2) = \Phi(S, \Xi_1 \cup \Xi_2)$$

moreover, if we define a measurement of a set of directions as a solid angle $\Omega(\Xi)$, then:

$$\text{if } \Omega(\Xi) = 0 \text{ then } \Phi(S, \Xi) = 0$$

There are several counter examples that show that this property does not always hold, among which interference is an excellent example. However, if we take the above property as valid we can define radiance as:

$$L(\boldsymbol{x}, \boldsymbol{\xi}) = \lim_{\Omega(\Xi)\to 0} \lim_{A(s)\to 0} \frac{\Phi(S,\Xi)}{A(S)\Omega(\Xi)} \; : \; \left[\frac{\text{Watts}}{\text{meters}^2 \cdot \text{steradians}}\right]$$

Radiance has the property of being invariant in vacuum. If we move a surface S placed at $\boldsymbol{x}$ to a different position $\boldsymbol{x} - t\boldsymbol{\xi}$ while pointing to the same infinitesimal cone of directions $\Xi(\boldsymbol{\xi})$, it will register the same amount of radiance.

Flux (or power)	Watt	$\Phi(S, \Xi)$	
Irradiance	$\frac{\text{Watt}}{\text{meter}^2}$	$E(\boldsymbol{x}, \Xi)$	$\lim_{A(s)\to 0} \frac{\Phi(S,\Xi)}{A(S)}$
Radiance	$\frac{\text{Watt}}{\text{meter}^2\cdot\text{steradian}}$	$L(\boldsymbol{x}, \boldsymbol{\xi})$	$\lim_{\Omega(\Xi)\to 0} \lim_{A(s)\to 0} \frac{\Phi(S,\Xi)}{A(S)\Omega(\Xi)}$

Table 1.1: **The Three Fundamental Radiometric Quantities**

1.1.4 Irradiance from Radiance

As a consequence of the definition of Radiance and of the cosine law1.1, Irradiance can be inversely defined as the integral of Radiance across all directions in the same hemisphere as $\boldsymbol{\xi}$, weighted by the cosine factor:

$$E(\boldsymbol{x}, \boldsymbol{\xi}) = \int_{\boldsymbol{\xi}\cdot\boldsymbol{\omega}\geq 0} L(\boldsymbol{x}, \boldsymbol{\omega}) \boldsymbol{\xi} \cdot \boldsymbol{\omega} \, d\Omega(\boldsymbol{\omega})$$

where $d\Omega(\boldsymbol{\omega})$ is a differential solid angle[4] around direction $\boldsymbol{\omega}$.

Since Irradiance is defined with respect to a direction (usually the normal of a surface at a point), direction-independent auxiliary measures are often used. In particular, *scalar Irradiance*

$$e(\boldsymbol{x}) = \int_{4\pi} L(\boldsymbol{x}, \boldsymbol{\omega}) \, d\Omega(\boldsymbol{\omega})$$

which is also called "fluence" or "energy density". In fact, it captures the notion that the amount of photons at a given point depends on the amount of light (Radiance) arriving from all direction and "passing through" the point. *Vector irradiance* is instead defined as,

$$\boldsymbol{E}(\boldsymbol{x}) = \int_{4\pi} L(\boldsymbol{x}, \boldsymbol{\omega}) \boldsymbol{\omega} \, d\Omega(\boldsymbol{\omega})$$

and another not so commonly used derived quantity is *hemispherical scalar irradiance*, defined as

$$e(\boldsymbol{x}, \boldsymbol{\xi}) = \int_{\boldsymbol{\xi}\cdot\boldsymbol{\omega}\geq 0} L(\boldsymbol{x}, \boldsymbol{\omega}) \, d\Omega(\boldsymbol{\omega})$$

All of the Irradiance-like quantities are measured in Watts per square meters.

[4] in Cartesian coordinates it would be $d\boldsymbol{\omega} = \sin\varphi \, d\varphi \, d\vartheta$

Irradiance	$E(\boldsymbol{x}, \boldsymbol{\xi})$	$\int_{\boldsymbol{\xi}\cdot\boldsymbol{\omega}\geq 0} L(\boldsymbol{x}, \boldsymbol{\omega})\boldsymbol{\xi}\cdot\boldsymbol{\omega}\, d\Omega(\boldsymbol{\omega})$
Scalar Irradiance *a.k.a. fluence*	$e(\boldsymbol{x})$	$\int_{4\pi} L(\boldsymbol{x}, \boldsymbol{\omega})\, d\Omega(\boldsymbol{\omega})$
Vector Irradiance	$\mathbf{E}(\boldsymbol{x})$	$\int_{4\pi} L(\boldsymbol{x}, \boldsymbol{\omega})\boldsymbol{\omega}\, d\Omega(\boldsymbol{\omega})$

Table 1.2: **Three Irradiance Measures**

1.2 Material Properties

1.2.1 Cross-sections

The two main properties used to describe scattering materials are the *absorption cross-section* and the *scattering cross-section*. The absorption cross-section, indicated as $\sigma_a(\boldsymbol{x})$, determines the probability that a light particle will be absorbed by the material at point x; while the scattering cross-section, indicated as $\sigma_s(\boldsymbol{x})$ determines the probability that a particle will "change direction"[5] by interacting with the material at point x. The sum of these two properties, $\sigma_t(\boldsymbol{x}) = \sigma_a(\boldsymbol{x}) + \sigma_s(\boldsymbol{x})$, is called the *extinction cross-section* of the material. The dimensions of these cross section is related to, but is not representative of[6], the actual dimensions of the atom. In fact, the unit of measure of the cross sections is the squared meter, an area measure. It is intuitive that the bigger is the area of interaction, the higher the probability that a particle will scatter or be absorbed.

$$\sigma_t(\boldsymbol{x}) \, : \, \left[\text{meter}^2\right]$$

The atom of gold, for example, has a cross section of $1.54 \cdot 10^{-24} cm^2$. Cross-sections have different effects on different ranges of the EM spectrum; therefore, for our purposes they will be represented with RGB tuples.

1.2.2 Coefficients

To obtain the scattering coefficients for a substance, the cross-sections defined above (per atom) must be multiplied by the material density $\rho(\boldsymbol{x})$. We define them as

$$\kappa_s(\boldsymbol{x}) = \sigma_s(\boldsymbol{x})\rho(\boldsymbol{x})$$

$$\kappa_a(\boldsymbol{x}) = \sigma_a(\boldsymbol{x})\rho(\boldsymbol{x})$$

$$\kappa_t(\boldsymbol{x}) = \sigma_t(\boldsymbol{x})\rho(\boldsymbol{x})$$

Scattering coefficients are measured in m^{-1}:

$$\kappa_t(\boldsymbol{x}) \, : \, \left[\frac{1}{\text{meter}}\right]$$

A useful derived quantity is the mean free path of a photon (average distance before a scattering event) which is simply $\frac{1}{k_t}$, and it is, as one could intuitively expect, measured in meters.

Another derived quantity, whose name can be misleading, is *scattering albedo*, a dimensionless quantity defined as

$$\Omega(x) = \frac{\kappa_s(\boldsymbol{x})}{\kappa_t(\boldsymbol{x})}$$

We will refer to this quantity through this dissertation simply as "albedo".

[5] More precisely, according to Quantum Electrodynamics, when a photon scatters there are no collisions. Instead there is an absorption of a quantum of EM radiation, followed by an immediate emission of radiation at the same frequency in all directions. Superposition and interference of waves then explains the apparent effect of scattering in specific directions, as modeled at a higher level by phase functions.

[6] shape of the particle and other electromagnetic properties of the atom also play a role. Sometimes it is used the term $\sigma_g(\boldsymbol{x})$ to indicate the actual geometric cross-section of the particle.

1.2.3 Index of refraction

In different materials, the apparent[7] speed of light propagation changes, a phenomenon called refraction. The index of refraction in its most general form is modeled as a complex number $\tilde{n}$ related to EM permettivity and permeability of the material. The real part of this quantity $\eta = \Re(\tilde{\eta})$ is connected to the ratio of the apparent speed v of light traveling in the material w.r.t. its speed in vacuum.

$$v = \frac{c}{\eta}$$

Although this quantity is usually presented in Computer Graphics as independent of the other quantities, including material density, the Imaginary part of η is actually related to the absorption coefficient as follows[55]:

$$\kappa_a = \frac{4\pi}{\lambda_0}\Im(\tilde{\eta})$$

where λ_0 is wavelength of light in vacuum.

Even if this is a wavelength dependent quantity like all scattering coefficients, it is often treated as a single value in most Computer Graphics literature. We will call this approximation, which precludes the computation of diffraction effects, as *scalar refraction.*

1.2.4 Phase Functions

The scattering coefficient κ_t describes only the amount of light that is out-scattered; in order to model the distribution of directions that light takes after a scattering event it is necessary to introduce phase functions. As is lamented in [14], the name "phase function" is confusing, because it has nothing to do with wave phases. We will nonetheless use this name since it is become a convention.

Phase functions describe the probability that a single photon, arriving from direction $\boldsymbol{\omega}$ will be deflected towards direction $\boldsymbol{\xi}$. Since the deflection probability usually depends only on the angle between the two directions (a phenomenon called incoherent scattering), it is customary to write phase functions as,

$$p(\boldsymbol{\omega} \cdot \boldsymbol{\xi}) \text{ or } p(\alpha)$$

All phase functions presented in this dissertation are normalized, that means that for every $\boldsymbol{\omega}$, $\int_{4\pi} p(\boldsymbol{\omega} \cdot \boldsymbol{\omega}')\, d\boldsymbol{\omega}' = 1$. A useful index of the function behavior is g, the mean cosine of the scattering angle:

$$g = \int_{4\pi} (\boldsymbol{\omega} \cdot \boldsymbol{\xi}) p(\boldsymbol{\omega} \cdot \boldsymbol{\xi})\, d\boldsymbol{\xi}$$

If g is positive the function is said to be "forward scattering", if g is negative then backwards scattering dominates. When function p is constant, then $g = 0$.

General theories like Rayleigh's yield complex functions, and the most general scattering model is provided by the Mie solutions to Maxwell's Equations in the presence of participating media. However, the level of accuracy of these models makes them inappropriate for real-time computations. Simpler and more common models are the isotropic and the Henrey-Greenstein phase functions.

Isotropic Phase Function The simplest and most common scattering phase function in Computer Graphics literature is the isotropic function:

$$p(\alpha) = k = \frac{1}{4\pi}$$

[7]This is only an apparent effect: in reality EM radiation always travels at speed c, what changes is phase velocity. This is due to phase shifts in the re-emitted waves in the excited atoms of the material.

Henrey-Greenstein Phase Function Even if originated for describing scattering of interstellar phenomena, the HG Phase function has been adopted, often inappropriately, to model many real-life materials. Its main features are ease of computation and direct use of the mean cosine in its definition.

$$p(\alpha) = \frac{1}{4\pi} \frac{1-g^2}{(1+g^2-2g\cdot\cos(\alpha))^{\frac{3}{2}}}$$

$\sigma_a(x)$	Absorption Cross-section
$\sigma_s(x)$	Scattering Cross-section
$p(\alpha)$	Phase Function
$\rho(x)$	Material Density
$\epsilon(x,\vec{\omega})$	Emissivity

Table 1.3: **Fundamental Material Properties**

$\sigma_t(x)$	$\sigma_a(x)+\sigma_s(x)$	Extinction Cross-section
$\kappa_t(x)$	$\sigma_t(x)\rho(x)$	Extinction Coefficient
Ω	$\frac{\sigma_s(x)}{\sigma_t(x)}$	Single Scattering Albedo
g	$\int_{\Xi}(\boldsymbol{\omega}\cdot\boldsymbol{\xi})p(\boldsymbol{\omega}\cdot\boldsymbol{\xi})\,d\Omega(\boldsymbol{\xi})$	First Moment of Phase Function
$\sigma_{tr}(x)$	$\sigma_t\left(1-\Omega\frac{\mu}{3}\right)$	Transport Coefficient
$\tau(x,y)$	$\int_x^y \kappa_t(z)dz$	Optical Distance
$tr(x,y)$	$e^{-d(x,y)}$	Transmittance
$\hat{\tau}(x,\vec{\omega})$	$\int_x^y \kappa_t(z)dz$	Optical Depth
$\tau_\infty(x,\vec{\omega})$		Optical Distance from Infinity

Table 1.4: **Derived Material Properties**

1.3 Light transmission

When dealing with rendering algorithms, a distinction is made between global illumination and local/direct illumination. The latter assumes that the appearance of objects is determined solely by their surface, and in turn, the color of the surface is determined solely by the light sources directly illuminating it. Global illumination, on the contrary, takes into account all possible paths that light can travel from a direct light source towards the eye, and as such it models all kinds of material interactions. Even if our solutions can be classified as Global Illumination algorithms, we are here interested in a specific subset of the macroscopic phenomena resulting from microscopic light-matter interaction. Namely, these are:

- Refraction
- Absorption
- Emission
- Out-scattering
- In-scattering

We will introduce the equations that describe singularly them and then combine them in a single equation for volumetric light transport.

1.3.1 Refraction Equations

The apparent change of speed of light waves when crossing the boundary of an object with a different refractive index, produces a change in direction of light rays. The angle of transmittance is related to the angle of incidence by *Snell's law*

$$\eta_i sin(\theta_i) = \eta_t sin(\theta_t) \tag{1.2}$$

However, a part of incoming energy will not be transmitted through the medium and will be in fact reflected back. A physically accurate calculation of the intensity of transmitted light F_t, predicted by Fresnel equations, would require the determination of the polarization of light. A common approximation consists in assuming light to be randomly polarized. Fresnel's law, under this assumption, is

$$F_t = 1 - \frac{1}{2}\left(\left(\frac{\eta_t cos(\theta_i) - \eta_i cos(\theta_t)}{\eta_t cos(\theta_i) + \eta_i cos(\theta_t)}\right)^2 + \left(\frac{\eta_i cos(\theta_i) - \eta_t cos(\theta_t)}{\eta_i cos(\theta_i) + \eta_t cos(\theta_t)}\right)^2\right) \tag{1.3}$$

If the refractive medium[8] is not constant in density or material type, then the refractive index will vary across space. This means that light waves will be constantly perturbed by superposition effects and hence light rays will not travel as straight lines. If we indicate with $\boldsymbol{s}(t)$ the tangent to the parametric ray at position $\mathbf{x}(t)$, then the eikonal equation form Gradient-index Optics predicts that:

$$\frac{d}{dt}(\eta \mathbf{s}) = \nabla \eta \tag{1.4}$$

This equation generalizes motion of light rays; in fact, it has as solution for constant refractive media the ray equation[9].

As a final note, we use the term "caustics" to denote the volumetric and surface patterns produced by refraction of light.

1.3.2 Beer-Lambert law and Optical Distance

Within any medium, the effects that arise from the encounter of external EM radiation with molecules of the solid produce an attenuation of energy along ray paths.

The *Extinction coefficient* $\kappa_t(\boldsymbol{x})$ captures these attenuation phenomena as the rate of reduction of light as described by the *Beer-Lambert law*,

$$\nabla_{\boldsymbol{\omega}} L(\boldsymbol{x}, \boldsymbol{\omega}) = -\kappa_t(\boldsymbol{x}) L(\boldsymbol{x}, \boldsymbol{\omega}) \tag{1.5}$$

This equation positively correlates the rate of reduction of light to the radiance of the light itself, multiplied by the extinction coefficient of the medium[10]. This First Order Homogeneous Ordinary Differential Equation is easy to solve by separation of variables. If we limit the discussion to a ray with starting point in $\boldsymbol{x}_0$ and we know the distribution of light at that particular point (acting as boundary condition for the equation), we can solve as follows. Parameterizing the above equation on ray $\boldsymbol{x}_0 + t\boldsymbol{\omega}$:

$$\frac{\partial L(\boldsymbol{x}, \boldsymbol{\omega})}{\partial t} = -\kappa_t(\boldsymbol{x}) L(\boldsymbol{x}, \boldsymbol{\omega})$$

Integrating from distance0 to distance t from the point $\boldsymbol{x}_0$

$$\int_0^t \frac{L'}{L} dt' = -\int_0^t \kappa_t(t') dt'$$

[8]by "refractive medium" we mean any medium with $n \neq 1$

[9]if η is constant, then $\nabla \eta = 0$ which implies $\frac{d}{dt}(\eta \mathbf{s}) = 0$, that is $\mathbf{s}(t)$ is a constant, say $\boldsymbol{\omega}$, therefore $\frac{d}{dt}\boldsymbol{x} = \boldsymbol{\omega}$ which implies $\mathbf{x}(t) = \boldsymbol{\omega} t + \mathbf{x}_0$, the straight line equation

[10]The extinction coefficient of perfect vacuum is in fact zero

gives us

$$L(\boldsymbol{x}(t'), \boldsymbol{\omega}) = L(\boldsymbol{x}_0, \boldsymbol{\omega}) e^{-\int_0^t \kappa_t(\boldsymbol{x}(t'))dt'}$$

The integral of the extinction coefficient along a straight path from x to y (in the equation above, from $\boldsymbol{x}(0)$ to $\boldsymbol{x}(t)$) is called the *optical distance* of the two points, and it is often indicated as

$$\tau(\boldsymbol{x}, \boldsymbol{y}) = \int_{\boldsymbol{x}}^{\boldsymbol{y}} \kappa_t(\boldsymbol{z})\, d\boldsymbol{z} \tag{1.6}$$

For convenience, when we consider points enclosed by a specific surface S, we will instead refer to the *optical depth* of a point. Such function is simply the optical distance between a given point inside a volume and the surface point found in a given direction,

$$\hat{\tau}(\boldsymbol{x}, \boldsymbol{\omega}) = \tau(\boldsymbol{x}, \boldsymbol{y}) \text{ s.t. } \boldsymbol{y} \in S,\, \boldsymbol{y} - \boldsymbol{x} = \boldsymbol{\omega} \tag{1.7}$$

Sometimes the whole exponential is considered and called *transmittance* or *exponential attenuation*:

$$tr(\boldsymbol{x}, \boldsymbol{y}) = e^{-\tau(\boldsymbol{x}, \boldsymbol{y})}$$

1.3.3 Volume Rendering Equation (Integro-Differential Form)

We denote the spontaneous emission of photons (for example by media like fire) with letter epsilon. This is simply the process by which light is introduced into the system.

$$\epsilon(\boldsymbol{x}, \boldsymbol{\omega})$$

However, from the perspective of the analysis of the change in energy transported along a specific rays, radiance can increase also because of scattering phenomena. The part of the energy that is not absorbed during the attenuation of other rays, is scattered in all other directions, eventually reaching a different ray. This phenomenon is called in-scattering and can be described with the following equation:

$$\nabla_{\vec{\omega}} L(\boldsymbol{x}, \boldsymbol{\omega}) = \kappa_s(\boldsymbol{x}) \int_{4\pi} p(\boldsymbol{\omega}, \boldsymbol{\omega}') L(\boldsymbol{x}, \boldsymbol{\omega}')\, d\Omega(\boldsymbol{\omega}') \tag{1.8}$$

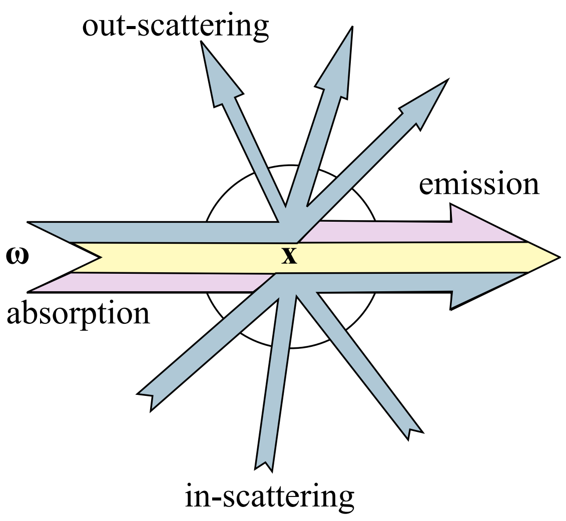


Figure 1.3: **The volumetric phenomena modeled by the RTE**

Combining equation 1.5 with equation 1.8 and adding the emission term we obtain the complete *volume rendering equation*:

$$\boxed{\nabla_{\boldsymbol{\omega}} L(\boldsymbol{x},\boldsymbol{\omega}) = \underbrace{-\kappa_t(\boldsymbol{x})L(\boldsymbol{x},\boldsymbol{\omega})}_{\text{extinction}} + \underbrace{\epsilon(\boldsymbol{x},\boldsymbol{\omega})}_{\text{emission}} + \underbrace{\kappa_s(\boldsymbol{x})\int_{4\pi} p(\boldsymbol{\omega},\boldsymbol{\omega}')L(\boldsymbol{x},\boldsymbol{\omega}')\,d\boldsymbol{\omega}'}_{\text{in-scattering}} \qquad (1.9)}$$

1.3.4 Source Function Formulation

Some texts, notably optic texts, will refer to a quantity called source function, which is just a rewriting of the quantities we have already introduced.

The source function is the ratio of the total apparent emission (in-scattering and emission) to total absorption (absorption and out-scattering), that is, using ε to indicate total emission:

$$S(x,\omega) = \frac{\varepsilon(x,\omega)}{\kappa_t(x)} = \frac{\epsilon(x,\vec{\omega}) + \kappa_s(x)\int_\Xi p(\vec{\omega},\vec{\omega}')L(x,\vec{\omega}')\,d\Omega(\vec{\omega}')}{\kappa_t(x)}$$

Note that, if there is no emission within the medium, the function reduces to

$$S(x,\omega) = \Omega(x)\int_\Xi p(\vec{\omega},\vec{\omega}')L(x,\vec{\omega}')\,d\Omega(\vec{\omega}')$$

Intuitively, the source function gives for a point and direction how much energy is available for redistribution.

Using the source function the radiative transport equation can be rewritten compactly as follows:

$$\frac{\nabla_{\vec{\omega}} L(x,\vec{\omega})}{\kappa_t(x)} = \underbrace{-L(x,\vec{\omega})}_{\text{radiance}} + \underbrace{S(x,\omega)}_{\text{source function}}$$

This separation of source term from the rest of the equation is a starting point for many computational models, including the Discrete Ordinates Method, outlined in chapter 2 and expanded in chapter 5. Note however that sometimes in literature by "source function" it is intended total emission ε.

1.3.5 Volume Rendering Equation (Integral Form)

The Volume Rendering Equation can more conveniently be rewritten as an integral equation. Since this connection is seldom made explicit, we believe it worth it to report here the complete derivation. Starting from equation 1.9:

$$\nabla_{\vec{\omega}} L(x,\vec{\omega}) = \underbrace{-\kappa_t(x)L(x,\vec{\omega})}_{\text{extinction}} + \underbrace{\epsilon(x,\vec{\omega})}_{\text{emission}} + \underbrace{\kappa_s(x)\int_\Xi p(\vec{\omega},\vec{\omega}')L(x,\vec{\omega}')\,d\Omega(\vec{\omega}')}_{\text{in-scattering}}$$

we can formulate it as a derivative over a single ray $\boldsymbol{x}(s) = \boldsymbol{x}_0 + s\boldsymbol{\omega}$ and, since the equation is on the spatial components of L only, we will omit the directional parameter $\boldsymbol{\omega}$ for the rest of the derivation

$$\frac{dL(\boldsymbol{x})}{ds} = -\kappa_t(\boldsymbol{x})L(\boldsymbol{x}) + \underbrace{\epsilon(\boldsymbol{x}) + \kappa_s(\boldsymbol{x})\int_\Xi p(\boldsymbol{\omega},\boldsymbol{\omega}')L(\boldsymbol{x},\boldsymbol{\omega}')\,d\Omega(\boldsymbol{\omega}')}_{q(\boldsymbol{x})}$$

To simplify the next passages, we will call the emission and in-scattering terms with the function name q and treat it as a constant term. The simplified differential equation is therefore of the form:

$$L' = -\kappa_t L + q$$

A first order differential equation of this type can be solved by using the artificial function $u(\boldsymbol{x})$ as an integrating factor. Multiplying both sides by this function and, moving the first term on the right to the left, we get:

$$uL' + u\kappa_t L = uq$$

Now, if u is such that $u' = u\kappa_t$, then $(uL)'$ would be exactly the left hand side of the previous equation. So in order for this to happen we need to solve $u' = u\kappa_t$. Therefore, similarly to Beer-Lambert's law, we set u as: $u(\boldsymbol{x}(t)) = e^{\int_0^t \kappa_t(\boldsymbol{x}(t'))dt'}$. We can now state that:

$$(uL)' = uq$$

and integrating both sides independently

$$\int_0^t \left(u(\boldsymbol{x}(t'))L(\boldsymbol{x}(t'))\right)' dt' = \int_0^t u(\boldsymbol{x}(t'))q(\boldsymbol{x}(t'))dt'$$

$$u(\boldsymbol{x}(t))L(\boldsymbol{x}(t)) - u(\boldsymbol{x}_0)L(\boldsymbol{x}_0) = \int_0^t u(\boldsymbol{x}(t'))q(\boldsymbol{x}(t'))dt'$$

now, since $u(\boldsymbol{x}_0) = 1$, by moving $L(\boldsymbol{x}_0)$ to the right hand side and substituting $u(\boldsymbol{x})$ with its definition,

$$e^{\int_0^t \kappa_t(\boldsymbol{x}(t''))dt''} L(\boldsymbol{x}(t)) = \int_0^t e^{\int_0^{t'} \kappa_t(\boldsymbol{x}(t''))dt''} q(\boldsymbol{x}(t'))dt' + L(\boldsymbol{x}_0)$$

To make things more clean we can divide both sides by the exponential factor, in order to leave only L on the left hand side:

$$L(\boldsymbol{x}(t)) = e^{-\int_0^t \kappa_t(\boldsymbol{x}(t''))dt''} \int_0^t e^{\int_0^{t'} \kappa_t(\boldsymbol{x}(t''))dt''} q(\boldsymbol{x}(t'))dt' + e^{-\int_0^t \kappa_t(\boldsymbol{x}(t'))dt''} L(\boldsymbol{x}_0)$$

Moving the exponential factor on the inside of the integral yields the final solution

$$L(\boldsymbol{x}(t)) = \int_0^t e^{-\int_{t'}^t \kappa_t(\boldsymbol{x}(t''))dt''} q(\boldsymbol{x}(t'))dt' + e^{-\int_0^t \kappa_t(\boldsymbol{x}(t'))dt''} L(\boldsymbol{x}_0)$$

And now, remembering how $q(\boldsymbol{x})$ has been defined,

$$L(\boldsymbol{x}(t), \boldsymbol{\omega}) = \underbrace{e^{-\int_0^t \kappa_t(\boldsymbol{x}(t'))dt''} L(\boldsymbol{x}_0, \boldsymbol{\omega})}_{\text{attenuated direct light}} +$$

$$\underbrace{+\int_0^t e^{-\int_{t'}^t \kappa_t(\boldsymbol{x}(t''))dt''} \left(\epsilon(\boldsymbol{x}(t'), \boldsymbol{\omega}) + \kappa_s(\boldsymbol{x}(t'), \boldsymbol{\omega}) \int_{4\pi} p(\boldsymbol{\omega}, \boldsymbol{\omega}')L(\boldsymbol{x}(t'), \boldsymbol{\omega}')\, d\Omega(\boldsymbol{\omega}')\right) dt'}_{\text{accumulation along ray of attenuated emission and in-scattering}}$$

Since we have already defined the convenient optical distance we can write more compactly what it is called the integral formulation of the volume rendering equation:

$$\boxed{\begin{aligned} L(\boldsymbol{x}, \boldsymbol{\omega}) \quad &= \quad \underbrace{e^{-\tau(\boldsymbol{x}_0, \boldsymbol{x})} L(\boldsymbol{x}_0, \boldsymbol{\omega})}_{\text{attenuated direct light}} \\ &+ \quad \underbrace{\int_{\boldsymbol{x}_0}^{\boldsymbol{x}} e^{-\tau(\boldsymbol{x}', \boldsymbol{x})} \left(\epsilon(\boldsymbol{x}', \boldsymbol{\omega}) + \kappa_s(\boldsymbol{x}', \boldsymbol{\omega}) \int_{4\pi} p(\boldsymbol{\omega}, \boldsymbol{\omega}')L(\boldsymbol{x}', \boldsymbol{\omega}')\, d\boldsymbol{\omega}'\right) d\boldsymbol{x}'}_{\text{accumulation along ray of attenuated emission and in-scattering}} \end{aligned}}$$

It is to be noted that in most volumetric simulation, there should be no difference between emissive media and light sources. Therefore the first term of the right hand side can be omitted, since the emission within the second term takes care of introducing light into the system.

1.3.6 Separation of Single and Multiple Scattering

Very often, in literature, scattering is artificially divided into a "single scattering" component and a "multiple scattering" one.

$$L(\boldsymbol{x},\boldsymbol{\omega}) = L_{SS}(\boldsymbol{x},\boldsymbol{\omega}) + L_{MS}(\boldsymbol{x},\boldsymbol{\omega})$$

Single scattering models light that comes directly from light sources in the medium and outside of it, i.e. external direct lighting and internal emission, and first order scattering of such light; while multiple scattering is the result of higher order scattering of single scattered light. This formulation neatly separates scattering into two distinct phenomena. In media where the albedo is low, single scattering represents a good approximation of total scattering (we will call this the "single scattering assumption"). In general cases, multiple scattering must also be accounted for.

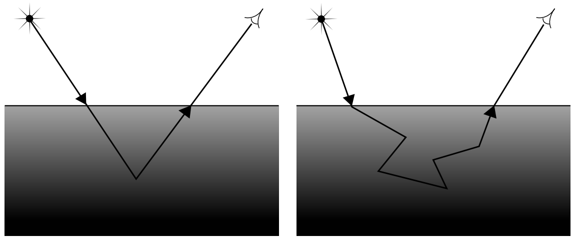


Figure 1.4: Scattering Orders

1.3.7 Bidirectional Functions

In dealing with scattering phenomena, often a clear distinction is made between participating media and translucent objects, the reason being that viewers cannot penetrate translucent objects while they can freely roam inside participating media (e.g. fog). When dealing with the former, one would be ideally interested in computing the appearance of the surface of the objects, without investigating the internal interactions of lights and matter.

1.3.7.1 BRDF

In Computer Graphics, the interaction between light and the surface of an object is modeled with the "Rendering Equation", introduced by Kajia in 1986[74]:

$$L_{out}(\boldsymbol{x},\boldsymbol{\omega}) = \int_{\boldsymbol{\omega}'\cdot\boldsymbol{n_x}>0} \underbrace{b(\boldsymbol{x},\boldsymbol{\omega},\boldsymbol{\omega}')}_{\text{brdf}} L_{in}(\boldsymbol{x},\boldsymbol{\omega}')\boldsymbol{\omega}'\cdot\boldsymbol{n_x}\,d\boldsymbol{\omega}'$$

The equation states that the outgoing radiance from the surface of an object can be expressed as the integral of the product of incoming radiance with a reflection function, weighted by a cosine factor. Reflection functions as used in the Rendering Equation are called Bidirectional Reflection Distribution Functions (BRDFs), and in fact relate the probability of a photon from direction $\boldsymbol{\omega}'$ to be reflected towards direction $\boldsymbol{\omega}$. Note that a model of this kind completely disregards any subsurface effect.

1.3.7.2 BSSRDF

When dealing with transparent objects, it is clear that outgoing light from the surface of the object at a given position is determined by light entering the system/object at any point along its surface. The equation can therefore be modified as such[70]:

$$L_{out}(\boldsymbol{y},\boldsymbol{\omega}') = \int_{\boldsymbol{x}\in\partial V}\int_{\boldsymbol{\omega}\cdot\boldsymbol{n_x}>0} \underbrace{b(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\omega},\boldsymbol{\omega}')}_{\text{bssrdf}} L_{in}(\boldsymbol{x},\boldsymbol{\omega})\boldsymbol{\omega}\cdot\boldsymbol{n_x}\,d\boldsymbol{\omega}$$

The reflection function is now called Bidirectional Subsurface Scattering Reflection Distribution Function (BSSRDF). One would ideally like to find a simple analytical bssrdf function that captures the appearance of translucent materials. However, even for homogeneous isotropic materials it is seldom possible to define such a function since it depends on the geometry of the specific object of interest.

1.3.7.3 BTDF

In order to model refractive materials, the Bidirectional Transmittance Distribution Function is sometimes used complementary to the BRDF. It specifies the amount of radiance that is instead transmitted below the surface of an object, taking into account deviation effects as modeled by Snell's law:

$$L_t(\boldsymbol{x}, \boldsymbol{\omega}) = \int_{\boldsymbol{\omega}' \cdot \boldsymbol{n_x} > 0} \underbrace{b(\boldsymbol{x}, \boldsymbol{\omega}, \boldsymbol{\omega}')}_{\text{btdf}} L_{in}(\boldsymbol{x}, \boldsymbol{\omega}') \boldsymbol{\omega}' \cdot \boldsymbol{n_x} \, d\boldsymbol{\omega}'$$

A BTDF coupled with a BRDF yields a Bidirectional Scattering Distribution Function (BSDF), not to be confused with BSSRDFs.

Chapter 2

Literature Review

Abstract

In this chapter we review the state of the art in rendering scattering and refraction effects, with a particular focus on some of the seminal works in this field and on the approaches most closely related to our work, namely the analytical solutions to Single Scattering, the Discrete Ordinates Method and its variants, and the Diffusion approximation, which is also relevant by itself for its wide adoption in the Computer Graphics community to solve the Multiple Scattering problem. We review some of the most interesting derivations of these methods, in order to render this volume as self contained as possible.

Research in scattering and refraction rendering in the last thirty years is vast, and different taxonomies have been proposed to classify this impressive body of work[1]. In this chapter we choose to finely categorize methods according to the main underlying paradigms, be it a physical model or a seminal algorithm.

The pioneering work in this field was carried out by Blinn[13], who gave analytical formulations for specific basic cases, e.g. a uniformly scattering medium with low albedo and a single light source. Later, Pattanaik et al.[99] employed the more general Monte Carlo (MC) path tracing approach, which had been long used for solving problems in particle transport. Their solution was the first to tackle the problem at a global level, and several other stochastic approaches followed. Among these, it is worth mentioning in particular the bidirectional path tracing approach of Lafortune and Willems[83] and the Metropolis Light Transport approach of Pauly et al.[100]. All of the aforementioned methods, despite being physically accurate and general, are suitable mostly for off-line rendering since the number of samples that they require in order to reduce noise in final images is extremely high. Hanrahan and Krueger [56] provided a restricted computationally more efficient model, which limits light transport in translucent materials composed of stratified layers.

Photon Mapping, introduced by Jensen et al.[71] greatly reduce the number of samples needed since light is first traced from the sources to the scene. However, only recently McGuire et al.[93] proposed an approximation of this method that is capable of running at interactive speeds, which however does not keep scattering effects into account.

The Zonal Method, proposed in thermal engineering by Hottel[62], is instead a completely deterministic approach. It was introduced by Rushmeier et al.[112] to the Computer Graphics community, and similarly to the Radiosity method, it iteratively computes the exchange of energy between finite elements of the scene. Contrarily to Radiosity, these finite elements are volume elements instead of surface patches, but it nonetheless requires the setting up of a large matrix correlating all elements to each other, thus imposing large memory requirements. While the method itself is designed for isotropic phase functions, Rushmeier et al.[111] later proposed an enhancement supporting weakly anisotropic materials. This method has also been extended by Sillion et al.[115]

[1]E.g., volumetric vs. surface-based vs. image-space[20], single vs. multiple scattering, online vs. offline, ray tracing vs. rasterization[8]

with a hierarchical approach in order to mitigate computational complexity, and by Bhate et al.[9] who employ a combination of the zonal method and spherical harmonics expansion.

Spherical Harmonics are widely adopted as a basis for representing spherical functions, and the algorithm we present in chapter3 is based on this fact. When they are used to model incoming light and arbitrary phase functions they give rise to the P_N family of methods, first introduced to the Computer Graphics community by Kajiya in 1984[75].

Radiosity & Zonal Method
[62, 112, 9, 40]
Monte Carlo Path Tracing / Photon Mapping
[71, 89, 37, 105, 82]

Table 2.1: Traditional paradigms applied to scattering & refraction

Another approach for dealing with anisotropic behavior of general phase functions, is the Discrete Ordinates Method[19], originated within the thermal engineering community. As reported by Barichello[5], under appropriate circumstances, this method provides the same level of accuracy as the P_N method, and it has become one of the most used methods due to its high generality. It will be further explained in section 2.1.

Another important line of work in scattering comes form the Diffusion Approximation (section 2.2), whose concept is highly relevant for the Lattice Boltzmann Lighting method, explained in the same section and expanded in chapter 4. From the diffusion approximation a large body of work has sprung out following the dipole approximation, which for its impact alone is worth mentioning in this dissertation and is thus presented in section 2.3. Precomputed Radiance Transfer is presented in section 2.6 and while its use of spherical harmonics for pre-computing transmission of light is not directly applied in our work, it has still influenced the method presented in chapter 3. Finally, since our work in chapter 3 is focused on single scattering computations, the most relevant analytical solutions are presented in section 2.5. An orthogonal body of work which is related to every chapter of our dissertation is the one in volumetric rendering, briefly outlined in chapter 2.4.

Interestingly, refraction is often treated as a separate topic from radiative transfer; the relevant algorithms will be highlighted in section 2.7.

Discrete Ordinates Method (§ 2.1)
[19, 123, 84, 41, 76, 15, 11]
Diffusion Approximation (§ 2.2)
[19, 66, 119, 54, 137, 128, 130, 3, 88, 47, 48, 120, 49]
Dipole (§ 2.3)
[70, 69, 86, 95, 27, 57, 129, 94, 34, 135, 113, 118, 20]
Precomputed Radiance Transfer (§ 2.6)
[117, 116, 135, 96]
Single Scattering Integral & Closed form Solutions (§ 2.5)
[98, 13, 56, 121, 46, 136, 103, 101, 102, 10, 39]
Volumetric Rendering & Splatting (§ 2.4)
[33, 80, 110, 65, 60, 32, 59]

Table 2.2: State of the art methods in Interactive Scattering & Refraction

A thorough classification of general approaches for scattering, up to 2005, can be found in the work of Cerezo et al.[18]. In this chapter we will focus on recent advances and in particular on those methods that are most closely related to our line of work.

2.1 Discrete Ordinates Method

One of the most widely adopted method for computing the Radiative Transfer Equation, mainly because of its tractability and generality, is the Discrete Ordinates Method (DOM). In this method, 3D space is discretized into finite elements and the 4π solid angle is divided into a discrete number m of solid angles, each corresponding to a principal direction $\boldsymbol{\omega}^m$. The RTE is then rewritten as a finite differences equation for the average radiance over the discrete directions, and solid angle integration is substituted with an appropriate quadrature scheme. The resulting method allows to treat any phase function and is capable of handling robustly any irregularity in the volumetric properties of objects.

