EFFICIENT SAMPLING METHODS FOR
PHOTOREALISTIC RENDERING ON THE GPU

A dissertation submitted to
the Budapest University of Technology and Economics
in fulfillment of the requirements
for the degree of Doctor of Philosophy (Ph.D)

by

Tóth Balázs György

Department of Control Engineering and Information Technology
Budapest University of Technology and Economics

advisor
Szirmay-Kalos László

2013
Contents

1 Introduction
  1.1 Formulation of the image synthesis process ........................................... 2
     1.1.1 Rendering equation ................................................................. 3
     1.1.2 Volumetric rendering equation .................................................... 4
  1.2 Solution of the rendering equation ....................................................... 5
     1.2.1 Global solution using numerical quadrature .................................... 6
     1.2.2 Local illumination model ............................................................ 6
  1.3 The modern rendering pipeline .............................................................. 7
  1.4 Objectives of this dissertation ............................................................. 10

2 Efficient sampling for screen-space algorithms ................................. 11
  2.1 Previous work ......................................................................................... 11
     2.1.1 Importance sampling ................................................................. 12
  2.2 Texture filtering with importance sampling ............................................ 13
  2.3 Tone mapping with bloom effect ............................................................ 15
     2.3.1 Implementation of bloom .............................................................. 16
  2.4 Depth of field ......................................................................................... 16
     2.4.1 Implementation of depth of field ..................................................... 18
  2.5 Results .................................................................................................... 19
  2.6 Conclusion ............................................................................................... 20

3 Real-time approximation of single scattering .................................. 21
  3.1 Previous work ......................................................................................... 21
  3.2 Volumetric lighting in participating media .............................................. 22
  3.3 Approximation of volumetric rendering equation .................................... 23
  3.4 Noise reduction with interleaved sampling ............................................. 25
  3.5 Results .................................................................................................... 26
  3.6 Conclusions ............................................................................................. 27

4 Approximation of ambient lighting ............................................... 29
  4.1 Previous work ......................................................................................... 29
     4.1.1 From the rendering equation to ambient occlusion and related models ... 30
     4.1.2 Ambient occlusion and indirect lighting on sampled geometry ............ 32
     4.1.3 Occlusion calculation for height fields defined by the depth buffer ...... 33
  4.2 Non-uniform sample distribution for point sampling ............................. 35
  4.3 A new integration method for volumetric ambient occlusion .................. 37
  4.4 Approximation of indirect lighting ........................................................... 40
     4.4.1 Adding local indirect lighting to ambient occlusion ......................... 40
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.2 Local approximation of the ambient transfer function</td>
<td>40</td>
</tr>
<tr>
<td>4.5 Implementation of ambient occlusion and indirect lighting</td>
<td>42</td>
</tr>
<tr>
<td>4.6 Results</td>
<td>44</td>
</tr>
<tr>
<td>4.7 Conclusions</td>
<td>45</td>
</tr>
<tr>
<td>5 Multiple-scattering in participating media</td>
<td>47</td>
</tr>
<tr>
<td>5.1 Previous work</td>
<td>47</td>
</tr>
<tr>
<td>5.1.1 Light attenuation and free path sampling</td>
<td>48</td>
</tr>
<tr>
<td>5.2 Free path sampling with virtual particles</td>
<td>50</td>
</tr>
<tr>
<td>5.2.1 Upper-bounding extinction coefficient function</td>
<td>51</td>
</tr>
<tr>
<td>5.3 The photon transport algorithm</td>
<td>54</td>
</tr>
<tr>
<td>5.3.1 Free path sampling</td>
<td>56</td>
</tr>
<tr>
<td>5.3.2 Absorption sampling</td>
<td>58</td>
</tr>
<tr>
<td>5.3.3 Scattering direction</td>
<td>58</td>
</tr>
<tr>
<td>5.4 Attenuation calculation with virtual particles</td>
<td>59</td>
</tr>
<tr>
<td>5.5 Parallel random number generation</td>
<td>59</td>
</tr>
<tr>
<td>5.6 Evaluation of the photon transport algorithm</td>
<td>60</td>
</tr>
<tr>
<td>5.7 Results</td>
<td>62</td>
</tr>
<tr>
<td>5.8 Conclusions</td>
<td>64</td>
</tr>
<tr>
<td>6 Summary</td>
<td>70</td>
</tr>
<tr>
<td>6.1 Thesis group 1: Efficient sampling for screen-space algorithms</td>
<td>70</td>
</tr>
<tr>
<td>6.2 Thesis group 2: Screen-space approximation of the ambient lighting</td>
<td>71</td>
</tr>
<tr>
<td>6.3 Thesis group 3: Multiple scattering simulation in participating media</td>
<td>72</td>
</tr>
</tbody>
</table>

Own publications   73

Bibliography   76

Index   82

Nomenclature   85

List of Figures   86
Chapter 1

Introduction

In the field of photorealistic image synthesis, the main goal is to create images which are indistinguishable from the ones we see in real life. To achieve this, we can use reality as our guide. The required formulation of the physical rules of the light transport should be as accurate as possible. The natural process that we try to mimic during image synthesis contains the following steps:

- Light is emitted by a light source which can be natural (e.g. the sun, a candle etc.) or artificial (e.g. a light bulb, a LED etc).
- Light photons interact with objects in the world. Some of them are absorbed, others are scattered and propagated in new directions.
- A fraction of the originally emitted light photons reach the sensor.

In the real world these steps are performed by a vast amount of light photons, so recreating this process is an enormous task. Global illumination methods used to solve the light transport problem in general form, but usually these algorithms are so computation intensive that they cannot be evaluated in real-time. For interactive purposes we use a more or less simplified version of the general formulae to suit the time constraint. As the computation power of the Graphics Processing Units (GPU) is steadily growing, more complex approaches become feasible for real-time photorealistic image synthesis.

This thesis work presents real-time methods that are based on the simplification of the light-transport problem and are feasible to real-time image synthesis on the GPU.

1.1 Formulation of the image synthesis process

The light can be modeled as photons, electromagnetic waves or even as geometric rays. Light is energy, which travels through the space between the objects in the virtual world. Light is emitted by light sources, which can be represented in many different ways. Besides the geometric properties (e.g. position, direction), the amount of illumination the light source emits needs to be specified. Usually it can be quantified by the amount of energy of the light photons passing through a unit area surface perpendicular to the light direction per second. This is called the irradiance. The amount of energy in unit perpendicular area and in unit solid angle is called the radiance. The radiance is measured on representative wavelengths, which correspond to the visible spectrum [LA98]. The radiance is the most basic measure of light intensity in computer graphics because if we have two surfaces of the same radiance, then the eye would perceive them
as having similar color [SKSS08]. It means that image synthesis should compute the radiance of a surface and then force the display to emit the same radiance. This way the monitor provides the same color illusion as the virtual world would do if it existed.

The model of the virtual world is usually specified analytically or with a finite set of geometric primitives. As the real world has almost infinite geometric details, we also have to simplify our model in this case. For real-time purposes the objects in the virtual world are often represented as surface objects with material properties [SKAC03]. The material properties are used to simulate the interaction of light with the object.

As the light travels through the world, the interaction with the environment can be traced back to two physical phenomena. One of them is absorption, which happens inside matter. This causes some of the light to disappear or converted into another kind of energy. The absorption between surfaces is often neglected in real-time applications, due to its computation complexity. Usually it is assumed that the virtual world contains vacuum between the objects. In this case the radiance remains constant along a ray.

The other phenomenon is scattering, which happens when light encounters an optical discontinuity. This causes change to the direction of the light, but does not change the energy. The most important optical discontinuity is at the boundary of the virtual objects. Scattering happens into two distinct directions depending on material properties. Some of the light passes into the object (refraction), and another part is reflected out of it.

Finally some of the light is absorbed in the imaging sensor. Usually the sensor is composed of many small sensors, each of them measures the irradiance over its surface. Since the irradiance is measured as the average of the light rays from all incoming directions, often a camera model is applied to produce a specific image. The virtual camera is directional and contains a small opening (pinhole camera) that restricts the directions from which light can reach the sensors. As the incoming direction is restricted, these cameras measure the average radiance of the visible surfaces.

### 1.1.1 Rendering equation

The interaction between the light and environment is formalized by the so called *rendering equation* [Ka86], which is based on the law of conservation of energy and describes the radiance at surface points in the virtual world. With the assumption that absorption and scattering can happen only at surface points, the *rendering equation* characterizes the radiance reflected off a surface as the sum of the emitted radiance and the illumination provided by other surfaces modified by the material properties at this point:

\[
L(\vec{x}, \vec{\omega}) = L^e(\vec{x}, \vec{\omega}) + L^r(\vec{x}, \vec{\omega}),
\]

where \(L^e(\vec{x}, \vec{\omega})\) is the emission radiance at point \(\vec{x}\) in direction \(\vec{\omega}\) and \(L^r(\vec{x}, \vec{\omega})\) is the reflected radiance. The reflected radiance is the sum of the incoming illumination scattered at \(\vec{x}\) to direction \(\vec{\omega}\):

\[
L^r(\vec{x}, \vec{\omega}) = \int_{\Omega'} L(\vec{x}', \vec{\omega}') f_r(\vec{\omega}', \vec{x}, \vec{\omega}) \cos^+ \theta_{x}' d\omega',
\]  

(1.1)

where \(\vec{x}'\) is the point visible from \(\vec{x}\) at direction \(\vec{\omega}'\), \(\Omega'\) is the directional sphere, \(f_r(\vec{\omega}', \vec{x}, \vec{\omega})\) is the *bidirectional reflectance distribution function* (BRDF), and \(\theta_{x}'\) is the incident angle between the surface normal and incoming direction \(\vec{\omega}'\) at the surface point. The superscript + denotes that negative \(\cos \theta_{x}'\) values should be replaced by zero when the incident angle is greater than 90°, since the light illuminates the back side of the surface. Position \(\vec{x}\) of the shaded point and direction \(\vec{\omega}\) specify illuminating point \(\vec{x}'\) exactly. As this equation contains a strong correspondence between
the surface points, it is a Fredholm type integral equation of second kind. These integral equations are hard to solve analytically [SKKA01] since the unknown radiance on the left side ($L(\vec{x}, \vec{\omega})$) is needed to evaluate the integral on the right side.