This method, or family of methods, has long been used within different communities, e.g. thermal engineering, fluid mechanics and neutron transport[2]. Its main advantage lays in reducing a global process to local easily-parallelizable interactions without losing in physical accuracy. The advantage can be readily seen if contrasted with a Radiosity-like method which needs, in order to operate, to precompute all form-factors between every pair of elements.

2.1.1 Discretization of the RTE

If we consider only a finite set of M directions $\boldsymbol{\omega}^m =< \mu^m, \zeta^m, \eta^m >$ the radiative transfer equation can be written as:

$$\mu^m \frac{\partial L(\boldsymbol{x}, \boldsymbol{\omega}^m)}{\partial x} + \zeta^m \frac{\partial L(\boldsymbol{x}, \boldsymbol{\omega}^m)}{\partial y} + \eta^m \frac{\partial L(\boldsymbol{x}, \boldsymbol{\omega}^m)}{\partial z} = \underbrace{-\kappa_t(\boldsymbol{x}) L(\boldsymbol{x}, \boldsymbol{\omega}^m)}_{\text{extinction}} + \underbrace{\epsilon(\boldsymbol{x}, \boldsymbol{\omega}^m)}_{\text{emission}} + \underbrace{\kappa_s(\boldsymbol{x}) \sum_{m'=1}^{M} p^{m,m'} L(\boldsymbol{x}, \boldsymbol{\omega}^{m'})\, w^{m'}}_{\text{in-scattering}}$$

Where $p^{m,m'}$ are discretized phase function coefficients, and w^m are quadrature weights corresponding to the solid angle associated with direction $\boldsymbol{\omega}^m$. Discretizing also with respect to space, we obtain:

$$\mu^m \frac{L(\boldsymbol{x} + h\hat{\boldsymbol{\imath}}, \boldsymbol{\omega}^m) - L(\boldsymbol{x} - h\hat{\boldsymbol{\imath}}, \boldsymbol{\omega}^m)}{2h} + \zeta^m \frac{L(\boldsymbol{x} + h\hat{\boldsymbol{\jmath}}, \boldsymbol{\omega}^m) - L(\boldsymbol{x} - h\hat{\boldsymbol{\jmath}}, \boldsymbol{\omega}^m)}{2h} + \eta^m \frac{L(\boldsymbol{x} + h\hat{\boldsymbol{k}}, \boldsymbol{\omega}^m) - L(\boldsymbol{x} - h\hat{\boldsymbol{k}}, \boldsymbol{\omega}^m)}{2h} =$$

$$= -\kappa_t(\boldsymbol{x}) L(\boldsymbol{x}, \boldsymbol{\omega}^m) + \epsilon(\boldsymbol{x}, \boldsymbol{\omega}^m) + \kappa_s(\boldsymbol{x}) \sum_{m'=1}^{M} p^{m,m'} L(\boldsymbol{x}, \boldsymbol{\omega}^{m'})\, w^{m'}$$

If cells have dimensions $\Delta y = \Delta z = \Delta x = 2h$ then, integrating on the volume $V = \Delta x \Delta y \Delta z$ of the cell, and explicitly indicating the set of the eight faces of a cell as $F = \{X^+, X^-, Y^+, Y^-, Z^+, Z^-\}$,

$$\mu^m A(L^m_{X^+} - L^m_{X^-}) + \zeta^m A(L^m_{Y^+} - L^m_{Y}) + \eta^m A(L^m_{Z^+} - L^m_{Z^-}) =$$

$$= -\kappa_t(\boldsymbol{x}) V L(V_{\boldsymbol{x}}, \boldsymbol{\omega}^m) + V \left(\epsilon(V_{\boldsymbol{x}}, \boldsymbol{\omega}^m) + \kappa_s(\boldsymbol{x}) \sum_{m'=1}^{M} p^{m,m'} L(V_{\boldsymbol{x}}, \boldsymbol{\omega}^{m'})\, w^{m'} \right)$$

where $A = 4h^2$ is the area of a voxel face, and with a slight abuse of notation, we used $\epsilon(V_{\boldsymbol{x}}, \boldsymbol{\omega}^m)$ and $L(V_{\boldsymbol{x}}, \boldsymbol{\omega}^{m'})$ to indicate respectively the average emission and average radiance over an entire voxel. Notice that the quantity within round parenthesis is $\kappa_t(\boldsymbol{x}) S(V_{\boldsymbol{x}}, \boldsymbol{\omega}^m)$, where S is exactly the source function as described in section 1.3.4.

Once boundary conditions are given, this system can be solved numerically by iteration. Each iteration provides the equivalent of one scattering as it propagates radiance from a cell to its neighbors. Gortler et al.[53] further provided different physical interpretations of alternative solving methods.

A large part of the literature about the DOM family is concerned with two important details of this discretization. The first regards the quadrature scheme used to choose the M directions and

[2]Note that when the quadrature scheme is based on the zeroes of Legendre Polynomials, then the method is often referred to as S_N method, surprisingly equivalent to the P_N method briefly outlined in the introduction.

the associated weights. A good quadrature scheme must be symmetric about all axes and energy preserving. The second regards the closure scheme related to the spatial discretization, examples of which are the step, diamond and CLAM scheme. In the remainder of this dissertation we will employ the popular diamond scheme[23].

2.1.2 Modified DOM and SHDOM

The Discrete Ordinates Method is not immune from limitations; much effort has been devoted into solving the two resulting main errors, namely false scattering and the ray effect.

False scattering is a form of numerical smearing which arises due to the spatial discretization of the volume. In fact, radiance transferred from a cell to the next along a specific direction is repeatedly averaged across the faces of the cell. This effect makes it impossible for rays traversing the medium to maintain a sharp profile.

The “ray effect” is instead due to the angular discretization, since energy is forced to be transmitted across specific directions. The name of this systematic error is due to the appearance of spurious beams of energy emanating from irradiated zones. Among the several solutions that have been proposed in literature to overcome these effects, the MDOM (Modified DOM) is one that has gained popularity[23, 109]. This method is based on the assumption that since basic DOM is not suited for maintaining high frequency detail, the direct component of radiation should be computed separately from the indirect component. This idea has been taken up in Computer Graphics by Languenou[84] and it forms the basis of the work we present in chapter 5 where the derivation of Languenou, adapted to our system, is presented.

The Spherical Harmonics DOM[41] switches from a discrete ordinates representation of radiance to a Spherical Harmonics one, employing the latter when computing the source function and the former when streaming radiance to a voxel to its neighbor. In this method Evans[41] also employed adaptive resolution, in order to provide extra accuracy where needed.

Another interesting development of DOM in Computer Graphics has come from Fattal[45] which introduced a decoupling of the angular resolution for propagation and of the angular resolution for storage. Moreover, since radiance is transported by a moving bidimensional map of rays, false scattering is greatly reduced. It is to be noted that this propagation mechanism is akin to the Long Characteristic (LC) computation in the Method of Characteristics (MOC).

Despite the large number of variants of the DOM, very few attempts have been made to port this family of algorithms to the GPU. The method we present in chapter 5 is one of these few. The reason why the Discrete Ordinates Method (DOM) has not been widely adopted within the Computer Graphics community is due to its large memory requirements, which do not make it amenable to a real-time GPU implementation. Only recently some approaches have been suggested to properly parallelize this method, most notably Gong et al.’s[52] who adapted the *sweep3d* variant introduced by Koch et al.[81] We decided to follow a similar approach and verify the feasibility of performing a full RTE simulation at interactive frame rates.

2.1.3 Light Propagation Volumes

Recently, Kaplanyan et al.[76] introduced a system to compute Global Illumination that is based on the ideas of the Discrete Ordinates Method. This method exploits several state of the art ideas like Virtual Point Lights, introduced by Keller et al [77] in their seminal Instant Radiosity paper, and has proven its capability of handling large fully dynamic scenes by being implemented in a popular commercial rendering engine.

In the system of Kaplanyan et al. grids at different resolution are used to represent indirect lighting (thus separating direct from indirect as in MDOM) and scene geometry. Within each cell, the spherical function is represented using a low frequency Spherical Harmonics approximation.

This method has been shown independently by Christensen[15] and Billeter[11] to be able to handle subsurface scattering.

2.2 Multiple Scattering as a Diffusion Process

By empirically studying the behavior of light under multiple scattering it becomes apparent that light distribution in a highly scattering material (i.e. with high albedo) tends to become isotropic. This implies that the directional information tends to be lost and therefore it is reasonable to avoid performing complex computations involving the full radiance distribution from the outset. This behavior of light can be also seen as a diffusion process: photons move in a highly scattering media according to a process similar to the one that governs diffusion of heat in a solid or diffusion of solvent in a solution, i.e.:

$$\frac{\partial \Phi}{\partial t} = \underbrace{D\nabla^2\Phi}_{\text{diffusion}} + \underbrace{S(x)}_{\text{source term}}$$

The Diffusion equation for light, also known as the Photon Diffusion Equation, must also account for the absorption process as described in the previous chapter, and it is therefore a differential equation of the form:

$$\frac{\partial \Phi}{\partial t} = \underbrace{D\nabla^2\Phi}_{\text{diffusion}} - \underbrace{\kappa_a\Phi}_{\text{absorption}} + \underbrace{S(x)}_{\text{source term}}$$

The most widely used diffusion equation in Computer Graphics is a steady-state variant of the form above; however, from a Lattice-Boltzmann model of light transmission it is possible to derive a time-varying equation which is particularly relevant to the work we present in chapter 4.

2.2.1 The Diffusion Equation

Although already known in different communities[19, 66], in 1995 Stam[119] introduced to Computer Graphics the diffusion approximation and the equation governing it. Since then, most methods accounting for Multiple Scattering in translucent objects and participating media have dealt either with this equation directly or with a simplified model, which we present in the next chapter. For this reason, and for the insights that it provides on light transmission that are exploited in chapter 4, we report here the full derivation. Furthermore, a complete and detailed explanation of such derivation is seldom found.

2.2.1.1 Considerations on Vector Irradiance

As a preliminary step, it is necessary to make some general considerations regarding Irradiance that can be directly derived from the integro-differential volume rendering equation. Starting with this equation again

$$\boldsymbol{\omega} \cdot \nabla L(\boldsymbol{x}, \boldsymbol{\omega}) = \underbrace{-\kappa_t(\boldsymbol{x})L(\boldsymbol{x}, \boldsymbol{\omega})}_{\text{extinction}} + \underbrace{\epsilon(\boldsymbol{x}, \boldsymbol{\omega})}_{\text{emission}} + \underbrace{\kappa_s(\boldsymbol{x}) \int_{4\pi} p(\boldsymbol{\omega}, \boldsymbol{\omega}')L(\boldsymbol{x}, \boldsymbol{\omega}')\, d\boldsymbol{\omega}'}_{\text{in-scattering}} \tag{2.1}$$

and recalling how we defined the accessory quantities *vector irradiance* $\boldsymbol{E}(\boldsymbol{x}) = \int_{4\pi} L(\boldsymbol{x}, \boldsymbol{\omega})\boldsymbol{\omega}\, d\boldsymbol{\omega}$ and *scalar irradiance* $e(\boldsymbol{x}) = \int_{4\pi} L(\boldsymbol{x}, \boldsymbol{\omega})\, d\boldsymbol{\omega}$ (which, if you divide it by 4π, can be seen as "mean radiance"). A useful property of directional derivatives is that they can be changed to divergence operations $\boldsymbol{\omega} \cdot \nabla L(\boldsymbol{x}, \boldsymbol{\omega}) = \nabla \cdot (\boldsymbol{\omega} L(\boldsymbol{x}, \boldsymbol{\omega}))$ which is easily verifiable[3]. So by making this switch, and integrating over the unitary sphere the left-hand side of the equation becomes[4]:

$$\int_{4\pi} \nabla \cdot (\boldsymbol{\omega} L(\boldsymbol{x}, \boldsymbol{\omega}))\, d\boldsymbol{\omega} = \nabla \cdot \int_{4\pi} \boldsymbol{\omega} L(\boldsymbol{x}, \boldsymbol{\omega})\, d\boldsymbol{\omega} = \nabla \cdot E(\boldsymbol{x})$$

[3]This is because in this case nabla operates on space, therfore directions are treated as constants that can be moved inside or outside the differentiation operation. Look for example at the x component and notice in fact that $\frac{\partial(\boldsymbol{\omega}_x L(\boldsymbol{x},\boldsymbol{\omega}))}{\partial x} = \omega_x \frac{\partial L(\boldsymbol{x},\boldsymbol{\omega})}{\partial x}$

[4]Here we exploit the fact that integration is on directions while differentiation is on space and therefore they can be safely exchanged in order

Now, by applying the same integration on the right hand side of the volume rendering equation, we discover that:

$$\nabla \cdot E(\boldsymbol{x}) = \underbrace{-\kappa_t(\boldsymbol{x})e(\boldsymbol{x})}_{\text{extinction}} + \underbrace{\int_{4\pi} \epsilon(\boldsymbol{x},\boldsymbol{\omega})d\boldsymbol{\omega}}_{\text{emission}} + \underbrace{\kappa_s(\boldsymbol{x})e(\boldsymbol{x})}_{\text{in-scattering}}$$

The phase function is normalized to yield 1 upon integration over the sphere, therefore by integrating first with respect to $\boldsymbol{\omega}$ instead of $\boldsymbol{\omega}'$ (this switch is possible since they are two indipendent varibales) you directly obtain $e(\boldsymbol{x})$. Now, remembering that $\kappa_t = \kappa_s + \kappa_a$, it is possible to compactly write:

$$\underbrace{\nabla \cdot E(\boldsymbol{x})}_{\text{divergence}} = \underbrace{-\kappa_a e(\boldsymbol{x})}_{\text{sinks}} + \underbrace{\int_{4\pi} \epsilon(\boldsymbol{x},\boldsymbol{\omega})d\boldsymbol{\omega}}_{\text{sources}} \tag{2.2}$$

This result may be backed up by intuition. The divergence of a vector field is positive where there is influx and negative where there is outflux. The two process that influence divergence of irradiance are therefore those that add irradiance or remove irradiance: absorption and emission. Scattering will only change direction but not the total energy circulating in the system. On the other side, in a medium with no absorption and no emission, the divergence is zero: if many photons enter the medium, the same amount will exit, none will be created or absorbed.

2.2.1.2 Taylor Approximation to Radiance

Kajiya and Von Herzen[75] approximated radiance using the first two levels of spherical harmonics expansions but did not notice the transformation of propagation of light in a diffusion process. Stam[119] instead made more clear that by using the first two terms of the Taylor expansion of radiance you invariably obtain a diffusion process. This interesting result is the one that will be presented here. By performing a Taylor expansion of radiance in the directional component only you obtain:

$$L(\boldsymbol{x},\boldsymbol{\omega}) \approx L(\boldsymbol{x},\boldsymbol{\xi}) + (\boldsymbol{\omega} - \boldsymbol{\xi}) \cdot \nabla L(\boldsymbol{x},\boldsymbol{\xi}) \tag{2.3}$$

By "directional component only" it means that above and in the following steps of the derivation $\boldsymbol{x}$ should be treated like a "constant", and the nabla operates on directions only, i.e. $\nabla = \langle \frac{\partial}{\partial\vartheta}, \frac{\partial}{\partial\varphi} \rangle$. Under this approximation, scalar irradiance becomes:

$$e(\boldsymbol{x}) \approx \int_{4\pi} \left(L(\boldsymbol{x},\boldsymbol{\xi}) + (\boldsymbol{\omega} - \boldsymbol{\xi}) \cdot \nabla L(\boldsymbol{x},\boldsymbol{\xi})\right) d\boldsymbol{\omega} = 4\pi L(\boldsymbol{x},\boldsymbol{\xi}) - 4\pi\boldsymbol{\xi}\nabla L(\boldsymbol{x},\boldsymbol{\xi})$$

while vector irradiance becomes (refer to the integration identities in the first chapter):

$$\boldsymbol{E}(\boldsymbol{x}) \approx \int_{4\pi} \boldsymbol{\omega} \cdot \left(L(\boldsymbol{x},\boldsymbol{\xi}) + (\boldsymbol{\omega} - \boldsymbol{\xi}) \cdot \nabla L(\boldsymbol{x},\boldsymbol{\xi})\right) d\boldsymbol{\omega} = \frac{4\pi}{3}\nabla L(\boldsymbol{x},\boldsymbol{\xi})$$

Combining these two equations together you get:

$$L(\boldsymbol{x},\boldsymbol{\xi}) \approx \underbrace{\frac{1}{4\pi}e(\boldsymbol{x})}_{\text{diffusive term}} + \underbrace{\frac{3}{4\pi}\boldsymbol{\xi} \cdot \boldsymbol{E}(\boldsymbol{x})}_{\text{directional term}} \tag{2.4}$$

Intuitively, radiance is approximated with its average across all directions (the first term) and corrected by comparing it to the "average direction" of transmission (the second term). To store the radiance function at a point $\boldsymbol{x}$ according to this approximation you would just need a single scalar value $e(\boldsymbol{x})$ and a single vector value $\boldsymbol{E}(\boldsymbol{x})$.

2.2.1.3 The Gradient of Scalar Irradiance

We want to explore the gradient of the scalar irradiance under the above approximation. Therefore from now on we will revert to using ∇ to indicate differentiation in space, not on directions: $\nabla =< \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} >$ and return to using $\boldsymbol{\omega}$ again to indicate the unknown direction instead of $\boldsymbol{\xi}$.

Let us substitute the previous equation (2.4) in the regular volume rendering equation. What you get on the left hand side is directly

$$\boldsymbol{\omega} \cdot \nabla L(\boldsymbol{x}, \boldsymbol{\omega}) \approx \boldsymbol{\omega} \cdot \nabla \left(\frac{1}{4\pi} e(\boldsymbol{x}) + \frac{3}{4\pi} \boldsymbol{\omega} \cdot \boldsymbol{E}(\boldsymbol{x}) \right) =$$

$$= \frac{1}{4\pi} \boldsymbol{\omega} \cdot \nabla e(\boldsymbol{x}) + \frac{3}{4\pi} \boldsymbol{\omega} \cdot \nabla \left(\boldsymbol{\omega} \cdot \boldsymbol{E}(\boldsymbol{x}) \right)$$

while for the right hand side, let us concentrate on the in-scattering term first.

Remember again that the phase function has been normalized ($\int_{4\pi} p(\boldsymbol{\omega} \cdot \boldsymbol{\omega}')\, d\boldsymbol{\omega}' = 1$) and that those quantities that do not depend on angle can be easily taken outside of integrals on directions:

$$\kappa_s(\boldsymbol{x}) \int_{4\pi} p(\boldsymbol{\omega}, \boldsymbol{\omega}') \left(\frac{1}{4\pi} e(\boldsymbol{x}) + \frac{3}{4\pi} \boldsymbol{\omega}' \cdot \boldsymbol{E}(\boldsymbol{x}) \right) d\boldsymbol{\omega}' =$$

$$= \frac{1}{4\pi} \kappa_s(\boldsymbol{x}) e(\boldsymbol{x}) + \kappa_s(\boldsymbol{x}) \frac{3}{4\pi} \boldsymbol{E}(\boldsymbol{x}) \cdot \underbrace{\int_{4\pi} p(\boldsymbol{\omega}, \boldsymbol{\omega}') \boldsymbol{\omega}'\, d\boldsymbol{\omega}'}_{\boldsymbol{G}(\boldsymbol{\omega})}$$

Now let us consider the vector quantity that has been labeled $\boldsymbol{G}(\boldsymbol{\omega})$. Since our phase functions are anisotropic, this vector quantity must have direction $\boldsymbol{\omega}$. This is because, when integrating on the sphere, directions $\boldsymbol{\omega}'$ at the same angle with respect to $\boldsymbol{\omega}$ will contribute the same amount and therefore their directional components will cancel each other out.

Now if take the dot product of $\boldsymbol{G}(\boldsymbol{\omega})$ with vector $\boldsymbol{\omega}$, which being a constant vector can be taken inside the integral, we would obtain g, the mean cosine of the scattering angle (remember the definition given in chapter 1).

$$g = \boldsymbol{G}(\boldsymbol{\omega}) \cdot \boldsymbol{\omega}$$

Since $\boldsymbol{G}(\boldsymbol{\omega})$ has direction $\boldsymbol{\omega}$, the above relation implies that the length of vector $\boldsymbol{G}(\boldsymbol{\omega})$ is g. Therefore we can conclude that:

$$\boldsymbol{G}(\boldsymbol{\omega}) = g\boldsymbol{\omega}$$

Including the emission term, which remains unvaried, and the extinction term, obtained by direct substitution, what you get therefore is:

$$\underbrace{\frac{1}{4\pi} \boldsymbol{\omega} \cdot \nabla e(\boldsymbol{x}) + \frac{3}{4\pi} \boldsymbol{\omega} \cdot \nabla \left(\boldsymbol{\omega} \cdot \boldsymbol{E}(\boldsymbol{x}) \right)}_{\text{was } \boldsymbol{\omega} \cdot \nabla L(\boldsymbol{x}, \boldsymbol{\omega})} =$$

$$= \underbrace{-\frac{1}{4\pi} \kappa_t(\boldsymbol{x}) e(\boldsymbol{x}) - \frac{3}{4\pi} \kappa_t(\boldsymbol{x}) \boldsymbol{\omega} \cdot \boldsymbol{E}(\boldsymbol{x})}_{\text{extinction}} + \underbrace{\frac{1}{4\pi} \kappa_s(\boldsymbol{x}) e(\boldsymbol{x}) + \frac{3}{4\pi} g \kappa_s(\boldsymbol{x}) \boldsymbol{E}(\boldsymbol{x}) \cdot \boldsymbol{\omega}}_{\text{in-scattering}} + \underbrace{\epsilon(\boldsymbol{x}, \boldsymbol{\omega})}_{\text{emission}}$$

Now, multiply by $\boldsymbol{\omega}$ and integrate over 4π. All non direction dependent terms disappear; most of the other terms simplify quickly according to the rule $\int \boldsymbol{\omega}(\boldsymbol{\omega} \cdot \boldsymbol{\xi}) = \frac{4\pi}{3} \boldsymbol{\xi}$. More effort is instead required to show that $\int \boldsymbol{\omega} \left(\frac{3}{4\pi} \boldsymbol{\omega} \cdot \nabla \left(\boldsymbol{\omega} \cdot \boldsymbol{E}(\boldsymbol{x}) \right) \right) = 0$. What you finally get is:

$$\frac{1}{3} \nabla e(\boldsymbol{x}) + 0 = 0 - \kappa_t(\boldsymbol{x}) \boldsymbol{E}(\boldsymbol{x}) + 0 + g \kappa_s(\boldsymbol{x}) \boldsymbol{E}(\boldsymbol{x}) + \int_{4\pi} \epsilon(\boldsymbol{x}, \boldsymbol{\omega}) \boldsymbol{\omega}\, d\boldsymbol{\omega}$$

Or, equivalently:

$$\nabla e(\boldsymbol{x}) = 3\left(g\kappa_s(\boldsymbol{x}) - \kappa_t(\boldsymbol{x})\right)\boldsymbol{E}(\boldsymbol{x}) + 3\int_{4\pi} \epsilon(\boldsymbol{x},\boldsymbol{\omega})\boldsymbol{\omega}\, d\boldsymbol{\omega}$$

The quantity $\kappa_s(\boldsymbol{x})(1-g) + \kappa_a(\boldsymbol{x})$ effectively approximates the anisotropic behavior of phase functions: it will be called reduced extinction coefficient and we will indicate it with $\kappa_t'(\boldsymbol{x})$.

$$\nabla e(\boldsymbol{x}) = -3\kappa_t'(\boldsymbol{x})\boldsymbol{E}(\boldsymbol{x}) + 3\int_{4\pi} \epsilon(\boldsymbol{x},\boldsymbol{\omega})\boldsymbol{\omega}\, d\boldsymbol{\omega} \tag{2.5}$$

2.2.1.4 The Diffusion Equation

Now that we know the gradient of the scalar irradiance in terms of the vector irradiance, and the divergence of the vector irradiance in terms of the scalar radiance, we can combine equation 2.5 and 2.2 to get an equation for scalar irradiance only. Let us introduce $D = \frac{1}{3\kappa_t'}$ and rewrite equation 2.5:

$$\boldsymbol{E}(\boldsymbol{x}) = -D(\boldsymbol{x})\nabla e(\boldsymbol{x}) + 3D(\boldsymbol{x})\int_{4\pi} \epsilon(\boldsymbol{x},\boldsymbol{\omega})\boldsymbol{\omega}\, d\boldsymbol{\omega}$$

Now this can be plugged in equation 2.2, yielding

$$\nabla \cdot D(\boldsymbol{x})\nabla e(\boldsymbol{x}) = \kappa_a(\boldsymbol{x})e(\boldsymbol{x}) - \underbrace{\int_{4\pi} \epsilon(\boldsymbol{x},\boldsymbol{\omega})\, d\boldsymbol{\omega}}_{q(\boldsymbol{x})} + 3\nabla \cdot D(\boldsymbol{x})\underbrace{\int_{4\pi} \epsilon(\boldsymbol{x},\boldsymbol{\omega})\boldsymbol{\omega}\, d\boldsymbol{\omega}}_{\boldsymbol{Q}(\boldsymbol{x})} \tag{2.6}$$

This equation describes a diffusion process and it can be more clearly seen in the case of constant coefficients:

$$\boxed{D\nabla^2 e(\boldsymbol{x}) = \kappa_a e(\boldsymbol{x}) - \underbrace{q(\boldsymbol{x}) + 3\nabla \cdot \boldsymbol{Q}(\boldsymbol{x})}_{S(\boldsymbol{x})}}$$

By adding proper boundary conditions, a system can be set up and solved to obtain the spatial distribution of scalar irradiance.

2.2.2 Solvers to the Diffusion Equation

When Stam[119] introduced the equation he contextually proposed a multigrid approach to solve the resulting system of equations. This approach was later extended to more general cases by Haber et al.[54], who dealt with the problem of boundary voxels not perfectly aligning with a mesh of arbitrary geometry. Instead of discretizing the equation on a grid, it is also possible to use as supporting structure any connected graph; this is the approach of Yajun Wang, Jiaping Wang et al. [128, 130] who employed first a static polygrid, and then a QuadGraph obtained from the tetrahedralization of the mesh.

Arbree et al.[2, 3] introduced very accurate boundary conditions for the diffusion approximation, capable of taking into account refraction at the boundary. Moreover, they introduced a query function to recover radiance from the computed fluence which smooths computational errors by averaging. In the second paper, they solved the resulting system using the Finite Element Method, which had not been used previously for heterogeneous scattering, yielding impressive results.

Finally, a recent incorporation of multiple scattering in a single scattering framework by Zhou et al.[137] solved the diffusion system by conjugate gradient.

2.2.3 Lattice-Boltzmann Diffusion

Lattice-Boltzmann models, introduced by Geist[47] for the computation of scattering effects, define global behavior exclusively through local interactions. Update rules are defined for each discrete element and through iterated application of these rules, a desired system evolution results, in a manner similar to the evolution of Cellular Automata. In particular, Geist showed that by modeling light distribution as in fluid mechanics, the system behaves according to a diffusion process.

In a Lattice-Boltzmann method, space is discretized into cells and each cell is connected to its neighbors through links. The quantity under simulation is defined over the M connecting links plus a rest distribution.

$$f_m(\boldsymbol{x}, t)$$

Total fluence at a site at time t is recovered by the sum of these distributions: $e(\boldsymbol{x}, t) = \sum_m f_m(\boldsymbol{x}, t)$.

If we indicate lattice spacing with λ and simulation step time with τ, then at every step the update rule is:

$$f_m(\boldsymbol{x} + \lambda\boldsymbol{\xi}_m, t + \tau) - f_m(\boldsymbol{x}, t) = \Omega_m \cdot f(\boldsymbol{x}, t)$$

where Ω_m denotes the m-th row of the $m \times m$ matrix describing local particle interactions. This defines the relation of the m-th distribution at a site connected along direction $\boldsymbol{\omega}_m$ to another site.

The choice of the Ω matrix (called *collision matrix*) is not unique apart from some reasonable constraints like conservation of mass. A reasonable choice, the one presented in the original paper, defines Ω as following:

for row 0 :

$$\Omega_{0j} = \begin{cases} -1 & j = 0 \\ \kappa_a & j > 0 \end{cases}$$

for the axial rows $i = 1, ..., 6$:

$$\Omega_{ij} = \begin{cases} 1/12 & j = 0 \\ \kappa_s/12 & j > 0,\, j \neq i \\ -\kappa_t + \kappa_s/12 & j = i \end{cases}$$

for the non-axial rows $i = 7, ..., 18$:

$$\Omega_{ij} = \begin{cases} 1/24 & j = 0 \\ \kappa_s/24 & j > 0,\, j \neq i \\ \kappa_t + \kappa_s/24, & j = i \end{cases}$$

However, it can be proven[49] that with a step size close to zero and a comparably small spatial discretization, the process tends towards a time varying diffusion process governed by equation:

$$\frac{\partial e}{\partial t} = D\nabla^2 e$$

with

$$D = \frac{\lambda^2}{\tau}\left(\frac{(2/\kappa_t) - 1}{4\,(1 + \kappa_a)}\right)$$

Geist, Steele et al.[48, 120] have shown in multiple papers the applicability of this method to the rendering of participating media, like clouds; and how to extend this approach to complex forest ecosystems. More details about the Lattice-Boltmann process are given in chapter 4 where it is used in the context of one of our solutions.

2.3 The Dipole Model

It is possible to find an analytical solution to the diffusion equation (2.6) for a point-like source of unit power $\Phi = 1$ immersed in infinite isotropic media:

$$e(\boldsymbol{x}) = \frac{1}{4\pi D}\frac{e^{-\kappa_{tr} r(\boldsymbol{x})}}{r(\boldsymbol{x})}$$

where $r(\boldsymbol{x})$ is the distance from the light source at point $\boldsymbol{x}$ and $\kappa_{rt} = \sqrt{3\kappa_a \kappa_t'}$ is the reduced transport coefficient. This provides the *diffusion Green's Function.*

Jensen et al.[70]introduced to the Computer Graphics community a model for media with infinite boundary which poses that the effects on a surface point $\boldsymbol{x}_o$ produced by a collimated beam of light impinging on a surface point $\boldsymbol{x}_i$ can be modeled by introducing two imaginary light sources: one below the surface at the distance of a mean free path ($z_r = {}^1\!/\!{}_{\kappa_{tr}}$) called the "real" source, and one (negative in power) over the surface at height[5] $z_v = z_r + 4A\kappa_{tr}$ called the virtual source. This is the Dipole Approximation. Under this model, outgoing fluence at point $\boldsymbol{x}_o$ can be computed as

$$e(x) = \frac{\Phi}{4\pi D}\left(\frac{e^{-\kappa_{tr} d_r}}{d_r} - \frac{e^{-\kappa_{tr} d_v}}{d_v}\right)$$

where d_r is the distance of $\boldsymbol{x}_o$ from the real source and d_v the distance from the virtual source (see figure 2.1). Even if this method is theoretically sound only for half-infinite media, it can successfully be applied to model objects of any geometry. Discarding direction of incoming light, except for computing Fresnel transmittance, it is possible to construct a BSSRDF using the dipole model as:

$$S_{MS}(\boldsymbol{x}_i, \boldsymbol{\omega}_i, \boldsymbol{x}_o, \boldsymbol{\omega}_o) = \frac{1}{\pi} F_t(\eta, \boldsymbol{\omega}_i) R_d(\boldsymbol{x}_i, \boldsymbol{x}_o) F_t(\eta, \boldsymbol{\omega}_o)$$

where

$$R_d(\boldsymbol{x}_i, \boldsymbol{x}_o) = \frac{\Omega'}{4\pi}\left[z_r(\kappa_{tr} d_r + 1)\frac{e^{-\kappa_{tr} d_r}}{\kappa_t' d_r^3} + z_v(\kappa_{tr} d_v + 1)\frac{e^{-\kappa_{tr} d_v}}{\kappa_t' d_v^3}\right]$$

with $\Omega' = \frac{\kappa_s(1-g)}{\kappa_t'}$.

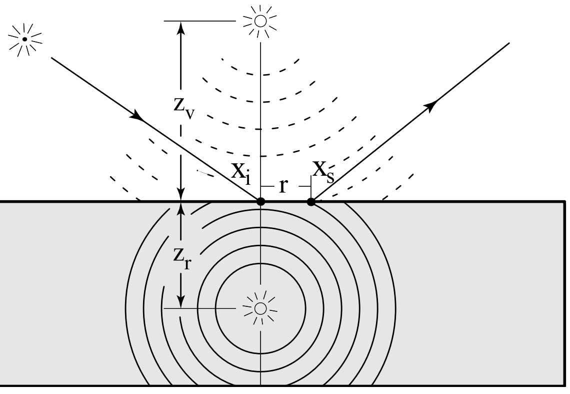


Figure 2.1: **Dipole Approximation**

This method has gained popularity due to the unique capability of providing an analytical BSS-RDF for translucent materials; however, it is not without shortcomings. In addition to the semi-infinite media assumption, this derivation assumes homogeneous materials and does not cleanly separate the Single Scattering contribution from the Multiple Scattering one. Papers in the last ten years have been geared towards overcoming these limitations. Worth mentioning are the multi-pole extension for layered materials[34], and the quadpole method, for handling sharp features[35]. Only recently a paper by D'Eon and Irving[31] has challenged the model from its core assumptions proposing an alternative formulation. Finally, it must be noted that other BSSRDF approaches

[5]The A factor in the equation accounts for Fresnel effects

have been explored, for example Holzschuch et al. [61, 36] devloped a model based on empirical obsersavtions about the correlation between multiple scattered light and secondorder scattering events.

2.3.1 Real-time Rendering with the Dipole Method

After its introduction the dipole model was initially used to accelerate off-line algorithms and continues to be a popular alternative to a full Monte Carlo or Photon Traced simulation[69, 34, 35, 89].

In a following paper[69], Jensen and Buhler decoupled irradiance evaluation from scattering computation with a two pass technique. In the first pass they sampled the incoming irradiance at selected points on the surface. In the second pass they applied the dipole diffusion approximation by cleverly integrating with a hierarchical method. Instead of summing for each point the contribution of every other point, or discarding a priori the contribution of distant samples, they clustered distant points for quick evaluation. Even if their method is considerably faster, they still did not achieve interactivity.

Lensch et al.[86] noticed how throughput factors R_d in the above equation are somehow similar to the geometric factors used in radiosity methods. But while geometry factors encode only distance and angular information between two patches, the subsurface scattering reflectance R_d encodes all the information about the volume that pertains to the diffusion of light from entering point to exiting point. The idea by Lensch et al. was then to discretize the subsurface scattering equation using Galerkin methods in a similar way as it is done in radiosity calculations. The authors employed a double discretization, one for the dominating local scattering (between close vertices) encoded as a texture, and one for the global scattering (between far apart vertices), encoded as hat functions. Although, this method is capable of producing real time results, its frame rate is still very limited.

Hao and Varshney[57], precomputed for every vertex, and for every incoming light direction, the contribution of the incoming radiance on the neighborhood to the outgoing radiance at the vertex. They stored such integral fully at selected points while using Spherical Harmonics to store at the remaining vertices the low-frequency difference from the selected ones. With such a scheme, they finally achieved a considerable interactive frame rate.

Mertens et al.[95] employ a hierarchical boundary element method for solving the light transport integral. However, their method is limited to interactive frame rates for moderately tessellated deformable objects. In a subsequent work, Mertens et al.[94] instead sped up the evaluation of R_d by implementing an importance sampling of the function in image space, such as to minimize the number of neighbor location accessed per vertex.

2.3.2 Screen space Subsurface Scattering

Among the attempts to develop an algorithm that could compute subsurface scattering in screen space we mention the simple method by Jimenez et al.[72] which performs a simple screen space blurring of the irradiance in order to approximate the effects of multiple scattering in human skin. A more precise approach is the one by D'Eon et al [30] who perform a similar blurring in texture space.

However, more accurate methods have been developed leveraging the dipole approximation.

The now popular work by Dachsbacher and Stamminger[27], Translucent Shadow Maps, provided a quick and conceptually simple algorithm. In their method the object is first rendered from the light position, such that all illuminated areas of the object correspond to visible rasterized fragments, as in shadow mapping methods. Contrarily to simple shadowing, this rendering is saved along with per-fragment depth information and normal information. During the view rendering stage, color is computed by accessing the translucent shadow map with a filter whose weights are given by R_d factors of the dipole approximation.

More recently, Shah et al.[113] proposed an image-space method based on irradiance splatting and gahtering, very efficient for homogeneous materials, but less suited for heterogeneous mixtures of short and long scattering ranges. A more straightforward approach is the one by Elek et al. [38], whose method is based on a filter in image space that can render the appearance on homogeneous fog given a rendered scene with depth information. Note that Lopez-Moreno et al. [91] had previously shown that perceptually plausible fog can be added to any scene if just a rough depth map, e.g. user sketched depth, is available.

2.4 Volume Rendering

The term “volume rendering” is mostly used to describe those methods that provide visualizations of volumetric datasets, ranging from CAT scans, to computer generated datasets. Although some of these methods are based on approximations of the RTE, in general the accent is placed on a real-time visualization that can carry meaningful information about the volumetric structure of the object rather than a physically correct appearance. A recent survey of volume rendering techniques can be found in Jonsson et al.[73] who divide state of the art methods according to the way light get transmitted from the sources to the volume.

We can subdivide methods based on the way the volume is visited during rendering. i.e., whether it happens in the direction from the viewer (marching, or Image Order methods) or towards the viewer (splatting, or Object Order methods).

2.4.1 Ray Marching

The method of Levoy[87] is still one of the most used for quickly computing the appearance of a volume under a given light direction. The gradient of the volume density is used as a normal and lighting is computed using a BRDF-like model. Effects are calculated locally and scattering is ignored. This is based on marching along every view-ray within the volume, i.e. sampling at regular interval the volume, accumulating the total opacity (more precisely, the exponential attenuation), and adding to the color of the pixel the computed local lighting multiplied by the opacity factor. This simple strategy has been the basis for most method to date dealing also with Single Scattering, whose main difference is that in ray marching single scattering, it is necessary to trace another ray towards all light sources at every sampling point.

The first algorithm of this kind has been provided by Nishita et al. in 1987[98] contextually to their introduction of the airlight integral (see section 2.5.1). To speed up the computation they proposed to bound each light by a cone as to restrict ray-marching to illuminated segments only.

Dobashi [33] subsequently proposed to approximate the integral of a product as a product of integrals, allowing the low frequencies to be coarsely evaluated on the CPU while higher-frequencies can be finely sampled via textured subplanes. Being also concerned about the trade-offs between aliasing and cost, Imagire et al.[65] proposed to divide space into a set of sampling hulls of which contributions are hardware-blended, allowing for higher sampling rates while limiting read-write memory accesses. A recent method by Engelhadt et al.[39] proposes instead to exploit the concepts of epipolar geometry to reduce the number of samples taken along the integral. A recent work by Baran et al. [4] accelerates the ray marching by constructing an acceleration structure from a shadowmap. Chen et al. [21] extended this work by adding support for textured light sources.

2.4.2 Splatting and Slicing

Instead of marching along rays for each pixel of the final image, another common technique in volume rendering is projection, that is, the volume is divided in smaller elements of constant optical properties (voxels, RBFs, metaballs), which at run time are sorted and rendered front-to-back with respect to the viewer, each piece independently of the others. The final color of the volume, as seen by the viewer, is given by a simple sum in image space of the colors of the elements. This approach is especially favored when the volume is not densely represented, but approximated using basis functions.

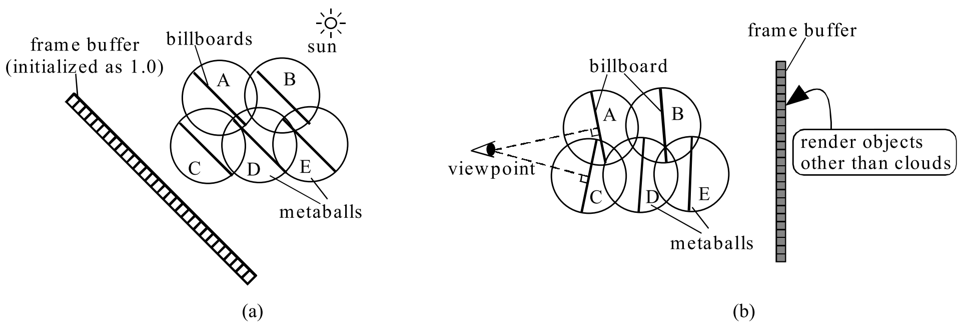


Figure 2.2: **Splatting** In step (a) of the illustration by Dobashi et al.[32], metaballs are rendered as simple billboards, ordered as B, E, A, D, C. In step (b), they are rendered in order C, A, D, B, E.

Dobashi et al.[32], for example, modeled volumetric information of density using metaballs of user defined radius. These metaballs were rendered, as billboards, two times: first from the light direction, to account for light-metaball attenuation, and then from the view to account for metaball-view attenuation, thus effectively computing single scattering.

Harris and Lastra[59], in the context of producing an as fast as possible cloud rendering algorithm for games, instead of rendering metaballs, rendered simpler impostors, precomputing multiple scattering and computing single scattering as in[32].

Another technique that allows the computation of multiple scattering at run time is given by half angle slicing, introduced by Kniss et al.[80], and particularly effective when volumetric data is stored as a 3D texture. The volume is sliced multiple times at an angle that is halfway between the view direction and the light direction (see figure 2.3). Slices are then rendered sorted by increasing light distance, each in two passes, first from the sun and then from the view. Lighting information computed at the previous pass is then spread at the next pass. With this method single and multiple scattering are accounted for, although for the latter, only its forward approximation is considered. This approach has the obvious drawback that it requires many renderings for any single view and it is not clear how it can be efficiently used with multiple lights or environment maps. Since Kniss et al. presented this algorithm with the intent of rendering clouds, this approach has been extended by Riley et al.[110], who added other atmospheric effects such as halos, coronas and rainbows.

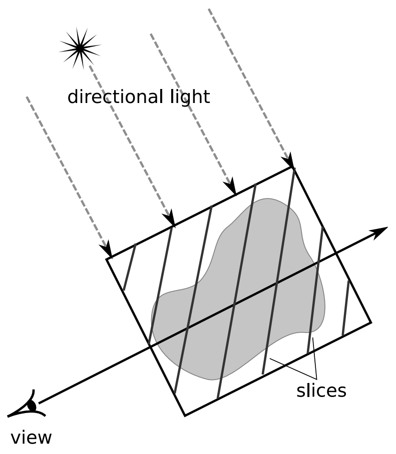


Figure 2.3: **Slice Accumulation.** The volume is divided in slices, rendered according to the light source direction. Each slice passes information about received and retransmittable light to the next.

Hegeman et al.[60] augmented the model by Kniss et al. by incorporating the Path Integration framework by Premože et al.[107]. By slicing, instead of ray marching, they achieved a modest form of interactivity (4 fps).

2.5 Single Scattering Integral and Closed Form Solutions

Even if ray marching computations have become feasible on modern hardware, a major goal in volumetric rendering literature has always been the development of an analytic and accurate equation for scattering. While, as we have seen in previous sections, diffusion based approximations permit some kind of analytic formulations, they are still based on arbitrary assumptions. However, for the case of single scattering in participating media, where the light source is immersed in the media itself, it has been shown possible to formulate a closed form solution of the integral form of the RTE. This integral is usually referred to as "airlight integral", since it captures the apparent phenomenon of air emitting light when a participating medium is present.

2.5.1 Air Light Integral

The first airlight model was developed by Nishita et al.[98], but a first attempt at an analytical solution is due to Lecocq et al. [85] who expanded the integral into a Taylor series. Biri et al. [126] followed this approach and incorporated volumetric shadows.

Sun et al.[121], contrarily to Biri et al.[126], built a method based on an explicit, exact, analytic solution of the single scattering integral for isotropic point lights in homogeneous participating media. However, the airlight integral is not completely solvable but requires the evaluation of a special function. Sun et al.[121] managed to numerically evaluate such function and store it in texture memory, allowing all computations to happen quickly at run-time.

In fact, the airlight integral due to a point light source of intensity Φ at location s scattered in the direction of a ray is given by,

$$L(\boldsymbol{x}_0, \boldsymbol{\omega}) = \int_0^{d_r} \kappa_s(\boldsymbol{x}) p(\alpha(\boldsymbol{x})) \frac{\Phi}{||\boldsymbol{x} - \boldsymbol{s}||^2} e^{-\tau(\boldsymbol{v},\boldsymbol{x}) - \tau(\boldsymbol{x},\boldsymbol{s})} \, dt \tag{2.7}$$

It can be shown that, by introducing the angle γ between the view-light and view-object lines, by approximating $p(\alpha(x))$ with k, and by integrating by substitution,

$$L(\boldsymbol{x}_0, \boldsymbol{\omega}) = \frac{\kappa_s^2 k \Phi e^{-\tau(s,v)\cos\gamma}}{\tau(\boldsymbol{s},\boldsymbol{v})\sin\gamma} \int_{\frac{\gamma}{2}}^{\frac{\pi}{4} + \frac{1}{2}\arctan\frac{\tau(v,p) - \tau(s.v)\cos\gamma}{\tau(s,v)\sin\gamma}} \exp\left[-\tau(\boldsymbol{s},\boldsymbol{v})\sin\gamma\tan\xi\right] \, d\xi$$

Although the expression seems fairly complicated, it reduces the evaluation of the integral to the integration of function

$$F(u, v) = \int_0^v e^{-u\tan(\alpha)} \, d\alpha$$

Even if function F is not analytically solvable, it is well behaved, and can be precomputed and stored as a 2D table.

The approach by Sun et al.[121] has been picked up by Zhou et al.[136] and by Wyman et al. [134] who independently combined it with ray-marching as to handle heterogeneous media in the first case, and light-shafts and anisotropic light sources in the second.

2.5.2 A closed form Integral Solution

More recently Pegoraro et al.[103, 101, 102] presented a solution of the airlight integral which does not require any tabulated function and which is capable of incorporating different phase functions into the solution by expansion via Legendre polynomials.[6] Their impressive result provided the first closed-form solution to the airlight problem.

Similarly to Sun et al.[121], the authors define in-scattered radiance due to a isotropic point light of power Φ, immersed in a homogeneous participating medium with phase function p as:

$$L_{in}(\boldsymbol{x},\boldsymbol{\omega}) = \Phi \frac{e^{-\kappa_t d(\boldsymbol{x},\boldsymbol{\omega})}}{d^2(\boldsymbol{x},\boldsymbol{\omega})} p(\boldsymbol{x},\boldsymbol{\omega})$$

where the directional distance $d(\boldsymbol{x},\boldsymbol{\omega})$ replaces the explicit $||\boldsymbol{x}-\boldsymbol{s}||^2$ term, and similarly, $\kappa_t d(\boldsymbol{x},\boldsymbol{\omega})$ is equivalent to the optical distance $\tau(\boldsymbol{x},\boldsymbol{s})$. Introducing h as the distance to the light from the given ray and $\boldsymbol{x}_h$ the coordinate of its projection onto it, the term can be rewritten as

$$L_{in}(\boldsymbol{x},\boldsymbol{\omega}) = \Phi \frac{e^{-\kappa_t \sqrt{h^2+(\boldsymbol{x}-\boldsymbol{x}_h)^2}}}{h^2+(\boldsymbol{x}-\boldsymbol{x}_h)^2} p \left(\arctan\left(\frac{\boldsymbol{x}-\boldsymbol{x}_h}{h} \right) + \frac{\pi}{2} \right)$$

Introducing the substitution $v(\boldsymbol{x}) = \frac{\boldsymbol{x}-\boldsymbol{x}_h}{h} + \sqrt{1+\left(\frac{\boldsymbol{x}-\boldsymbol{x}_h}{h}\right)^2}$, then the following holds:

$$\int_{x_0}^{x_\infty} e^{-\tau(\boldsymbol{x}_0,\boldsymbol{x})} \kappa_s(\boldsymbol{x}) L_{in}(\boldsymbol{x},\boldsymbol{\omega}) d\boldsymbol{x} = \Phi \frac{\kappa_s}{h} e^{\kappa_t(\boldsymbol{x}_0-\boldsymbol{x}_h)} \frac{2}{4\pi} I_0(-H, v_0, v_\infty)$$

where$H = \kappa_t h$ is the optical distance between the source and the ray, and $I_0(a,v_a,v_b) = i_0(a,v_b) - i_0(a,v_a)$ with i_0 an auxiliary function defined as

$$i_0(a,v) = \sin(a)\Re(Ei(av+ia)) - \cos(a)\Im(Ei(av+ia))$$

and Ei the exponential integral function.

2.6 Precomputed Radiance Transfer

In their seminal paper, Sloan et al.[117] used a Spherical Harmonics basis to encode, for each vertex, a transfer function that maps incoming illumination to "transferred" radiance.

By exploiting linearity of light transfer, they encoded distant, low frequency illumination in the same basis as the surface transfer function, thus reducing the evaluation of the rendering integral to a simple dot product between two 9-25 coefficient vectors (see § A.1).

During the preprocessing stage, global lighting computations are performed to record how an object shadows and scatters light onto itself. The results of this computation is stored in vectors (for diffusely reflecting objects) or matrices (for glossy objects).

At run time, global incident illumination is projected on the Spherical Harmonics basis. If the object's surface is diffuse, the transfer function coefficients at each vertex are dotted with the lighting coefficients, thus producing the final correct shading. If the object's surface is glossy, the transfer matrix is applied to the lighting coefficients in order to produce the exiting radiance coefficients. These coefficients are then convolved with the BRDF of the object and the resulting spherical function is evaluated at the view direction.

Exiting radiance at a surface point p of a diffuse object can be simplified to

$$L(\boldsymbol{x},\boldsymbol{\omega}) = \int_{\Xi} L(\boldsymbol{x},\boldsymbol{\omega}') \underbrace{\frac{\rho}{\pi} \max(\boldsymbol{n}_p \cdot \boldsymbol{\omega}', 0) V(\boldsymbol{\omega}')}_{\text{transfer function}} d\Omega(\boldsymbol{\omega}')$$

with ρ the diffusive coefficient and $V(\omega)$ a visibility term that equals zero if there is self-occlusion of the object in direction ω.

The transfer function at each point depends only on the normal at the point. So by virtue of being completely know during preprocessing, it can be projected on the SH basis

[6]the expansion of the phase function via Legendre polynomials is central to the P_N method and the SHDOM.

$$L(\boldsymbol{x}) = \int_{\Xi} L(\boldsymbol{x}, \boldsymbol{\omega}')T(\boldsymbol{\omega}')\, d\Omega(\boldsymbol{\omega}')$$

Then, thanks to SH properties if incoming light has been projected on the same basis

$$L(\boldsymbol{x}) = \sum_{i=0}^{n^2} l_i t_i$$

If the object is glossy, its exiting radiance is

$$L(\boldsymbol{x}, \boldsymbol{\omega}) = \int_{\Xi} L(\boldsymbol{x}, \boldsymbol{\omega}')P(\boldsymbol{\omega}', \boldsymbol{\omega}, r)V(\boldsymbol{\omega}')\, d\Omega(\boldsymbol{\omega}')$$

Since P depends on both view direction and light direction, it cannot be precomputed, but thanks to SH properties it is possible to precompute visibility (and interreflections)

$$L(\boldsymbol{x}, \boldsymbol{\omega}) = \left(P_r^* * \left(\sum_{j=0}^{n^2} l_i T_{i,j} y_{i,j} \right) \right) (\boldsymbol{\omega})$$

where $P_r^*(\boldsymbol{\omega}) = P(\boldsymbol{\omega}', (1, 0, 0), r)$ and the operator $*$ represents the convolution operator.

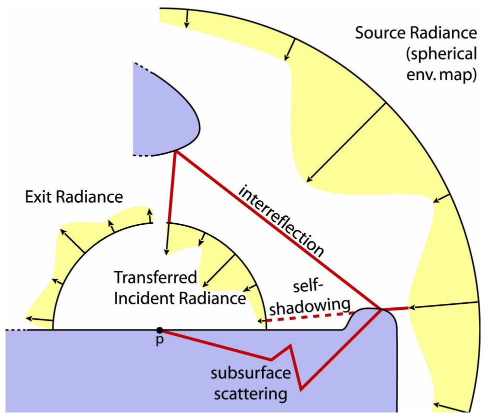


Figure 2.4: **Precomputing Radiance Transfer.** Interreflections, self shadowing and subsurface scattering[116]

In a following paper[116], Sloan et al. explicitly treat, among other global illumination effects, the subsurface scattering case, defining a compact way to store the matrices representing transfer functions. To account for subsurface scattering it is sufficient to substitute the simple visibility term V with a more complex function that performs some expensive traversal of the solid to determine light paths inside the medium. Since this computation happens during the offline stage, this traversal can be solved with a Monte Carlo simulation. The main limitation is again the assumption of distant lighting and the necessity to store a matrix for each surface point.

2.7 Refraction

Even if refraction has often been treated as a separate subject from scattering, Computer Graphics researchers have been interested in this phenomenon since the off-line ray tracing engine of Turner Whitted[132]. In fact, refraction can be trivially incorporated in a ray traced paradigm by employing Snell's equations at the boundary of objects, or a more complex transmission function modeled with an appropriate BDTF.

In the area of real-time rendering, refraction has often been simplified by considering only the first interaction between light and object boundaries. This is the case of [90] where a normal map is used to deviate rays and index into a texture containing background information.

A more accurate approximation is proposed by Wyman et al., in their two-pass method[133], capable of accounting both for the incoming and outgoing interactions with the volume, but is inaccurate for non convex objects. More recent methods are capable of dealing with more complex effects, for example Rousier et al.[29] incorporated non specular BTDFs, transcending the assumptions that for every incoming light ray only one transmitted ray needs to be tracked. Ihrke et al.[64] introduced a particle-based method derived from the Eikonal equation that achieves interactive frame rates. Sun et al.[122] instead presented a real-time renderer which constantly bends light rays at each voxel intersection in a fashion similar to volumetric photon mapping. More recently, this method has been extended in order to handle a closed form analytical formulation of light ray trajectory on constant gradient refractive media[17]. Finally, Walter et al.[127] showed a real-time formulation that determines the focal points on the boundary of an object.

Note that even if there has been one example of derivation of the diffusion equation with refraction[78], this formula, to the best of our knowledge, has never been put to practical use.

Chapter 3

Efficient Compression of Material Properties for Single Scattering

Abstract

In this chapter we present a general and efficient algorithm to render single scattering effects in translucent objects of arbitrary geometry composed of heterogeneous materials or having visible internal details. The developed technique is based on an adaptive volumetric point sampling, done in a preprocessing stage, which associates to each sample the optical depth for a predefined set of directions. This information is then used by a rendering algorithm that combines the object's surface rasterization with a ray tracing process, implemented on the graphics processor. This approach allows us to simulate single scattering phenomena for inhomogeneous isotropic materials in real time with an arbitrary number of light sources. We tested our algorithm by comparing the produced images with the result of ray tracing and show that the technique is effective.

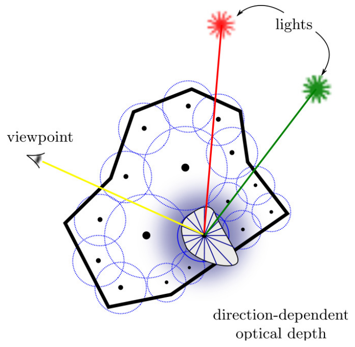


Figure 3.1: **An overview image of our algorithm**. the direction-dependent optical depth is sampled and stored in a set of locations and it is used at rendering time to compute the scattering of light in a single pass.

In translucent objects composed of materials with low albedo ($\Omega \ll 1$), multiple scattering effects add little to the global appearance of the object; consequently, a full solution to the Radiative Transfer Equation is often considered not to be necessary[75, 13]. Both in the offline and real-time application domains, numerical methods or Diffusion/Dipole based solutions can bring an unnecessary overhead to the rendering time of such materials. On the other hand, simple volumetric solutions, which compute single scattering in arbitrary dense volumetric representations, do not scale well with the number of lights, and are often confined to work with a specific rendering paradigm. Moreover, precomputation strategies like Precomputed Radiance Transfer either impose a simplification on directional effects or require a large amount of storage in order to produce high quality results.

In this chapter we therefore introduce a method that is tailored at producing convincing visual results conveying the volumetric appearance of low albedo objects, obtained with an adaptive sampling strategy. This strategy is guided by material properties in a way that we believe manages to capture discontinuities inside the material while producing data-sets of reasonable size. Although preprocessing times can be considerable for complex meshes, they are balanced by a frame rate at rendering time that is always above the interactivity threshold (10 fps in the worst case).

In summary, our contribution to the state of the art is two-fold:

- An efficient, single-pass algorithm for real-time rendering of single scattering effects which completely decouples scattering phenomena from superficial light reflection computations, and that can be well integrated in standard rendering pipelines.
- A strategy for an adaptive opacity-dependent point sampling of the volume enclosed by a surface that is able to capture and represent the internal appearance of the object.

3.1 Algorithm Overview

The presented algorithm is divided in a preprocessing stage and a rendering stage.

In the preprocessing stage, the internal volume of the object is sampled with a set of points, each of which covers a spherical portion of space. This set is chosen to maximally cover the outermost subsurface stratus and to capture the most important internal discontinuities. To each sample is associated a directional function that describes the attenuation due to Beer-Lambert's law between the surface and the point (see figure 3.1). The function is computed by tracing rays through the material, from each sample towards a predefined set of directions, compactly encoding the result using Spherical Harmonics.

At run-time, the boundary surface is normally rendered with the rasterization pipeline to account for the pure surface reflection. Subsequently, the spheres associated with the sampling points are ray traced with an algorithm implemented in a GPU fragment shader, which computes the contribution of each spherical portion to the color of the corresponding pixel by integration.

The substantial difference with similar approaches for participating media is that, thanks to the precomputation of the optical depth, we do not need to perform a rendering pass for each light in order to accumulate Irradiance on the samples.

3.1.1 Reformulating the Radiative Transfer Equation

Translucent objects, as opposed to participating media, are enclosed by a definite boundary. We exploit this fact in the following derivation without posing limitations on the topology of the enclosing surface. Moreover, since most translucent objects do not emit radiance[1], we can drop the emission term from the Radiative Transfer Equation (1.9):

$$\nabla_{\boldsymbol{\omega}} L(\boldsymbol{x},\boldsymbol{\omega}) = \underbrace{-\kappa_t(\boldsymbol{x}) L(\boldsymbol{x},\boldsymbol{\omega})}_{\text{extinction}} + \underbrace{\kappa_s(\boldsymbol{x}) \int_{\Xi} p(\boldsymbol{\omega},\boldsymbol{\omega}') L(\boldsymbol{x},\boldsymbol{\omega}')\, d\Omega(\boldsymbol{\omega}')}_{\text{in-scattering}} \tag{3.1}$$

Since points belonging to surface S are the only receivers of direct, unattenuated lighting, we can treat them as the sole emitters of transmitted light within the volume, which we indicate it in this chapter as L_t.

We simplify refraction of light trough the boundary using equation 1.3 and therefore set the ratio of transmitted light to incoming light as F_t.

$$L_t(\boldsymbol{y},\boldsymbol{\omega}) = F_t L_i(\boldsymbol{y},\boldsymbol{\omega})$$

[1]we ignore incandescent and fluorescent materials

We rewrite the integral form of the RTE on the ray between a point $\boldsymbol{x}$ and a point $\boldsymbol{y} \in S$ on the surface of the object. (see figure 3.2)

$$L(\boldsymbol{x},\boldsymbol{\omega}) = \underbrace{e^{-\tau(\boldsymbol{x},\boldsymbol{y})}L_t(\boldsymbol{y},\boldsymbol{\omega})}_{\text{attenuated direct light}} + \underbrace{\int_{\boldsymbol{x}}^{\boldsymbol{y}} e^{-\tau(\boldsymbol{x}',\boldsymbol{y})}\kappa_s(\boldsymbol{x}') \int_{\Xi} p(\boldsymbol{\omega},\boldsymbol{\omega}')L(\boldsymbol{x}',\boldsymbol{\omega}')\,d\Omega(\boldsymbol{\omega}')\,d\boldsymbol{x}'}_{\text{in-scattered lighting}}$$

Since we developed a system which deals with point-lights, which are by definition invisible sources, we can safely drop the first term of the above equation and work on the second.

$$L(\boldsymbol{x},\boldsymbol{\omega}) = \int_{\boldsymbol{x}}^{\boldsymbol{y}} e^{-\tau(\boldsymbol{x},\boldsymbol{y})}\kappa_s(\boldsymbol{x}') \int_{\Xi} p(\boldsymbol{\omega},\boldsymbol{\omega}')L(\boldsymbol{x}',\boldsymbol{\omega}')\,d\Omega(\boldsymbol{\omega}')\,d\boldsymbol{x}'$$

Under the single scattering assumption (§ 1.3.6), $L(\boldsymbol{x}'\boldsymbol{\omega}')$ must simply be the attenuated direct lighting coming from some point $\boldsymbol{y}'$ on the surface.

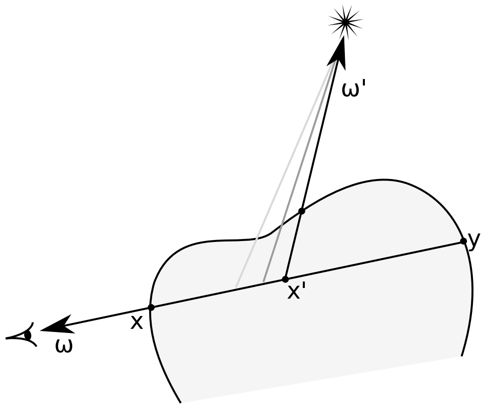


Figure 3.2: **Single Scattering of Light** The contribution of single scattering of light at point $\boldsymbol{x}$ in direction $\vec{\omega}$ is computed by integrating, for each point x' along the ray from x to y, the attenuated amount of light received from direction $\vec{\omega}'$ and retransmitted towards ω.

The integral can then be explicitly written using the definition of optical depth $\hat{\tau}$, as

$$L_{SS}(\boldsymbol{x},\boldsymbol{\omega}) = \int_{\boldsymbol{x}}^{\boldsymbol{y}} e^{-\tau(\boldsymbol{x}',\boldsymbol{y})}\kappa_s(\boldsymbol{x}') \int_{\Xi} p(\boldsymbol{\omega},\boldsymbol{\omega}')\underbrace{e^{-\hat{\tau}(\boldsymbol{x}',\boldsymbol{\omega}')}L_t(\boldsymbol{x}',\boldsymbol{\omega}')}_{\text{attenuated direct light}}\,d\Omega(\boldsymbol{\omega}')\,d\boldsymbol{x}' \tag{3.2}$$

where we abused the notation to indicate with $L_t(\boldsymbol{x}',\boldsymbol{\omega}')$ light arriving on the surface at the point $\boldsymbol{y}$ found by traversing the volume in direction $\boldsymbol{\omega}$ from $\boldsymbol{x}$.

On a surface point $\boldsymbol{y}$, the total amount of light that appears to be outgoing towards viewer direction $\boldsymbol{\omega}$ is given by the sum of the reflected radiance according to the rendering equation and the retransmitted radiance arriving according to the above equation (3.2). Since we ignore in this model the directional effects of crossing a boundary between materials with different indexes of refraction we will show how this effect can be modeled further on.

As shown in figure 3.3, radiance at point $\boldsymbol{x}$ towards $\boldsymbol{\omega}$ is therefore, according to our purposes, given by,

$$\underbrace{L_o(\boldsymbol{x},\boldsymbol{\omega})}_{\text{total outgoing light}} = \underbrace{L_{ss}(\boldsymbol{x},\boldsymbol{\omega})}_{\text{single scattered light}} + (1-F_t)\underbrace{\int_{4\pi} f(\boldsymbol{\omega},\boldsymbol{\omega}')L_i(\boldsymbol{x},\boldsymbol{\omega}')d\boldsymbol{\omega}'}_{L_r(\boldsymbol{x},\boldsymbol{\omega})}$$

The term $L_r(\boldsymbol{x},\boldsymbol{\omega})$, in the context of our on-line rendering application, will be calculated using one of the popular shading models used by OpenGL.

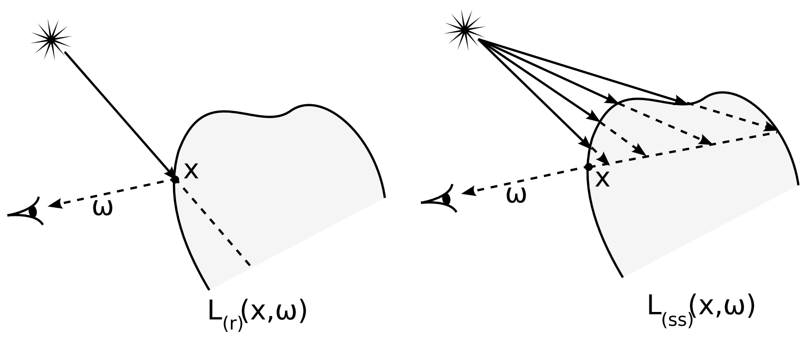


Figure 3.3: **Radiance leaving point x** is the contribution of radiance directly reflected by the surface (component r) and subsusrface scattering (component ss). On the left, light directly incident on the surface partially penetrates the object F_t and partially is reflected $1 - F_t$. On the right, light that exits the surface at x is determined by the integral of all points on the ray. Total outgoing radiance is simply the sum of the two.

For the sake of simplicity, we assume that scattering is isotropic, hence $p(\boldsymbol{\omega}, \boldsymbol{\omega}') = \frac{1}{4\pi}$; moreover, we will refer to the integrand function in the next steps as the function $F(\boldsymbol{x}')$,

$$L_{ss}(\boldsymbol{x}, \boldsymbol{\omega}) = \int_{\boldsymbol{x}}^{\boldsymbol{y}} \underbrace{e^{-\tau(\boldsymbol{x}',\boldsymbol{y})}\kappa_s(\boldsymbol{x}')\frac{1}{4\pi}\int_{4\pi} e^{-\hat{\tau}(\boldsymbol{x}',\boldsymbol{\omega}')}L_t(\boldsymbol{x}', \boldsymbol{\omega}')\, d\Omega(\boldsymbol{\omega}')}_{F(\boldsymbol{x}')}\, d\boldsymbol{x}'$$

We would like now to evaluate $F(\boldsymbol{x}')$ only at discrete points within the volume and interpolate them when integrating along a ray. To this end, we will properly choose a set of points $B = \{b_i\}$.

Let us assume that, given B, it is possible to build a set of Gaussian radial basis functions $\Gamma = \{\gamma_i(\boldsymbol{x})\}$, where $\gamma_i(\boldsymbol{x}) = e^{a_i||\boldsymbol{x}-\boldsymbol{b}_i||}$, such that, if we call V the internal volume of an object, then

$$\begin{cases} \gamma_i(\boldsymbol{b}_i) = 1 & \forall i \\ \gamma_i(\boldsymbol{b}_j) = 0 & \forall j \neq i \\ \sum_i \gamma_i(\boldsymbol{x}) = 0 & \forall \boldsymbol{x} \notin V \\ \sum_i \gamma_i(\boldsymbol{x}) = 1 & \forall \boldsymbol{x} \in V \end{cases} \tag{3.3}$$

If that be the case[2], then the following relation is satisfied

$$\int_{\boldsymbol{x}}^{\boldsymbol{y}} F(\boldsymbol{x}')\, dx = \int_{\boldsymbol{x}}^{\boldsymbol{y}} F(\boldsymbol{x}') \sum_i \gamma_i(\boldsymbol{x}') = \sum_i \int_{\boldsymbol{x}}^{\boldsymbol{y}} F(\boldsymbol{x}')\gamma_i(\boldsymbol{x}')\, d\boldsymbol{x}'$$

If function $F(\boldsymbol{x}')$ is such that its values, when $\gamma_i(x')$ is not null (i.e. in the surroundings of the Gaussian center), do not vary significantly, then we can assume $F(\boldsymbol{x}')$ to be constant and take its value at point $\boldsymbol{b}_i$.

$$\sum_i \int_{\boldsymbol{x}}^{\boldsymbol{y}} f(\boldsymbol{x}')\gamma_i(\boldsymbol{x}')\, d\boldsymbol{x} \approx \sum_i f(\boldsymbol{b}_i) \int_{x}^{y} \gamma_i(\boldsymbol{x}')\, d\boldsymbol{x}'$$

It is intuitive that, if Gaussians have small radii of influence with respect to the dimensions of the surface, then $F(\boldsymbol{x})$, whose value strongly depends on the distance from the surface, will have little variance inside said radius. Therefore, we can write,

[2]Note, however, that conditions 3.3 cannot be actually satisfied using simple Gaussian functions. While we will base our derivation on the assumption that a perfect set Γ can instead be built, in practice we will only attain a rough approximation.

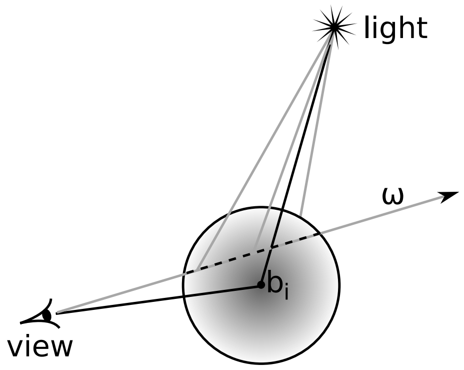


Figure 3.4: **Gaussian Radial Basis Functions** Instead of computing the correct integral of function $F(\boldsymbol{x})$ (light gray paths), the function is computed only at $\boldsymbol{b}_i$ (dark path) and multiplied by the integral of the Gaussian function (dashed path)

$$L_{SS}(\boldsymbol{x},\boldsymbol{\omega}) \approx \sum_{i\in\Gamma} \underbrace{\frac{1}{4\pi}\kappa_s(\boldsymbol{b}_i)e^{-\hat{\tau}(\boldsymbol{b}_i,\boldsymbol{\omega})}\int_{\Xi} e^{-\hat{\tau}(\boldsymbol{b}_i,\boldsymbol{\xi})}F_t L(\boldsymbol{b}_i,\boldsymbol{\xi})d\Omega(\boldsymbol{\xi})}_{F(\boldsymbol{b}_i)} \int_{\boldsymbol{x}}^{\boldsymbol{y}} \gamma_i(\boldsymbol{x}')\,d\boldsymbol{x}'$$

By encoding the exponential attenuation due to the optical depth of each Gaussian center with Spherical Harmonics ψ_i, we arrive at the final formula used in our on-line computations:

$$L_{ss}(\boldsymbol{x},\boldsymbol{\omega}) \approx \sum_{i\in\Gamma} \frac{1}{4\pi}\kappa_s(\boldsymbol{b}_i)\psi_i(\boldsymbol{\omega})\int_{\Xi} \psi_i(\boldsymbol{\xi})F_t L(\boldsymbol{b}_i,\boldsymbol{\omega})\,d\Omega(\boldsymbol{\xi}) \int_{\boldsymbol{x}}^{\boldsymbol{y}} \gamma_i(\boldsymbol{x}')\,d\boldsymbol{x}' \tag{3.4}$$

Lighting computations are extremely fast if light comes only from a limited set of directions D. In such case, instead of integrating over Ξ, it is sufficient to evaluate and sum for all directions in D. On the other hand, if lighting is defined over the whole sphere of directions, like when environment maps are used, then by projecting it over the Spherical Harmonics basis the above integral will reduce to a simple dot product of two vectors of coefficients (see Appendix A.1).

3.2 Point Sampling Algorithm

The core of the preprocessing stage revolves around finding good positions for the samples inside the volume. The sampling algorithm is critical not only to minimize memory occupation and rendering times, but also to ensure that the assumptions made in our derivation are valid. Preprocessing is done in two stages; in the first stage, great care is taken in choosing the right sampling set (samples in algorithm 1), which corresponds to set B in our derivation. The construction of such set is explained in detail in the next section (§ 3.2.1)

In the second part, the remaining preprocessing steps consist only of associating, with each point, the compressed attenuation function coefficients (vector SH) and the Gaussian radial basis function width (value Width). This relatively simple process is presented in section 3.3.

3.2.1 Sampling Algorithm

Producing a good stochastic sampling of a region of space, delimited by a generic mesh, is not trivial. Instead of adapting an existing sampling algorithm (see figure 3.5), we sought an *ad hoc* solution based on considerations on the behavior of light.

From simple observations it can be noticed that, for most scattering materials, light will penetrate only by a small distance beneath the surface of an object, proportionally to the mean free path. A consequence of the exponential attenuation predicted by Beer-Lambert's law 1.5 is that

Algorithm 1 Preprocessing Stage

```
Preprocess(Mesh mesh)
{
    //Poisson Disk Sampling with variable disk radius.
    //Variation is given by a Density Heuristic Function
    Sample[] samples = LayeredPoissonDiskSampling(mesh);

    //Associate with each sample the Spherical Harmonics
    //coefficients and the gaussian width.
    foreach sample in samples
    {
        sample.Width = WidthFromDensity(sample);
        sample.SH = ProjectFunctionAt(sample);
    }
    return samples;
}
```

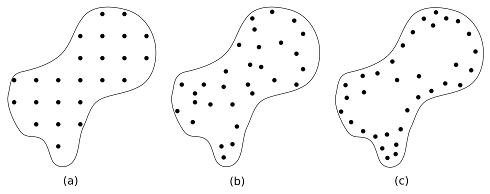


Figure 3.5: **Sampling Strategies** Using a uniform grid approach (a), in order to capture small details, and ensure that variance with respect to the Gaussian functions is minimal, it would be necessary to over-sample the whole volume, leading to an excessive amount of data. Similarly, a completely random sampling (b), while possibly eliminating visual artifacts produced by the uniform sampling, would lead to an even more accentuated oversampling of the volume in order to ensure that all areas are adequately covered. Our approach (c) strives to put samples only where they are actually needed.

the majority of light rays will be scattered or absorbed within a thin subsurface layer. Therefore, in order to adequately represent small distance lighting effects, it is necessary to place many samples as close as possible to the surface of objects. Furthermore, as seen in section 3.4, the nature of our rendering algorithm is such that, if an area remains accidentally uncovered by any sample, it will be displayed as unnaturally dark. Surfaces presenting high frequency geometric details must be sampled with great accuracy, so as to ensure that even small convex features are covered.

On the other hand, chosen points will be used with the principal intent of encoding function $\psi_i(\boldsymbol{\omega}) = e^{-\hat{\tau}(\boldsymbol{b}_i,\boldsymbol{\omega})}$, which represents the amount of attenuation to which light arriving at the point is subject. If we allow scattering coefficients to be variable, it follows intuition that there will possibly be areas of lower or higher light penetration. In low lighting areas, deep within the surface, or even at the surface level if the material is extremely thick, there will be no need to place sampling points as they would be used to encode a function which is approximately always zero. Furthermore, as some models will undoubtedly present areas across which scattering behavior is constant, it would be a reasonable choice to use fewer sampling points, with Gaussians of bigger radii, in those areas.

With these considerations in mind, our sampling algorithm proceeds by iteratively building concentric meshes inside the input one using the OffsetSurface algorithm (§ 3.2.2), separated by a constant, tiny, offset. Sampling points will be successively placed only on the surfaces of these internal meshes, which we will call layers. In this way, sampling positions, while still random, will be forced to follow the shape of the surface.

Algorithm 2 Layered Poisson Disk Sampling

```
Samples[] LayeredPoissonDiskSampling(Mesh mesh)
{
    for (offset = delta; offset < max_offset; offset += delta)
    {
        layers += OffsetSurface(mesh, offset);
    }
    HeuristicDensityPerVertex(layers);
    Sample[] samples = PoissonDiskSampling(layers);
    return samples;
}
```

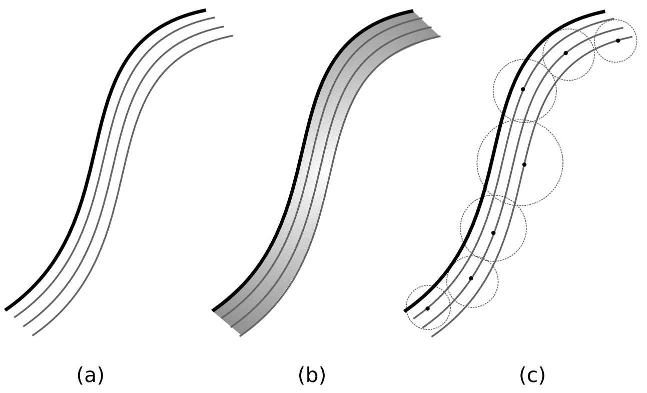


Figure 3.6: **Our Sampling Strategy** (a) layers are build underneath the original mesh at regular intervals, (b) density is heuristically estimated (darker areas represent higher density), (c) samples are chosen from all layers using a Poisson disk algorithm guided by density (note that disks have a bigger radius in areas of lower density)

Before the actual sampling, HeuristicDensityPerVertex is used to estimate areas where more or less samples are needed to follow variation of encoded function $\psi_i(\boldsymbol{\omega})$. This heuristic is based on the divergence of the first moment of function $\psi_i(\boldsymbol{\omega})$ and it is computed and stored at all vertices of the internal layers (§ 3.2.3).