1.1.2 Volumetric rendering equation

Without the simplification that the virtual world does not contain matter between the surfaces, the rendering problem gets even more difficult. We have to incorporate the absorption and scattering in participating media into the rendering equation. As the incoming radiance at a given surface point does not only depend on the geometric properties of the virtual world, we should consider the change of radiance along a path as light travels through the virtual world (Figure 1.1). Let us consider a differential path $ds$ of direction $\vec{\omega}$. The change of the radiance $L$ is caused by the following phenomena:

- The radiance along the path may be increased by the photons emitted by the participating media itself. This change can be formalized as $L' ds$ where $L'$ is the emission density.

- When light photons collide with the particles of participating media some of them are absorbed or scattered out from its path. These two effects can be described by absorption coefficient $\sigma_a$ and scattering coefficient $\sigma_s$ (also called cross sections) of the media. As these two phenomena cannot be distinguished just by measuring the light beam, the two effects are usually described by attenuation coefficient $\sigma_t = \sigma_a + \sigma_s$, also called extinction coefficient. The ratio of scattering and extinction cross sections is the probability that a photon does not get lost at a material interaction and is called the albedo $a = \sigma_s / \sigma_t$. These cross sections express the probability that an event happens in unit length path. The change of the radiance due these phenomena in differential length $ds$ can be formalized as $-\sigma_t L ds$.

- Radiance can also increase due in-scattering, when photons along different paths get scattered into the considered direction. The amount of scattered photons in a given direction $\vec{\omega}'$ from a differential solid angle $d\omega'$ equals to the product of the amount of incoming photons and the probability of the scattering from $d\omega'$ to $\vec{\omega}'$. The probability density of the scattering direction is called the phase function. This probability density can be expressed analytically or measured from real world medium. The total radiance increase from all directions can be formalized as

$$\sigma_s ds \int_{\Omega'} L(s, \vec{\omega}') P(\vec{\omega}', \vec{\omega}) d\omega',$$

where $\Omega'$ is the directional sphere.
Incorporating the discussed phenomena into the rendering equation, we obtain the following formula for radiance $L$ of a ray at $s + ds$:

$$L(s + ds, \bar{w}) = L(s, \bar{w}) - \sigma_t ds L(s, \bar{w}) + L^e(s, \bar{w}) ds + \sigma_s ds \int_{\Omega'} L(s, \bar{w}') P(\bar{w}', \bar{w}) d\omega'.$$

This equation can also be presented in an integro-differential form:

$$\frac{dL(s, \bar{w})}{ds} = -\sigma_t L(s, \bar{w}) + L^e(s, \bar{w}) + \sigma_s \int_{\Omega'} L(s, \bar{w}') P(\bar{w}', \bar{w}) d\omega'.$$

or in pure integral form

$$L(s', \bar{w}) = \int_{s'} \left(-\sigma_t L(s, \bar{w}) + L^e(s, \bar{w}) + \sigma_s \int_{\Omega'} L(s, \bar{w}') P(\bar{w}', \bar{w}) d\omega'\right) ds.$$

(1.2)

This integral equation is the volumetric rendering equation. Since the in-scattering term of the volumetric rendering equation makes the evaluation difficult, we can assume various simplifications for real-time purposes. The most drastic approach is the complete elimination of the in-scattering term and using only the emission and attenuation parts of the equation. We can restrict the use of the volumetric equation only for light paths between the surface points and the camera, which can simulate fog like effects. If we use the equation for light paths between the surface points and the light sources taking into account the objects along the rays, then the shadow of the participating media shows up with the additional light shafts caused by the opaque objects (Chapter 3). Another possible simplification is the restriction of the directions where the volumetric integral is calculated (Chapter 4). We can also choose to solve the entire volumetric equation (Chapter 5).

### 1.2 Solution of the rendering equation

As stated, the image synthesis process can be traced back to solving the rendering equation. Both the original rendering equation and the volumetric rendering equation are Fredholm type integral equations of the following operator form

$$L = E + TL$$

where $E$ is the direct effect of emission and $T$ is a linear integral operator. Using the expansion as suggested by the Neumann series approach [SK08], let us substitute the right side’s unknown function by the complete right side recursively:

$$L = E + TL = E + TE + TL = \sum_{n=0}^{\infty} T^n E,$$

which exists if light transport operator $T$ is a contraction. As the terms of this Neumann series involve the integral operator more and more times, the Neumann series is a sequence of ever increasing dimensional integrals. These high dimensional integrals are evaluated by numerical quadratures [DM85]. The classical quadrature rules become problematic in higher dimensions as the required function evaluations to obtain a prescribed precision grows exponentially with the dimension. In order to avoid the dimensional explosion, the Monte Carlo integration method [Se66] can be applied to approximate the integrals, since Monte Carlo quadrature requires the same number of samples to keep the variance below a threshold independently of the dimension [Kel98].
1.2.1 Global solution using numerical quadrature

Monte Carlo quadrature is a numerical integration technique that relies on repeated random sampling of the domain to obtain the numerical result. The main idea behind the Monte Carlo quadrature is to transform the integral to an expected value. This expected value can be estimated by the average of random samples.

Let us consider the integral of a function \( f(u_1, \ldots, u_D) \), depending on \( D \) variables \( u_1, \ldots, u_D \) over the unit cube integration domain. If \( f \) is square-integrable, and the function can be evaluated in every point of the integration domain \( (f(x) = f(u_1, \ldots, u_D)) \), then the Monte Carlo estimate of the integral

\[
I = \int f(x)dx = \int f(u_1, \cdots, u_D)d^D u,
\]

is given by

\[
\hat{I} = \frac{1}{N} \sum_{n=1}^{N} f(x_n)
\]

where \( N \) is the number of random samples. The law of large numbers ensures that the Monte Carlo estimate converges to the true value of the integral:

\[
\lim_{N \to \infty} \frac{1}{N} \sum_{n=1}^{N} f(x_n) = I.
\]

As this method relies on random sampling of the integration domain, the exact solution can be calculated only with infinite set of samples. With a finite set of samples the estimate is different from the actual solution. In order to discuss the error estimate for finite set of samples \( (N) \), we first introduce the variance \( \sigma^2(f) \) of the function \( f(x) \):

\[
\sigma^2(f) = \int (f(x) - I)^2 dx.
\]

We can then show that

\[
\int dx_1 \cdots \int \left( \frac{1}{N} \sum_{n=1}^{N} f(x_n) - I \right)^2 dx_N = \frac{\sigma^2(f)}{N}.
\]

This can be interpreted as the mean of error is on the average \( \sigma(f)/\sqrt{N} \), and scales like \( 1/\sqrt{N} \) independent of the dimension of the integral. As this convergence is relatively slow, there are several techniques to reduce the variance of the estimation [SK08, Ciska02, Cso05, VG95, SK99a]. In this dissertation, we focus on importance sampling as a primary means of variance reduction.

1.2.2 Local illumination model

The modern graphics hardware is a parallel machine, which has a special architecture to accelerate the image synthesis process. The graphics processing unit is a hardware realization of the incremental pipeline that uses a drastic approach to approximate the incoming radiance [SK99b].

The base of the simplification is the approximation of the incoming radiance at a surface point by a known term \( L^{in}(\vec{x}, \vec{ω}') \), thus the rendering equation becomes:

\[
L(\vec{x}, \vec{ω}) = L^c(\vec{x}, \vec{ω}) + \int_{Ω'} L^{in}(\vec{x}, \vec{ω}') f_r(\vec{ω}', \vec{x}, \vec{ω}) \cos^+ \theta'_x d\omega'.
\] (1.3)
This form of the equation assumes that the change of radiance in a given surface point depends only on local properties. This assumption enables a broad range of simplification during the image synthesis process and is feasible for real-time applications.

As visible surface points can be searched from the camera and the incoming radiance is usually approximated by using abstract light sources which provide illumination from one direction, the rendering process can calculate the radiance in each surface point independently. Since this equation does not contain the geometric relation between surface points, the primary visibility should be computed with special care. To determine what is visible from the camera, the graphics accelerator hardware uses \textit{rasterization} and the so called \textit{z-buffer algorithm}. As the visibility calculation is decoupled and the radiance depends only on local properties, this method can be highly parallelized.