Finally, layers are densely sampled using an adapted PoissonDiskSampling. The algorithm, using previously computed density information, produces a random subset without generating accidentally overcrowded or under-sampled zones (§ 3.2.4).

3.2.2 Offset Surfaces Construction

In the first phase of the algorithm, internal meshes are built at regular intervals beneath the surface of the object. These meshes need to be placed at short distances from each other so as not to influence the final density distribution of sampling points. While the optimum would be to separate them by infinitesimal intervals, good results can still be obtained by setting distances as fractions of the predicted mean distance[3] between sampling points. The construction of these internal surfaces is performed by iteratively applying a Marching Cubes[92] algorithm on the distance field determined by the outer mesh.

[3]This distance is estimated given the dimensions of the bounding box of the input mesh, and the number of requested samples by the user.

3.2.3 Density Heuristic

Before the actual sampling is applied, a strategy to predetermine optimal density guided by the behavior of function $\psi_i(\boldsymbol{\omega})$ must be employed.

Since such function decreases exponentially with the optical depth $\hat{\tau}(\boldsymbol{b}_i, \boldsymbol{\omega})$, it follows that it will be strongly peaked at directions corresponding to minimal $\hat{\tau}(\boldsymbol{b}_i, \boldsymbol{\omega})$. As an extreme case, the function $\psi_i(\boldsymbol{\omega})$ defined at a point $\boldsymbol{x}$ placed at an infinitesimal distance beneath the surface, will present an extreme peak in the same direction as the normal to the surface. (see figure 3.7)

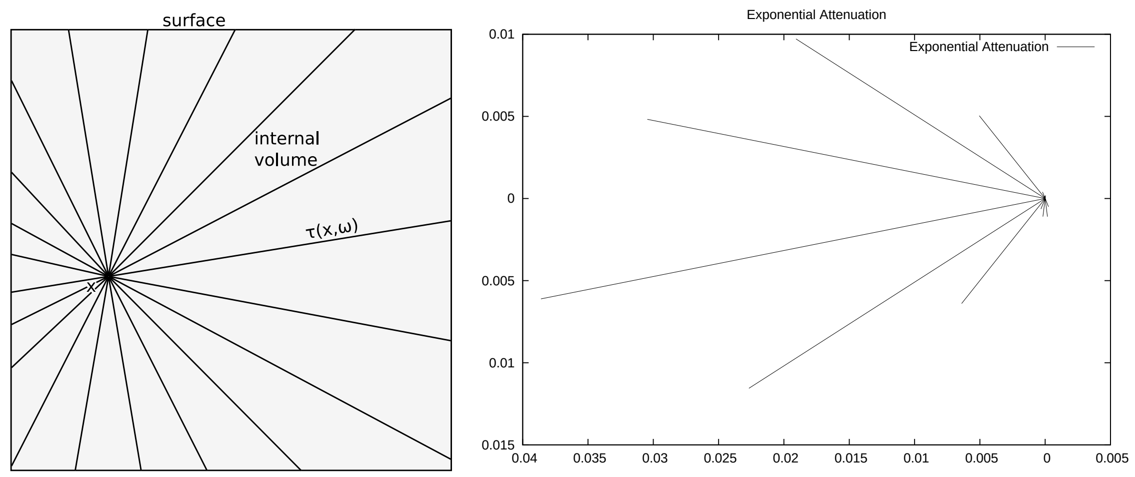


Figure 3.7: **Optical Depth Peakiness** On the left-hand side a point x is placed near the surface of a cube; lines indicate the optical distance $\tau(\boldsymbol{x}, \boldsymbol{\omega})$ between point and surface. On the right-hand side an impulse plot of the function $e^{-\tau(\boldsymbol{x},\boldsymbol{\omega})}$, or exponential attenuation function, i.e. the function that is actually encoded using Spherical Harmonics. It can be noticed how such function has an extremely pronounced peak at the direction of minimum optical distance from the surface.

Since the function is strongly peaked, a good indicator of its shape and behavior is its first moment:

$$\boldsymbol{T}(\boldsymbol{x}) = \int e^{-\hat{\tau}(\boldsymbol{x},\boldsymbol{\omega})} \cdot \boldsymbol{\omega} \, d\Omega(\boldsymbol{\omega}) \tag{3.5}$$

Function $\boldsymbol{T}(\boldsymbol{x})$ defines a vector field whose values point towards the mean direction of minimum optical distance at each point. Moreover, the norm of these vectors, $||\boldsymbol{T}(\boldsymbol{x})||_2$, would give us a weighted mean of the attenuation factor over all directions. However, this value does not concern us because, as sampling density needs to reflect changes in shape of $e^{-\hat{\tau}(\boldsymbol{x},\boldsymbol{\omega})}$ with respect to $\boldsymbol{x}$, the quantity that is more apt to estimate optimal density is the divergence of the aforementioned vector field, $\nabla \cdot \boldsymbol{T}(\boldsymbol{x})$. In fact, in areas where the principal direction of light changes rapidly, e.g. at corners, divergence will be higher; similarly in areas where the magnitude of the principal direction decreases rapidly, this value will be high as well, e.g. near the surface, exactly where we expect scattering phenomena to be more pronounced (see figure 3.8).

As divergence is a signed quantity, we will set desired density ρ as its absolute value. Note also that this quantity cannot be considered the optimal density but rather a heuristic approximation.

$$\rho = |\nabla \cdot \boldsymbol{T}(\boldsymbol{x})|$$

After layers have been built at the previous stage, for each vertex $\boldsymbol{v}_j$ of those internal meshes the value $\rho_j = |\nabla \cdot \boldsymbol{T}(\boldsymbol{v}_j)|$ is computed. The computation of theses values is simply done by finite differences, choosing an appropriately small h:

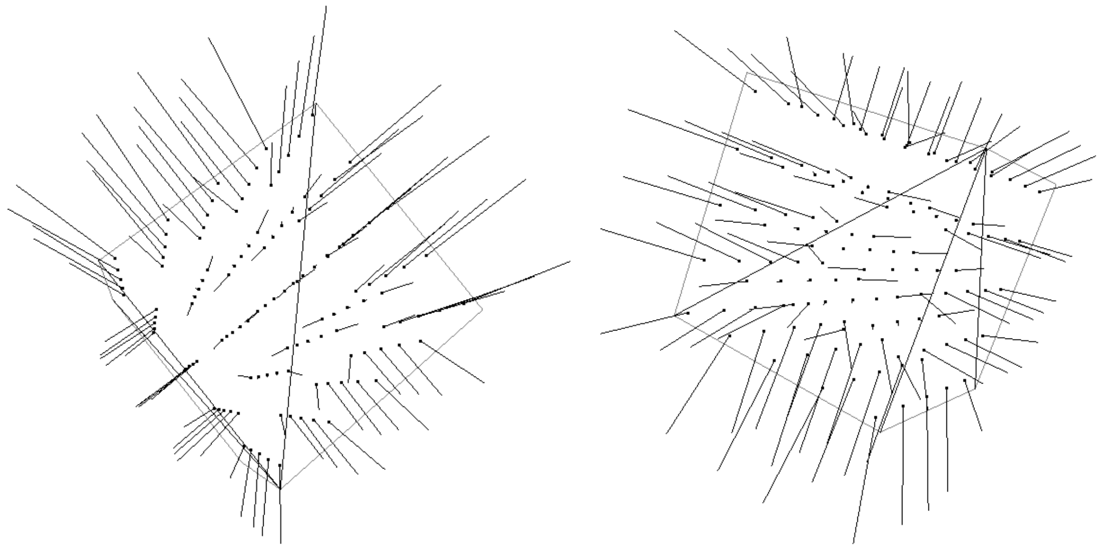


Figure 3.8: **Vector Field** In the two figures, lines indicate vectors at sampling points of a simple triangulated cube with a homogeneous material. Notice how direction is influenced by surface curvature, while magnitude by surface distance.

$$
\begin{aligned}
\nabla \cdot \boldsymbol{T}(\boldsymbol{\chi}) \approx & \frac{\boldsymbol{T}(\boldsymbol{\chi}_x + h, \boldsymbol{\chi}_y, \boldsymbol{\chi}_z)_x - \boldsymbol{T}(\boldsymbol{\chi}_x - h, \boldsymbol{\chi}_y, \boldsymbol{\chi}_z)_x}{2h} \\
+ & \frac{\boldsymbol{T}(\boldsymbol{\chi}_x, \boldsymbol{\chi}_y + h, \chi_z)_y - \boldsymbol{T}(\boldsymbol{\chi}_x, \boldsymbol{\chi}_y - h, \boldsymbol{\chi}_z)_y}{2h} \\
+ & \frac{\boldsymbol{T}(\boldsymbol{\chi}_x, \boldsymbol{\chi}_y, \boldsymbol{\chi}_z + h)_z - \boldsymbol{T}(\boldsymbol{\chi}_x, \boldsymbol{\chi}_y, \boldsymbol{\chi}_z - h)_z}{2h}
\end{aligned}
$$

with an error proportional to $O(h)$.

This, in turn, requires the computation of $\boldsymbol{T}(x)$ at six different points. Such integrals (3.5) are obtained by simple Monte Carlo evaluation using stratified sampling. Since the resulting values are used only as a reference for the final sampling, we allow precision of Monte Carlo integration to be lower by sampling the integral only at a limited number d of directions.

Computing equation 3.5 via Monte Carlo requires, for each sampling direction, the computation of $\hat{\tau}(\boldsymbol{x}, \boldsymbol{\omega})$. This calculation is again non trivial, as it requires a ray marching integration. Firstly, all intersections between a ray, with origin $\boldsymbol{x}$ and direction $\boldsymbol{\omega}$, and the mesh are computed. Such process would usually require an $O(m)$ time, where m is the number of triangles of the mesh, but we accelerate this computation by using an indexing data structure for the outer mesh. Once all intersections are known, the scattering coefficient is sampled at small random intervals along the ray between each couple of intersections.

To preemptively reduce the number of samples to be processed at the final stage, all points whose total receivable light is under a given threshold (i.e. such as $\int_{\Xi} e^{-\hat{\tau}(\boldsymbol{x},\boldsymbol{\omega})}\, d\Omega(\boldsymbol{\omega}) < \epsilon$) will be discarded as inconsequential. In fact, those points will always contribute an infinitesimal part to the lighting of a surface point for every lighting condition.

As can be easily seen, this stage is extremely expensive, as it requires $O(m \cdot v \cdot d)$ time, where v is the total number of vertices of the internal meshes. Even if this density value is needed at the next stage, after a large number of uniformly random samples is produced, we chose to perform it before and later interpolate computed values. Performing density estimation at this point in the overall algorithm allows for a balanced compromise between accuracy and computation times.

3.2.4 Poisson Disks Sampling

In the final stage, we seek to sample layers, using the computed desired density, to produce samples that, even if randomly placed, do not interfere with each other forming clusters of overcrowded areas or holes. Poisson Disks sampling produces a set with the guarantee that no two samples can be closer than a certain distance. By varying this minimum distance according to desired density, we obtain a set with the desired properties.[24]

The algorithm can be naively implemented with a dart throwing strategy, by discarding each sample that lays within an already chosen sample. Such approach would require $O(n^2)$ time where n is the number of samples. Instead, the algorithm chosen is a variant that uses a hierarchical data structure to rapidly find out if a vertex is good or bad. This algorithm due to White and Egbert[131] has been shown to run in $O(n)$ time.

Although this algorithm is extremely fast and works well even if the desired density is not uniform, it has been observed that radius variance cannot exceed a given limit. Our algorithm then assigns the predicted mean radius[4] r to all samples, and allows it to vary between $2*r$ and $^r/_2$ according to normalized heuristic density $\frac{\rho-\rho_{min}}{\rho_{max}-\rho_{min}}$. The final, actual radius used is again stored with each vertex.

3.3 Setting Samples Parameters

In the second stage, once the final set of samples is known, we associated to each of them a floating point value Width representing the parameter a_i of the Gaussian function $\gamma_i(x) = e^{a_i||\boldsymbol{x}-\boldsymbol{b}_i||^2}$ and a vector SH of 25 floating point coefficients that represents the Spherical Harmonics compression of function $\psi_i(\boldsymbol{\omega}) = e^{-\hat{\tau}(\boldsymbol{b}_i,\boldsymbol{\omega})}$.

3.3.1 Spherical Harmonics Compression

As reviewed in Appendix A.1, projecting a function on a Spherical Harmonics basis is performed by integrating the product of the function with each basis function. Such integration is again achieved with a Monte Carlo method. Instead of limiting the number of sampling directions as we did for the estimation of the density, this time we set such number to be reasonably high. We empirically found that 128 directions are enough to produce plausible results. Moreover, sampling at more than 256 directions does not sensibly change the quality of results.

Compression error can be estimated by evaluating the integral of the squared optical depth[108]. As Spherical Harmonics compression is theoretically loss-less for an infinite degree (and infinite number of coefficients), the following holds:

$$\sum_{l=0}^{\infty}\sum_{m=-l}^{l} |\hat{\tau}_{x\,l,m}|^2 = \int_{\Xi} \hat{\tau}^2(\boldsymbol{x},\boldsymbol{\xi})\, d\Omega(\boldsymbol{\xi})$$

Then, to obtain accuracy $(1-\varepsilon)$, the maximum order n must be such that

$$\sum_{l=0}^{n}\sum_{m=-l}^{l} |\hat{\tau}_{x\,l,m}|^2 \geq (1-\varepsilon)\int_{\Xi} \hat{\tau}^2(\boldsymbol{x},\boldsymbol{\xi})\, d\Omega(\boldsymbol{\xi})$$

Based on such estimate it would be possible to increase the number of coefficients when the error is over the desired threshold and reduce them when the function is very smooth; this optimization, however, is left for a future work. In our work we have empirically found that the fifth order (25 coefficients) is enough to represent the exponentiated optical depth with a reasonable error margin, producing data-sets with a small memory footprint.

[4]see note 3

3.3.2 Gaussian Functions Width

From the Poisson Disk sampling algorithm we know the radius of each sample, i.e. the distance under which there are no other sample. Poisson Disk sampling, however, guarantees that there cannot be other samples in the same radius, but does not place any bound on the maximum distance of the closest sample. Fortunately, in practice this distance is not far from the requested radius and a small scaling factor is sufficient to account for this discrepancy. We use this information to set the parameters a_i of Gaussian functions centered on the sampling points. As it is nearly impossible to produce a set with the properties described in equation 3.3, we will produce a reasonable approximation by setting the function to have full width at half maximum[5] proportional to the radius produced by Poisson disk sampling.

Gaussian Functions of the form $\gamma_i(\boldsymbol{x}) = e^{a_i||\boldsymbol{x}-\boldsymbol{b}_i||^2}$ have a width at half maximum of $\delta = \frac{2\sqrt{\ln(2)}}{a_i}$, hence we will set

$$a_i \sim \frac{2\sqrt{\ln(2)}}{\delta}$$

where δ is the requested radius of the associated Poisson disk.

3.4 Rendering Algorithm

We implemented our algorithm using the cross-platform *OpenGL* rendering API. In a first phase, the outer mesh is rendered using basic OpenGL functions, while, in a second phase, Spherical Harmonics are evaluated and passed, along with the width of the related Gaussian function, to a shader program written with *GLSL*, the OpenGL language for shader programming.

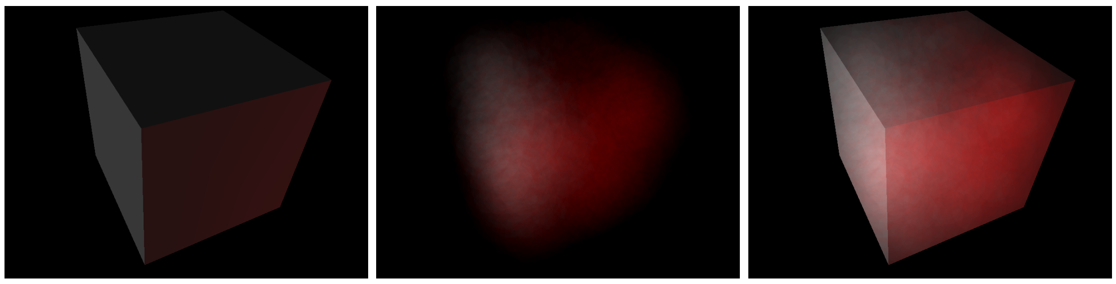


Figure 3.9: **Rendering Passes** The mesh is firstly rendered using Phong shading, then samples contributions are computed on the GPU. The final result is given by the clipped sum of the two passes.

3.4.1 First Pass

The input mesh is firstly rendered, according to simple Phong shading, using color and normal information defined at each vertex. This pass is necessary to compute, on one hand, the reflected component $L_r(\boldsymbol{x}, \boldsymbol{\omega})$ of light at each surface point, and on the other hand, to determine the region where samples must be rendered (clipping region).

This rendering is not immediately shown to the user, as it is saved on a hidden color buffer stored on the graphics device memory. As part of the normal order of operation, a depth buffer is produced together with the color buffer. Each pixel of the depth buffer contains the distance from the viewer of the rendered primitive visible at that pixel. In order to allow quick access to perform clipping computations, this depth buffer is saved as a texture and kept in device memory.

[5]the width of the Gaussian “bell” when the function has half peak strength

3.4.2 Second Pass

To simplify rendering procedures, in our implementation we used only point light sources (array lights in algorithm 3) located at a limited number of positions $\boldsymbol{l}_j$ with associated intensities I_j, specified as colors. For each sampling point $\boldsymbol{b}_i$, the following multiplication is performed

$$v_i = \underbrace{\frac{1}{4\pi}}_{\text{phase function}} \cdot \underbrace{\kappa_s(\boldsymbol{b}_i)}_{\text{scattering coefficient}} \cdot \underbrace{\psi_i(\boldsymbol{\omega})}_{\text{view attenuation}} \cdot \sum_j \underbrace{I_j \cdot \psi_i(\boldsymbol{l}_j - \boldsymbol{b}_i)}_{\text{attenuated light}}$$

A shader program is then activated, and its execution begins by forwarding Gaussian centers $\boldsymbol{b}_i$, computed colors v_i and Gaussian widths a_i.

Algorithm 3 Rendering Stage

```
Render(Mesh mesh, Sample[] points, Light[] lights)
{
    //First Pass - Compute lighting component L_r
    Render(mesh);

    //Second Pass - Compute lighting component L_ss
    for i = 0 ... points.size
    {
        Color color = BLACK;
        for j = 0 ... lights.size Light enters the system from the surface boundary ;
        {
            light_direction = lights[j].position -
                points[i].position;

            color += lights[j].color *
                points[i].SH.Eval(light_direction);
        }
        color *= k * points[i].scattering_coeff *
            points[i].SH.Eval(view_direction);

        //Render all points using fragment/geometry shaders
        RayGaussianIntegration(points[i], color);
    }
}
```

Fragment shaders are perfectly suited to perform parallel execution of code that computes a value associated with a pixel on screen. To each pixel, according to the pinhole camera model, corresponds exactly a ray[6], originating in the viewer location and directed towards the position of the pixel on the virtual image plane.

In order to trigger fragment shader evaluation for each ray, and for each Gaussian separately, a primitive covering the desired area of evaluation must be submitted to the rastering engine. In our scenario, the desired area is composed of all pixels whose associated ray intersects the Gaussian function within a certain radius from its center. In fact, even if Gaussian radial functions are never null over their entire domain, their values become rapidly negligible, and it would be useless to integrate over infinitesimal values. Let us then define $\varepsilon = 10^{-6}$, i.e. a small value under which we can assume the Gaussian function to be zero. The distance $r = ||\boldsymbol{x} - \boldsymbol{b}_i||$ for which $e^{-a_i^2||\boldsymbol{x}-\boldsymbol{b}_i||^2} < \varepsilon$, is

[6]Excluding multisampling strategies to avoid aliasing effects

$$r = \frac{\sqrt{\log(1) - \log(\varepsilon)}}{a_i}$$

To activate fragment shaders only when rays cross the Gaussian function within radius r from the center, the optimum would be to rasterize a sphere. However, a common compromise between number of useless fragment processed and number of submitted vertices is to use a cube of side $2r$ centered on $\boldsymbol{b}_i$.

Furthermore, instead of submitting cube triangles to the raster engine, we can reduce the number of costly data transfers from the host to the device by forwarding just a_i and $\boldsymbol{b}_i$ to a geometry shader. The geometry shader computes r from the above formula and generates at run time all vertices and necessary topology to draw a cube.

The raster engine is designed to interpolate values, defined at vertices, across the surface of a triangle. With programmable shaders, it is possible to set cube vertices positions as these values. When the fragment shader is finally invoked it will not only know the position of the vertex at the center of the cube, but also the position in space of the portion of the cube that has triggered the fragment shader. This two pieces of information, along with the knowledge of the user position, are enough to determine the angle ξ and the distance between the viewer and the Gaussian center $||\boldsymbol{b}_i - \boldsymbol{v}||$, the only two variables required to analytically compute the integral of the Gaussian function. (see figure 3.10)

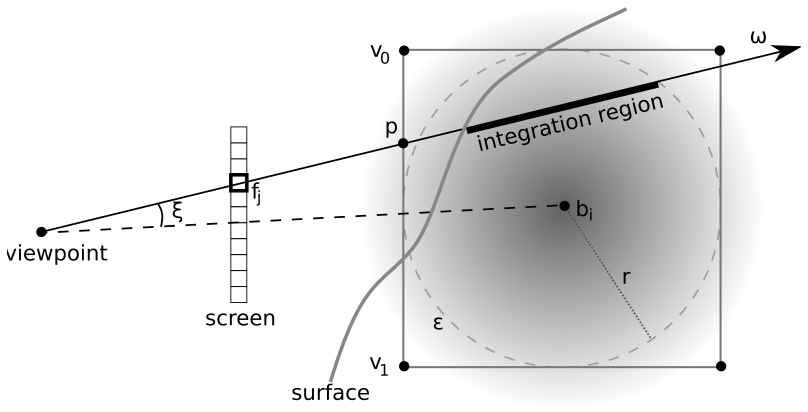


Figure 3.10: **Gaussian Rendering** The area in which the Gaussian function has values above ε is approximated using a cube. The contribution of the Gaussian function i to pixel f_j is given by the integral of the function along the ray $\boldsymbol{\omega}$. To compute such integral it is necessary to know angle ξ, which can in turn be computed by knowing $\boldsymbol{b}_i$, the viewpoint and $\boldsymbol{p}$. Coordinates of point $\boldsymbol{p}$ are automatically generated by the rastering engine by simple interpolation of the coordinates at the vertices ($\boldsymbol{v}_0$ and $\boldsymbol{v}_1$ in the figure)

As fragment shaders parallely complete their calculation, their results are blended together by simple summation, adding to the previously computed color of the outer mesh, the color contribution of each Gaussian. Finally, the result is displayed to the user.

3.5 Results

We implemented our algorithm in C++ and OpenGL, using OpenCL for the preprocessing phase and and GLSL for the shader programs in the rendering phase. All the tests were run on a Core 2 Duo @ 3.0 GHz 4GB RAM equipped with Nvidia GeForce 9800 GX2 512MB graphics card.

3.5.1 Comparison with ground truth

Our method relies on an approximation scheme configurable with a three parameters: the number of samples, the number of directions and the number of SPH coefficients. We do not give a formal proof of the bound on the error committed as a function of these parameters, but we can empirically show their influence by comparing images generated with our algorithm with the ground truth image obtained by ray tracing the original density field and integrating single scattering along the rays. Note that in this context "ground truth" is only referred to the single scattered contribution, which is the focus of this work, therefore the other phenomena are ignored in producing the image.

The first row of Table 3.11 shows the ground truth image for a view of a homogeneous sphere, while the next 3 rows show images taken with our method at an increasing number of sampling directions (left column) and the pixel-by-pixel difference with the ground truth image (right column). Note that the number of final sampling points increases with the number of directions. This is due to the fact that more directions capture more features and confirms that the divergence criterion adopted to sample the direction- dependent optical depth is effective.

The Figure 3.12 shows two artificial examples. In the first example we place a torus inside a cube as shown in Figure 3.12.a and process the dataset considering the cube as the boundary of a translucent object containing an opaque part (the torus). Figure 3.12.b shows the result of our rendering algorithm putting behind the cube a single white light. Note that the geometric description of the torus is no more part of the dataset, but the sampling of the direction dependent optical depth captures its effect on the scattering of light inside the volume. A similar example is shown in Figure 3.12.c, where two lights (a white and a red one) are placed roughly symmetrically about the cube. Figure 3.12.d shows a dataset composed of a cube which volume is procedurally defined as a set of layers with two different density values, visible in the rendering.

3.5.2 Efficiency

The preprocessing time of our algorithm is dominated by the time to compute the directional dependent optical depth, which in turn is linearly proportional to the number of sampling directions. Table 3.1 show the preprocessing times decomposed for the steps described in Section 3.2. The number of frames per time is linearly proportional to the number of samples used, and in all our experiments with a single light is constantly over 60 for a number of samples ranging from $8k$ to $11k$.

Note that it could be possible to implement a fully GPU version of our sampling algorithm, thus severely reducing preprocessing times. However, in the next chapter we present a method that is capable of running without a preprocessing stage.

DDOD	Div	PDS	SPH coeff
43.02	1.76	15.39	28.92
168	1.71	17.64	112
664	1.78	19.16	446

Table 3.1: **Preprocessing times.** Construction times (seconds) for the gargoyle model varying the number of directions for constructing the direction-dependent optical depth. DDOD: time for direction dependent optical depth computation; Div time for computing the divergence; PDS: time for the Poisson Disk Sampling algorithm; SPH: time for the spherical harmonic coefficients.

Table 3.2 shows that the fps decrease is only sub-linearly the increasing of the number of lights. This results is obtained because, thanks to the precomputation of the direction-dependent optical depth, adding a light only costs one more evaluation of the spherical harmonics. Table 3.3 shows the linear relation between rendering time and number of samples. It can be seen that the algorithm produces over 30 FPS with $100K$ samples.

n. lights	2	4	7	8
fps	62	51	42	38

Table 3.2: **FPS when augmenting the number of lights in a scene.** Tests where made with 58297 samples with 25 SPH coefficients on a 800×800 viewport.

n. samples	25000	50001	107833
fps	117	55	31

Table 3.3: **FPS in relation to the number of samples** on a 800×800 viewport.

3.5.3 Discussion

As in the works of Zhou et al. [136, 137] our approach represents the volume with radial basis functions to accelerate the computation of scattering effects. However, instead of storing density information, we use sampling points to encode a direction dependent optical depth. In this way, ray marching can be done without sorting the spheres front-to-back and therefore avoiding the overlinear cost of a sorting algorithm and finally allowing us to use a much greater number of particles. As a consequence, we have a reduced approximation error which allows us to obtain satisfying visual results and to sustain higher fps. Clearly this is possible because the object is assumed to be static and the direction dependent optical depth does not change over time. On the downside, only single scattering effects have been considered, thus reducing the applicability of the method to materials of low albedo. Furthermore, refraction effects, which are important for the realistic appearance of semi-transparent materials, have been largely ignored. Even if it is possible to implement a ray tracing strategy capable of accounting for refraction at the first interaction with the object, in order to perform a physically accurate simulation of refraction in solid materials, it is necessary to resort to a different approach.

An intrinsic limitation with this approach is due to the use of non perfectly overlapping Gaussians. Either a different (and potentially more expensive) space partitioning strategy is employed, or else a noise reduction algorithm operating a posteriori in image space. Most of the artifacts as visible in the images (e.g. bottom left of figure 3.13) arise from the early cut-off of the region of influence of the Gaussians, since a cube of limited area is used to activate their integration. However, increasing the cube's areas is not a solution as it would drastically increase render times. Other noise reduction strategies can be considered interesting directions for future work.

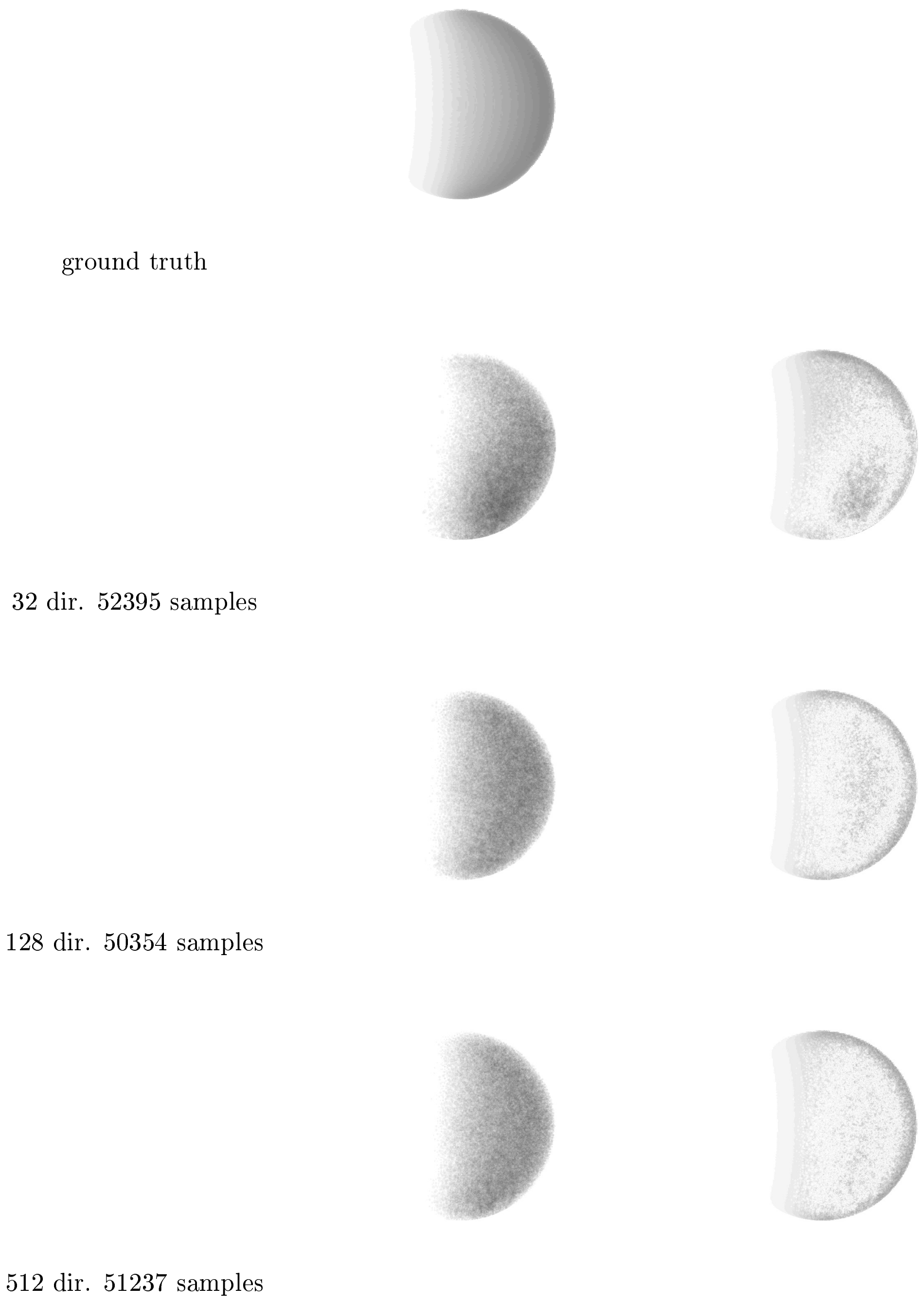


Figure 3.11: **Ground truth comparison.** Dependency of the approximation on the number of sampling directions. The top image is the ground truth, on the left column our approximation and on the right the difference with the ground truth. Original pixels values are inverted compressed in a darker range to make the result visible when printing.

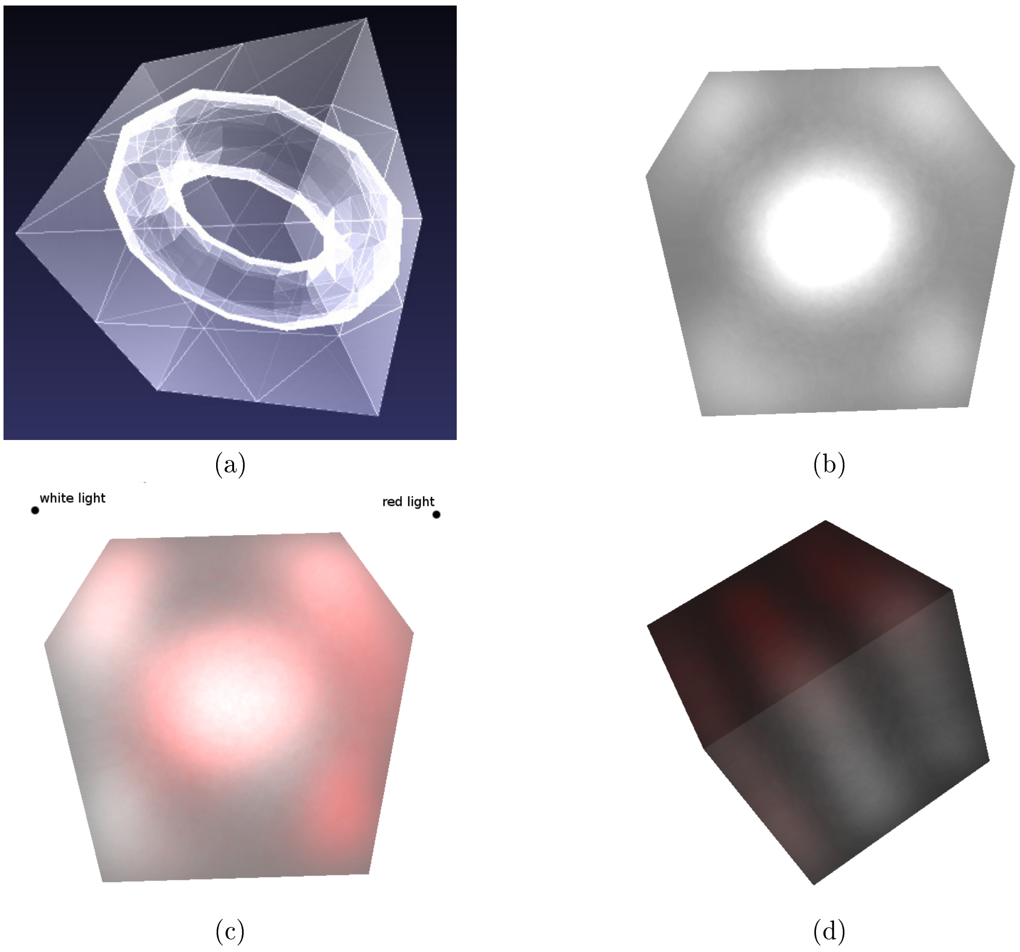


Figure 3.12: **Capturing subsurface details.** (a) a cube with a torus inside. The cube is taken as external boundary of a translucent object and the torus as an opaque blocker. (b-c) Two renderings that show how the effect of the blocker is captured by the sampling. (d) a procedural volume with two values of density arranges in parallel layers.

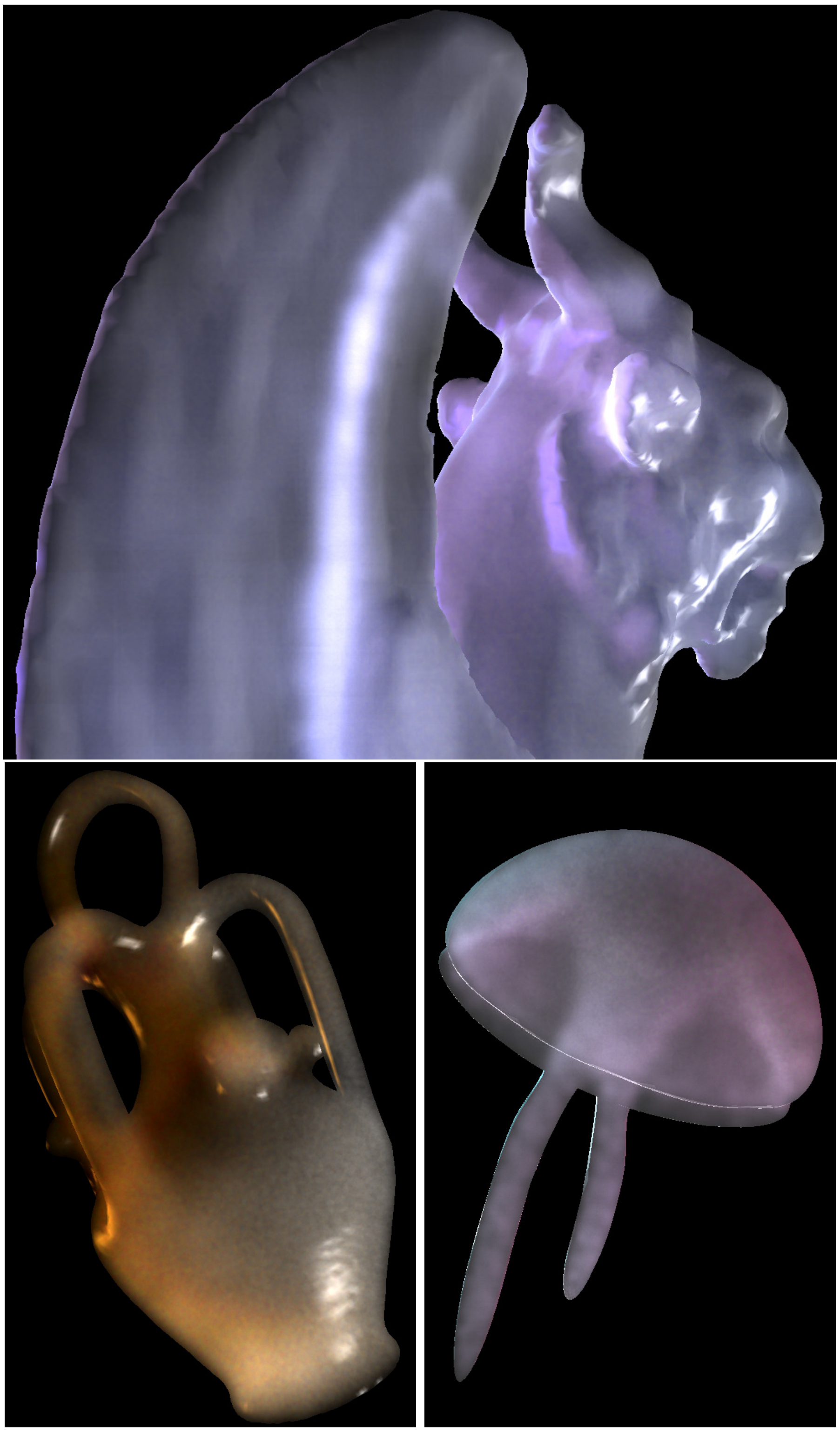


Figure 3.13: **Rendering of translucent objects**: a statue, a bottle (botijo) and a jellyfish.