Since this local illumination model ignores multiple light bounces, the result will be an under-estimation of the rendering equation. An important drawback of this estimation is that surfaces that are not directly visible from light sources will be completely dark. A usual way to restore the missing energy is adding a special light source to the scene. This special light source cause a constant light intensity for every direction in every point. The reflected radiance of this \textit{ambient light} is depends on the probability that a photon arriving at a surface point from the view direction is reflected by the surface. This probability is called \textit{albedo}, which is defined as a ratio of reflected radiance from the surface to incident radiance upon it. According to the rendering equation this can be expressed as

\[
a(\bar{x}, \bar{\omega}) = \int_{\Omega'} f_r(\bar{\omega}', \bar{x}, \bar{\omega}) \cos^+ \theta'_e d\omega',
\]

where \(a\) is the \textit{albedo} and the reflected radiance at surface point \(\bar{x}\) to a given viewing direction \(\bar{\omega}\) is

\[
L(\bar{x}, \bar{\omega}) = \int_{\Omega'} L^a f_r(\bar{\omega}', \bar{x}, \bar{\omega}) \cos^+ \theta'_e d\omega',
\]

where \(L^a\) is the intensity of the ambient lighting. Although the global energy state can be conserved with this approach, ignoring the geometry properties during radiance calculation results in an apparent loss in visual quality. Fortunately the modern GPUs — with various programmable pipeline stages — enable the transition towards more complex approaches [LSK10].

\section{The modern rendering pipeline}

The rendering pipeline is controlled by the CPU, usually through a general 3D API. The most popular 3D APIs are OpenGL and Direct3D, which are similar in many fundamental manner.

The functionality of the pipeline is exposed through \textit{render states} that describe the applied models of the render stages. The actual workload is defined through \textit{render calls} that describe the input of the pipeline. The output of image synthesis process contains the final image defined by a color buffer and may contain various supplemental information (e.g. depth information, user defined data, etc).

The input of the rendering pipeline describes the virtual world by a bunch of objects. Usually every object is defined in its own coordinate system (\textit{model space}), and then transformed into the coordinate system of the virtual world (\textit{world space}). Although quite a few mathematical methods exist to describe the virtual objects, the most commonly used are \textit{triangle meshes} extended by point and line primitives. These primitives are described by the \textit{vertex position}, \textit{normal vectors}, \textit{vertex colors} and \textit{texture coordinates}. 

The modern rendering pipeline consists of several data manipulation stages, they work together like a data flow machine:

- **Input assembler**: The purpose of the first stage is to read primitive description from the user-defined buffers and assemble them into primitives. This stage can assign various hardware dependent data (e.g. primitive id, instance id, vertex id) to the primitives to help the later stages work more efficiently. Once the input assembler processed a primitive, it is delivered to **vertex shader** stage.

- **Vertex shader**: This stage processes vertices from the previous stage, performing per-vertex operations like transformations, skinning, morphing or per-vertex lighting. Vertex shader operates on a single input vertex and produce a single output vertex. The main goal of this stage is the preprocessing of the input world description. To enable later stages to perform shading, the application defined vertex attributes (e.g. position, normal) have to be transformed into *world space* coordinate system, where the camera and light sources are defined. These transformations define where the objects are in the virtual world. The *camera transformation* modify the vertex parameters according to the camera parameters (e.g. camera position, direction) and the *perspective projection* distorts the scene to make view rays meeting in the camera parallel, effectively imitating a physical camera by the abstract pinhole camera. Finally applying the *viewport transformation* the input primitives take place where they must be drawn on the screen.

- **Tessellator**: The recent addition of **tessellator** stage enables an efficient decomposition of the input primitives. This stage incorporates three sub-stages, such as **tessellation control**, **tessellation evaluation**, and **output merging**.
 CHAPTER 1. INTRODUCTION

fixed-function tessellator and tessellation evaluation. The tessellation control stage is used to read an input patch and emit a parameterized output patch, with custom attributes to the hardware tessellator. The most important attribute is the tessellation level, which is used to control the number of subdivisions performed by the tessellation primitive generator. The tessellation evaluation stage performs post-processing on the generated primitives and calculates their final position and other attributes of the vertices.

• Geometry shader: After vertices are processed, they are arranged into renderable primitives. The geometry shader stage operates on a single primitive and outputs one or more output primitives, all of the same type. At the end of this stage the input primitives are discarded and the emitted new primitives are then processed like equivalent primitives specified by the application.

• Rasterizer: For further processing the rasterization stage converts the primitives’ vector information into a raster image for the purpose of displaying the 3D information on a 2D display. During rasterization each primitive is clipped to the view frustum and mapped into a 2D coordinate system (screen-space fragment coordinates) to identify where they should be drawn in the output buffer. Then the per-vertex primitive attributes are interpolated linearly across the primitive providing per-fragment attributes.

• Fragment shader: The fragments generated by the rasterizer contains all information required by the final fragment color calculation, thus can be processed independently. This stage enables rich shading techniques such as per-pixel lighting or post-processing.

• Output merging: The final stage of the rendering pipeline is responsible for arranging the fragments into the output buffer. During this stage not only the fragments color may used, but additional information, such as the fragments depth or stencil masks. With this information the merging stage is able to reconstruct the coverage relation between the fragments (z-buffer). As a result, the image of primitives will occlude each other properly. In case of transparent primitives the merging stage uses blending utilizing the transparency information of the fragments. The transparency factor is usually incorporated into the fragments color as a fourth component \((r, g, b, \alpha)\). During the blending the processed fragments color is combined with the actual color stored in the output buffer with a predefined blending operation.

The rendering pipeline exposes many of the hardware capabilities through programmable stages, where the applied algorithm can be described through special programs (e.g. shader programs). These stages are the vertex shader, tessellator control and evaluator, geometry shader and fragment shader. These programs can be written in high-level shading languages, which extend the capabilities of the applied 3D API. The most popular shading languages are GLSL for OpenGL and HLSL for Direct3D. The shading languages define frequently used mathematical functions and various texture sampling methods for common rendering problems.

To fully utilize the highly parallel rendering pipeline, we have to reconsider the applied algorithms. Instead of direct evaluation of the rendering equation (e.g. ray-tracing techniques) a better approach is to extend the local illumination model. An obvious way to separate the geometry processing and visibility calculations from the lighting calculations. Quite a few classic illumination algorithm can be rephrased into screen-space easily. The screen space techniques work on the sampled representation of the scene, where the geometry and shading attributes are sampled at discrete points (e.g. at the pixels of the final image) and this representation contains
only the visible parts of the scene. The crucial part of the screen space techniques is the efficient handling of the sampled information.

1.4 Objectives of this dissertation

The first main part of this thesis work introduces an efficient method of the texture sampling which is commonly used in various screen space post-processing methods. This texture filtering method is built upon importance sampling and also exploits the bi-linear filtering hardware component of the modern GPUs.

The second part proposes an efficient approximation of the volumetric single scattering problem. The proposed algorithm can be applied as a post-processing effect in real-time.

The third part proposes an importance sampling based method to construct a non-uniform sample distribution for containment test based screen-space ambient occlusion calculation. We propose a new volumetric integration formula for occlusion calculations, which efficiently reduces the noise over the containment test based methods. This local ambient illumination model is further extended to take into account the material properties (albedo) of the scene.

The fourth main chapter introduces an efficient way to calculate physically correct multiple scattering in participating media.
Chapter 2

Efficient sampling for screen-space algorithms

Texture filtering is a critical part in many rendering and post-processing algorithms. In case of typical filter based approaches, the applied shader program samples not only the shaded point but its surrounding area too. Thus, if the sampling is done naively, the shader needs to access the texture memory proportionally with the applied filter kernel size. On the modern graphics hardware the texture fetching operation is at least an order of magnitude slower than the arithmetic operations, so it is crucial to minimize the number of samples for real-time rendering.

2.1 Previous work

Many rendering and post-processing algorithms are equivalent to the evaluation of spatial integrals \cite{SP92, Ber10, Pra91}. The general form of a two-dimensional image filter is:

\[ L'(X, Y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} L(X - x, Y - y)w(X, Y, x, y)dydx, \]

where \( L'(X, Y) \) is the filtered value at pixel \( X, Y \), \( L(X, Y) \) is the original image, and \( w(X, Y, x, y) \) is the filter kernel for this pixel. If the same filter kernel is used for all pixels, i.e. when kernel \( w \) is independent of pixel coordinates \( X, Y \), then the filter is called spatial-invariant. For example, the spatial invariant two-dimensional Gaussian filter of variance \( \sigma^2 \) has the following kernel:

\[ w(x, y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2 + y^2}{2\sigma^2}}. \]

The square root of the variance is called the standard deviation and is denoted by \( \sigma \). The integrals of the filter are usually approximated by finite sums:

\[ L'(X, Y) \approx \sum_{i=-N/2}^{i=N/2} \sum_{j=-N/2}^{j=N/2} L(X - i, Y - j)w(i, j). \]

This discrete integral approximation requires the evaluation of \( N^2 \) kernel values, multiplications, and additions, which is rather costly when repeated for every pixel of the screen.

The computation cost can be reduced for spatial-invariant separable filters. In case of separable filters the two-variate filter kernel can be expressed in a product form:

\[ w(x, y) = w_x(x) \cdot w_y(y). \]
For spatial-invariant separable filters, the double integral can be computed in two passes. The first pass results in the following 2D function:

\[ L_h(X, Y) \approx \sum_{i = -N/2}^{i = N/2} L(X - i, Y)w_x(i). \]

Then the resulting image is filtered again with a similar one-dimensional filter, but in the vertical direction:

\[ L'(X, Y) \approx \sum_{i = -N/2}^{i = N/2} L_h(X, Y - i)w_y(i). \]

With this technique the filtering cost of a pixel can be reduced from \( N^2 \) to \( 2N \) kernel evaluations and multiplications, which may still be too high in interactive applications.

We can further reduce the number of required samples to evaluate the filter if we apply Monte Carlo integration method instead of classical numerical integration methods. The error of Monte Carlo integration is independent of the number of dimensions, and scales like \( 1/\sqrt{N} \), where \( N \) is the number of samples taken. This convergence rate is quite slow for real-time applications, where the texture filtering problem is only a small part of the required work. Fortunately there are several techniques to improve the efficiency of Monte Carlo integration.

If we have an estimation of the behavior of the integrated function, we can use this knowledge to refine the sample points where the integral is evaluated.

### 2.1.1 Importance sampling

A widely used variance reduction scheme is *importance sampling*. The main idea behind importance sampling is that certain values of the random distribution used in Monte Carlo integration have more impact on the estimation than others [SKS09]. If we can focus the sample points to these regions, we get lower variance with the same number of samples. Mathematically, importance sampling is a change of integration variables:

\[
\int f(x)dx = \int \frac{f(x)}{p(x)}p(x)dx = \int \frac{f(x)}{p(x)}dP(x),
\]

where

\[
p(x) = \frac{\delta^D}{\delta x_1...\delta x_D}P(x).
\]

If \( p(x) \) is a positive-valued function and

\[
\int p(x)dx = 1,
\]

then \( p(x) \) can be interpreted as a probability density function. If we draw our random samples according to probability distribution \( P(x) \), then we may estimate the integral as

\[
\hat{I} = \frac{1}{N} \sum_{n=1}^{N} \frac{f(x_n)}{p(x_n)}.
\]

The variance of the estimate is given by

\[
\sigma^2 \left( \frac{f}{p} \right) = \frac{1}{N} \sum_{n=1}^{N} \left( \frac{f(x_n)}{p(x_n)} \right)^2 - \hat{I}^2.
\]
It is advantageous to choose \( p(x) \) as close in shape to \( f(x) \) as possible, to reduce the variance of the estimation. A disadvantage of this method is that \( p(x) \) function should be chosen carefully. If \( p \) approaches zero quickly or goes to zero where \( f \) is not zero, then \( \sigma(f/p) \) may be infinite.

In the following sections we propose an efficient filtering algorithm that minimizes the number of texture fetches. The algorithm is based on importance sampling and also exploits the bi-linear filtering hardware.

### 2.2 Texture filtering with importance sampling

In order to reduce the number of samples, instead of sampling the integration domain regularly, **importance sampling** takes more samples where the filter kernel is large. Let us consider the one-dimensional convolution,

\[
L'(X) = \int_{-\infty}^{\infty} L(X - x)w(x)dx,
\]

and find integral \( \tau(x) \) of the kernel and also its inverse \( x(\tau) \) so that the following conditions hold:

\[
\frac{d\tau}{dx} = w(x),
\]

that is,

\[
\tau(x) = \int_{-\infty}^{x} w(t)dt.
\]

If kernel \( w(t) \) is a probability density, that is, it is non-negative and integrates to 1, then \( \tau(x) \) is non-decreasing, \( \tau(-\infty) = 0 \), and \( \tau(\infty) = 1 \). In fact, \( \tau(x) \) is the cumulative distribution function of the probability density.

If filter kernel \( w \) is known, then \( x(\tau) \) can be computed and inverted off-line for sufficient number of uniformly distributed sample points. Substituting the \( x(\tau) \) function into the filtering integral we obtain

\[
L'(X) = \int_{-\infty}^{\infty} L(X - x)w(x)dx = \int_{-\infty}^{\infty} L(X - x)\frac{d\tau}{dx}dx = \int_{\tau(-\infty)}^{\tau(\infty)} L(X - x(\tau))d\tau = \int_{0}^{1} L(X - x(\tau))d\tau.
\]

Approximating the transformed integral taking uniformly distributed samples in \( \tau \) corresponds to a quadrature of the original integral taking \( M \) non-uniform samples in \( x \). In one-dimension we compute \( x(\tau) \) for \( \tau = 1/(2M), 3/(2M), \ldots, (2M - 1)/(2M) \) as

\[
L'(X) = \int_{0}^{1} L(X - x(\tau))d\tau \approx \frac{1}{M} \sum_{i=1}^{M} L \left( X - x \left( \frac{2i - 1}{2M} \right) \right).
\]

This way we take samples densely where the filter kernel is large and fetch samples less often farther away, but do not apply weighting (Figure 2.1).
The non-uniform sample positions are not necessarily on the grid, but may also be in between the texel centers. Such samples can be obtained assuming the original function to be piece-wise linear and exploiting the texture filtering hardware to provide us with these interpolated values at no additional cost [SH05].

Note that non-uniform sampling allows us to use a smaller number of samples than uniform sampling ($M < N$) while providing the same accuracy. Non-uniform sampling does not access every texel in the neighborhood since far from the center of the filter kernel the weighting would eliminate the contribution anyway, so taking dense samples far from the center would be waste of time.

The implementation of this approach is quite straightforward. Function $x(\tau)$ is computed by integrating the filter kernel and inverting the integral. For the Gaussian filter, $\tau(x)$ is the cumulative probability distribution function of the normal distribution, i.e. the famous $\Phi$-function, $\tau(x) = \Phi(x/\sigma)$ [Enc13a]. Values of its inverse can be hardwired into the shader. For example, if $M = 5$, we need the following pre-computed values:

\[
\begin{align*}
x(1/10) &= -1.282, & x(3/10) &= -0.524, & x(5/10) &= 0, & x(7/10) &= 0.524, & x(9/10) &= 1.282.
\end{align*}
\]

The fragment shader implementing this idea along one dimension is shown below (Listing 2.1). The shader gets the texture coordinates of the current fragment in Tex, and filters the input image stored in texture InputImage. The horizontal resolution of the image is hres. The standard deviation of the Gaussian filter is passed in global variable sigma.

---

**Listing 2.1: 1D Gaussian filter**

```c
uniform sampler2D InputImage;
uniform float sigma;
uniform float hres;

in vec2 Tex;
out vec4 Color;

void FilterImportancePS() {
    vec2 du1 = vec2(0.524/hres * sigma, 0);
    vec2 du2 = vec2(1.282/hres * sigma, 0);
```
In one dimension, importance sampling takes an optimally uniform series \((1/(2M), 3/(2M), \ldots, (2M - 1)/(2M))\) in the unit interval, and transforms its elements with the inverse of the cumulative distribution \((\tau^{-1})\) to obtain the non-uniform set of offsets for sampling. In two or higher dimensions the same concept is applied but unfortunately we do not have the optimally uniform distribution of points in a unit square or in higher dimensional unit cubes. Regular grids, which repeat the same one-dimensional series independently along the coordinate axes, get very non-uniform in higher dimensions (we have large gaps between the rows and columns). Better options are the low-discrepancy series, such as the Halton or Hammersley series [Nie92], or we can also produce our own uniform sample set with an iterative relaxation algorithm. An initial set of points are put into an (arbitrary dimensional) cube, and we assume that points repel each other. Moving the points in the direction of the resulting force and repeating this step iteratively, a uniform distribution, the Poisson disc distribution can be obtained.

In the following sections we present applications for the discussed importance based filtering scheme. Tone mapping and glow (also called bloom) require spatial-invariant Gaussian filtering, which can be executed as a pass of one-variate horizontal, then vertical filtering. Then depth of field is attacked, where the filter size is not spatial-invariant. Thus, the two dimensions cannot be simply separated, but we apply the two-dimensional version of the discussed importance sampling scheme. This filtering scheme can be applied to three-dimensional problems too, an example of low-variance computation of screen space ambient occlusion is in Chapter 4.

### 2.3 Tone mapping with bloom effect

Off the shelf monitors can produce light intensity just in a limited, low dynamic range (LDR). Therefore the values written into the frame buffer are unsigned bytes in the range of \([0x00, 0x0f]\), representing values in \([0,1]\), where 1 corresponds to the maximum intensity of the monitor. However, realistic rendering often results in high dynamic range (HDR) luminance values that are not restricted to the range of the monitors. The mapping of HDR image values to displayable LDR values is called tone mapping [RWPD06, MNP99]. The conversion is based on the luminance the human eye is adapted to. Assuming that our view spans over the image, the adaptation luminance will be the average luminance of the whole image. The main steps of tone mapping are as follows.