Chapter 4

A Parallel Ray Tracing and Lattice-Boltzman Method

Abstract

In this chapter we present a different solution which combines accurate tracing of light rays in inhomogeneous refractive media to compute high frequency phenomena (i.e. Single Scattering and refraction) with a Lattice-Boltzmann method to account for low-frequency Multiple Scattering effects. The presented technique is designed for parallel execution of these two algorithms on modern graphics hardware. In our solution, light marches from the sources into the medium, taking into account refraction, scattering and absorption. As a light ray marches inside the volumetric representation of the scene, Irradiance is accumulated and it is diffused with the Lattice-Boltzmann method to produce multiple scattering effects. The resulting architecture is capable of running at real-time speeds without precomputations and produces results which are comparable to state of the art methods.

Materials of albedo within the low to mid range ($\Omega \leq 0.8$) cannot be captured adequately under a single scattering approximation. Moreover, if the albedo is in said range, the effects of refraction greatly contribute to the realism of the produced pictures and should not be ignored. If materials are heterogeneous in their scattering coefficients then it follows that refractive indexes throughout the volume should be heterogeneous as well. Hence, a standard straight-line integral cannot be computed for this kind of materials and a form of "eikonal tracing" becomes a necessity: rays of light naturally bend when traversing heterogeneously refractive objects.

In order to overcome the limitations of the method of the previous chapter we set out to develop an architecture which could enable single scattering, multiple scattering and refraction effects, by fully exploiting the power of modern GPUs as exposed by the OpenCL application programming interface.[79]

Multiple papers have shown[137, 109, 31] that scattering is best treated by separating the single component from the multiple component in order not to loose the highly directional information associated with single scattering. However, a ray tracing or a photon tracing paradigm seems best suited for SS computations, while Multiple Scattering can benefit from a more volume-centered approach. Lattice-Boltzmann lighting is such a volume-centered method, and it is capable of effectively capturing diffusion-like multiple scattering effects. We therefore set out to develop an architecture that is capable of supporting these two approaches and to run them in parallel, performing multiple scattering diffusion to single scattering marching. It is nowadays apparent that Moore's law applied to GPUs is more related to the number of cores per unit than to their clock speed, therefore our goal is to use as much as possible the multiplicity of cores instead of just their speed.

Summarizing, the proposed algorithm brings a twofold contribution to the state of the art:

- a real-time rendering engine that takes into account refraction, scattering and absorption for inhomogeneous media;

- a novel parallel rendering framework specifically designed to harness the modern GPU architecture and amenable to scale well to large scenes.

To the best of our knowledge the proposed method constitutes the first real-time approach to simulate refraction and multiple scattering effects in heterogeneous media.

4.1 Overview of the algorithm

In our architecture, single scattering is computed on a per ray basis, while multiple scattering is computed on a per voxel basis. For each light source, rays are generated and traced through the scene, leaving irradiance within the voxels that are visited. While rays are still traversing the volume, a diffusion process (LBL) starts spreading the irradiance deposited by the rays to neighboring voxels, simulating the effect of multiple scattering. When both the ray traversal is complete and diffusion have finished, view rays are shot, traversing the scene and reading back the irradiance.

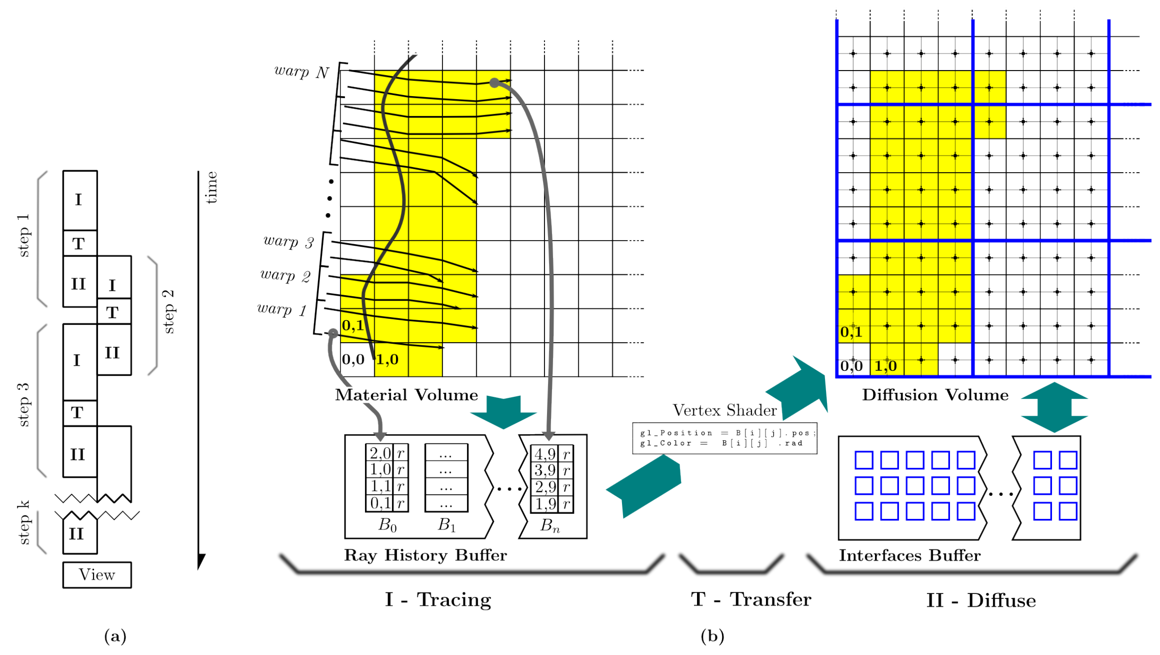


Figure 4.1: **A schematic description of our parallel rendering pipeline.** On the left: a timeline of our pipeline, where for each iteration step n of the algorithm, phase I (tracing) is run concurrently with phase II (diffuse) of iteration step $n-1$. Note that the transfer phase T must run by itself because it accesses data structures from phase I and II. On the right: the three phases. During the tracing phase (I), multiple threads (grouped in work-groups) run concurrently on the GPU, tracing light through a limited number of voxels (around 8 in most of our tests), and storing their deposited irradiance in the Ray History Buffer. The transfer phase then takes these values and transfers them to the diffusion volume through a shader program. Finally, during the Diffuse stage (II) on the GPU a thread per each voxel computes the diffused local irradiance distribution based on the light incoming from neighbors.

In summary, we can logically divide the operations performed by our algorithm in the following stages:

1. **Init**: computes the voxelization of the scene and the gradient of the refractive index, and initializes the light rays.

2. **Marching**: traces the light rays from the sources through the volume, accounting for refractive index, scattering and absorption coefficients. Finally, it stores in the voxels the percentage of attenuated light that has to be scattered. The process ends when all rays have left the scene or there is no more light to carry.

3. **Diffusion**: computes the multiple scattering contribution of the light deposited in the Marching pass.

4. **View**: casts rays from the point of view and gathers the irradiance.

Our algorithm is built around the observation that the diffusion stage should run concurrently with the marching stage. Theoretically, in order for a scattering simulation to be physically correct, there should be no temporal gap between the moment light reaches a particle of a volume and the moment this particle starts re-emitting energy. Therefore, in our parallel architecture as soon as a portion of volume has been traversed by light rays, the irradiance accumulated in the volume elements gets diffused throughout the volume without waiting for the end of ray traversal. During the diffusion pass, rays may continue to march through the volume. This structure ensures that if an early termination of the algorithm is requested, provided that the tracing stage has been completed, then multiple scattering will be approximated to the number of local bounces it could perform up to that point, yielding plausible results. Figure 4.1.a shows a temporal diagram of the algorithm.

Note that since the tracing pass and the diffuse pass are built around two different paradigms, there is a necessity to implement a custom stage to transfer accumulated energy from the buffer of rays to the voxels used in diffusion. This is the transfer stage (T in figure 4.1) and it is outlined in section 4.4.4.

4.2 GPU Parallel Programming

Despite the power of Graphics Programmable Units exposed by the OpenGL interface, which constituted the backbone of the method of the previous chapter, this graphics-oriented API limits programmability to those algorithms that can be mapped efficiently to the shader model. Thankfully, the advances in Graphics Hardware and the availability of more open APIs in the last decade have finally allowed programmers to perform general purpose computing on GPUs. The two most complete and stable API at the time of this writing are OpenCL and CUDA. We decided to use OpenCL because of its greater portability and ease of integration with existing programming environments; furthermore, recent investigations (e.g. [42]) have shown that in term of performance the two APIs are equivalent. Despite the different naming conventions that they adopt, their architecture is similar, since it closely maps on the actual hardware architecture (see figure 4.2).

The abstract architecture upon which the OpenCL specifications[79] are built has at its lowest level the concept of *Processing Element* (PE), that is, a register-based, programmable computing unit which is able to run user-defined code routines, or *kernels*, and to access memory in read and write mode. In addition to operate on its own registers, a PE can access a *private memory*, which is typically very small and extremely fast. PEs are grouped together to form a *Streaming Multiprocessor* or *Compute Unit* (CU), and can concurrently access a dedicated small amount of fast *local memory*, with the ability to implement communication mechanisms using synchronization primitives. A group of CU form the top of the abstract architecture, called a *Computing Device* (CD), equipped with a large amount of relatively slow *global memory*. This memory resource represents the only communication channel between PEs belonging to different CUs and between the *host* OpenCL application and the CD.

A fundamental difference in the execution model between a multi-cpu system and the actual hardware implementation of a CU is that all the PEs belonging to the same CU share their instruction pointer: this implies that, to obtain maximum processing throughput, all the instances of instructions which conditionally modify the control flow (e.g. conditional jumps) must generate the same execution path in every PEs of the CU. Whenever this condition is not met (*branching*

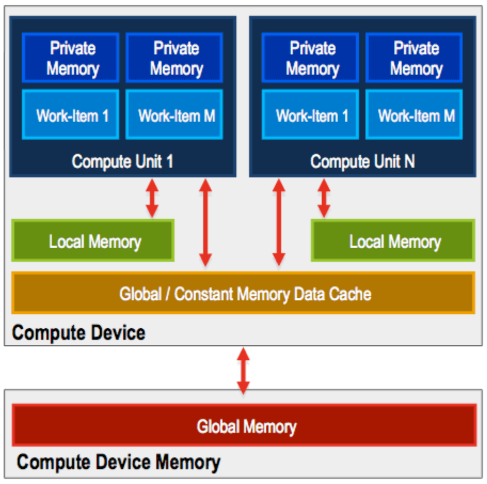


Figure 4.2: **Memory structure in OpenCL**

divergence), e.g. when the condition of an *if-then-else* statement evaluates differently among PEs, the shared instruction pointer must proceed through all the branches of the execution flow and every PE is in charge of masking out all the instructions which are not executed, thus degrading the performances.

Running an OpenCL computation means instancing a kernel on a virtual N-dimensional[1] *computing grid*, virtually represented by the CD. Kernel execution instances is furthermore grouped into *work groups*, each virtually mapped to a CU. This architecture model implies that, to obtain a high level of performances, a kernel should exhibit as little as possible branching divergence and should mainly use private and local memory, possibly avoiding synchronization.

In general, the execution of a parallel OpenCL computation consists of: (1) transferring data from host memory to the device global memory, (2) defining the execution shape (computing grid and work group size), (3) instancing a parallel execution of a kernel over the computation grid and (4) reading back data from device global memory to host memory.

In the context of our work it is important to note that, differently from previous generation hardware, recent hardware implementations allow a computation involving a *different* kernel to be executed before the previous one has completely finished: this means that, at the same time, different CUs on the same CD can execute different kernels. In addition, our technique exploits efficient resource (buffers and textures) sharing between OpenGL and OpenCL kernels.

4.3 Init pass

The input to our algorithm is a scene described as a set of watertight meshes, and the role of the Init pass is to fill the *Material Volume* and to initialize the light rays.

The Material Volume is stored in two separate 4-channel textures, one containing the 3-channel extinction coefficient κ_t and the occupancy value o (the percentage of the voxel that is actually occupied by the object), the other containing the scalar refractive index η and its three-component gradient $\nabla\eta$. Memory consumption when dealing with this kind of data structures can be high, and we strove to optimize memory usage for auxiliary data structures, especially since access to global memory can be the main bottleneck in an OpenCL/CUDA algorithm. Nonetheless, we observed a significant increase in performance for the tracing part of the algorithm if the gradient of the heterogeneous refractive index is computed in advance and stored.

The occupancy value is initialized from the mesh description of the scene by performing a voxelization at a higher resolution (eight times the final target size in our experiments), in a

[1]where N can be 1, 2, 3

manner similar to Sun et al.[122] and using the algorithm of Fang et al.[43] implemented using OpenGL fixed pipeline operations. According to this algorithm the mesh is rendered in orthogonal projection multiple times seen from a camera positioned at a negative z value, one for each discrete value of z, and a value of 1 is recorded if a backfacing triangle is seen, 0 otherwise. This volume is then downsampled to the target size, thus setting more accurate fractionary values for the occupancy. The method has been chosen for its speed; however, it requires all meshes in a scene to be watertight. This is not a serious limitation as most production-quality meshes are already closed, and there exist many automatic techniques for closing holes in meshes.

Extinctions coefficients in Material Volume are initialized either by a user provided volumetric texture (which is upsampled or downsampled to the resolution of Material Volume) or by a procedural generator. Values are premultiplied by the occupancy value, similarly to alpha-premultiplication in images. Note that we do not allocate a separate texture for scattering and absorption coefficient and let the user specify a global albedo Ω for the material. Scattering and absorption are then recovered simply with the relation $\kappa_s = \Omega\kappa_t$, $\kappa_a = (1-\Omega)\kappa_t$. This approximation makes sense in most real world situation, where the variation in one coefficient is positively correlated to the variation in the other[2].

It is to be noted that we allow meshes to change shape and topology, and as such the entire process, which is implemented in OpenGL/OpenCL in a straightforward way, can be repeated for each frame of the animation. The added cost is noticeable but still keeps the algorithm above the interactivity threshold.

Scalar refractive index is as well multiplied by the occupancy (space not occupied by meshes is always assumed to have refractive index 1); however, after computing its gradient using finite differences, as in [64], we filter the gradient with a Gaussian kernel to avoid aliasing effects when refracting rays. These Gaussian kernels are uniform and with a standard deviation of 0.5-1.0 voxels, thus producing object boundaries that extend over 2-3 voxels. We avoid this blurred boundaries to be visible by employing a clipping operation using the original mesh geometry, similarly to what we did in the previous chapter. Note that the settings that have been chosen for the Gaussian derive from the direct observation that an increase in kernel size decreases the visual quality without positively affecting the stability of ray traversal.

Finally, additional data structures used for bookkeeping propagation of light, i.e. the Raymap, the Ray History Buffer, the Irradiance volume and the interfaces buffers, are all initialized to zero values.

Data Structure	**Quantity Stored**
Material Volume	κ_t, η, $\nabla\eta$, o
Irradiance Volume	$e(\boldsymbol{x})$
Raymap	$\boldsymbol{x}$,$\boldsymbol{\omega}$, Φ_{p_b}, *active_flag*
Ray History Buffer	$e(\boldsymbol{x})$, *voxel_id*
Interfaces Buffer	$e(\boldsymbol{x})$

Table 4.1: **Data structures used in the algorithm**

4.4 Ray tracing

The first step is implemented as a parallel photon marching pass[68], where each execution thread (or Work Group Item, in one-on-one correspondence with Processing Elements) is responsible for a single light ray. Instead of writing directly to the global Irradiance Volume, rays add entries in the Ray History Buffer, storing voxel_id and deposited light. The second step of of this pass is then to render these entries as single vertexes into the Irradiance Volume, exploiting the fast blending capabilities of OpenGL.

[2]Usually, it is variation in density in participating media that accounts for the change in coefficients. In such cases, a fixed global albedo is a valid assumption.

4.4.1 Initializing photon rays

For each light source, we render the scene to build a depth map that will be used in the *view pass* for handling shadows. Furthermore, the depth values are converted in world space and used to set the starting point for the rays, so avoiding the marching of rays through the empty volume. We set the starting point slightly before the position found with the depth value, more precisely by a distance equal to the number of voxels used for the Gaussian smoothing, as shown in Figure 4.3.(a). This is an optimization that may be disabled if the light source is inside a participating media. Rays are slightly jittered in their starting position to reduce artifacts. Although we could also have applied a small jittering in step size during the actual tracing, it would have introduced inefficient branching in the execution process.

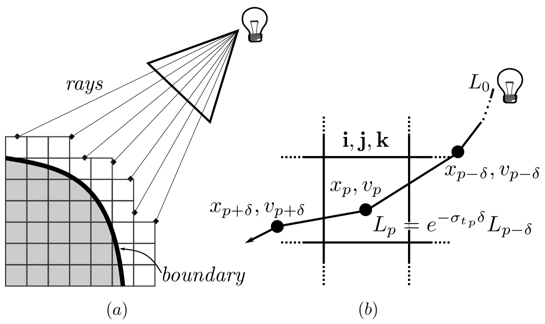

rays
boundary
$\mathbf{i},\mathbf{j},\mathbf{k}$
L_0
$x_{p-\delta}, v_{p-\delta}$
x_p, v_p
$x_{p+\delta}, v_{p+\delta}$
$L_p = e^{-\sigma_{t_p}\delta} L_{p-\delta}$
(a)
(b)

Figure 4.3: **Photon Marching** (a) Setting the starting point of rays a few voxel from the actual volume. (b) Update of particle motion and radiance attenuation according to the Eikonal equations.

We divide power of light source among photons according to the subtended solid angle of a pixel in the Raymap with respect to the light source. In order to avoid areas of low photon penetration, the raymap size must be dense enough such that the spacing between rays does not exceed a voxel of distance. We have observed that for a volume of 128^3 a raymap of 512^2 is more than enough for general cases.

Note that we still use the same single ray map strategy even when multiple lights or an environment map are present. In case of multiple lights, we subdivide the light map equally between the light sources; in the case of Environment maps, we perform importance sampling on the values within and reduce to the case of multiple spot lights. Empirical tests have shown that keeping to 512^2 the total number of rays, even when dividing them between multiple lights, does not significantly degrade the quality of produced scenes.

The internal state of rays as stored in the Raymap keeps track of registering position, carried energy and termination status (active_flag). This flag is activated when:

- the power carried by the beam falls below a given threshold.
- the beam has reached the end of the volume.
- a time limit has been reached (thus preventing total internal reflection rays from hijacking the GPU)

4.4.2 Marching Pass

In our model light gets propagated according to a process that Jarosz et al. termed[68] "photon marching", an hybrid process between photon tracing and ray marching integration. However, contrarily to the algorithm presented in their paper, we march curved beams to account for refraction, in a manner similar as Sun et al.'s[122] who however did not give a precise account of the mathematical foundation of their photon mapping process.

Photons from a ray, or stream, are deposited at regular intervals in the volume. The energy that is deposited corresponds to the amount that the stream has not lost due to extinction at the beginning of each interval, reduced by the amount that is scattered along the length of the next interval. Even if this process is not equivalent to the physical process of photon absorption and emission, energy is guaranteed to be conserved. This energy is stored at discrete locations, within the voxel in which the beginning of the interval falls. During the marching pass, only a maximum of k different voxels can be filled up with energy. This strategy allows for less memory access and removes the need for computing expensive ray-voxel intersections.

At the end of each step, the path direction is updated using the constant-time discretization of the eikonal equation (see Figure 4.3.(b)):

$$\mathbf{x}_{t+\Delta_t} = \mathbf{x}_t + \frac{\Delta_t}{n}\mathbf{v}_t \tag{4.1}$$

$$\mathbf{v}_{t+\Delta_t} = \mathbf{v}_t + \Delta_t \, \nabla n \tag{4.2}$$

where Δ_t is the integration time step, corresponding to the size of the integration interval, and $\boldsymbol{x}_t$ and $\boldsymbol{v}_t$ are respectively the position and the velocity of the particle along its path at distance t from the source, and n is the refractive index. In order to further reduce artifacts, the value ∇n in a point inside the volume is trilinearly interpolated from the values in the *material volume*.

The power of a deposited photon at $\boldsymbol{x}_t$ is determined by Beer's law of attenuation: [68]

$$\Phi_{p_b} = \kappa_s e^{-\int_0^t \kappa_t(t')\,dt'} \Phi_b \Delta t \tag{4.3}$$

where Φ_b is the power of the beam upon entering the medium and Φ_{p_b} is the power of deposited photon. A photon is deposited at each $\boldsymbol{x}_t$ as defined by the eikonal equations with t at discrete increments with a small initial perturbation to avoid aliasing.

The area of photon reception is a voxel, and this has volume V. Incoming radiance as determined by photons is thus:

$$L(\boldsymbol{x}, \boldsymbol{\omega}) = \frac{1}{V} \sum_{photons} p(\theta_b)\kappa_s \Phi_b e^{-\int_0^t \kappa_t(t')\,dt'} \Delta t$$

We switch to irradiance for storage since we are dealing with isotropic phase functions, thus, since we know that phase functions are normalized to one, we can just add the value:

$$e(\boldsymbol{x}) = \int_{4\pi} L(\boldsymbol{x}, \boldsymbol{\omega}) d\boldsymbol{\omega} = \frac{1}{V} \sum_{photons} \Phi_{p_b}$$

Note that this is akin to performing a photon mapping operation, using a regular grid as a storage volume and performing a local search of photons restricted to one cubic cell per query position. While it is true that this approach, although correct from an energy-balance point of view, can potentially lead to artifacts, these are reduced by employing a small sampling step (in our case it is never much more than two voxels) and a sufficiently high number of photons.

4.4.3 Parallel Implementation

Traditional serialization methods, e.g semaphores, though partially supported on the latest GPUs, are still highly inefficient due to the nature of the SIMD execution model. Therefore we resorted to a custom buffering mechanism which is similar to the one employed by Ihrke et al.[64].

Figure 4.4 shows a C-like description of the algorithm for a ray. Each ray is assigned to a dedicated OpenCL thread and executed in a Processing Element (PE). We start this pass by instancing a 2-dimensional computation grid with R threads, with R the number of rays. At the beginning, the rays are assigned to the same Streaming Multiprocessor (SM) in packets, so that, at least at the first marching steps, access to global memory tends to be coherent. Unlike in [64], we use a fixed *length* step, Δt, because all threads in the same SM share the same instruction pointer. Furthermore, to obtain maximum processing throughput, all the instances of instructions which conditionally modify the control flow (e.g. conditional jumps) must generate the same execution path in every PEs of the SM have to complete their execution at the same time.

The *material volume* resides in slow global memory and it is read-only, while the *LocalRay-HistoryBuffer* is a write-only vector of k positions and resides in the fast local memory of the SM. When the i-th voxel is traversed, the program writes to this vector at position i (the position of the voxel) the amount of radiance it is carrying. The last instruction of the program, *UploadToGlobalMemory*, copies the *LocalRayHistoryBuffer* to the appropriate slot in the global *Ray History Buffer*. When all threads terminate, we will have a buffer in global memory, the *Ray History Buffer*, which is filled with the index of all voxels traversed by all rays and the amount of radiance to be left in each one (see Figure 4.1.b).

```
RayMarching(x,v,L0){
  buff = 0; L = L0;
  do{
      (i,j,k) = voxel containing x;
      sigma,grad_n,n,occupancy =
         MaterialVolume[x,v];
      if (occupancy != 0){
        L = exp(- delta * (sigma)) * L;
        LocalRayHistoryBuffer[buff] =
           [(i,j,k),L];
      }
      x=x+v/n *delta; v=v+ delta * grad_n/n
      ++buff;
    }while(buff < max);
    UploadToGlobalMemory();
}
```

Figure 4.4: **The algorithm executed on each thread.** Note that the only potential source of branching divergence is provided by the instruction checking the occupancy. This is not an issue because different threads generally come up with different results only at the very beginning, when entering the volume.

4.4.4 Transfer

This operation completes the marching pass by accumulating the content of the *Ray History Buffer* onto the *Diffusion Volume*. The *diffusion volume* is another grid in one-to-one correspondence with the *material volume*. A voxel of the *diffusion volume* stores a single RGB irradiance value. We perform the transfer operation in OpenGL by binding the *diffusion volume* texture as a render target and issuing the rendering of a batch of $R \times k$ points. In a vertex shader, we fetch the voxel coordinates and the radiance from the *Ray History Buffer*, and use them as output position and color, respectively. Since multiple rays contribute to the same voxel, we enable blending to accumulate each contribution into a total irradiance value. The transfer is the only stage where we use OpenGL. Note that this operation cannot be done directly in OpenCL due to the lack of atomic floating point operation needed for irradiance accumulation. When the OpenGL accumulation step ends, we bind again the *Ray History Buffer* as output and restart the threads assigned to the marching pass and the threads assigned to Diffusion pass. Even though blending is an expensive operation, it is significantly faster than any other approach. Note however, that an efficient parallel implementation of stream compaction can perform similarly. We experiment with this alternative approach in chapter 5.

4.5 Diffusion pass

In this pass, we propagate the irradiance stored in the *diffusion volume*. As we did in the Marching step, we want to exploit the parallelism of the GPU by breaking up the problem into subproblems and assigning each one to a dedicated SM. Therefore, we partition the *diffusion volume* in blocks of b^3 voxels (highlighted in blue in figure 4.1.b) and compute the diffusion of light within each block. We proceed iteratively: for each iteration we run the diffusion process in parallel for each block and store the leaving photon density in an ad hoc *interfaces buffer* of size $6 \times b \times b$ that will serve as input the neighbor blocks at the next iteration.

4.5.1 Diffusion inside a block

The transport of light inside a block is computed by means of the LBL approach [47]. LBL has proven to be a viable method to simulate the light diffusion process in a medium (see chapter 2) The transport of light is discretized in space and time, by modeling the volume with a lattice where each node (i.e. each voxel) stores the photon density along a predefined set of directions and updates these densities at discrete time steps. In the original paper, each node is connected to its 6 neighbor nodes along the principal axis, and the 12 nodes along the diagonals on the 3 planes $X = 0$, $Y = 0$ and $Z = 0$. To comply with current memory limits of the SM, we use blocks of 4^3 voxels and only the 6 principal directions. Following the notation of the original paper, the photon density, $f_i(\boldsymbol{x}, t)$, arriving at lattice site (i.e. the voxel) r at time t along direction c_i is computed as:

$$f_i(\boldsymbol{x} + \lambda \boldsymbol{\omega}_i, t + \delta) = \Theta_{ij} \; f_j(\boldsymbol{x}, t) \tag{4.4}$$

where Θ_{ij} is defined as:

$$\begin{gathered}\Theta_{0j} = \begin{cases} 0 & j = 0 \\ \kappa_a & j > 0 \end{cases} \\ \Theta_{ij} = \begin{cases} 1/6 & j = 0 \\ \kappa_s/6 & j > 0 \\ 1 - \kappa_t + \kappa_s/6 & j = i \end{cases}\end{gathered} \tag{4.5}$$

The photon density is initialized from the Irradiance Volume as:

$$f_i(\boldsymbol{x}, t) = \frac{1}{6} e(\boldsymbol{x})$$

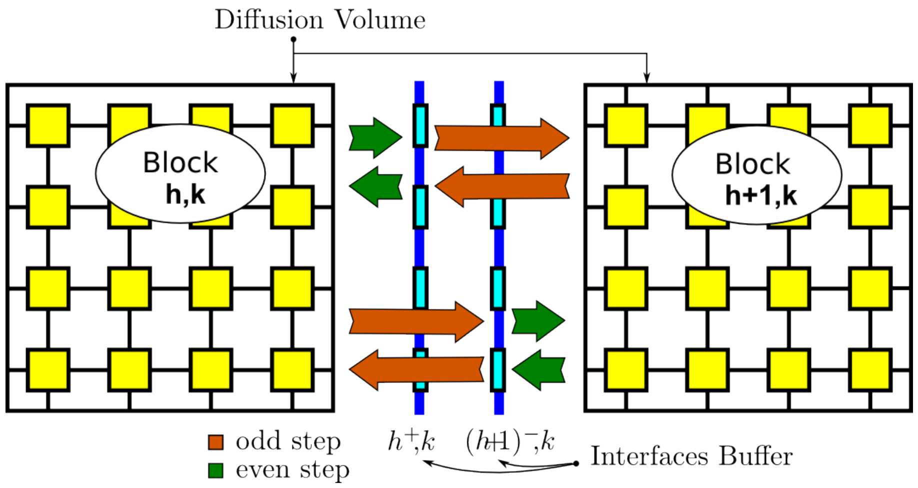


Figure 4.5: **The Interface Buffer.** The swapping mechanism which avoids conflicts in writing/reading to/from the Interface Buffer. The outgoing buffer at a block at iteration n is the incoming buffer at its neighbor at iteration $n + 1$

The memory consumption of the LBL data structure for $b = 4$ (i.e. the vector of photon density f) is 6144 bytes (6 directions, 4^3 voxels, 4 channels, and 4 bytes for storing the density). The *interface buffer* requires 1536 byte for a block (b^2 elements, 6 faces, 4 channels, 4 bytes for the density). In total, $6144 + 1536 = 7680$ bytes.
At the end of the Diffusion pass, the irradiance in the voxels is accumulated in a copy of the Diffusion Volume, that we call *Irradiance Volume*, which will be the one ultimately used in the View pass. Note that the Interfaces Buffer is accessed both for reading and writing by concurrent threads. Therefore, we must guarantee that there are no access conflicts. Figure 4.5 shows two adjacent blocks and the Interface Buffer among the two. Note that, within a block, we can use the same locations in the *interfaces buffer* both for reading and for writing, because each location is read and written only by the thread controlling the single voxel corresponding to it. Moreover, the irradiance written to the interface (h^+, k) has to be read in the subsequent step by the block $h + 1, k$ (and vice-versa). Therefore, we simply alternate h^+, k and $(h + 1)^-, k$ as *interfaces buffer* for the two blocks to avoid conflicts. This is shown in Figure 4.5.

4.6 View pass

4.6.1 Termination condition

Before performing the view pass and displaying the frame, we must guarantee that the system has reached an equilibrium state. This means that performing additional steps will not significantly change the current state of irradiance on each voxel. The number of steps typically varies with the characteristics of the volume, so it cannot be fixed beforehand. We test for convergence by monitoring the amount of irradiance in a sparse set of points and stopping when the relative change is under a predefined threshold.

4.6.2 View

The View pass is the last step of our rendering algorithm. We instantiate a 2-dimensional computation grid of the size of the framebuffer such that the calculation of the color at each pixel will be assigned to an OpenCL thread. In this pass, the rays are shot from the observer's viewpoint toward the *irradiance volume* and marched through it, taking into account refraction. At each step of the marching, a view ray accumulates a radiance contribution from the *irradiance volume*, calculating the color of the pixel. Finally, the direct lighting contribution is calculated evaluating the BRDF of the surface (i.e. Lambertian BRDF), where shadow maps computed in the Init pass are used for the visibility test.

Radiance is gathered back as a discretized beam query[68], i.e. at discrete steps, and attenuation coefficients are assumed constant throughout the query segment. Also the energy is assumed constant trough the query segment.

$$L(\boldsymbol{x}, \boldsymbol{\omega}) = \int_0^{t_{max}} e^{-\int_0^t \kappa_t(t')\,dt'} \kappa_s(\boldsymbol{x}) \frac{1}{4\pi} e(\boldsymbol{x}(t))dt$$

Note that as before, we only trace those rays that hit the volume according to the depth map, the others are immediately discarded. This has also the benefit of allowing us to clip on the actual mesh geometry, similarly to the method of the previous chapter.

4.7 Results

We performed several tests with different materials and lighting conditions on a Intel Core2 Duo 2.66 GHz, 2 GB RAM, equipped with a NVidia GeForce GTX 465.

4.7.1 Comparison with Ground Thruth

Figure 4.6 shows a comparison between our approach and a ground truth image of a sphere of homogeneous material lit by a spot light. All ground truth images have been produced using classic volumetric photon mapping. Some bandization artifacts are visible because of the blockization; moreover, due to the iterative nature of the method, propagation of energy to the far end of the object with respect to beam entry point is slow, thus producing a slightly darker image.

Particle tracing was adapted to match the same refraction model used in our technique; more precisely, we employed the Eikonal equation to bend photons when moving inside the medium. However, when moving from air into the medium, the photon tracer uses the normals of the mesh to compute Snell's equations, thus achieving higher quality renderings. The resulting difference can be noticed in figure 4.11 in details such as the Armadillo's hands or the Bunny's ears.

4.7.2 Approximation of LBL

The first test aims at evaluating the effect of the subdivision of the volumes in blocks operated in Section 4.5. Applying LBL inside each block of voxels and transmitting the photon density through the *interfaces buffer* is not the same as performing a LBL globally over the entire volume. Therefore, we ran a simple test with a single light in a homogeneous material to evaluate the difference. Figure 4.7 shows a comparison for two different albedo values, showing how the two algorithms achieve comparable results.

Figure 4.8.(a) shows a model of a homogeneous elephant placed inside an otherwise homogeneous sphere. The difference in refractive indices between the two materials produces some caustics within the object.

Figure 4.8.(b) shows a similar situation with two spheres inside a cube. Renderings in figure 4.8.(d,f) are computed from low scattering coefficient materials and therefore refraction related effects are more prominent. For example, caustics on the checkerboard and image of the checkerboard on the surface of the green sphere can be seen. Figure 4.8.(e) shows a bunny immersed in a cube and Figure 4.8.(g) show 4 frames of the animation we produced of a running elephant. For these last two examples, we also used an environment map, that we are able to easily incorporate into our framework.

4.7.3 Performances

We conducted a number of experiments to analyze the efficiency of our algorithm when changing its parameters.

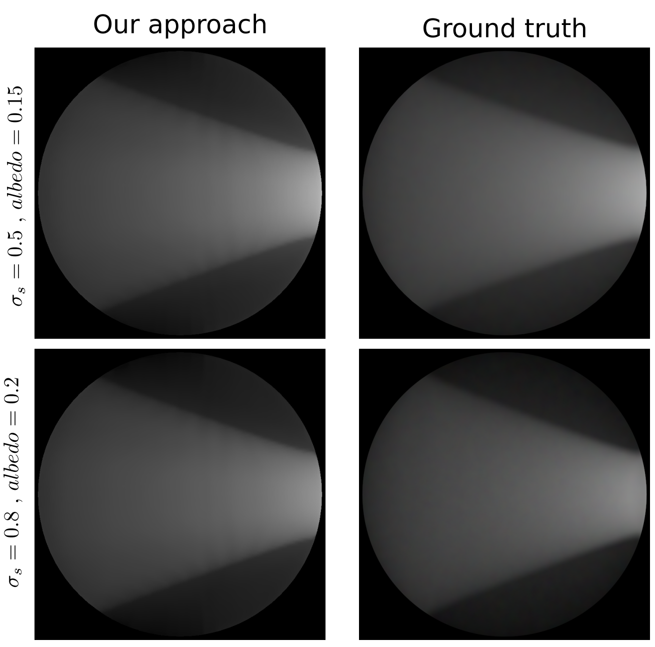

Our approach
Ground truth
$\sigma_s = 0.5$, $albedo = 0.15$
$\sigma_s = 0.8$, $albedo = 0.2$

Figure 4.6: **Comparison of our approach against a ground truth image.** Different values of scattering and albedo.

An important number is the size of the *Ray History Buffer*, because it determines the granularity of the parallelization between *diffusion pass* and *marching pass* of consecutive steps and the total number of steps to convergence. We studied the relation between the size of the *Ray History Buffer* and the total number of steps to convergence. Small values of k correspond to a dense interleaving of marching and diffusion pass and a higher number of steps of the algorithm. Large values of k means less dense interleaving and small number of steps. In the extreme case of unlimited *Ray History Buffer*, we would have a single marching phase followed by a single diffusion phase. Figure 4.9.(a) shows the rendering time for different sizes of the *Ray History Buffer* from 2 to 64 for the bubbly cube dataset. Small sizes of the *Ray History Buffer* lead to longer rendering times, mainly due to the overhead of the OpenGL transfer. For our hardware setting, *Ray History Buffer* with size $k = 16$ turned out to be an optimal value. This suggests that the proposed architecture is effective w.r.t. executing the two passes in a sequence, otherwise greater values of k should correspond to shorter times.
Figure 4.9.(b) shows the dependence between the albedo of the material and the number of steps required to reach convergence for a 128^3 volume and produce a 1024^2 image. As expected, the algorithm takes more steps and hence is slower with materials with high values of albedo, but it never falls under 8 fps even for albedo approaching 1.

A test more directed to prove the effectiveness of the framework has been conducted by running a sequential execution of a single marching and diffusion phase against our algorithm, which shows that our algorithm is on the average 30% faster. Note that this gain is only due to the parallel execution of the diffuse phase of the i-th step with the marching phase of the $(i + 1)$-th step. We expect a further improvement in the out-of-core implementation of our framework, where our approach will also benefit from the cache coherent access to memory. Note that in the present implementation the entire dataset resided in video memory.

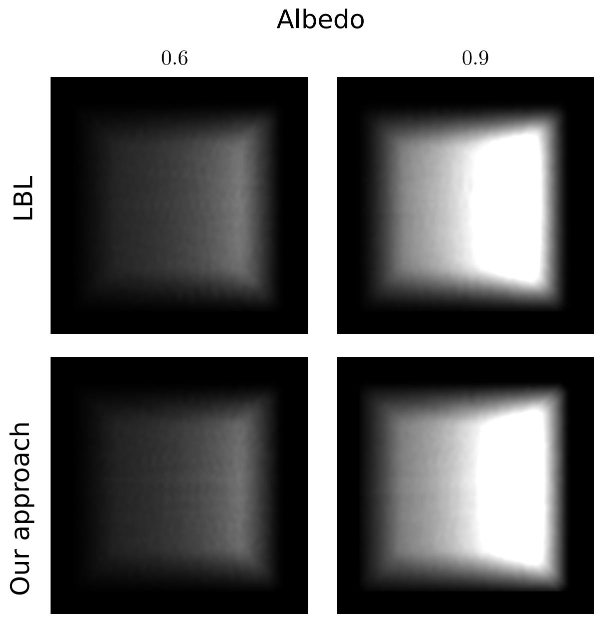


Figure 4.7: **Comparison of Lattice Boltzmann Lighting with our approach for two albedo values**. Blockization artifacts disappear after few global iterations.