The luminance value of every pixel is obtained with the standard CIE XYZ transform [WS82, Sch96], for example when the tristimulus in a pixel represented with sRGB components in the range \([0,1]\):

\[
C_{\text{linear}} = \begin{cases} 
\frac{C_{sRGB}}{12.92} & C_{sRGB} \leq 0.04045 \\
\left( \frac{C_{sRGB}}{1+\alpha} \right)^{2.4} & C_{sRGB} > 0.04045
\end{cases}
\]

where \(C\) is \(R, G\) or \(B\), and \(\alpha = 0.055\). Followed by a matrix multiplication of the linear values to get \(XYZ\)
where $Y$ is the luminance of the pixel. These values are averaged to get adaptation luminance $Y'$. Having adaptation luminance $Y'$, pixel luminance values $Y$ are first mapped to relative luminance $Y_r$:

$$Y_r = \frac{\alpha Y}{Y'}$$

where $\alpha$ is a user defined constant of the mapping, which is called the key [KMS05]. The relative luminance values are then mapped to displayable pixel intensities using the following function:

$$D = \frac{Y_r(1 + Y_r/Y_w^2)}{1 + Y_r},$$

where $Y_w$ is another user defined value representing the relative luminance that is expected to be mapped onto the maximum monitor intensity. Colors of relative luminance higher than $Y_w$ will burn out.

Having the display luminance, the original $[R,G,B]$ data is scaled with it to provide color information $[r,g,b]$: $[r,g,b] = [R,G,B]D$.

The user defined key value controls where the average luminance is mapped in the $[0,1]$ region [Rei02].

Glow or bloom occurs when a very bright object causes the neighboring pixels to be brighter than they would be normally. It is caused by scattering in the lens and other parts of the eye, giving a glow around the light and dimming contrast elsewhere. To produce glow, first we distinguish pixels where glowing parts are seen from the rest by selecting pixels where the luminance is significantly higher than the average. The HDR color of glowing parts is distributed to the pixels nearby, which is a Gaussian blur, which results in a glow map. The glow map is added to the HDR image before tone mapping.

### 2.3.1 Implementation of bloom

Note that in this process we use Gaussian filtering during the computation of the glow map. The variance is constant, thus this is a spatial-invariant separable filter that can be realized by two 1D filtering steps, which can exploit the proposed importance sampling scheme, which fetches samples non-uniformly, but with a distribution specified by the cumulative distribution of the Gaussian (Listing 2.1).

### 2.4 Depth of field

Computer graphics algorithms usually apply the pinhole camera model, while real cameras have lenses of finite dimensions, which let through rays coming from different directions. As a result, parts of the scene are sharp only if they are located at a specific focal distance.

According to geometric optics (see Figure 2.2), if the focal length of a lens is $f$ and an object point is at distance $t$ from the lens, then the corresponding image point will be in sharp
focus on an image plane at distance $k$ behind the lens, where $f$, $t$, and $k$ satisfy the following equation [Abr97]:

$$\frac{1}{f} = \frac{1}{k} + \frac{1}{t}.$$ 

If the image plane is not at proper distance $k$ from the lens of diameter $D$, but at distance $d$, then the object point is mapped not onto a point but onto a circle of radius $r$ [SKe95]:

$$r = \frac{|k - d| D}{k}.$$ 

This circle is called the circle of confusion corresponding to the given object point. It expresses that the color of the object point affects the color of not only a single pixel but all pixels falling into the circle.

![Figure 2.2: The computation of the circle of confusion.](image)

A given camera setting can be specified by the focal distance $P$, which is the distance of those objects from the lens, which appear in sharp focus on the image plane (note that the focal distance must not be confused with the focal length). The focal distance and the distance of the image plane also satisfy the basic relation of the geometric optics:

$$\frac{1}{f} = \frac{1}{d} + \frac{1}{P}.$$ 

Putting the three equations together, we obtain the following formula for the radius of the circle of confusion:

$$r = \left| \frac{1}{t} - \frac{1}{P} \right| D d.$$ 

According to this formula the radius is proportional to the difference of the reciprocals of the object distance and of the focal distance. Since the projective transform and the homogeneous division translates camera space depth $z$ to screen space depth $Z$, as $Z = a + b/z$ where $a$ and $b$ depend on the front and back clipping space distances, the radius of the circle of confusion is just proportional to the difference of the object’s depth coordinate and the focal distance, interpreting them in screen space:

$$r = |Z - P'| S,$$

where $Z$ is the screen space depth of the point, $P'$ is the distance of the focal plane transformed to screen space, and $S = D db/2$ is the camera’s scaling parameter composed from the size of the lens, the distance of the image plane, and the front/back clipping plane distances.
This theory of depth of field describes the phenomenon from the point of view of object points projected onto the view plane. The power reflected by a point is distributed in a circle, usually non-uniformly, producing higher power density in the center. The affected region grows with the area of the circle, but the contribution to a particular point decreases proportionally. However, in rendering we take an opposite approach and should consider the phenomenon from the point of view of the pixels of the image. If we can assume that the depth is similar in the neighborhood to the depth of the current fragment, the colors of the neighborhood pixels should be blended with the current color, using a filter that decreases with the distance to a negligible value when the distance is greater than the radius of the circle of confusion. A good candidate for such filters is the Gaussian filter setting its standard deviation to the radius of the circle of confusion.

2.4.1 Implementation of depth of field

Depth of field simulation consists of two phases. In the first pass, the scene is rendered into textures of color and depth values. In the second pass, the final image is computed from the prepared textures with blurring. Blurring is performed using a variable-sized Gaussian filter since the variance (i.e. the circle of confusion) changes from pixel to pixel, making the filtering process not spatially invariant. Note that this prohibits us from replacing the 2D filtering with two 1D filtering phases. So we have to implement the method as a real 2D filtering scheme, which makes importance sampling even more indispensable. The depth of field shader should use 2D offset positions stored in a pre-computed array, which is generated by the previously discussed importance sampling method. We have two options. On the one hand, we can exploit the separability of the 2D Gaussian filter and repeat the method proposed for 1D filtering to independently find the $x$ and $y$ coordinates of the sample locations. The other possibility is to replace Cartesian coordinates $r, \phi$ in the filtering integral:

$$
\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} L(X-x, Y-y) \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}} \, dx \, dy = \int_{\phi=0}^{2\pi} \int_{\phi=0}^{2\pi} L(X-r \cos \phi, Y-r \sin \phi) \frac{1}{2\pi\sigma^2} e^{\frac{r^2}{2\sigma^2}} r \, dr \, d\phi.
$$

This filter can be separated to a filter for angles $\phi$, where the filter kernel is constant $1/2\pi$ in the filtering domain $[0, 2\pi]$, and to filter for radius $r$, where the filtering kernel is

$$w(r) = \frac{1}{\sigma^2} e^{-\frac{r^2}{2\sigma^2}} r.
$$

Since the filtering kernel of polar angle $\phi$ is constant, such samples are taken uniformly in the angular domain $[0, 2\pi]$. Considering distance $r$, the cumulative distribution

$$\tau(r) = \int_0^r \frac{1}{\sigma^2} e^{-\frac{t^2}{2\sigma^2}} \, dt = 1 - e^{-\frac{r^2}{2\sigma^2}}
$$

can be analytically inverted:

$$r = \sigma \sqrt{-2 \log(1 - \tau)}.
$$

Despite the analytical expression, it is still worth pre-computing the sample location offsets and passing them to the shader as a global variable ($\text{filterTapSi}$ in the program below). This array is constructed by taking $\text{NUM\_DOF\_TAPS}$ number of uniformly distributed $(r, \phi)$ samples in the unit square, transforming them as $(\sqrt{-2 \log(1 - \tau)}, 2\pi \nu)$ to $(r, \phi)$ polar coordinate pairs, and finally obtaining the offset positions in Cartesian coordinates. The offset positions will be
multiplied by the radius of the circle of confusion ($\sigma$) in the shader since this parameter varies from fragment to fragment.

The fragment shader gets the location of the current fragment and also its depth value in screen coordinates, computes the circle of confusion radius $\text{sizeCoC}$, and scales the sample offset with this value. The original color image is fetched at the offset positions and the colors are added up without any additional weighting.

Listing 2.2: Example fragment shader of depth of field effect

```glsl
uniform sampler2D InputImage;
uniform sampler2D DepthMapSampler;
in vec2 Tex;
out vec4 Color;

void main()
{
  vec4 colorSum = texture(InputImage, Tex); // Center sample
  float depth = texture(DepthMapSampler, Tex).a; // Current depth
  float sizeCoC = abs(depth - FOCAL_DISTANCE) * DOF_SCALE;
  for (int i = 0; i < NUM_DOF_TAPS; i++) { // Filter
    vec2 tapCoord = Tex + filterTaps[i].xy * sizeCoC;
    colorSum += texture(InputImage, tapCoord);
  }
  Color = colorSum / (NUM_DOF_TAPS + 1); // Normalize
}
```

Note that this implementation assumes that the depth values are similar in the neighborhood as the depth of the current pixel. If this assumption fails, artifacts may show up, which can be reduced by skipping those candidate fragments where the depth difference is too large.

### 2.5 Results

In order to demonstrate the results, we implemented tone-mapping, glow and depth of field effects as post-processing filters in an open source game engine called *Ogre3D* [Tea12].

![Figure 2.3: The Moria scene with tone mapping and glow.](image)

Figure 2.3 shows a game scene rendered with tone-mapping and glow. Figure 2.4 demonstrates the depth of field effect.
2.6 Conclusion

This chapter proposed an efficient texture filtering algorithm for post-processing effects. The new filtering scheme is built upon importance sampling and exploits the interpolation capability of modern graphics hardware. We demonstrated the applicability of the proposed method on typical screen-space algorithms like tone-mapping, bloom and depth of field.
Chapter 3

Screen space approximation of volumetric single scattering

Volumetric scattering can greatly improve the overall realism of rendered images. The main idea behind these simulations is that the space between the visible surface and the camera is filled with tiny particles that absorb and scatter light photons colliding with them. The exact computation of scattering is hard as taking into account multiple light bounces between particles yields to a strong dependence between incoming and outgoing radiance at a particle. However in many cases multiple scattering does not contribute too much to the overall appearance of the final image, and computing single scattering (one light bounce only) can capture the main characteristics of light transport.

3.1 Previous work

Volumetric effects are well known in the context of off-line rendering. Typical renderers use ray marching [KH84, NN94] for illumination accumulation and some kind of Monte Carlo quadrature [Rus94] or volumetric photon maps [JZJ08, JC98] to account for scattering. These methods require expensive computations, which restricts their application in interactive systems. As the performance of graphics processors (GPU) grows, researchers have proposed new methods for interactive volumetric lighting effects. These techniques commonly fall into two categories, they can be based either on shadow volumes or on ray marching.

Methods using shadow volumes [BAM06, Jam03] identify the shadowed parts of the eye rays via silhouette polygons extruded from the light source. These shadow volumes are rendered in back to front order, which has an additional sorting cost.

Ray marching methods accumulate scattering along eye rays by advancing on the ray in small steps and updating the accumulated parameters iteratively. In real-time rendering a slice based volume rendering technique can be used to provide efficient per pixel ray marching where shadowing of solid objects can be computed with shadow mapping [DYN02, Mit04]. There also exist methods that combine shadow volumes and ray marching to speed up the classical ray marching algorithm [WYC08].

Finally, it is also possible to use an image-processing radial blur operator to visualize light-shafts. These techniques work only if the light source is visible from the camera [K07].

In the following sections we propose a new single scattering approximation model, which can be applied as post-processing in real-time.
3.2 Volumetric lighting in participating media

Let us consider a ray of equation \( \vec{x}(s) = \vec{x}_0 + \vec{\omega} s \), defined by origin \( \vec{x}_0 \), direction \( \vec{\omega} \), and ray parameter \( s \). The change of radiance \( L(\vec{x}, \vec{\omega}) \) along this ray in non-emissive homogeneous participating media is expressed by the \textit{volumetric rendering equation} [SKS08]:

\[
\frac{dL(\vec{x}(s), \vec{\omega})}{ds} = -\sigma_t L(\vec{x}(s), \vec{\omega}) + \sigma_s \int_{\Omega'} L(\vec{x}(s), \vec{\omega}') P(\vec{\omega}', \vec{\omega}) d\vec{\omega}'.
\]

where \( \sigma_t \) is the \textit{attenuation} describing the probability of collision in a unit distance, \( \sigma_s \) is the \textit{scattering coefficient}, and \( P(\vec{\omega}', \vec{\omega}) \) is the \textit{phase function} describing the probability density of the scattering direction.

This integro-differential equation is difficult to solve since the unknown radiance appears in derivative, normal, and integrated forms. Such equations can be solved by finite element Monte Carlo methods [SK08, J8], but they are far too slow for real-time applications. Thus, we completely ignore multiple scattering and approximate the in-scattering integral assuming single scattering only. Let us denote the in-scattering term by \( L_i \) in the following way:

\[
\sigma_s \int_{\Omega'} L(\vec{x}(s), \vec{\omega}') P(\vec{\omega}', \vec{\omega}) d\vec{\omega}' \approx L_i(\vec{x}(s), \vec{\omega}).
\]

Due to the simplifying assumption, \( L_i \) no longer depends on unknown radiance \( L \). The resulting differential equation

\[
\frac{L(\vec{x}(s), \vec{\omega})}{ds} = -\sigma_t L(\vec{x}(s), \vec{\omega}) + L_i(s, \vec{\omega})
\]

(3.1)
can be solved analytically (the correctness of the solution can be proven by inserting it into the differential equation):

\[
L(\vec{x}(s), \vec{\omega}) = e^{-\sigma_t s} L(\vec{x}_0, \vec{\omega}) + \int_0^s L_i(\vec{x}(l), \vec{\omega}) e^{-\sigma_t (s-l)} dl.
\]

The integral on the right hand side of the equation can be approximated with a finite Riemann summation:

\[
L(\vec{x}(s), \vec{\omega}) \approx L(\vec{x}_0, \vec{\omega}) e^{-\sigma_t s} + \sum_{n=0}^N L_i(\vec{x}(l_n), \vec{\omega}) e^{-\sigma_t (s-l_n)} \Delta l,
\]

(3.2)
where the step size is \( \Delta l = s/N \), i.e. it is proportional to the length of the ray and is inversely proportional to the number of sample points.

We should note here that the consecutive samples of the summation are independent which can be exploited to reduce the number of required samples per pixel.

Using the same argument for the incident radiance showing up in the in-scattering term, we get the following formula for \( L_i \). If the scene has a single point light of power \( \Phi \), then only one \( \vec{\omega}' \) direction needs to be taken into account, thus we have:

\[
L_i(\vec{x}, \vec{\omega}) = \sigma_s \frac{\Phi}{4\pi d^2} v(\vec{x}) e^{-\sigma_t d} P(\vec{\omega}, \vec{\omega}).
\]

where \( d \) is the distance between the considered point and the light source and \( \vec{\omega} \) is the direction of the light source from the sample point. Function \( v(\vec{x}) \) indicates the visibility of the sample point from the light source. It returns zero for sample points that are in shadow and one for lit sample points (Figure 3.1).
3.3 Approximation of volumetric rendering equation

We can approximate the volumetric rendering equation with a ray marching method that iteratively evaluates equation 3.2. The algorithm executes the following steps (Figure 3.2):

1. In every pixel it determines the visible surface point and its reflected radiance that will be the boundary condition for the volume radiance.

2. It iterates along the ray from the surface to the camera making small steps. In a particular sample point on the ray
   - in-scattering term \( L_i(\vec{x}(l_n), \vec{\omega}) \) is computed,
   - absorption factor \( e^{-\tau(s-l_n)} \) from the sample point to the eye is obtained, and
   - their product is added to the accumulated radiance.

3. The accumulated radiance is stored in the pixel spearheaded by the ray.

For the calculation of the illumination term \( L_i(l_n, \vec{\omega}) \) we can use shadow mapping. Depending on the type of the light source we can generate either a normal 2D shadow map for spot lights or a cube map for point lights.
For the accumulation of the radiance while ray marching we can use the following shader code (Listing 3.1) that regularly calls shadow test function \textit{shadowMC} to get visibility indicator \( v \):

\begin{verbatim}
float radiance() {
  float L = L0 * exp(-s * sigma_t);
  for(float l = s - dl; l >= 0; l -= dl) {
    x += viewDir * dl;
    float v = shadowMC(shadowMap, x);
    float d = length(x);
    Lin = exp(-d * sigma_t) * v * Phi/4/M_PI/d/d;
    Li = Lin * sigma_s * P(x, viewDir);
    L += Li * exp(-l * sigma_t) * dl;
  }
  return L;
}
\end{verbatim}

This function initializes the radiance of the ray to the radiance of the surface at the beginning \((L0)\) multiplied by the total absorption along the ray. Then the ray is marched by making steps of size \(\Delta l\). Ray marching is executed in light’s space, that is, in a coordinate system where the light is in the origin since this makes calculation simpler. In this space the distance of point \(\vec{x}\) from the light source is \(|\vec{x}|\) and the light direction is also parallel with \(\vec{x}\). This direction is checked by shadow test. Working in light’s space requires the transformation of the viewing direction in this space. The transformed viewing direction is denoted by \(\text{viewDir}\).

Taking the view ray the sample point can be obtained by a single addition \((x)\). By calling the shadow test function, we can determine whether or not the sample point is visible from the light source, i.e. whether scattering may happen here. Then the source power \(\Phi\) is attenuated by the absorption resulting in incident radiance \(Lin\). The incident radiance is multiplied by the albedo and the phase function and the result is accumulated to the ray radiance. Complicated phase functions like the Heney-Greenstein phase function can be stored in a look up table, and fetched by a single texture read.

![Figure 3.3: Single scatter as post-processing pass. The scatter image (right) calculated from the shadow map of the light source (left) and the depth map of the virtual camera (middle)](image)

The ray marching algorithm can be implemented as a post-processing method. The necessary inputs of the method are the shadow map and a depth map taken from the camera to identify visible surface points. As these maps are usually present in the texture memory, the algorithm requires no additional special rendering passes. If multiple lights should be simulated, the post
process can be run separately for each light sources, and their contribution can be added together.

Figure 3.4: The color buffer (left) and the scatter image (middle) are composited to the final image (right).

3.4 Noise reduction with interleaved sampling

In the method discussed so far, ray marching evaluates $N$ samples for every pixel, which would slow down rendering if $N$ is high. However, if the number of sample points $N$ is reduced, then smooth lightshaft boundaries are replaced by abrupt changes. One way of attacking this problem without sacrificing performance is the application of interleaved sampling [KH01].

Interleaved sampling exploits the fact that the volume lighting and the visible surfaces are similar at neighboring pixels, thus the information gained at a particular pixel can be well used in its neighbors as well.