4.7.4 Discussion

The method presented in this chapter is capable of presenting real-time physically-based simulations of volumetric effects. The precision of the single scattering computation has considerably increased with respect to the previous method, and we are able to account also for Multiple Scattering and refraction at the same speed as before but without any lengthy precomputation. Moreover, results show that the method produces results which agree with ground truth photon mapping simulations.

While the Marching phase and the View pass of our algorithm is mostly performed like in[122], we were also able to add MS at little additional cost, while preserving the possibility of changing light and material as in their method. The algorithm by Wang et al.[130] also implements multiple scattering of heterogeneous materials and does not suffer from the limitations due to voxelization in handling sharp features. However, it requires a complete tetrahedralization of the model which takes several minutes of computation (10 minutes for the gargoyle model) and does not take into account refraction. Light Propagation Maps[45] provide a more accurate solution for light transport in participating media, but still require minutes to produce a single image. The mesh based approach by Walter et al.[127] produces more precise refraction effects than ours but it only handles single scattering and only for one boundary between two constant refractive-index materials.

However, as the albedo reaches 1.0 (values greater than 0.8), convergence timings can become an issue and the LBL simulation becomes less accurate, especially if the grid is not extremely dense and if the time step of the simulation is not small enough. At the same time, the limited amount of memory available on contemporary GPU poses a serious limit on spatial resolution which can only be overcome by implementing an Out Of Core (OOC) system for efficiently handling virtual GPU memory. These considerations have prompted the development of the method presented in the next chapter.

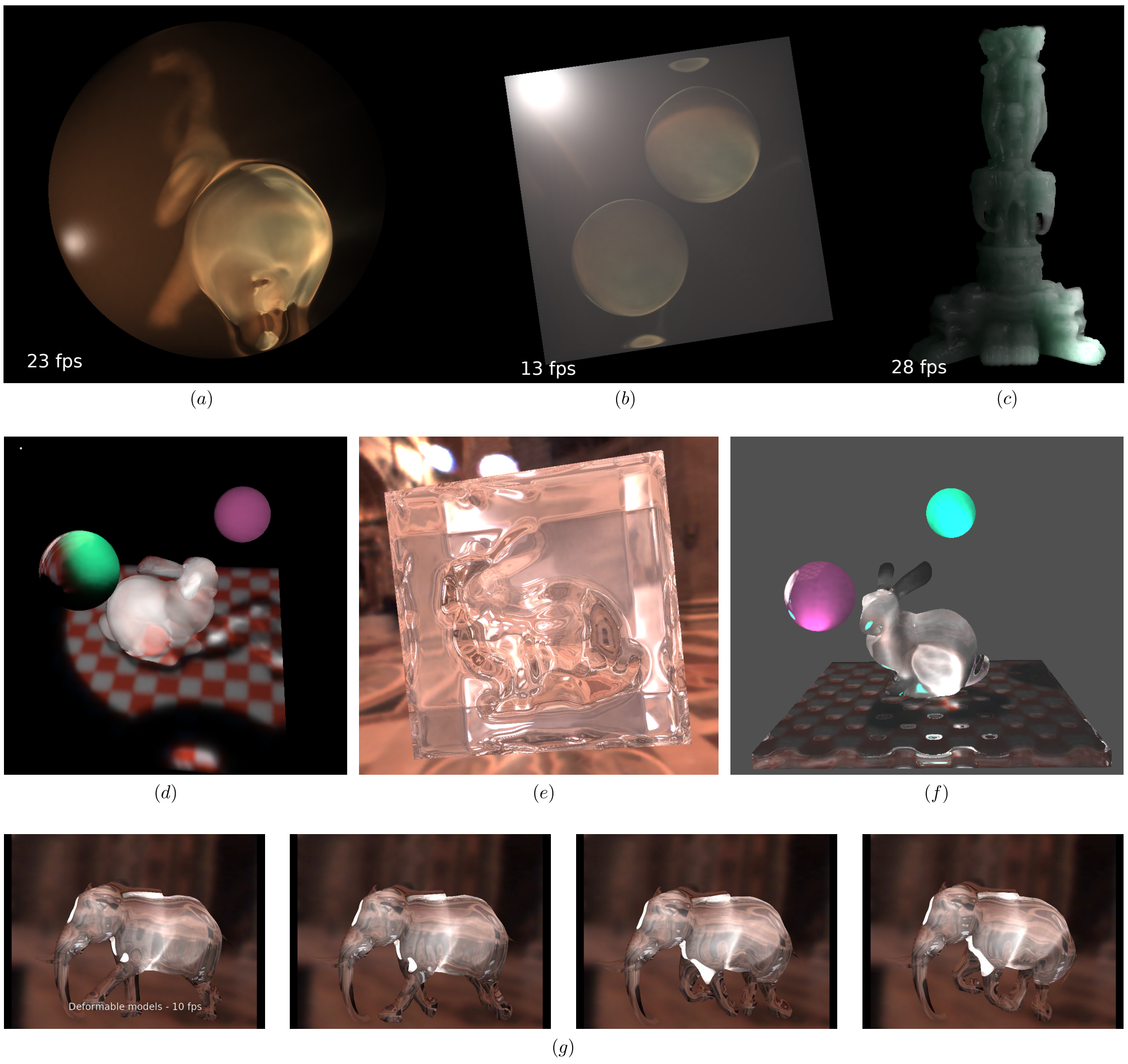
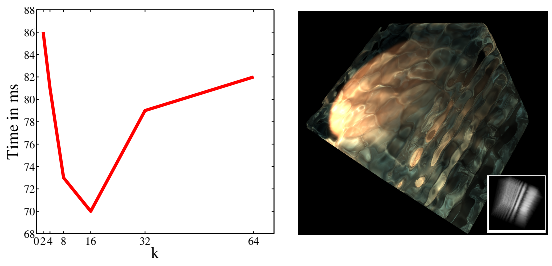
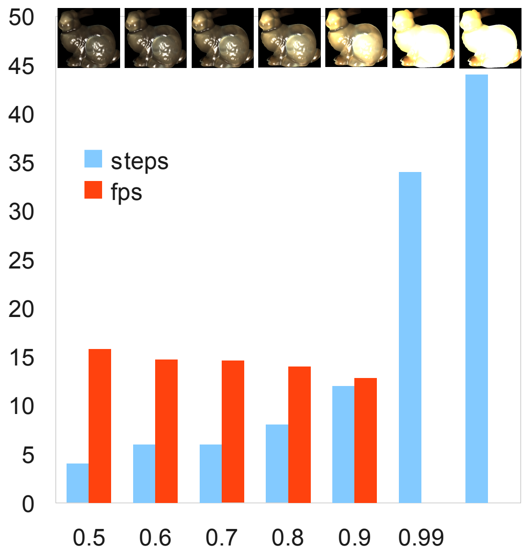


Figure 4.8: **A range of results computed by our method.** (a,b,c) Renderings showing refraction and multiple scattering. (d, f) Renderings showing refraction and multiple scattering in heterogeneous media. (e) Including an environment map. (g) A few frames from a real time animation (see accompanying video).



(a)



(b)

Figure 4.9: (a) **Rendering time** (in ms) for different sizes of *Ray History Buffer* k for the case of a 128^3 volume entirely filled. (b) **Performances of our algorithm** for different albedo values.

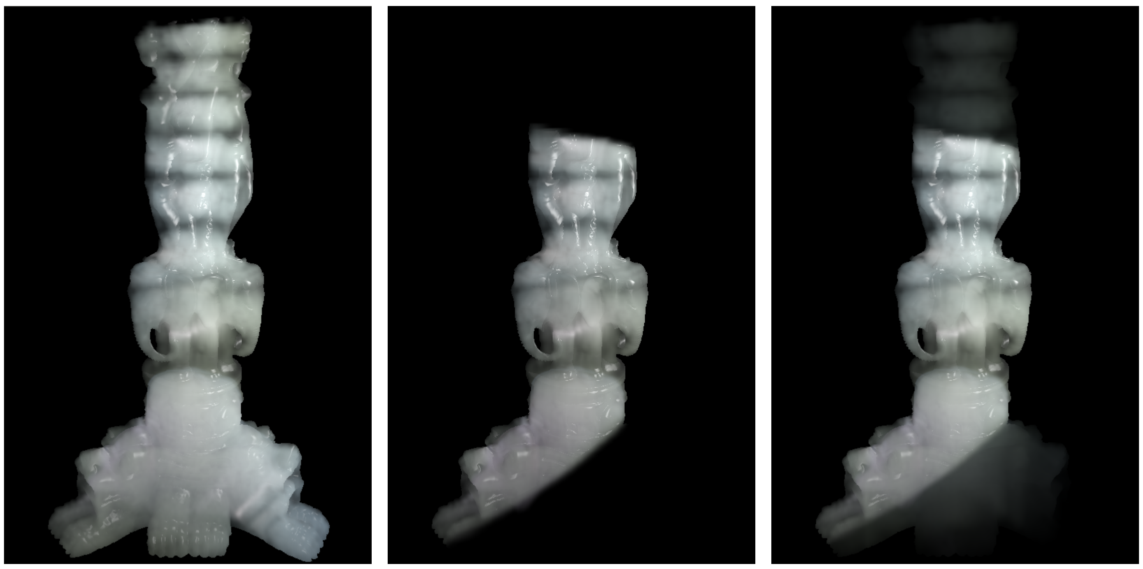


Figure 4.10: **Multiple and single scattering**. Left: The Thai model lit by a point light placed at its right; center: The same model lit by a spot light placed at the same position, **single** scattering only; right: Same as center, with **multiple** scattering re-enabled.

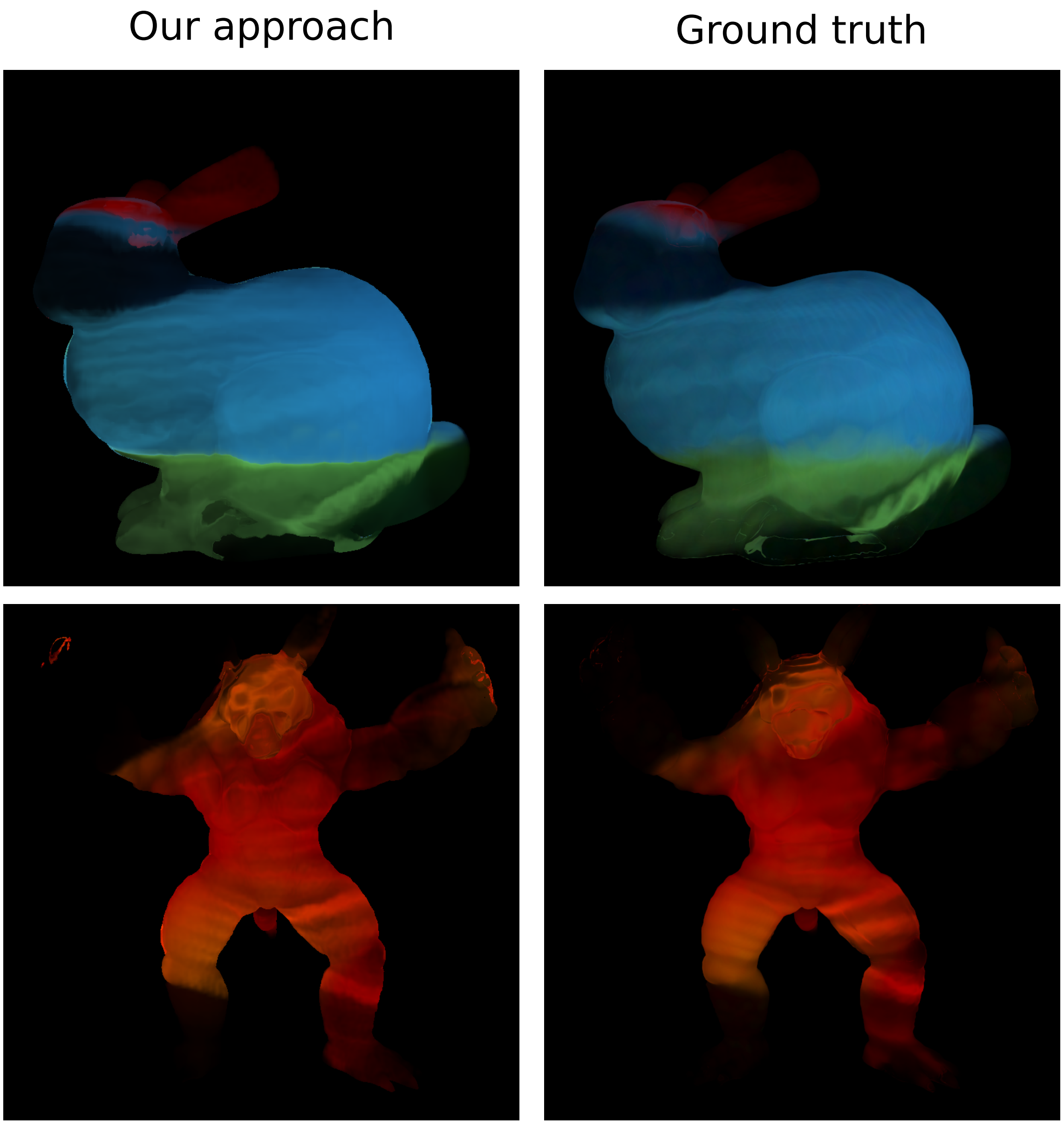


Figure 4.11: **Ground Truth comparisons.** On the left column, two renderings of the Bunny and Armadillo model at 12 fps made of heterogeneous materials. On the right column the scenes are rendered with photon mapping (228M and 63M of photons, respectively). The same spot light was placed on the right of the Bunny and on the left of the Armadillo.

Chapter 5

A Scalable Approach to the MDOM

Abstract

In this chapter we propose an algorithm which increases the directional resolution of our scattering solution with respect to the previously presented one, by building on a Modified Discrete Ordinates Method (MDOM) approach. Spatial resolution is also increased by implementing a Not Recently Used (NRU) paging algorithm which can support datasets many times larger than what can be currently fitted in device memory. The MDOM algorithm is then adapted to work efficiently on a paged volume, introducing innovations in light transmission with respect to current state of the art techniques. Our technique is parallel, scalable and can achieve interactive frame rates. Results demonstrate the ability of this method to render high quality multiple scattering effects in heterogeneous and anisotropic materials with physical accuracy.

The work we present in this chapter is aimed at overcoming the limitations that are inherent in using a scattering model based on a diffusion approximation. In order to model any kind of material, including materials with anisotropic phase functions, a more physically precise model of computation must be adopted. We chose to build on the ideas of the Discrete Ordinate family of methods (see chapter 2), because of their generality and accuracy.

Similarly to what we proposed in the previous chapter, the algorithm is divided in a first ray tracing stage (*direct* component computation) and a subsequent multiple scattering pass (*indirect* component computation), finally followed by a view tracing step. In the context of the DOM family, performing this subdivision places the method within the MDOM subfamily[84], introduced in chapter 2.

For the direct component, we apply a similar photon marching solution as the one proposed in the previous chapter, introducing a different transfer strategy by using a *sort-search-compact* algorithm.

For the indirect component, we propose a parallel wavefront propagation scheme for solving the DOM iterations. These wavefronts propagate light from voxel to voxel along discrete directions. Each scattering iteration constitutes eight such wavefronts propagating from each corner of the volume.

The newly proposed method has been designed as to be compatible with data of arbitrary size. However; contrarily on what is possible with visualization methods (see for example the technique introduced by Gobbetti et al.[51, 50] or the Gigavoxel system[26]), when dealing with scattering phenomena it is not possible to adaptively employ a lower resolution representation of the dataset for those parts of it that are far from the viewer. In fact, interactions that happen on the far side of an object have meaningful implications on the apparent surface color of the parts of the object that are close to the viewer. For this reason it was necessary to implement an efficient block-based streaming mechanism for transferring data between the CPU and the GPU, with the advantage of knowing and exploiting some of the access patterns of the algorithm on the dataset for accelerating page pre-loading.

Moreover, in order to reduce memory requirements for the data structure that we use during the indirect computation, we employ a lower angular resolution volume for storage and up-sample Irradiance data to a higher resolution during propagation between voxels. A comparable approach to this one is the one proposed by Fattal's[45]; however, we proceeded in short characteristic form instead of his long characteristic approach. This difference implies, as explained in Nikolaeva et al.[97], that we transfer indirect energy at every iteration from one face of a voxel to the face of a neighbor one, instead of tracing it up to the boundary of the volume.

In summary, this method allows for a physically accurate, high resolution computation of scattering in volumes at a fraction of the time of a comparable off-line render engine. Moreover, we add to the state of the art by introducing:

- An accurate and fast GPU implementation of the Discrete Ordinates Method capable of interactively displaying intermediate results.
- A streaming mechanism tailored at computing Single and Multiple Scattering effects.

5.1 Outline of the Algorithm

The algorithm is designed to work with volumes that have already been voxelized and it is particularly suited to handle voxelized representations of participating media like smoke and clouds; however, it can also take as input a watertight mesh together with material properties and, similarly to the previous method, perform a voxelization. Together with a set of light sources of any kind and a camera position for the viewer, the algorithm produces a realistic rendering of the volume.

The algorithm works by performing three sequential operations, i.e. computation of the direct component followed by computation of the indirect component and terminated with a view-tracing pass. As with the previous method, the result of the computation of light transmission within the volume are independent of view position, and as such a change in viewpoint leaving all other parameters unaltered can be performed efficiently at real-time speeds. We can conceptually divide the operation of our algorithm in separate modules. Figure 5.1 shows the interdependence of modules for the computation of the final solution. Note that the block scheduler is orthogonal to all operations and as such will be explained contextually to these.

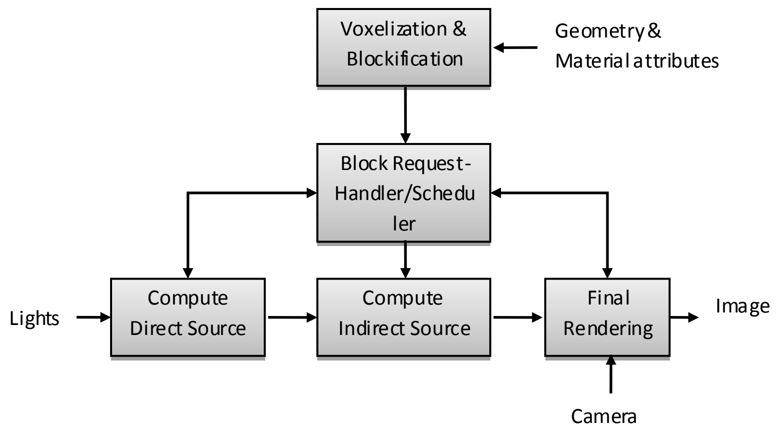

Voxelization & Blockification
Geometry & Material attributes
Block Request-Handler/Scheduler
Lights
Compute Direct Source
Compute Indirect Source
Final Rendering
Image
Camera

Figure 5.1: **Pipeline Overview.** Conceptual separation of our system into modules

5.1.1 Adapting the DOM

Recalling back the initial discretization of the RTE for $m = 1 \dots M$ angular bins as presented in chapter 2:

$$\mu^m \frac{\partial L(\boldsymbol{x},\boldsymbol{\omega}^m)}{\partial x} + \zeta^m \frac{\partial L(\boldsymbol{x},\boldsymbol{\omega}^m)}{\partial y} + \eta^m \frac{\partial L(\boldsymbol{x},\boldsymbol{\omega}^m)}{\partial z} = \underbrace{-\kappa_t(\boldsymbol{x})L(\boldsymbol{x},\boldsymbol{\omega}^m)}_{\text{extinction}} \underbrace{+\kappa_s(\boldsymbol{x}) \sum_{m'=1}^{M} p^{m,m'} L(\boldsymbol{x},\boldsymbol{\omega}^{m'})\, w^{m'}}_{G^m(\boldsymbol{x})}$$

We call $G^m(\boldsymbol{x})$ "source term" with a slightly different definition with respect to the customary use in literature (see chapter 1), and the equivalent term resulting from the integration of the above equation over the volume of a voxel (see chapter 2) will be denoted with $\bar{G}^{m,i,j,k}$. We will not use the superscripts i, j, k, except when needed for disambiguation, in order to keep the notation light.

As in the MDOM family, the source term is separated in a direct component, which is attenuated direct lighting as computed in the previous method, and an indirect component resulting from the multiple scattering of the direct component.

$$\bar{G}^m = \bar{G}^m_{dir} + \bar{G}^m_{dif}$$

Moreover, in order to disambiguate, we will indicate with a different letter, $S^{i,j,k}(\boldsymbol{\omega})$, the interpolation of the source components:

$$S^{i,j,k}(\boldsymbol{\omega}) = \sum_{\boldsymbol{\omega}^m \cdot \boldsymbol{\omega} > 0} (\boldsymbol{\omega} \cdot \boldsymbol{\omega}^m) \cdot \bar{G}^{m,i,j,k}$$

Memory consumption for the Discrete Ordinates Method is in the order of $m \times v^3$ where v^3 is the spatial resolution. In order to ameliorate these requirements we allow M to be a low number and use a different number of directions N (with $N \gg M$) during propagation using sweeps. The discretization of the interpolated source according to the propagation resolution is indicated as $S^{n,i,j,k}$.

5.2 Direct Component

In order to compute the direct component we march photons according to the technique presented in the previous chapter (4), with two notable differences. Since we are neglecting refraction effects in order to be able to use the DOM method, we can trace rays with the standard parametric ray formula instead of the eikonal equations. On the other hand, we are supporting a more precise directional resolution; therefore, we cannot store a single irradiance value for the voxel but need instead to store its discretized scattered directional distribution. Contribution to bin m is therefore:

$$\bar{G}^{m,i,j,k}_{dir} = (V^m)^{-1} \int_V \int_{\Xi(\omega^m)} \kappa_s(\bar{\boldsymbol{x}}) \int_{4\pi} L(\bar{\boldsymbol{x}},\boldsymbol{\omega}')p(\boldsymbol{\omega},\boldsymbol{\omega}')d\boldsymbol{\omega}' dx d\boldsymbol{\omega}$$

where $V^m = \Delta x \Delta y \Delta z \Omega^m = V\Omega^m$ is the "dimension" of the angular bin in the cross product space of 3D Cartesian dimension and solid angles and V is the volumetric space of the voxel i, j, k, centered in position $\bar{\boldsymbol{x}}$.

Calling $\Phi_{p,\boldsymbol{\omega}}$ the power of photon p coming from direction $\boldsymbol{\omega}$:

$$\bar{G}^m_{dir} = (V^m)^{-1}\kappa_s \int_{\Xi(\omega^m)} \sum_p \Phi_{p,\boldsymbol{\omega}'} p(\boldsymbol{\omega},\boldsymbol{\omega}')d\boldsymbol{\omega}$$

where the power of $\Phi_{p,\boldsymbol{\omega}}$ is computed as in the previous method (equation 4.3) and $\Xi(\boldsymbol{\omega}_m)$ indicates the solid angle spanned by bin m, centered around direction $\boldsymbol{\omega}_m$. We will refer to the increments on the source function induced by single photons as *source deltas*.

Finally, since in integration quadrature terms cancel each other out,

$$\bar{G}^{m}_{dir} = (V)^{-1}\kappa_s \sum_{p} \Phi_{p,\boldsymbol{\omega}'}\bar{p}(\boldsymbol{\omega},\boldsymbol{\omega}')$$

where we have introduced the normalized phase function $\bar{p}$ which ensures energy conservation when using a quadrature scheme.

Similarly to what presented before (chapter 4), we cannot directly store the contribution for efficiency reasons and instead store it in an intermediate buffer. However, due to the paged nature of the volume and the increased resolutions, the algorithm for merging the results of the buffer has been modified with respect to the previous chapter. These modifications are presented in the following subsections.

5.2.1 Tracing and Storing Light

As previously, rays are generated for every light source and their state is stored in a dedicated global raymap. A computational grid with the same size of the raymap is launched after ray initialization and each thread is responsible for marching a photon stream through the volume.

Instead of updating the source function volume directly, each ray writes the increments of the source function G along with the intersecting voxel_id to an intermediate buffer of fixed size termed here "source_delta_buffers". The operation proceeds as in the previous chapter, with a similar use of an intermediate local buffer which is moved to global memory at the end of the execution of the entire WorkGroup. However, in the present variant, there are M source_delta_buffers, each corresponding to a storage angular resolution of the DOM. Furthermore, voxel_id's are not stored along with the value in each buffer, but in a separate buffer (the index_buffer), the reason for which will be clear shortly. Every ray is allocated a fixed number of slots in each of these buffers. Hence, the size of these individual buffers equals the number of write-slots-per-ray $\times$ number of rays in the raymap.

To compute the sum of the writes (compaction) efficiently we sort the intermediate buffers by the voxel_id and then do a binary-search for the voxel_id followed by linear compaction operation. Figure 5.2 depicts the various steps involved in this sort-search-compact serialization of the source function deltas. Notice that in order to reduce global memory access, sorting is performed only on the index_buffer which stores, for every voxel_id, also the offset to the specific source increment in every source_delta_buffer. The resulting effect is that we perform the sort-search-gather pass in unison for all storage directions. We use a customization of the open source OpenCL implementation of the CUDPP [58] radixsort provided by the NVIDIA OpenCL SDK for sorting the index_buffer.

The reason for choosing this particular approach is twofold. The OpenGL blending implementation presented in the previous method is not capable of scaling well with the dimension of the Irradiance buffer. In particular, the cost of the blending solution is proportional to the number of elements that need to be written. In order to support a paging system like the one presented in the next section the entire blend operation would need to be repeated for each block in which the volume is subdivided. Therefore transforming an $O(p)$ solution to a $O(p \times b^3)$ one where p is the number of accessed voxels during the tracing stage, and b^3 is the total number of blocks.

On the other hand, accessing directly the buffer elements without a previous sort-compact operation, would require scanning, for each voxel, the entire index_buffer in order to find the values that it needs to add to its own source function, an extremely expensive operation.

We finally note that, for the hardware setting we used for testing, our experiments have shown that the optimal value of write-slots-per-ray is 32, which is higher than the value observed in the previous chapter. As before, the raymap dimensions used for all our experiments were 512×512.

5.2.2 Page Handling

In order for our scattering simulations to be properly scalable, we decided to implement a page handling mechanism. After an initial subdivision of the volume into blocks of uniform size, these

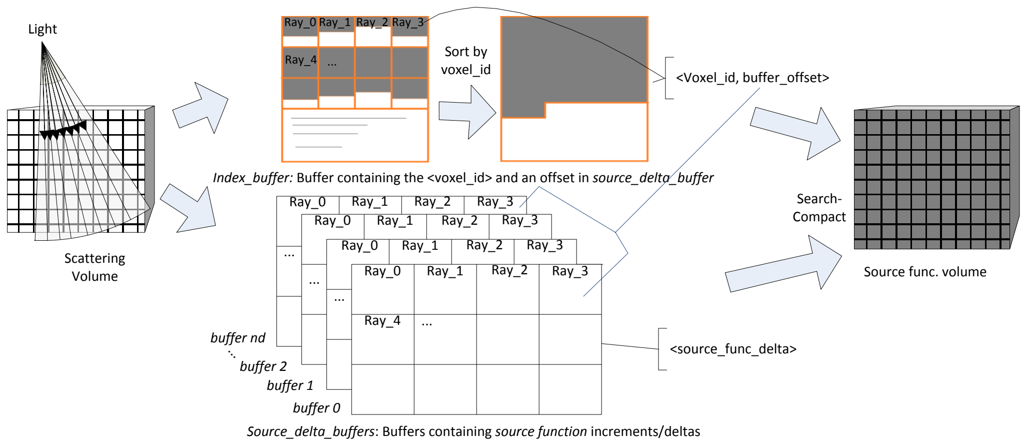

Light
Scattering Volume
Ray_0 Ray_1 Ray_2 Ray_3
Ray_4 ...
Sort by voxel_id
<Voxel_id, buffer_offset>
Index_buffer: Buffer containing the <voxel_id> and an offset in source_delta_buffer
Search-Compact
Source func. volume
buffer nd
buffer 2
buffer 1
buffer 0
<source_func_delta>
Source_delta_buffers: Buffers containing source function increments/deltas

Figure 5.2: **Sort and Compact** Photons marched through the volume store the angular distribution of the scattered irradiance in multiple buffers (one per discrete direction) which are then sorted with a parallel sorting algorithm according to the voxel_id they refer to. Afterwards a compaction algorithm sums all write requests to the same voxel as a single request, which is then performed directly by writing in the source function volume.

blocks are then stored in main/host memory: these are called “volume blocks”, storing the extinction coefficients. Space must also be allocated and cleared for the writing of the source function. We will refer to these blocks as the “source function blocks”.

At the time of this writing, handling 3D texture is still relatively inefficient as compared to 2D textures. For this reason, blocks are logically stored as rectangular areas of a 2D texture, named “block_pool”. The full memory required for the algorithm is allocated in advance, since we have the advantage of knowing that the algorithm will eventually use the entire volume. We allocate as much memory as it is possible, in order to minimize the expensive streaming operations, but this can be modified for real-world usages of the algorithm. In order to keep track of blocks of the two types it is necessary to use three tables: **block_request_table**, **block_dirty_table** and **block_index_table**. The first two are bitmaps, used respectively to flag a block as needed and as written. The block_dirty_table is not used for the volume blocks since only source function blocks can be written to in this architecture. The block index table, which is a 3D texture of b^3 elements (where b^3 is the number of blocks) is used to store the location of the block within the block pool. This same table indicates non present blocks by setting the table to an invalid value.

However, there is no way to predict the loading pattern of the volume/source blocks as requested by rays; therefore, it is necessary to implement an on-demand mechanism that integrates streaming into this phase. When a GPU thread encounters a missing block, it sets the corresponding dirty bit and terminates its computation. However, if the thread has used up all of its slots in the buffer, it still has to mark explicitly that it will need the same block when computation resumes. This allows us to implement a simple yet effective Not Recently Used (NRU) policy[25] when deciding which pages to remove.

At the end of every stage of the algorithm, blocks which have been set as requested need to be loaded. The order of preference, when having to choose where to put a block is:

- free space
- a non-dirty non-requested occupied block
- a dirty non-requested occupied block
- a dirty requested occupied block

This means that it is possible that a block will need to be swapped out even if it is requested by some of the rays, that is, there can be a number of request that exceeds the amount of available space. This strategy however does not induce starvation, since rays will eventually leave a block in which they are working.

Notice also that the search-compact pass includes the block tables for the source function blocks. It is obvious that the volume blocks visited by the light rays during the ray marching pass correspond to the source function blocks to be written during the compaction pass, hence we use the block_index_table of the volume blocks for pre-loading the source function blocks in its block_memory_pool for the compaction pass. This strategy of tracking the volume blocks streamed during the last ray marching pass and using this information to pre-load source function blocks, facilitates a single pass compaction.

Some different strategies could also have been implemented; for example, we could have ordered swap-out preference according to the number of individual thread requests. However, this would have required either a semaphore or a counting operation, i.e. a scan, which is still an expensive operation. Moreover, this does not guarantee efficiency, because it could lead to the starvation of a single ray, lagging behind when all others move forward because it is the only one requesting a specific block and eventually delaying the application.

Another conceivable strategy would be to sort rays and group them together according to requests. This is, for example, the approach taken in ray tracing through scenes when rays are packed together in order to speed up traversal of the data structure for the scene (Bounding Volume Hierarchies or Octrees). However, it is reasonable to believe that the overhead induced by the sorting operation counters any benefit that can be had from a speedup due to coherent memory access.

Finally, note that the blocks where it will be necessary to write source function values are exactly the ones that have been loaded for traversal. Therefore, during the OpenCL sorting of the buffer values it is possible to pre-load source function blocks using the request bits of the previous pass as indications of which block to load.

5.3 Parallel Wavefront Propagation

After rays have completed their travel through the volume and source deltas have been transfered to the source volume, the wavefront DOM solver starts. The solution of the DOM equation for the face to face propagation is based on the assumption that both the extinction coefficient and the source function are constant within a voxel. Moreover, in-scattering due to energy deposited at the current iteration is ignored, since it will be propagated at the next iteration. Under these assumptions, the integral equation for light transfer presented in chapter 1 reduces to

$$L(\boldsymbol{x},\boldsymbol{\omega}) = e^{-\kappa_t||\boldsymbol{x}-\boldsymbol{x}_0||}L(\boldsymbol{x}_0,\boldsymbol{\omega}) + \int_{\boldsymbol{x}_0}^{\boldsymbol{x}} e^{-\kappa_t||\boldsymbol{y}-\boldsymbol{x}||}S(\boldsymbol{\omega})d\boldsymbol{y}$$

which can be analytically solved as

$$L(\boldsymbol{x},\boldsymbol{\omega}) = e^{-\kappa_t||\boldsymbol{x}-\boldsymbol{x}_0||}L(\boldsymbol{x}_0,\boldsymbol{\omega}) + \frac{S(\boldsymbol{\omega})}{\kappa_t}\left(1 - e^{-\kappa_t||\boldsymbol{x}-\boldsymbol{x}_0||}\right)$$

This equation can be used to determine the amount of radiance leaving a face along a specific direction as determined by the incoming radiance at a different face; and the total scattered radiance within the voxel.

5.3.1 Radiance of the outgoing faces

We define the average radiance on a face Out as

$$\bar{L}^n_{Out} = \frac{1}{A_{Out}}\int_{Out} L^n_i d\boldsymbol{i}$$

where we use $\boldsymbol{i}$ to index points belonging to the face.

For a propagation direction n and a point on the surface of the outgoing face, there is exactly one corresponding point in one of the other faces, therefore we can turn the integral on the face into an integral on the projected faces

$$\bar{L}^n_{Out} = \frac{1}{A_{Out}} \sum_{In} \int_{A_{In,Out}} L^n_i di$$

where $A_{In,Out}$ is the area of projection of face In over face Out (see figure 5.3)

Using the equation of the integral above, and assuming the radiance on each face to be constant, then:

$$\bar{L}^n_{Out} = \frac{S(\boldsymbol{\omega}^n)}{\kappa_t} + \sum_{In} \frac{A_{In,Out}}{A_{Out}} \left(\bar{L}^m_{In} - \frac{S(\boldsymbol{\omega}^n)}{\kappa_t} \right) \frac{1}{A_{In,Out}} \int_{A_{In,Out}} e^{-\kappa_t ||\boldsymbol{i}-\boldsymbol{j}||} di$$

Languenou[84] asserts that according to experiments, if κ_t is small, then $\frac{1}{A_{In,Out}} \int_{A_{In,Out}} e^{-\kappa_t ||\boldsymbol{i}-\boldsymbol{j}||} di \approx e^{-\kappa_t s_{avg(In,Out)}}$ where $s_{avg(In,Out)} = \frac{1}{A_{In,Out}} \int_{A_{In,Out}} ||\boldsymbol{i} - \boldsymbol{j}|| di$, which can be easily precomputed. Therefore,

$$\bar{L}^n_{Out} = \frac{S(\boldsymbol{\omega}^n)}{\kappa_t} + \sum_{In} \frac{A_{In,Out}}{A_{Out}} \left(\bar{L}^n_{In} - \frac{S(\boldsymbol{\omega}^n)}{\kappa_t} \right) e^{-\kappa_t s_{avg(In,Out)}}$$

and this is the quantity that gets propagated to an adjacent voxel during wavefront sweeps.

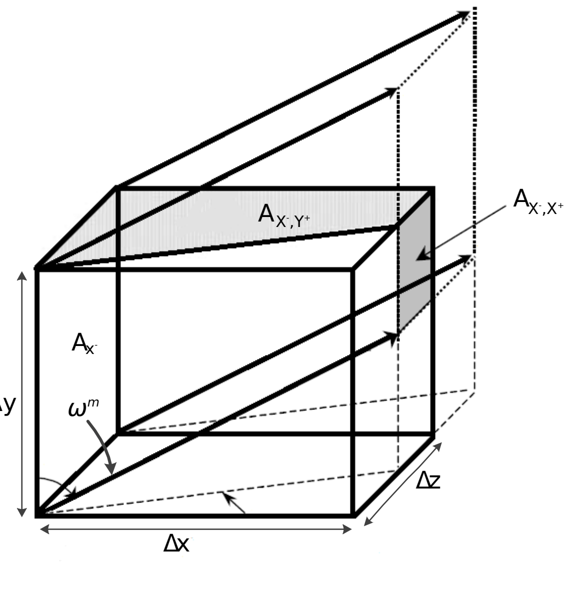


Figure 5.3: **The $A_{In,Out}$ factors.** Projection of face A_{X^-} on A_{X^+} is indicated in gray (A_{X^+,X^-})

5.3.2 Average radiance of the voxel

The average radiance is the radiance that will be deposited in the voxel after the sweep has passed and that will be scattered back at the next iteration. Similarly to the previous section we define it as:

$$\bar{L}^n_V = \frac{1}{V} \int_{\boldsymbol{x} \in V} L(\boldsymbol{x}, \boldsymbol{\omega}^n) dV$$

where $L(\boldsymbol{x}, \boldsymbol{\omega}^n)$ is the amount of radiance reaching a point $\boldsymbol{x}$ within the volume of the voxel from one of the boundary faces. That is,

$$L(\boldsymbol{x}, \boldsymbol{\omega}^n) = \bar{L}^n_{In} e^{-\kappa_t ||\boldsymbol{x}-\boldsymbol{i}||} + \frac{S(\boldsymbol{\omega}^n)}{\kappa_t} (1 - e^{-\kappa_t ||\boldsymbol{x}-\boldsymbol{i}||})$$

The entire volume can be divided into non overlapping zones determined by an incoming face and an outgoing face (see figure 5.4)

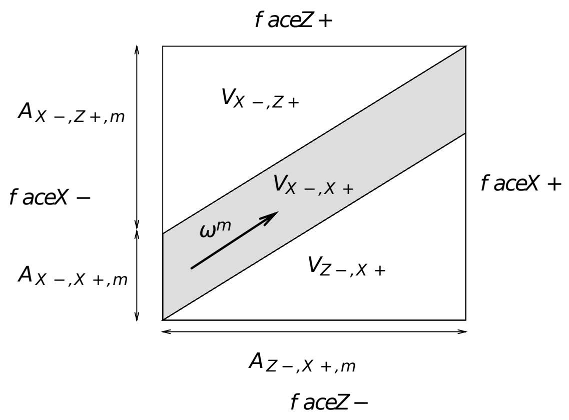


Figure 5.4: **Non-overlapping zones**. Bidimensional representation of the non-overlapping zones between faces induced by a direction $\boldsymbol{\omega}^m$

$$\bar{L}_V^n = \frac{S(\boldsymbol{\omega}^n)}{\kappa_t} + \frac{1}{V}\sum_{In}\sum_{Out}\left(\left(\bar{L}_{In}^n - \frac{S(\boldsymbol{\omega}^n)}{\kappa_t}\right)\frac{V_{In,Out}}{V} \times \frac{1}{V_{In,Out}}\int_{V_{In,Out}} e^{-\kappa_t||\boldsymbol{x}-\boldsymbol{i}||}dV\right)$$

Where we approximate $\frac{1}{V_{In,Out}}\int_{V_{In,Out}} e^{-\kappa_t||\boldsymbol{x}-\boldsymbol{i}||}dV$ with $e^{-\kappa_t\frac{s_{avg(In,Out)}}{2}}$ and is precomputed in advance for all faces and all directions.