Let us consider the sum of $N$ terms of equation 3.2. We divide the screen into $M \times M$ pixel blocks, and the $N$ terms are distributed in them in a periodic way. That is, in a particular pixel we evaluate just $N/M^2$ terms. If we assume that the samples in the $M \times M$ pixel blocks are lying approximately on the same view ray — which is reasonable if the visible surface points are close to each other — we can add their contributions together to obtain a solution similar to taking all $N$ samples on the ray. This requires additional post-processing passes to add the
pixel contributions in a block together. As the addition filter is separable we can use an addition once in a horizontal and once in a vertical direction instead of taking all the $M \times M$ samples in a single filter operation.

![Figure 3.6: Comparison of scatter image generated with 8 (left), 128 (middle) and 8 interleaved samples (right) per ray.](image)

### 3.5 Results

The proposed method has been implemented in OpenGL and tested on an NVidia 8800GTX GPU. The scene consists of 66000 triangles and is illuminated by a point light (Figures 3.7-3.8). Without volumetric single scattering the application renders 110 frames per second at 800 × 600 resolution. With volumetric scattering but without interleaved sampling the performance drops to 42 frames per second. However, if we enable interleaved sampling, the performance goes back to 100 frames per second again.

![Figure 3.7: Sample scene without volumetric scattering on the left side and with scattering on the right side.](image)
Figure 3.8: Sample scene without volumetric scattering on the left side and with scattering on the right side.

3.6 Conclusions

In this chapter we proposed an efficient volumetric single scattering simulation model, which can be applied as a post-processing effect in real-time. The algorithm uses a shadow mapping like method to check whether or not a point in the participating media may be directly illuminated
by the light source. Executing ray marching in light’s space makes the algorithm particularly simple. We also demonstrated the application of interleaved sampling that significantly increases the rendering speed while maintaining high image quality.
Chapter 4

Screen-space approximation of ambient lighting

Computer games and real-time systems require realistically looking images at high and reliably sustainable frame rates. The realism may be provided by an accurate simulation of the physics laws, but this approach has unpredictable and usually long computation time if artifact free results are expected. Real-time global illumination algorithms [RDGK12, SKSS08] are available, but may not scale up for larger scenes or may not fit into the game production pipeline. Thus in games and virtual reality systems, we prefer approximations that provide images that may not be physically accurate but have no noticeable artifact, and can be efficiently evaluated on the GPU. A usual simplification is the local illumination model, which computes only the direct contribution of the light sources and adds a constant ambient term for the missing indirect illumination. However, constant ambient lighting ignores the geometry of the scene, which results in plain and unrealistic images.

We need better compromises that approximate indirect lighting in a more believable way but are not significantly more difficult to compute than the local illumination model. A cheap, simple and flexible compromise is the local indirect lighting approach that examines only a finite neighborhood of the shaded point during illumination calculation.

4.1 Previous work

Nearby occlusions, accessibility, or openness of points on a surface were first examined with the objective of ambient lighting calculations in [ZIK98, IKSZ03], where the obscurances method was introduced. The obscurances method, which is also called the ambient occlusion [Lan02, PG04] computes just how “open” the scene is in the neighborhood of a point, and scales the ambient light accordingly. Obtaining their radiance, occluders can be treated as indirect light sources, so even color bleeding effects can be simulated [MSC03, C7, J6, RGS09].

Ambient occlusion and related methods approximate the look of an object illuminated by a cloudy sky of constant radiance. The constant radiance of the distant source does not mean that the incident radiance of a surface point would be constant in all directions. Due to occlusions or inter-reflections on the object, the sky radiance might be altered in the local neighborhood. In this sense, ambient occlusion plays the role of shadow computation, similarly as shadow maps handle directional or point sources [SWP10].

A physically correct approach would be too expensive computationally when dynamic scenes need to be rendered in real-time, therefore the mentioned models simplify the problem. On the
one hand, they limit the bounces of indirect light to zero, when only occlusions or shadowing is considered, or allowing at most one bounce in a simple form. On the other hand, they rely on the observation that distant parts do not significantly alter the illumination of the object, thus, when a point is shaded, only a local neighborhood is examined.

Early ambient occlusion and obscurances calculation methods were originally off-line processes, which stored the result in a texture. For a review of these off-line techniques see [MFS08] and for older GPU based approaches refer to [Kne07, KL05].

Occlusions in a finite neighborhood or accessibility of a point have been examined in other fields as well, including molecular simulation, CAD/CAM, robot path planning, etc. The information regarding how easily a surface may be touched by a spherical probe can be defined by the radius of the largest possible tangent sphere. The authors of the method of accessibility shading [Mil94] proposed this measure to simulate the patina on certain tarnished surfaces.

4.1.1 From the rendering equation to ambient occlusion and related models

For the sake of simplicity, ambient occlusion or obscurances models assume that the surfaces are diffuse and have albedo \( a(\vec{s}) \) and consequently their BRDF is \( f_r(\vec{s}) = a(\vec{s})/\pi \). According to the rendering equation, the reflected radiance \( L' \) in shaded point \( \vec{s} \) can be obtained as:

\[
L'(\vec{s}) = \int_{\Omega} L^{in}(\vec{s}, \vec{\omega}) a(\vec{s})/\pi \cos^+ \theta d\omega,
\]

where \( \Omega \) is the directional sphere, \( L^{in}(\vec{s}, \vec{\omega}) \) is the incident radiance from direction \( \vec{\omega} \) and \( \theta \) is the angle between illumination direction \( \vec{\omega} \) and the surface normal.

If no surface is seen from \( \vec{s} \) at direction \( \vec{\omega} \), then shaded point \( \vec{s} \) is said to be open in this direction, and incident radiance \( L^{in} \) is equal to ambient radiance \( L^a \). If there is an occluder nearby, then the point is called closed at this direction and the incident radiance is the radiance of the occluder point \( \vec{o} \).

Note that this means that the radiance of arbitrary far occluder points may have an effect on the shaded point. However, this does not meet our intuition and everyday experience that the effect of very far surfaces becomes negligible. This experience is due to that the real space is not empty but is filled by participating media. If the extinction coefficient is \( \sigma_t \) and the albedo is 1 in the participating media, then the radiance along a ray changes as:

\[
L^{in}(\vec{s}, \vec{\omega}) = \exp(-\sigma_t d) L'(\vec{o}) + (1 - \exp(-\sigma_t d)) L^a
\]

where \( d \) is the distance between the shaded point and the occluder point. Note that in this equation factor

\[
\mu(d) = (1 - \exp(-\sigma_t d))
\]

and its complement \( 1 - \mu(d) = \exp(-\sigma_t d) \) express the effects of the ambient lighting and of the occluder on the shaded direction, respectively. The effect of the occluder diminishes with the distance. Function \( \mu \) can be interpreted as a fuzzy measure that defines how strongly the given direction belongs to the set of open directions based on distance \( d \) of the occlusion at this direction.

The exponential function derived from the physical analogy of participating media has a significant drawback [IKSZ03]. As it is non-zero for arbitrarily large distances, very distant surfaces need to be considered that otherwise have a negligible effect. Thus, for practical fuzzy measures, we use functions that are non-negative, monotonically increasing from zero and reach 1 at finite distance \( R \) instead of only converging to 1 when the distance goes to infinity. This
allows the consideration of only those occlusions that are nearby, i.e. in the neighborhood sphere of radius $R$. The particular value of $R$ can be set by the application developer. When we increase this value, shadows due to ambient occlusions get larger and softer.

To define the fuzzy measure that increases from zero to one in $[0, R]$, we can use a simple polynomial:

$$
\mu(d) = \left( \frac{d}{R} \right)^\alpha.
$$

Using the fuzzy measure, the reflected radiance of the shaded point can be expressed in the following way:

$$
L^r(\vec{s}) = a(\vec{s}) (L^aO(\vec{s}) + I(\vec{s}))
$$

where $O(\vec{s})$ is the ambient occlusion (obscurance) of the shaded point [ZIK98, IKSZ03]:

$$
O(\vec{s}) = \frac{1}{\pi} \int_{\Omega} \mu(d) \cos^+ \theta d\omega,
$$

and $I(\vec{s})$ is the irradiance due to nearby indirect lighting:

$$
I(\vec{s}) = \frac{1}{\pi} \int_{\Omega} (1 - \mu(d))L^r(\vec{s}) \cos^+ \theta d\omega.
$$

This integral is traced back to the ambient occlusion. Replacing occcluder radiance $L^r(\vec{s})$ by the average of surface radiance values in the neighborhood of the shaded point $\tilde{L}^r(\vec{s})$, we can express the irradiance as:

$$
I(\vec{s}) \approx \frac{1}{\pi} \int_{\Omega} [1 - \mu(d)]\tilde{L}^r(\vec{s}) \cos^+ \theta d\omega = \tilde{L}^r(\vec{s})(1 - O(\vec{s})).
$$

The original definition of ambient occlusion [Lan02, PG04, KA06] can be interpreted as the simplification of the obscuration when the fuzzy measure becomes a clear separation of occlusion closer than $R$ and no occlusion closer than $R$. On the other hand, it was also proposed to approximate the solid angle and the average direction where the neighborhood is open, thus we can use environment map illumination instead of the ambient light. However, nowadays many authors include a weighting function into the ambient occlusion formula, thus ambient occlusion has become equivalent to the original obscuration model. As the term ambient occlusion has got more popular than the older obscuration, we shall name the general method as ambient occlusion.