Finally this will be added to the source term as:

$$\bar{G}_{dif}^m = \frac{\kappa_s}{4\pi}\sum_{n=1}^{N} w_m \bar{p}_{m,n}\bar{L}_V^n$$

Note that even if propagation is carried along the N directions, re-projecting back to the lower M directions is straightforward since we chose an angular quadrature scheme that always incorporates its subset levels.

5.3.3 Plane Sweeping

Since every voxel solves independently the equation relating radiance at the incoming faces to radiance at the outgoing ones, it could be reasonable to construct a parallel algorithm like the one presented in the previous chapter, where each thread attached to each volume element runs concurrently with every other. However, we can also notice that one of the issues with the previous approach resides in the time that it takes for energy deposited in a voxel to reach the opposite side of the mesh. The sweep3d method[81] reduces this issue by parallelizing within wavefronts but sequentialzing their motion through the volume. It is based on the assumption that for any possible propagation direction, say $\boldsymbol{\omega}^n$, there can be only three faces in a cube for which this direction represents an incoming one, casting the remaining three as outgoing faces. Therefore there exists a diagonal dependency which can be exploited to avoid synchronization issues. Conversely, it is possible to group directions into octants according to the face dependency they induce.

Consequently, we propagate all directions in an octant simultaneously, inducing a diagonal propagation sweep starting from each corner of the volume. This sweep when propagated in parallel, diagonally, is akin to propagating a wave starting from a corner. These wavefronts pick out-scattered light at voxels and propagate it until they reach the end of the volume. Eight such wavefronts starting from each corner of the volume constitute one scattering iteration. Figure 5.5.(a) illustrates a two dimensional version of this process. For the 3D version, this is similar to a 3D hyper-plane propagating diagonally from a grid corner to its opposite corner. A voxel indexed by i, j, k belongs to this 3D hyperplane if it satisfies the equation.

$$w = i + j + k$$

With $0 \leq w \leq 3v - 3$ the wavefront step number, and v the number of voxels per side in the total volume.

Notice again that waves do not propagate in-scattered radiance from the wave itself, but only radiance that has been deposited during the previous cycle of eight sweeps. For this reason, apart from maintaining the source function values $\bar{G}^m$ for each direction at every voxel (which constitutes the final solution), we also need to maintain source function contributions from the previous iteration ($\bar{G''}^m$) and the current iteration ($\bar{G'}^m$). After the photon marching phase, $\bar{G''}^m$ is initialized with $\bar{G}^m_{dir}$ and $\bar{G'}^m$ is initialized to zero. Note that this strategy is akin to the one employed by the Progressive Radiosity algorithm.

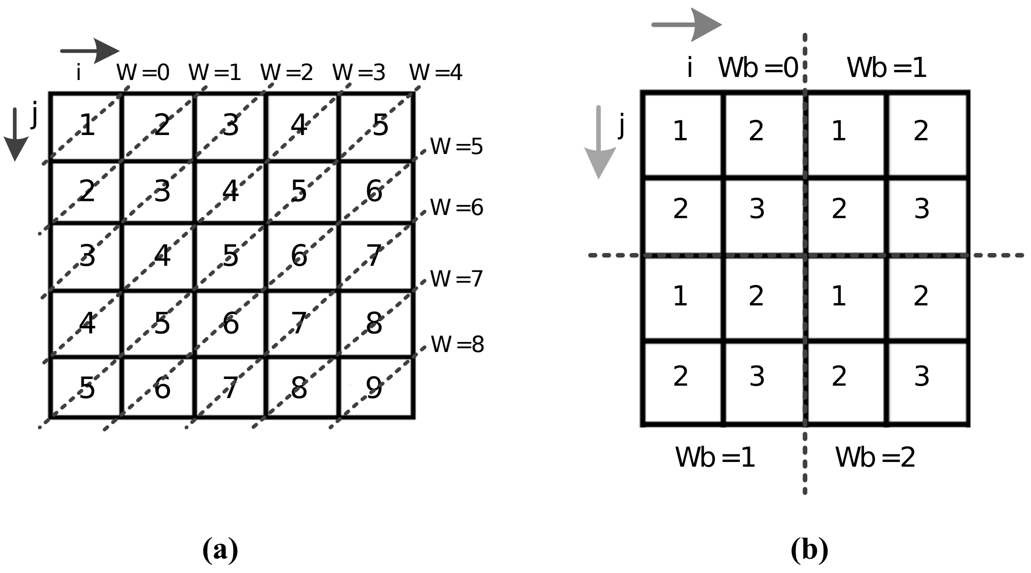


Figure 5.5: **Wavefront propagation.** On the left: order of computation of different waves (a); on the right: order of loading of blocks and internal order of computation. (b)

5.3.4 Streaming Blocks

We avoid any necessity for wavefront collision handling by executing only one sweep at a time. This also enables us to pre-compute the loading sequence for the Volume-blocks and Source-function-blocks along the direction of sweep. This further implies that instead of taking the on-demand streaming approach of the direct pass, we can simply schedule the propagation of block-wavefronts based on the pre-computed sequence. This scheduling of the wavefronts is handled by the block scheduler. The blocks in the current wavefront step are independent of each other and can be loaded or operated upon in any order. This innate flexibility implies that, if there is not enough GPU memory available, we can do the propagation only for the blocks which will fit in the GPU memory. After completing this task, we load the next set of blocks and continue the process till the wavefront propagation is finished.

The propagation of a wavefront of blocks is further decomposed into a wavefront of voxels which runs independently in each block. This decomposition is performed on the GPU, where a WorkGroup of GPU threads operates within the confines of a single block. Figure 5.5.(b) illustrates a 2D version of this approach. However, block size is larger than the available memory for a local WorkGroup. In all of our experiments on our machine, a WorkGroup size eight times smaller than block size proved to be optimal. The reason why this is so, is because keeping track of blocks adds an overhead which is inversely proportional to block size. Therefore, using blocks of the same size of WG memory would not be more efficient.

We can finally notice that the incoming radiances for the boundary voxels of a block is the outgoing radiances from the ones belonging to the blocks of the previous step. To store these outgoing radiances we maintain a 2D interface of voxels, similarly to the interface buffers used in the previous chapter. These interfaces store outgoing radiance for each outgoing face along the directions in the current octant. Hence, each interface voxel stores $\frac{N}{8}$ directions. This memory is only accessed by voxels on the boundary of the blocks and we use local memory instead for synchronizing and propagating information within a WorkGroup.

5.3.5 View Gathering

The final pass, view gathering, is essentially unchanged with respect to the previous method, with the sole exception of having an angular distribution of radiance instead of an average isotropic one. Blocks must be streamed again for this operation and no data on their previous usage can be employed in this stage. However, in the view pass it could be possible to employ lower resolution blocks through a mipmapping strategy similar to the one by Gobbetti et al.[51]. However, it is to be noted that the view gathering stage of the computation is already the most efficient one with respect to the other operations.

For the tracing of rays, a small jittering is used as in the previous method in order to avoid introducing artifacts. The equation used for view gathering is the integral formulation of the radiative transfer equation with interpolated source function as in chapter 4.

5.4 Results

We implemented our pipeline using C++ and OpenCL. The results presented in this section are from our implementation running on a 3.0 GHz Intel Core 2 Duo machine with 8 GB RAM and a GeForce GTX 580 GPU with 3 GB's of physical memory.

In Figure 5.7, an isotropic botijo model is rendered with its extinction coefficients perturbed. The spatial grid resolution is 256^3 and the propagation angular resolution is 24 directions. We illustrate the difference in the multiple and the single scattering results. This difference is more pronounced in areas with shorter optical depth due to a lesser exponential fall-off. The difference is most noticeable in the handles of the botijo. In Figure 5.8, we depict images of isotropic smoke simulations rendered from our pipeline. The two images on the left show results for a 64^3 grid with 24 propagation directions. The smoke grid is lit by a bright parallel projector light placed on top of the volume. The two images on the right show results for an environment lighting of a 128^3 smoke grid using 120 propagation directions. For environment lighting, we skip the direct component stage as we have a significant number of light rays coming from all directions. As an alternative, we use an environment irradiance map for incoming boundary radiance during the indirect component stage. This is similar to solving the standard DOM equation with detached angular resolutions. Since the environment light is discretized at the volume boundary, we use a higher propagation angular resolution, 120 directions. The smoke simulations were rendered with two multiple scattering iterations at 1.2 fps and 0.1 fps respectively.

5.4.1 Performance

5.4.1.1 Memory requirements

The Volume-blocks require n^3 storage and each of the Source-function-blocks ($\bar{G}^m$, $\bar{G'}^m$ and $\bar{G''}^m$) require mn^3 storage. For a 256^3 grid with each voxel containing 3 channel extinction coefficients stored as float4 values, the Volume-blocks storage amounts to 256 MB. For isotropic media, the

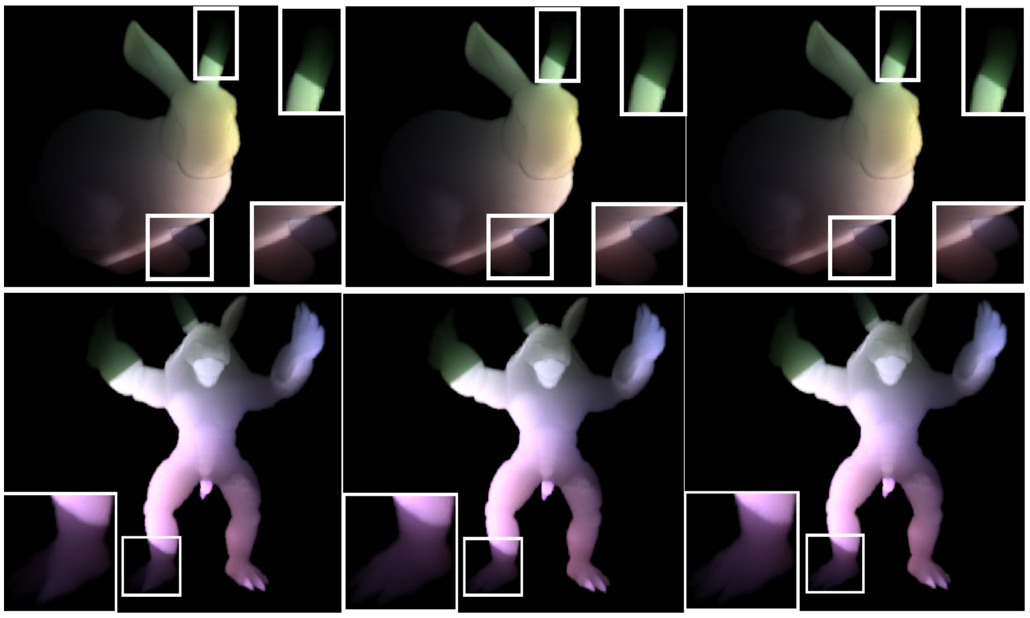


Figure 5.6: **Differences in light scattering when changing propagation directions.** Top row, left to right: Armadillo rendered with 24, 48 and 120 propagation directions. Bottom row, left to righ: Bunny rendered with 24, 48 and 120 propagation directions. Difference are notable between the 24 and 48 versions but not much between the 48 and th 120 one.

Source-function-blocks together require a 256×3 MB storage, making the total volumetric storage requirement equal to 1 GB. For a 512^3 grid this total increases to 8 GB. These allocations happen on the CPU memory. The corresponding GPU memory allocations are determined by the sizes of their respective block_pools. The block_pool sizes are configured in terms of number of blocks they can accommodate and are the same for all the block_pools. For our hardware setting, our experiments have shown that the optimal value for block_size is 32^3.

Grid size (n^3)	N	Seconds/frame
128^3	24	5.3
	48	9.86
256^3	24	41.55
	48	68.39
512^3	24	579
	48	780.39

Table 5.1: **Timings for the bunny scene (5.6).** Different grid sizes with 24 and 48 propagation directions. The timings were recorded for a combined block_pool size of 2 GB. Even if several seconds per frame are necessary in this setup to produce an image matching the ground truth, the timings are considerably faster than comparable images produced by an offline path tracer (see section 5.4.2)

5.4.1.2 Timings

Table 5.1 shows the timings for an isotropic bunny scene (figure 5.6) with different grid sizes and propagation angular resolutions, after three scattering iterations. The timings were recorded for a combined block_pool size of 2 GB. Since we use OpenCL for our implementation, we can also compare timings for GPU and CPU executions of our pipeline. Graph 5.10 plots the speedup

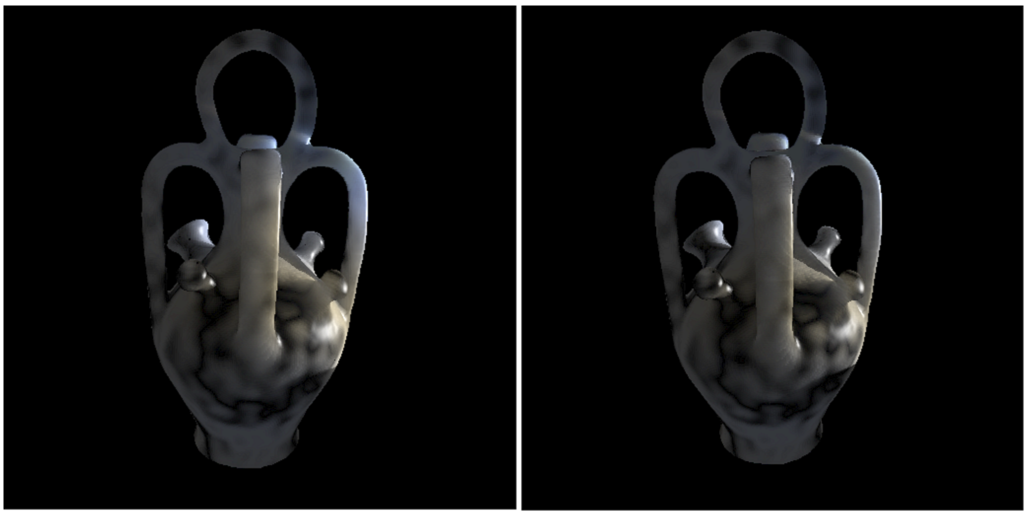


Figure 5.7: **Multiple vs Single scattering.** MS on the left, SS on the right. The botijo was rendered with a marble material and surface lighting enabled.

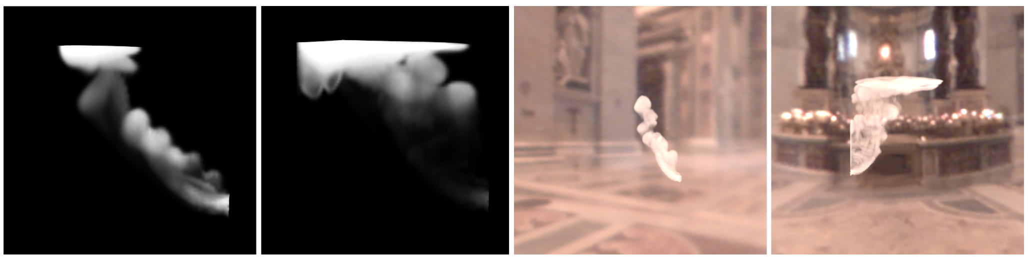


Figure 5.8: **Rendering smoke under different lighting conditions.** Left: with a parallel projector placed above the volume . Right: with environment mapping.

achieved by the GPU executions over the CPU executions of our pipeline for different grid sizes, and combined block_pool sizes. The combined block_pool size is analogous to the size of the total streaming cache. The CPU used for these experiments had an effective 12 execution cores (6 core Intel CPU with hyper-threading enabled) which execute the OpenCL kernels in parallel. As the graph 5.10 shows, when the combined block_pool size is equal to the total memory requirement of a particular grid size, the overhead of our streaming approach over a non-streaming solution is negligible. It should be mentioned here that an 8X speedup is achieved for a 512^3 grid with a combined block_pool size of just 256 MB, which is less than 4% of the total volumetric memory requirement. These results further corroborate the need for a GPU based parallel technique for solving the DOM equations.

5.4.2 Ground Truth Comparison

We performed all a Ground Truth comparisons against the Mitsuba volumetric path tracer[1] which solves the full RTE, and with the Lattice-Boltzmann solution of the previous chapter. We choose a heterogeneous isotropic medium for this comparison; depicted in Figure 5.11. The ground truth result was rendered with path lengths of 12 and 64K samples per pixel and it took several hours to compute. The corresponding result produced from our method has spatial grid resolution of 256^3 and propagation angular resolution of 48 directions. Our new method took 68 seconds to converge. For the same grid resolution, the diffusion approximation solution took few milliseconds

[1]The Mitsuba rendering engine http://www.mitsuba-renderer.org/

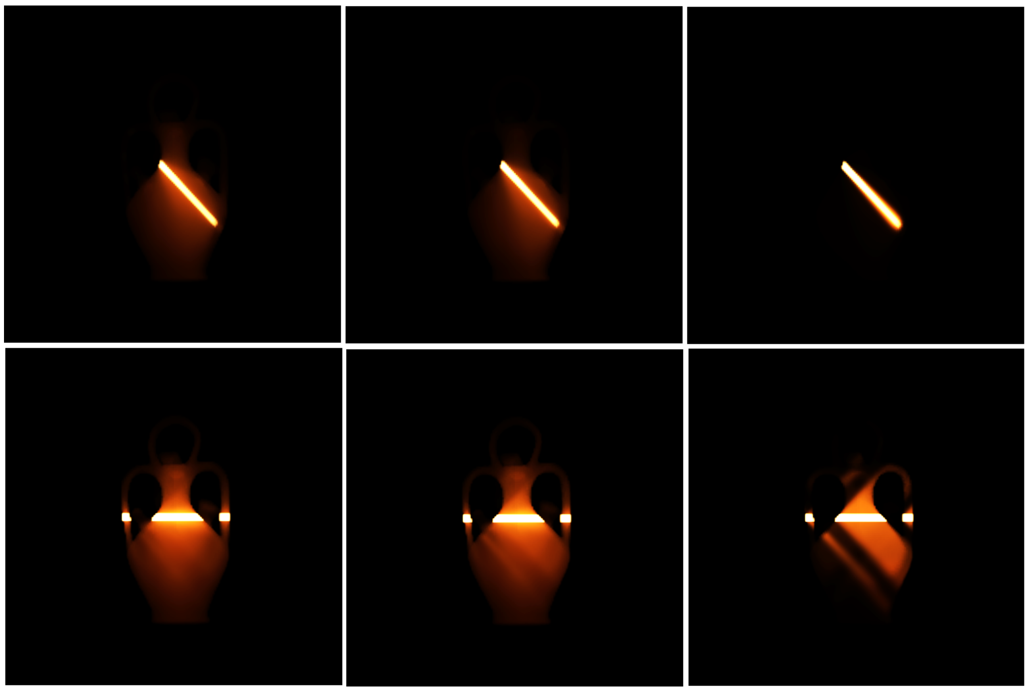


Figure 5.9: **Multiple scattering results.** From left to right: with g = 0.4, 0.6 and 0.9. When light direction is aligned with one of the storage directions (Top) and (Bottom) not aligned with any of the storage directions.

per frame to render the scene. In the center image, there is some numerical smearing noticeable on the right hand and right foot of the armadillo. On the rightmost image, it is possible to notice how, since the LBL scattering is mostly localized, the result differs from the ground truth in thin areas which have not been reached by the time the user specified iterations come to an end. Though the method presented in this chapter is slower than the LBL method, it is much more general and can more effectively handle multiple scattering in heterogeneous media.

5.4.3 Discussion

We begin the discussion by examining the effect of the storage angular resolution on the quality of results. In case of isotropic media, as the phase function is constant in all directions, the storage resolution does not impact the quality of the results, thus agreeing with the method of chapter 4. However, the capability of our system to handle anisotropic media is determined by the storage angular resolution. At an 8 direction storage our system can effectively handle Henyey Greenstein phase functions with values of mean cosine g up to 0.4. For values of g greater than 0.4 the results exhibit ray effects as shown in Figure 5.9. The image shows the renderings of an anisotropic botijo with homogeneous scattering coefficients and increasing values of g, which is lit by a beam of light. The spatial grid resolution is 128^3, with storage angular resolution of 8 and propagation angular resolution of 48. If the direction of the light beam aligns with one the storage directions (top images in Figure 5.9), then as expected, the medium exhibits higher forward scattering effects as g increases. But when the direction of light does not align with any of the storage directions (bottom images in Figure 5.9), then the ray effect increases as g takes values beyond 0.4. Although this can be mitigated by increasing the storage angular resolution, it is a matter of quality vs. performance trade-off.

The propagation direction resolution can also be considered as a quality vs performance trade-off parameter. In case of a convex geometry such as the bunny shown in Figure 5.6 (Top images), the differences in the 24 and 48 propagation direction results are very subtle. This indicates that we can

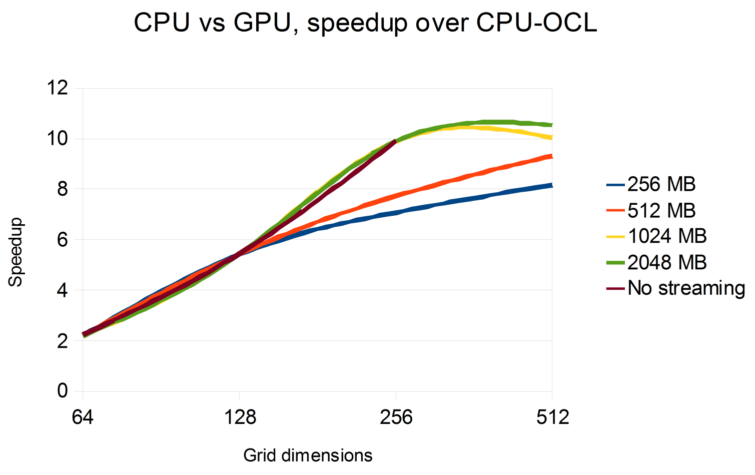


Figure 5.10: **GPU vs CPU.** Speedup of the GPU executions over the CPU executions for increasing grid sizes (x axis) and combined block_pool allocations (lines). Also, plotted are speedups for non-streaming cases (this excludes 512^3 grids, because there is not enough GPU memory available.)

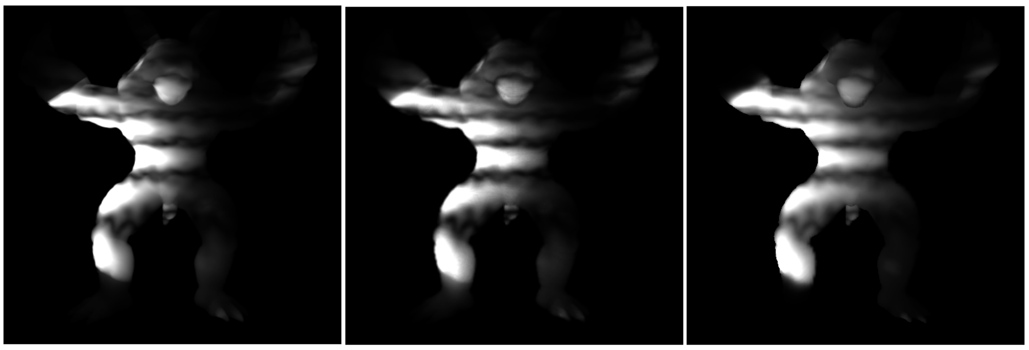


Figure 5.11: **Comparisons**. From left to right: results rendered by a ground truth Monte Carlo based raytracer, our method and our diffusion approximation based method.

afford a low resolution propagation in this case and thus a better performance without sacrificing too much on quality. While in case of the armadillo shown in Figure 5.6 (Bottom images), the result for 24 directions differ significantly from the 48 or 120 direction results. However, in both cases there are negligible differences between the results of 48 and 120 propagation directions.

Regarding our data streaming process, although it virtually eliminates the GPU memory limitation, overall scalability is still limited by the memory available on the CPU. For example, a 1024^3 volume would require around 64 GB of memory allocations on the CPU. This could be ameliorated by adding another layer of streaming, from a large scale storage device to the CPU.

In conclusion, we have presented a method which is capable of providing accurate results comparable to those produced with an offline photon tracer but at a fraction of the time. Although the method does not match the real-time speed of our LBL proposal (chapter 4), for some simple cases it is nonetheless capable of achieving a limited form of interactivity while still providing physically accurate single and multiple scattering effects on heterveogeneous and anisotropic volumes. The data streaming mechanism further ensures that datasets much bigger than GPU memory can still be effectively visualized, which was not possible in the previously presented method.

Chapter 6

Collateral Work

Abstract

In this chapter we examine a collateral work not related to the rendering of volumetric effects. In particular, we present a method designed for assisting visually impaired people in navigating indoors environments through a partial on-the-fly reconstruction of the environment obtained with a Kinect©device. The system compares favorably with traditional navigational aids and is composed of cheap off-the-shelf components.

6.1 A navigation system for visually impaired people

6.1.1 State of the Art in Computer-Assisted User Navigation

Exploiting state-of-the-art technology for improving people's everyday life is always a compelling challenge, especially when the people in question are impaired in some way. Over the recent years, the rapid evolution of color acquisition devices and computing hardware and their affordability has spawned a number of solutions to assist blind people in indoor and outdoor mobility. Most of the edge-breaking technologies have been applied to assist or train them: RFID [44][28][22], haptic devices [67], ultrasonic systems [12], virtual reality [124][63]. One of the latest steps of this evolution is the availability of low cost 3D scanners which are able to supply a three dimensional description of the scene in front of the device. It is straightforward to connect this novelty to the design of yet another system that would analyze the context and allow a blind person to move freely. This has some points in common with the robot guidance immense literature (see [16] for an application for impaired people), but the kind of feedback needed by the user is different. More specifically, given the 3D description of the scene, the algorithm identifies obstacles in the path, but there's no real way to constrain the movement of the user. Hence, the system should assist her/him during the exploration. Several ad-hoc sensors have been developed in the past [125][104][114], even able to recognize known features of inhabited environments, such as stairs or sidewalks. However, when considering 3D scanning technologies, we must confront ourselves with the fact that, while a color camera provides information that the blind person has no way to know, i.e. the colors, a 3D scanner-based system has two strong competitors against with it should be compared: the *white cane* and the *guide dog*. The white cane allows a person to explore the terrain in front of his/her foot in a range of approximately one meter (it depends on the length of the cane and the stature of the person), it is done with light weight materials, it is foldable and easily stored, it does not need any non-human power source and therefore is extremely fault-tolerant and finally it is very reliable, since the human brain does an excellent job in elaborating the haptic information returned by the cane. For this reason, some of the proposed solutions are integrated in it [22][114]. A trained guide dog does not provide a punctual information like the cane, but in this case it is not necessary since it does the brain work and leads the way. Obviously owning a dog is much more committing that owning a white cane and it may not even be possible for some people. In this section we present an early implementation of an assistive navigation system for blind people based on a 3D scanner. We use a recent device, called Kinect©, released in the context of videogames and entertainment. The Kinect is essentially a low-cost short-range 3D scanner that is able to acquire a 640×480 range map of the scene at the pace of 20 to 30 times per second. Since the Kinect works in a range between 0.5 to 4 meters, it is perfect for the guidance of the movement of a person, and the quality data it provides is sufficient for the task of analyzing and detecting obstacles. The concept of our system is quite the same as [125] and similar works but there are substantial differences. Firstly, the prototype of our system is built with off-the-shelf technologies and it costs less than 500 Euros while the solutions proposed in the past were essentially costly prototypes. We consider this aspect of primary importance since, in our opinion, the market for blind people is not big enough to scale down the cost of ad-hoc technologies [104]. Secondly, we aim at exploiting the availability of solutions in the field of computational geometry to analyze 3D data and provide accurate understanding of the scene in front of the user. As we will see later on, these solutions are not directly applicable as they are, since almost all of them are not concerned with time-critical interruptible computation but only with a general concept of efficiency. We designed a simple framework, shared in a open source repository, for time-critical computation that is able to incorporate new algorithms in the system in a collaborative fashion.

6.1.2 System Description

Figure 6.1 illustrates the setup of the system. We fix the Kinect sensor to the belt so it can have an ideal coverage of both floor and object at human height. The Kinect is connected to a $12V - 3mAh$ battery pack and to a computing device such as a smartphone, which provides audio feedback to the user. The use of audio as interface is a temporary solution and it will be replaced by an haptic device currently being designed. Although the design of most systems for visually impaired people include audio as primary interface to provide feedback to the user, there is a fairly strong argument against this solution: we are essentially depriving the person of the use of hearing, which for a blind person is more than one sense.

6.1.2.1 A Time-Critical Framework

The feedback from data analysis must be responsive. Unlike with a robot, we cannot make a person *freeze* while waiting for system response, therefore the computation must be not only as fast as possible but also *interruptible*, i.e. the control must return in the time assigned with a meaningful, although conservative, answer. This constraint gives rise to a number of new problems in the field of computational geometry. At the present time there are a large number of techniques to analyze 3D data in order to discover regularities, symmetries, known shapes etcetera. While the corresponding algorithms are designed to be fast, very few of them are also interruptible. In order to make a practical example, none of the several algorithms for surface curvature computation, which is often used as a building block for shape recognition algorithms, may be run with a time constraint or interrupted in the middle of the computation.

Our control software is organized as a series of interruptible tasks that run concurrently (the ellipses in Figure 6.2) and a series of data (the rectangles). Each task has its input data and produces its output data, and it is assigned with a time lapse by which it must complete its computation. In the current implementation we use the following tasks:

- **ReadDevice**: reads the data from the device and does the necessary conversion to express it as 3D points;
- **Registration**: detects the floor and registers the data in a reference system centered at the user's feet;
- **OccupancyDetector**: detects the occupancy of the volume in front of the user;

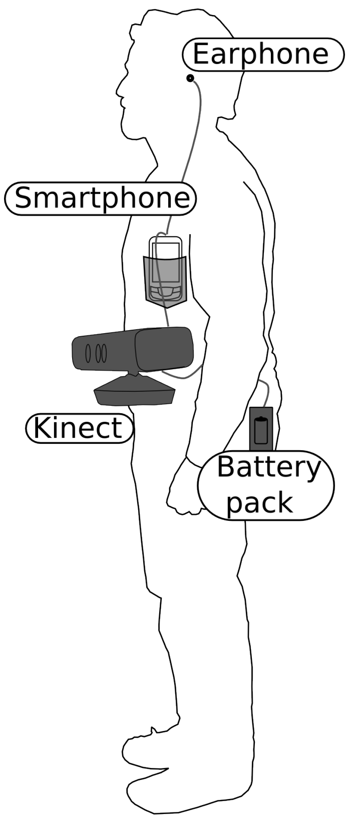


Figure 6.1: **Illustration of the proposed system**.

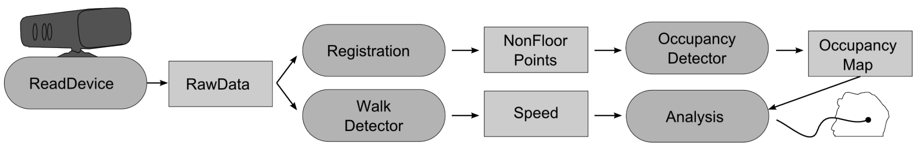


Figure 6.2: **A scheme of our time-critical framework**.

- **WalkDetector**: analyzes the output of the accelerometer to establish if the user is walking and at what speed;
- **Analysis**: provides the feedback to the users on the base of the result of the other tasks.

In the following we provide more details on how the single tasks are carried out with time constraint.

6.1.2.2 Registration

This task consists of applying a geometric transformation to bring the 3D points from the reference system of the Kinect, indicated with K_{rf} in Figure 6.3, to the reference system centered at the user's feet with the z axis passing trough the barycenter and the head of the user, indicated with H_{rf}. Since the Kinect is mounted on the waist, it goes under ample movements when the user walks, which means that the transformation between these two reference systems, a 4×4 matrix called T from now on, must be recomputed many times per seconds. We do this by identifying the points belonging to the floor in the data input, i.e. in the frame K_{rf}, fitting a plane to those points and computing T as the roto-translation that express such plane in the frame H_{rf}. The transformation T is computed by assuming that the normal to the plane in world coordinates

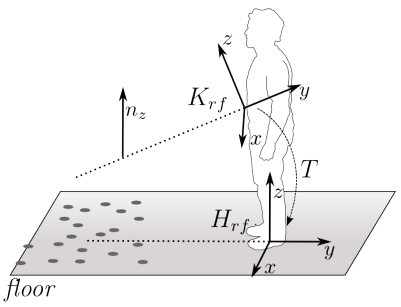


Figure 6.3: **Coordinate frames used by the *Registration* task**.

coincides with the z axis of H_{rf}. The translation part is simply the difference between the origin of H_{rf} and the origin of K_{rf} while the rotational part is given by:

$$R_{\vec{X}}(acos(n_z))) \cdot R_{\vec{Y}}(acos(\sqrt{(n_x{}^2 + n_z{}^2)}))$$

where n is expressed in the K_{rf} frame, $\vec{X}$ and $\vec{Y}$ are canonical axes and $R_{ax}(r)$ indicates the rotation matrix of r radians around the ax axis.

We identify the points belonging to the floor as those closer than a threshold to the plane fitted at the previous step. The plane at step 0 is computed in a *calibration phase* where the user stays motionless without obstacles in front of him/her, so that it is easy to identify points belonging to the floor.
This simple technique relies on frame-to-frame coherency, therefore it is crucial that the time lapse between consecutive plane fitting operations is minimal, which means that the task, like any other, must complete within the assigned time interval.

Completing within the assigned time The time required for computation is directly proportional to the size of the input depth map, i.e. the number of points, therefore the algorithm may subsample the input data or the assigned time interval is otherwise insufficient. The subsampling procedure consists of a *reduce* operation, in which an input depth map is reduced to a smaller one using a *min* operator. That is, to guarantee conservative results, i.e. that no obstacle goes undetected, each used sample is assigned with the minimum depth among the original samples it represents.

6.1.2.3 Occupancy Detector

This tasks must tell, for each direction the user could walk, the distance to the closest obstacle. Because the user needs a free space corresponding to his/her height and width, the task first computes a low-resolution, one-dimensional $D \times 1$ depth map using a conservative process as in the Registration task. The resulting map is therefore a quantization of the space in front of the user into D values, each one corresponding to a vertical volume spanning a range of possible directions. The produced depth map is then used to find, for each direction, the farthest point reachable by the user considering his/her width. Figure 6.4 shows a depth map of size 8, in which the most right region is not reachable although *visible* from the device. Note that we have no use of a high resolution depth map since ultimately the user makes imprecise movements. In our tests we empirically found that a reasonable size of the depth map is 32.

Completing within the assigned time Like for the Registration task, the time for the OccupancyDetector is proportional to the size of the depth-map that can be reduced to fulfill the time constraint. However, note that this task is very simple and fast to complete, so that its interruption was never necessary in practical cases.

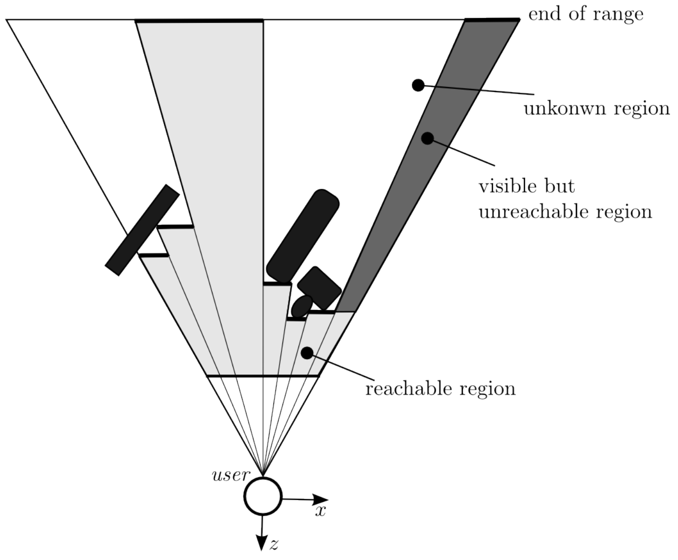


Figure 6.4: **Linear depth map of size** 8.

6.1.2.4 Walk Detector

This task uses the accelerometer mounted with the Kinect to detect if the user is walking and at, approximately, at what speed. Since the device is fixed at the height of the pelvis, when the user walks it moves from left to right and vice versa. Therefore, just like any step counter does, we track the values provided by the accelerometer and count the frequency of changes of sign of derivatives. To avoid errors introduced by high-frequency noise, the input signal is first smoothed using a simple *finite impulse response* filter.

6.1.2.5 Analysis

This task reads the input provided by the OccupancyDetector and the WalkDetector and decides what feedback to give to the user. The algorithm consists of tracking the result of the OccupancyDetector to detect if the user should be told to change direction or to stop. Note that the occupancy of a region of space is an obstacle only if the user is moving towards said region. If, for example, the user is walking along a corridor, the walls will occupy the space on left and right, but they are not obstacles. Another example is represented by someone walking in front of the user in the same direction. In this case, the distance to the occupied region will be roughly constant. However, roughly means that we must filter off oscillations due to human walking which means, in turn, that we should know if the person is walking, and this is where WalkDetector is comes into play.
In the current implementation, the Analysis task monitors the occupancy map and tests if an obstacle is approaching the user. If this is the case it suggests the closest direction to take to avoid the obstacle. If there is no such direction, it notifies the user the he/she is approaching to a dead end. The speed estimated by the WalkDetector is used to assign the dynamically set time constraints to the other tasks. If the user is walking slowly or is standing still, we need a lower refresh rate than if the user is walking fast. In other words, we may say we express the update frequency per meter. As a conservative policy, if the OccupancyData is not updated on time the Analysis task tells the user to stop.

6.1.2.6 Audio feedback

We tried two alternative audio feedback responses. The first consisted in using few simple messages (“proceed”,“keep left”,“keep right”,“stop”) that were given according to the simple goal of walking towards the farthest reachable place visible from the scanner. The second was a continuous tone used to indicate the direction to the farthest reachable place by mapping the direction on the stereo balance, and the distance to the place in question to the tone pitch.

6.1.3 Experimental Results

We implemented our platform-independent framework in C++, using Qt for multithread/multi-process foundations and OpenAL for audio feedback and run the system on a Eee PC 1015PD 1.66 GHz, 1 GB RAM. Note that the Eee PC is not a smart phone but it is less powerful of many currently available smarphone such as the Motorola Atrix (dual core with an NVidia Tegra 2 graphics card). During development and test we used OpenGL to have a three-dimensional overview of the acquired data.
We tested the system on a blindfolded, non visually impaired person. This choice was made for the twofold reason that the system is not yet in its final release and that a born blind person is used to walk without seeing and it would be harder to evaluate the actual contribution of our system. The subject was brought to an unfamiliar location and asked to reach a point identified as the source of a sound. The location was filled with obstacles such as chairs, desks and books on the ground and there were other people moving in the room. As expected, the user was able to safely reach the target place avoiding the obstacles and taking the shortest way. It is clear that this is still a controlled environment where, for example, there are no stairs, therefore the tasks essentially work as proximity detectors. However, even at this early stage we could observe the effects of the time-critical system. After the first test was completed, we asked the user to do another one by walking as fast as he felt like. In this case the system asked the user to stop when the first obstacle was approaching. This happened not because the OccupancyDetector could not compute the direction to move to, but simply because it could not do it fast enough.

We observed that, after few minutes, the use of the continuous tone for providing feedback was more profitable that single messages. This did not come as a surprise since the tone gives the essentially same information processed by the Analysis task, but lets the brain do the analysis.

Conclusions

In this dissertation we have presented solutions for computing volumetric light-matter interaction effects on commodity hardware. While on one hand it can be affirmed that the majority of fundamental challenges in Computer Graphics have been solved in the last thirty years, the intrinsic difficulty of light simulation still keeps the design of efficient algorithms an open problem. Even if processing power has steadily increased, the demand for realism and speed has grown as well. It is apparent that the efficient exploitation of parallelism is one of the promising directions of research, since light propagation is, by its nature, an inherently parallelizable process. As the design of hardware changes, and restrictions are lifted on the types of operations that can be performed, so software architecture will evolve as well. At the same time, perfect physical simulations are not always necessary in end-user applications and for most purposes it is sufficient to achieve a plausible rendition of lighting effects. Suitable approximations and reasonable assumptions seem to be always necessary in order to ameliorate the difficulties of the rendering problem, and the algorithms presented in this thesis conform to this approach.

The first method described has been based on the single scattering approximation, which is valid for all materials with low albedo. We have presented a novel way to solve the resulting problem by sampling the volume and storing at each sample site the directional optical depth function. This function is compressed by projecting it on a Spherical Harmonics basis, such that storage is reduced and evaluation at run-time can be performed efficiently. Spherical Harmonics projection however acts as a low pass filter, and even if the exponential attenuation function is smooth, some details are unavoidably lost unless a high number of coefficients is employed. The entire algorithm relies on a precomputation strategy, which in its current CPU implementation is very expensive; however a GPU implementation could be substituted in that stage with relatively low effort. The resulting method is capable of dealing with any mesh, even topologically problematic meshes, and it is easy to incorporate in an existing rendering pipeline, adding to the realism of translucent materials.

In the second method we relaxed the single scattering assumption and allowed for the presence of multiple scattering. We decided to attain a higher spatial resolution and work on a voxel grid of appropriate dimensions. Single Scattering in this method is computed by photon marching light from sources and performing view-ray marching as a final pass. Refraction effects are accounted for by employing eikonal equations instead of standard ray equations used in marching processes. Multiple Scattering is instead computed by leveraging on the diffusion approximation, which has been shown to be valid for highly scattering and isotropic materials. Under this assumption it is possible to compute MS by a Lattice-Boltzmann method, which had never been used on a GPU at the time. However, we noticed that it could be possible to parallelize efficiently the SS computation and the MS one. We therefore developed a parallel architecture which assigns GPU WorkGroups to the tracing task or to the diffusion task, exploiting the load balancing mechanisms of current generation GPUs. We further modified standard LBL computation to make it compatible with a strictly blockized structure and implementing a synchronization method using interfaces between blocks. The resulting system produces images which are comparable to ground truth pictures produced with a state-of-the-art offline stochastic photon tracer.

In the third method we dropped the diffusion approximation and worked on a method that could be capable of capturing the effects on light scattering due to anisotropic phase functions. The method can be classified in the Discrete Ordinates Method family, as it performs a full discretization of directions in addition to the discretization of space employed in the previous method. Transfer of energy is accurately computed using the Radiative Transfer Equation not only between voxels, but also between different faces of the same voxel. In order to keep the dimensions of the light-storage dataset manageable, we detach the propagation angular resolution from the storage angular resolution, and employ a wave-like propagation scheme similar to the sweep3d algorithm. This allows in-scattered light to be propagated at a higher resolution and at a higher distance in less time with respect to a Lattice Boltzmann propagation. Furthermore, in order to support bigger datasets, we developed an Out Of Core streaming mechanism that exploits known access patterns to move volumetric data to and from GPU memory. This allows the method to accurately visualize scattering effects even in datasets many times bigger than the available memory.

In a future work, we could explore the interaction of lighting effects with volumes defined at different resolutions in different areas (eg. Sun et al's work[122]), especially in our multiple scattering solutions. Such reframing of the problem would introduce in the first of such algorithms (chapter 4) issues regarding the energy exchange between Workgroups operating at different resolutions, while in the second (chapter 5) it would impose a different paradigm of wave propagation. However, the presented algorithms would be more amenable to be integrated in an end-user rendering engine, where dynamically switching dataset resolution is a common acceleration technique. Furthermore, there have been little studies on the correlation between the resolution needed for subsurface lighting data structures and the resolution of the input dataset. Among the algorithms working in this direction, we recall here Lightcuts[1] and Light Propagation Volumes[15, 11]. On a different branch of research, further improvements could be made on the second paper, namely a modified DOM which can take into account Eikonal refraction through volumes. To the best of our knowledge, such modification has not been introduced yet. In general, few investigations has been made on the correlation between refraction and scattering, and it is telling how, despite being two phenomenas springing from the same physical properties, they are generally treated as mutually independent effects. For example, even if a refracting reformulation of the diffusion equation has been introduced [78], no algorithm based on such a model has yet been produced.

In conclusion, the current dissertation has added to the state of the art in the domain of real-time rendering of volumetric effects and in collateral fields, and it has resulted in the following publications:

- [7] D. Bernabei, F. Ganovelli, N. Pietroni, P. Cignoni, S. Pattanaik, and R. Scopigno, “Real-time single scattering inside inhomogeneous materials,” The Visual Computer, vol. 26, no. 6–8, pp. 583–593, Apr. 2010.
- [8] D. Bernabei, A. Hakke Patil, F. Banterle, F. Ganovelli, M. Di Benedetto, S. Pattanaik, and R. Scopigno, “A Parallel Architecture for Interactive Rendering of Scattering and Refraction Effects,” IEEE Computer Graphics and Applications, vol. 30, no. 3, 2011.
- [6] D. Bernabei, F. Ganovelli, M. Di Benedetto, M. Dellepiane, and R. Scopigno, “Walka-Ware : A Low-Cost Time-Critical Obstacle Avoidance System for the Visually Impaired,” in International Conference on Indoor Positioning and Indoor Navigation (IPIN), 2011, Short Paper.

Bibliography

[1] Adam Arbree, Bruce Walter, and Kavita Bala. Single-pass Scalable Subsurface Rendering with Lightcuts. *Computer Graphics Forum*, 27(2):507–516, April 2008.

[2] Adam Arbree, Bruce Walter, and Kavita Bala. Diffusion formulation for heterogeneous subsurface scattering. Technical report, Cornell University, 2009.

[3] Adam Arbree, Bruce Walter, and Kavita Bala. Heterogeneous Subsurface Scattering Using the Finite Element Method. *IEEE Transactions on Visualization and Computer Graphics*, 17(7):956–969, 2011.

[4] Ilya Baran, Jiawen Chen, Jonathan Ragan-Kelley, Frédo Durand, and Jaakko Lehtinen. A hierarchical volumetric shadow algorithm for single scattering. In *ACM SIGGRAPH Asia 2010 papers on - SIGGRAPH ASIA '10*, volume 29, page 1, New York, New York, USA, December 2010. ACM Press.

[5] LB Barichello and CE Siewert. On the equivalence between the discrete ordinates and the spherical harmonics methods in radiative transfer. *Nuclear science and engineering*, pages 79–84, 1998.

[6] Daniele Bernabei, Fabio Ganovelli, Marco Di Benedetto, Matteo Dellepiane, and Roberto Scopigno. WalkaWare : A Low-Cost Time-Critical Obstacle Avoidance System for the Visually Impaired. In *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, number September, pages 21–23, 2011.

[7] Daniele Bernabei, Fabio Ganovelli, Nico Pietroni, Paolo Cignoni, Sumanta Pattanaik, and Roberto Scopigno. Real-time single scattering inside inhomogeneous materials. *The Visual Computer*, 26(6-8):583–593, April 2010.

[8] Daniele Bernabei, Ajit Hakke Patil, Francesco Banterle, Fabio Ganovelli, Marco Di Benedetto, Sumanta Pattanaik, and Roberto Scopigno. A Parallel Architecture for Interactive Rendering of Scattering and Refraction Effects. *IEEE Computer Graphics and Applications*, 30(3), 2011.

[9] N Bhate and A Tokuta. Photorealistic Volume Rendering of Media with Directional Scattering. In *Third Eurographics Workshop on Rendering*, pages 227–245, 1992.

[10] Markus Billeter, Erik Sintorn, and Ulf Assarsson. Real Time Volumetric Shadows using Polygonal Light Volumes. In *Proceedings of the Conference on High Performance Graphics*. Eurographics Association, 2010.

[11] Markus Billeter, Erik Sintorn, and Ulf Assarsson. Real-time multiple scattering using light propagation volumes. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D '12*, pages 119–126, New York, New York, USA, 2012. ACM Press.

[12] Pettitt S Blenkhom P. and Evans D G. An ultrasonic mobility device with minimal audio feedback. In *12˜ Annual International Conference on Technology and Persons with Disabilities*, 1997.

[13] James F Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *SIGGRAPH Comput. Graph.*, 16:21–29, 1982.

[14] Craig F. Bohren and Eugene Edmund Clothiaux. Fundamentals of atmospheric radiation: an introduction with 400 problems. page 472, 2006.

[15] J Borlum and BB Christensen. SSLPV: subsurface light propagation volumes. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, pages 7–14. ACM, 2011.

[16] R Bostelman, P Russo, J Albus, T Hong, and R Madhavan. Applications of a 3D Range Camera Towards Healthcare Mobility Aids. *2006 IEEE International Conference on Networking, Sensing and Control*, pages 416–421, 2006.

[17] Chen Cao, Zhong Ren, Baining Guo, and Kun Zhou. Interactive Rendering of Non-Constant, Refractive Media Using the Ray Equations of Gradient-Index Optics. In *Computer Graphics Forum*, volume 29, pages 1375–1382. Wiley Online Library, 2010.

[18] Eva Cerezo, Frederic Perez-Cazorla, Xavier Pueyo, Francisco Seron, and François Sillion. A Survey on Participating Media Rendering Techniques. *The Visual Computer*, 2005.

[19] S Chandrasekhar. *Radiative Transfer.* Dover Publications, Inc, 1950.

[20] Guojun Chen, Pieter Peers, Jiawan Zhang, and Xin Tong. Real-time rendering of deformable heterogeneous translucent objects using multiresolution splatting. *The Visual Computer*, 28(6-8):701–711, April 2012.

[21] Jiawen Chen, Ilya Baran, Frédo Durand, and Wojciech Jarosz. Real-time volumetric shadows using 1D min-max mipmaps. In *Symposium on Interactive 3D Graphics and Games on - I3D '11*, page 39, New York, New York, USA, February 2011. ACM Press.

[22] S Chumkamon, P Tuvaphanthaphiphat, and P Keeratiwintakorn. A blind navigation system using RFID for indoor environments. In *ECTI-CON 2008*, volume 2, pages 765–768, May 2008.

[23] P. J. Coelho. A modified version of the discrete ordinates method for radiative heat transfer modelling. *Computational Mechanics*, 33(5):375–388, April 2004.

[24] Robert L Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5:51–72, 1986.

[25] Michael Cox and David Ellsworth. Application-controlled demand paging for out-of-core visualization. In *Proceedings of the 8th conference on Visualization '97*. IEEE Computer Society Press, 1997.

[26] Cyril Crassin, Fabrice Neyret, S Lefebvre, and E. Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. *Proceedings of the 2009 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*, d, 2009.

[27] Carsten Dachsbacher and Marc Stamminger. Translucent shadow maps. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 197–201, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[28] E D'Atri, C M Medaglia, A Serbanati, and U B Ceipidor. A system to aid blind people in the mobility: A usability test and its results. In *Systems, 2007. ICONS '07. Second International Conference on*, page 35, 2007.

[29] Charles de Rousiers, Adrien Bousseau, Kartic Subr, Nicolas Holzschuch, and Ravi Ramamoorthi. Real-time rough refraction. In *Symposium on Interactive 3D Graphics and Games on - I3D '11*, page 111, New York, New York, USA, 2011. ACM Press.

[30] E D'Eon, David Luebke, and Eric Enderton. Efficient rendering of human skin. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 147–157. Eurographics Association, 2007.

[31] Eugene D'Eon and Geoffrey Irving. A Quantized-Diffusion Model for Rendering Translucent Materials. 30(4):1–13, 2011.

[32] Yoshinori Dobashi, Kazufumi Kaneda, Hideo Yamashita, Tsuyoshi Okita, and Tomoyuki Nishita. A Simple, Efficient Method for Realistic Animation of Clouds. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 19–28. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[33] Yoshinori Dobashi, Tsuyoshi Yamamoto, and Tomoyuki Nishita. Interactive Rendering of Atmospheric Scattering Effects Using Graphics Hardware. pages 1–10, 2002.

[34] Craig Donner and Henrik Wann Jensen. Light diffusion in multi-layered translucent materials. *ACM Transactions on Graphics*, 24:1032–1039, July 2005.

[35] Craig Donner and HW Jensen. Rendering translucent materials using photon diffusion. *ACM SIGGRAPH 2008 classes*, 2008.

[36] Craig Donner, Jason Lawrence, Ravi Ramamoorthi, Toshiya Hachisuka, Henrik Wann Jensen, and Shree Nayar. An empirical BSSRDF model. *ACM Transactions on Graphics*, 28(3):1, July 2009.

[37] Julie Dorsey, Alan Edelman, Henrik Wann Jensen, Justin Legakis, and Hans Kohling Pedersen. Modeling and rendering of weathered stone. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*, pages 225–234, New York, New York, USA, July 1999. ACM Press.

[38] Oskar Elek, Tobias Ritschel, and Hans-Peter Seidel. Real-Time Screen-Space Scattering in Homogeneous Environments. *IEEE Computer Graphics and Applications*, pages 1–1, 2013.

[39] Thomas Engelhardt and Carsten Dachsbacher. Epipolar sampling for shadows and crepuscular rays in participating media with single scattering. *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, page 119, 2010.

[40] Thomas Engelhardt, Jan Novak, and Carsten Dachsbacher. Instant Multiple Scattering for Interactive Rendering of Heterogeneous Participating Media. Technical Report December, 2010.

[41] KF Evans and LH Chambers. The Spherical Harmonics Discrete Ordinate Method for Three-Dimensional Atmospheric Radiative Transfer. *Program*, pages 0–4, 1998.

[42] Jianbin Fang, Ana Lucia Varbanescu, and Henk Sips. A Comprehensive Performance Comparison of CUDA and OpenCL. *2011 International Conference on Parallel Processing*, pages 216–225, September 2011.

[43] Shiaofen Fang and Hongsheng Chen. Hardware accelerated voxelization. *Computers & Graphics*, 2000.

[44] J Faria, S Lopes, H Fernandes, P Martins, J Barroso, Lopes St, Fernandes Ht, and Barroso J. Electronic white cane for blind people navigation assistance. In *World Automation Congress (WAC), 2010*, pages 1–7, 2010.

[45] Raanan Fattal. Participating media illumination using light propagation maps. *ACM Transactions on Graphics*, 28(1):1–11, January 2009.

[46] Guillaume François, Sumanta Pattanaik, Kadi Bouatouch, and Gaspard Breton. Subsurface texture mapping. In *SIGGRAPH {'}06: ACM SIGGRAPH 2006 Sketches*, page 172, New York, NY, USA, 2006. ACM.

[47] Robert Geist, Karl Rasche, James Westall, and Robert Schalkoff. Lattice-Boltzmann Lighting. In Alexander Keller and Henrik Wann Jensen, editors, *Eurographics Symposium on Rendering*, pages 355–362, Norrkoping, Sweden, 2004. Eurographics Association.

[48] Robert Geist and Jay Steele. A lighting model for fast rendering of forest ecosystems. *2008 IEEE Symposium on Interactive Ray Tracing*, pages 99–106, August 2008.

[49] Robert Geist and James Westall. Lattice-Boltzmann Lighting Models. In Wen-mei W. Hwu, editor, *GPU Computing Gems Emerald Edition*, chapter 25. Morgan Kaufmann, 2011.

[50] Enrico Gobbetti and Fabio Marton. Far voxels. *ACM Transactions on Graphics*, 24(3):878, July 2005.

[51] Enrico Gobbetti, Fabio Marton, and José Antonio Iglesias Guitián. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer*, 24(7-9):797–806, June 2008.

[52] Chunye Gong, Jie Liu, Zhenghu Gong, Jin Qin, and Jing Xie. Optimizing Sweep3D for Graphic Processor Unit. pages 416–426, 2010.

[53] S. Gortler, M.F. Cohen, and P. Slusallek. Radiosity and relaxation methods. *IEEE Computer Graphics and Applications*, 14(6):48–58, November 1994.

[54] Tom Haber, Tom Mertens, Philippe Bekaert, and Frank Van Reeth. A computational approach to simulate subsurface light diffusion in arbitrarily shaped objects. In *Graphics Interface*, Proceedings of the Graphics Interface 2005 Conference, May 9-11, 2005, Victoria, British Columbia, Canada, pages 79–86. Canadian Human-Computer Communications Society, 2005.

[55] David W Hahn. Light Scattering Theory. (July):1–13, 2009.

[56] Pat Hanrahan and Wolfgang Krueger. Reflection from layered surfaces due to subsurface scattering. *of the 20th annual conference on*, page 174, 1993.

[57] Xuejun Hao and Amitabh Varshney. Real-time rendering of translucent meshes. *ACM Transactions on Graphics*, 23:120–142, 2004.

[58] Mark Harris, S Sengupta, and JD Owens. Parallel prefix sum (scan) with CUDA. *GPU Gems*, (April), 2007.

[59] Mark J Harris and Anselmo Lastra. Real-time cloud rendering. In *Computer Graphics Forum*, pages 76–84. Blackwell Publishing, 2001.

[60] Kyle Hegeman, Michael Ashikhmin, and Simon Premože. A lighting model for general participating media. *Proceedings of the 2005 symposium on Interactive 3D graphics and games - SI3D '05*, page 117, 2005.

[61] Nicolas Holzschuch and Jean-Dominique Gascuel. Double and Multiple-scattering effects in translucent materials. *IEEE Computer Graphics and Applications*, 3:1–1, 2013.

[62] H. C. Hottel and A. F. Sarofim. Radiative transfer. *AIChE Journal*, 15(5):794–796, 1969.

[63] Andreas Hub and Joachim Diepstraten. Augmented indoor modeling for navigation support for the blind. *CPSN'05-The International Conference on Computers for People with Special Needs*, 2005.

[64] Ivo Ihrke, Gernot Ziegler, Art Tevs, and Christian Theobalt. Eikonal rendering: efficient light transport in refractive objects. *ACM Transactions on Graphics*, 26(3), 2007.

[65] T Imagire, H Johan, N Tamura, and T Nishita. Anti-aliased and real-time rendering of scenes with light scattering effects. *The Visual Computer*, 2007.

[66] A Ishimaru. *Wave propagation and scattering in random media*. 1978.

[67] G Jansson, H Petrie, and C Colwell. Haptic virtual environments for blind people: Exploratory experiments with two devices. *Journal of Virtual Reality*, 3:10–20, 1999.

[68] Wojciech Jarosz, Derek Nowrouzezahrai, Iman Sadeghi, and Henrik Wann Jensen. A comprehensive theory of volumetric radiance estimation using photon points and beams. *ACM Transactions on Graphics*, 30(1):1–19, January 2011.

[69] Henrik Wann Jensen and Juan Buhler. A rapid hierarchical rendering technique for translucent materials. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 576–581, New York, NY, USA, 2002. ACM.

[70] Henrik Wann Jensen, S.R. Marschner, Marc Levoy, and Pat Hanrahan. A practical model for subsurface light transport. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 511–518, New York, NY, USA, 2001. ACM New York, NY, USA.

[71] Henrik Wann HW Jensen and PH Per H. Christensen. Efficient simulation of light transport in scences with participating media using photon maps. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 311–320, New York, NY, USA, July 1998. ACM.

[72] Jorge Jimenez, Veronica Sundstedt, and Diego Gutierrez. Screen-space perceptual rendering of human skin. *ACM Transactions on Applied Perception*, 6(4):1–15, September 2009.

[73] Daniel Jönsson, Erik Sundén, Anders Ynnerman, and Timo Ropinski. State of The Art Report on Interactive Volume Rendering with Volumetric Illumination. *Star*, 1, 2012.

[74] James T. Kajiya. The rendering equation, 1986.

[75] James T. Kajiya and Brian P Von Herzen. Ray Tracing Volume Densities. In *Computer Graphics (ACM SIGGRAPH '84 Proceedings)*, volume 18, pages 165–174, July 1984.

[76] Anton Kaplanyan and Carsten Dachsbacher. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D '10*, page 99, New York, New York, USA, 2010. ACM Press.

[77] Alexander Keller. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*, pages 49–56, New York, New York, USA, 1997. ACM Press.

[78] T Khan. On derivation of the radiative transfer equation and its diffusion approximation for scattering media with spatially varying refractive indices. *Clemson University Mathematical Sciences Technical Report*, 2003.

[79] Khronos OpenCL Working Group. OpenCL Specification. Technical report, 2011.

[80] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Multi-Dimensional Transfer Functions for Interactive Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, July 2002.

[81] K.R. Koch, R.S. Baker, and R.E. Alcouffe. Solution of the first-order form of the 3-D discrete ordinates equations on a massively parallel machine. January 1991.

[82] Jens Krüger, Kai Bürger, and Rüdiger Westermann. Interactive Screen-Space Accurate Photon Tracing on GPUs, 2006.

[83] Eric P Lafortune and Yves D Willems. Rendering Participating Media with Bidirectional Path Tracing. In Xavier Pueyo and Peter Schröder, editors, *Rendering Techniques '96*, Eurographics, pages 91–100. Springer-Verlag Wien New York, 1996.

[84] E. Languénou, Kadi Bouatouch, and M. Chelle. Global illumination in presence of participating media with general properties. In *Proceedings of the Fifth Eurographics Workshop on Rendering*, pages 69–85, 1994.

[85] P Lecocq, S Michelin, D. Arques, and A. Kemeny. Mathematical approximation for real-time lighting rendering through participating media. *Pacific Graphics Short Papers*, pages 2–3, 2000.

[86] Hendrik P A Lensch, Michael Goesele, Philippe Bekaert, Jan Kautz, Marcus A Magnor, Jochen Lang, and Hans-Peter Seidel. Interactive Rendering of Translucent Objects. *Computer Graphics and Applications, Pacific Conference on*, 0:214, 2002.

[87] Marc Levoy. Display of Surfaces from Volume Data. D:1–10, 1988.

[88] Dongping Li, X Sun, Z Ren, Stephen Lin, Y Tong, B Guo, and K Zhou. TransCut: Interactive Rendering of Translucent Cutouts. pages 1–11, 2012.

[89] Hongsong Li, Fabio Pellacini, and Kenneth E. Torrance. A hybrid monte carlo method for accurate and efficient subsurface scattering. *Eurographics Symposium on Rendering*, pages 283–290, June 2005.

[90] Erik Lindholm, Mark J. Kligard, and Henry Moreton. A user-programmable vertex engine. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*, pages 149–158, 2001.

[91] J Lopez-Moreno, Angel Cabanes, and Diego Gutierrez. Image-based participating media. *Proc. CEIG*, 5, 2008.

[92] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21:163–169, 1987.

[93] Morgan McGuire and David Luebke. Hardware-accelerated global illumination by image space photon mapping. *Proceedings of the Conference on High Performance Graphics*, 2009.

[94] Tom Mertens, Jan Kautz, Philippe Bekaert, Frank Van Reeth, and Hans-Peter Seidel. Efficient Rendering of Local Subsurface Scattering. *Computer Graphics Forum*, 24(1):41–49, March 2005.

[95] Tom Mertens, Jan Kautz, Hans-Peter Seidel, Philippe Bekaert, and F. Van Reeth. Interactive rendering of translucent deformable objects. In *Proceedings of the 14th Eurographics workshop on Rendering*, pages 130–140, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[96] Jonathan T Moon, Bruce Walter, and Steve Marschner. Efficient multiple scattering in hair using spherical harmonics. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–7, New York, NY, USA, 2008. ACM.

[97] OV Nikolaeva and LP Bass. Radiative transfer in horizontally and vertically inhomogeneous turbid media. In *Light Scattering Reviews 2*. Springer Praxis Books, 2007.

[98] Tomoyuki Nishita, Yasuhiro Miyawaki, and Eihachiro Nakamae. A shading model for atmospheric scattering considering luminous intensity distribution of light sources. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 303–310, New York, NY, USA, 1987. ACM.

[99] SN Pattanaik and SP Mudur. Computation of global illumination in a participating medium by monte carlo simulation. *The Journal of Visualization and Computer Animation*, 4(3):133–152, 1993.

[100] Mark Pauly, T Kollig, and A Keller. *Metropolis light transport for participating media.* 2000.

[101] Vincent Pegoraro. A closed-form solution to single scattering for general phase functions and light distributions. 2010.

[102] Vincent Pegoraro. A mathematical framework for efficient closed-form single scattering. In *Proceedings of Graphics Interface 2011*, pages 151–158. Canadian Human-Computer Communications Society, 2011.

[103] Vincent Pegoraro and Steven G. Parker. An Analytical Solution to Single Scattering in Homogeneous Participating Media. *Computer Graphics Forum*, 28(2):329–335, April 2009.

[104] F G Peris, L Dunai, P V Santiago, and I Dunai. CASBliP - a new cognitive object detection and orientation aid system for blind people. *CogSys2010 Conference*, 2010.

[105] Matt Pharr and Pat Hanrahan. Monte Carlo evaluation of non-linear scattering equations for subsurface reflection. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, pages 75–84, July 2000.

[106] Rudolph W Preisendorfer. *Hydrologic Optics.* U.S. Department of Commerce - National Oceanic and Atmospheric Administration Enviromental Research Laboratories, 1976.

[107] Simon Premože, Michael Ashikhmin, and Peter Shirley. Path integration for light transport in volumes. In *Proceedings of the 14th Eurographics workshop on Rendering*, page 63. Eurographics Association, 2003.

[108] Ravi Ramamoorthi and Pat Hanrahan. Frequency space environment map rendering. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 517–526, New York, NY, USA, 2002. ACM.

[109] M.a. Ramankutty and A.L. Crosbie. Modified discrete ordinates solution of radiative transfer in two-dimensional rectangular enclosures. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 57(1):107–140, January 1997.

[110] Kirk Riley, David S Ebert, Martin Kraus, Jerry Tessendorf, and Charles Hansen. Efficient Rendering of Atmospheric Phenomena, 2004.

[111] Holly Rushmeier. *Realistic Image Synthesis for Scenes with Radiatively Participating Media.* PhD thesis, 1988.

[112] Holly E. Rushmeier and Kenneth E. Torrance. The zonal method for calculating light intensities in the presence of a participating medium. *ACM SIGGRAPH Computer Graphics*, 21(4):293–302, August 1987.

[113] Musawir A. Shah, Jaakko Konttinen, and Sumanta Pattanaik. Image-space subsurface scattering for interactive rendering of deformable translucent objects. *Computer Graphics and Applications*, 29(1):66–78, 2009.

[114] Shraga Shoval, Iwan Ulrich, and Johann Borenstein. *Computerized obstacle avoidance systems for the blind and visually impaired*, pages 413–448. CRC Press, Inc., Boca Raton, FL, USA, 2001.

[115] François X. Franqois X Sillion. A unified hierarchical algorithm for global illumination with scattering volumes and object clusters. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):240–254, 1995.

[116] Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. Clustered principal components for precomputed radiance transfer. *ACM Transactions on Graphics*, 22:382–391, 2003.

[117] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques - SIGGRAPH '02*, page 527, New York, New York, USA, 2002. ACM Press.

[118] Ying Song, Xin Tong, Fabio Pellacini, and Pieter Peers. SubEdit: a representation for editing measured heterogeneous subsurface scattering. *ACM Transactions on Graphics (* ..., 2009.

[119] Jos Stam. Multiple scattering as a diffusion process. In *Eurographics Rendering Workshop 1995*, pages 41–50. Citeseer, 1995.

[120] J Steele and Robert Geist. Relighting forest ecosystems. *Advances in Visual Computing*, pages 1–12, 2009.

[121] Bo Sun, Ravi Ramamoorthi, Srinivasa G Narasimhan, and Shree K. Nayar. A Practical Analytic Single Scattering Model for Real Time Rendering. In *ACM SIGGRAPH 2005 Papers*, number c, page 1049. ACM, 2005.

[122] Xin Sun, Kun Zhou, Eric Stollnitz, Jiaoying Shi, and Baining Guo. Interactive relighting of dynamic refractive objects. In *ACM SIGGRAPH 2008 papers*, page 35. ACM, 2008.

[123] Nelson Max This, F L July, Istribution Of, This Document, and Nelson L. Max. Efficient light propagation for multiple anisotropic volume scattering. In *Proc. of the Fifth Eurographics Workshop on Rendering*, volume pages, pages 87–104. Citeseer, 1994.

[124] M A Torres-Gil, O Casanova-Gonzalez, and J L Gonzalez-Mora. Applications of virtual reality for visually impaired people. *W. Trans. on Comp.*, 9(2):184–193, February 2010.

[125] Tatsuro Ueda, Hirohiko Kawata, Tetsuo Tomizawa, Akihisa Ohya, and Shin'ich Yuta. Visual Information Assist System Using 3D SOKUIKI Sensor for Blind People, System Concept and Object Detecting Experiments. *IECON 2006 - 32nd Annual Conference on IEEE Industrial Electronics*, pages 3058–3063, November 2006.

[126] Biri Venceslas, Michelin Sylvain, and Didier Arques. Real-Time Single Scattering with Volumetric Shadows, May 2003.

[127] B Walter, S Zhao, N Holzschuch, and Kavita Bala. Single scattering in refractive media with triangle mesh boundaries, 2009.

[128] Jiaping Wang. Modeling and rendering of heterogeneous translucent materials using the diffusion equation. *ACM Transactions on Graphics*, 27:1, 2008.

[129] Rui Wang, John Tran, and David Luebke. All-frequency interactive relighting of translucent objects with single and multiple scattering. *ACM Transactions on Graphics*, 24(3):1207, 2005.

[130] Y. Wang, J Wang, N Holzschuch, K. Subr, J.H. Yong, and Baining Guo. Real-time Rendering of Heterogeneous Translucent Objects with Arbitrary Shapes. In *Computer Graphics Forum*, volume 29, pages 497–506. John Wiley & Sons, 2010.

[131] D White K. . Cline and P Egbert. Poisson disk point sets by hierarchical dart throwing. *Symposium on Interactive Ray Tracing*, pages 129–132, 2007.

[132] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[133] Chris Wyman. An approximate image-space approach for interactive refraction. *ACM SIGGRAPH 2005 Papers on - SIGGRAPH '05*, 1:1050, 2005.

[134] Chris Wyman and Shaun Ramsey. Interactive volumetric shadows in participating media with single-scattering. In *Interactive Ray Tracing, 2008. RT 2008. IEEE Symposium on*, pages 87–92. IEEE, 2008.

[135] Kun Xu, Yue Gao, Yong Li, Tao Ju, and Shi-Min Hu. Real-time homogenous translucent material editing. *Computer Graphics Forum*, 26(3):545–552, September 2007.

[136] Kun Zhou, Qiming Hou, Minmin Gong, John Snyder, Baining Guo, and H.Y. Shum. Fogshop: Real-time design and rendering of inhomogeneous, single-scattering media. *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pages 116–125, 2007.

[137] Kun Zhou, Zhong Ren, Stephen Lin, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Real-time smoke rendering using compensated ray marching. *ACM Transactions on Graphics*, 27(3):1, August 2008.

Appendix A

Spherical Harmonics and Legendre Polynomials

A.1 Spherical Harmonics

Spherical Harmonics are a set of twice continuously differentiable orthogonal spherical functions which satisfy Laplace Equation $\nabla^2 f = 0$. They can be defined from Legendre Polynomials as follows,

$$Y_{l,m}(\theta, \varphi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} P_{l,m}(\cos\theta) e^{im\varphi}$$

where l is called the *degree* and m the *order* of the harmonic. $P_{l,m}(x)$ is an Associated Legendre Polynomial . For the sake of notation simplicity, we introduce the scaling factor $K_{l,m}$, defined as

$$K_{l,m} = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}}$$

Spherical Harmonics have Complex domain; since we intend to use them to approximate real valued functions, we will restrict to Real Spherical Harmonics only, indicated as $y_{l,m}(\theta, \varphi)$. They are defined from Complex Spherical Harmonics as follows:

$$y_{l,m}(\theta, \varphi) = \begin{cases} \frac{1}{\sqrt{2}} \left(Y_{l,m}(\theta, \varphi) + Y^*_{l,m}(\theta, \varphi) \right) & m > 0 \\ Y_{l,0}(\theta, \varphi) & m = 0 \\ \frac{-i}{\sqrt{2}} \left(Y_{l,|m|}(\theta, \varphi) - Y^*_{l,|m|}(\theta, \varphi) \right) & m < 0 \end{cases}$$

where $Y_{l,m}(\theta, \varphi)$ is the Complex Spherical Harmonic and $Y^*_{l,m}(\theta, \varphi)$ its conjugate. Separating the real part from the imaginary one, we have,

$$\Re\left(Y(\theta, \varphi)\right) = K_{l,m} P_{l,m}(\cos\theta)\cos(m\varphi)$$

$$\Im\left(Y(\theta, \varphi)\right) = K_{l,m} P_{l,m}(\cos\theta)\sin(m\varphi)$$

Finally, the real Spherical Harmonics are,

$$y_{l,m}(\theta, \varphi) = \begin{cases} \sqrt{2}\,\Re\left(Y_{l,m}(\theta, \varphi)\right) & m > 0 \\ Y_{l,0}(\theta, \varphi) & m = 0 \\ \sqrt{2}\,\Im\left(Y_{l,|m|}(\theta, \varphi)\right) & m < 0 \end{cases}$$

with

$$Y_{l,0}(\theta,\varphi) = K_{l,0}P_{l,0}(\cos\theta)$$

Spherical Harmonics have many application domains. Their main use in Computer Graphics is to approximate spherical functions, in a manner similar to the Fourier Transform. Any function $f(\theta,\varphi)$ can be projected on the Spherical Harmonics basis by integrating its product with the basis.

$$f_{l,m} = \int_{\Xi} f(\theta,\varphi)y_{l,m}(\theta,\varphi)\,d(\theta,\varphi)$$

The resulting coefficients $f_{l,m}$ may be used to reconstruct the original function in the following manner

$$\tilde{f}(\theta,\varphi) = \sum_{l=0}^{n-1}\sum_{m=-l}^{l} f_{l,m}y_{l,m}(\theta,\varphi)$$

Spherical Harmonics act as low-pass filters on functions, producing smooth, anti-aliased approximations.

A very useful property of SHs is that they considerably speed up the operation of integrating the product of two functions. If f and g are both projected on the Spherical Harmonics basis, then

$$\int_{\Xi} f(\theta,\varphi)g(\theta,\varphi)\,d(\theta,\varphi) = \sum_{l=0}^{n-1}\sum_{m=-l}^{l} f_{l,m}(\theta,\varphi)g_{l,m}(\theta,\varphi)$$

This means, essentially, that any integral of such form can be reduced to a dot product of two vectors of coefficients.

A similar property allows for the partial precomputation of a product of two functions in the SH basis. Suppose function $a(\theta,\varphi)$ is known while function $b(\theta,\varphi)$ will be known only at a later time. Let us call $c(\theta,\varphi) = a(\theta,\varphi)b(\theta,\varphi)$. It is then possible to build a matrix that represents the multiplication by $a(\theta,\varphi)$(in the SH basis) as follows:

$$A_{i,j} = \int_{\Xi} a(\theta,\varphi)y_i(\theta,\varphi)y_j(\theta,\varphi)\,ds$$

so that the coefficients of c can be obtained by

$$c_i = \sum_{j=1}^{n^2} A_{i,j}b_i$$

Spherical Harmonics have other interesting properties regarding rotations and convolutions of functions, which will not be covered in this text.

A.2 Legendre Polynomials

Associated Legendre Polynomials are solutions to General Legendre Differential Equation, and they can be defined from *Legendre Polynomials* using recurrence relations.

The first two Legendre Polynomials are:

$$P_0(x) = 1$$

$$P_1(x) = x$$

To compute successive polynomials in the series, the following recurrence relation can be used:

$$P_{l+1}(x) = \frac{(x(2l+1)P_l(x) - lP_{l-1}(x))}{l+1}$$

Regarding Associated Legendre Polynomials, the following recurrence relations are used,

$$P_{l,0}(x) = P_l(x)$$

$$P_{m,m}(x) = (-1)^m(2m-1)!!\sqrt{(1-x^2)^l}$$

$$P_{m,m+1}(x) = x(2m+1)P_{m,m}(x)$$

$$P_{m,l+1}(x) = \frac{(x(2l+1+m)P_{m,l}(x) - lP_{m,l-1}(x))}{l+1-m}$$