Ambient occlusion measures the openness of a point based on the solid angle corresponding to the visible sky. Volumetric ambient occlusion [J6] is yet another characterization for the openness, which is defined as the relative volume of the unoccluded part of the tangent sphere $T$:

$$
O(\vec{s}) = \frac{\int_{T} V(\vec{p}) d\vec{p}}{|T|},
$$

where $|T|$ is the volume of the tangent sphere, which makes sure that the volumetric ambient occlusion is also in $[0, 1]$. Volumetric ambient occlusion provides very similar values than the ambient occlusion, and thus can also be used as an ambient lighting factor, if the following conditions are at least approximately satisfied [J6]:

- The radius of the tangent sphere is the half of the radius of the neighborhood sphere used in ambient occlusion.
The fuzzy membership function of ambient occlusion is $\mu(d) = (d/R)^3$.

The surface is relatively smooth, so visibility rays can intersect it at most once in the neighborhood sphere.

The advantage of volumetric ambient occlusion over solid angle based occlusion is that its low-variance estimator is easier to compute.

4.1.2 Ambient occlusion and indirect lighting on sampled geometry

Local approaches require only the consideration of the geometry in the neighborhood of the shaded point instead of the complete geometry of the scene, which is a great advantage. However, even the consideration of the neighborhood poses problems. A GPU processes elements like triangles, vertices, and fragments independently of other elements, thus geometric information on mutual occlusions should be gathered from textures.

There are many possibilities to encode geometric information in textures, from which the direct exploitation of the depth buffer has become the most popular. One important reason is that this texture need not be computed separately, but is automatically created by the GPU by the end of a rendering pass. The depth buffer was first used as a sampled representation of a part of the scene geometry in the depth buffer shadow algorithm [Wil78]. Later, this idea established a new branch of real-time global illumination algorithms, called \textit{screen-space} methods [Mit07, SA07, Kaj09].

The screen-space representation is the sampled version of the distance $h(x, y)$ from the eye as a function of pixel coordinates $x, y$. This representation is similar to the definition of \textit{height-fields} defined by $z = h(x, y)$ or to \textit{implicit surfaces} of form $f(x, y, z) = z - h(x, y) = 0$. Thus, processing of the screen-space representation can exploit algorithms developed for height-field or implicit surface rendering.

It must be emphasized, however, that the depth buffer represents only that portion of the scene geometry which is directly visible from the camera while occlusions and indirect lighting may be caused by other surface elements as well. Extensions of screen space methods consider more and more portions of the geometry to make the computation more robust and accurate. These extensions include the application of multi-view camera or storing multiple layers of depth images [BS09]. These layers can be obtained with \textit{depth peeling}, which requires multi-pass rendering [SKP98, Eve01, LWX06, Thi08, BM08] or can take advantage of the \textit{structured buffers} of Shader Model 5 class GPUs [GT10, SBK12].

The running time of screen space algorithms is independent of the scene complexity, but is strongly affected by the resolution of the screen. Thus, the performance can be improved if the depth map is down-sampled, which should be executed carefully not to destroy fine details [BS09].

Screen-space techniques assume that the height-field defined by the current content of the depth-buffer is an appropriate representation of the screen geometry, and the color buffer stores the radiance values of the represented points. In screen-space viewing rays are parallel to axis $z$, which greatly simplifies calculations. However, the transformation to this space is not angle and distance preserving (it is not even affine), thus this space is not appropriate for angle and distance computation.

If we need to compute angles and distances, we should rather work in camera-space, which means that we store camera-space $z$ values in textures and also the camera-space normal vectors of the visible points. The transformation from world-space to camera-space is angle and distance preserving since it is basically a translation and a rotation, thus these spaces are equivalent.
when distances and angles are computed. The disadvantage of camera-space is that in case of a perspective camera the viewing rays are not parallel, but intersect each other at the origin of the coordinate system.

Thus, to solve the distortion problem of screen-space but to keep the advantages of parallel rays, we usually work in camera-space but use a quasi-orthogonal approximation, which is called as weak perspective approximation in machine vision. When large-scale information is obtained, we follow the structure of the camera-space. However, when smaller neighborhoods are explored, which happens during the evaluation of the ambient occlusion integral, we assume that in this small neighborhood the viewing rays are parallel with axis $z$. This is an approximation, but is a reasonable compromise between accuracy and simplicity.

4.1.3 Occlusion calculation for height fields defined by the depth buffer

Indirect illumination computation requires those hit points that are visible from the shaded point, which can usually be obtained with ray-tracing. For height fields, it is very simple to decide whether a point is below or above the surface (Figure 4.1), but is quite difficult to analytically determine the ray-surface intersection point. For height fields and implicit surfaces, the ray tracing problem is thus usually solved by ray marching, which takes small steps along the ray and finds the step in which the surface is first crossed, i.e. the start point is above and the end point is below the surface. If the step size is too large with respect to the expected accuracy, the identified step can be subdivided or the intersection point can be refined with halving or secant methods [SKU08].

![Figure 4.1: Surfaces represented as height field (left) and by the depth buffer (right).](image)

Ray marching and point sampling

As the shaded point belongs to the set of points that are visible from the camera and the considered potential hit points are nearby, we may assume that the hit point is also visible from the camera. Two points are visible from each other if the ray originated at one of the points has zero intersection with the surfaces before it arrives in the other point. The requirement of both of them being in the visible part of the height-field, on the other hand, means that the ray intersects the surface by even number of times, i.e. zero, two, four, etc. times. If the neighborhood is small, and at most one intersection is possible, then the two criteria are similar.

Note that we may make a series of simplifying assumptions, i.e. use more and more drastic approximations:
• Assuming that the hit point is also visible from the camera and is represented in the depth map is fundamental in screen-space methods and allows us to look for the hit point in the depth map. However, this assumption fails when the hit point is out of the camera frustum or is occluded by other another surface.

• Working with ray marching instead of ray tracing and considering the sample representation instead of the original geometry allows the application of height field visibility methods, but may miss intersections when the surface exhibits high frequency variations that are either not captured by the sampled representation or skipped by ray marching steps.

• Assuming that in the local neighborhood at most one intersection can occur we can replace ray marching by testing individual points but impose further restriction on the surface variation.

![Diagram showing ray tracing and ray marching with intersection points and containment tests.](image)

Figure 4.2: Replacing ray tracing by containment tests. If a test point along a ray starting at the shaded point being in the outer region or on the region boundary is also in the outer region, then the ray either has not intersected the surface (green point) or has intersected at least two times (blue point). Supposing that the ray is short enough with respect to the radius of the neighborhood sphere and thus may intersect the surface at most once, the condition of being in the outer region is equivalent to the condition that no intersection happened.

If we take $N$ rays and each ray is marched with $M$ steps, then we have to test $N \times M$ points, which is prohibitive in most cases. The problem is that sample points which are uniformly distributed along rays are very non-uniformly fill the neighborhood sphere. Non-uniform sampling is responsible for large quadrature error unless the number of sample points is high. From another point of view, as we need only the first intersection, marching along a ray with small steps is waste of computational resources since those points that are behind the first intersection do not provide useful information. We may think that breaking from the loop of ray marching when an intersection is found could solve this problem, but it does not. The problem is that the GPU is a quasi-SIMD parallel machine, thus it prefers to execute the very same machine instruction in parallel computational threads. Thus, if threads execute different number of cycles of a loop, the performance degrades.
The solution is to take very few, i.e. a single sample along each ray. First, we assume that the neighborhood is small with respect to the local curvature of the surface, so we can suppose that a ray may intersect the surface at most once. It means that if a sample point is not in the occluded region, then the ray has not intersected the surface yet. Thus, we ignore cases when the ray intersected the surface two, four, etc. in even number of times, and while starting at the unoccluded region, it visits the unoccluded region again.

If a single sample is taken per ray, and we check whether or not this particular sample is occluded, then we can ignore the ray direction since the sample itself determines the direction, so it needs not be explicitly represented. Samples fill the volume of neighborhood sphere and their weighted sum provides the ambient occlusion estimate. The questions are how the samples should be placed for effective evaluation and how the volumetric interpretation can be exploited to obtain a low-variance estimator.

In the following sections we propose a sample distribution scheme to construct non-uniform sample locations for real-time obscurance, and a new integration scheme for the evaluation of the volumetric ambient occlusion integral. We also propose a new approximation for the ambient transfer function, which approximates the indirect lighting between diffuse surfaces.

4.2 Non-uniform sample distribution for point sampling

The obscurance [IKSZ03] model approximates the indirect lighting in point \( \bar{s} \) of the scene by,

\[
L^i(\bar{s}) \approx a(\bar{s})W(\bar{s})L^a
\]

where \( a \) is the albedo, \( L^a \) is the ambient light intensity, and \( W \) is the obscurance value of the point, which expresses how open this point is for ambient illumination. A point is open in a direction if no occluder can be found close by. The obscurance value is defined by

\[
W(\bar{s}) = \frac{1}{\pi} \int_{\Omega} \mu(d(\bar{s})) \cos^+ \theta \, d\omega,
\]

where \( \Omega \) is the directional hemisphere. In order to find an efficient method for the evaluation of obscurance, we express it as a three dimensional integral. First the fuzzy measure is expressed as

\[
\mu(d) = \int_0^d \frac{d\mu(r)}{dr} dr = \int_0^R \frac{d\mu(r)}{dr} \epsilon(d-r)dr
\]

where \( \epsilon(r) \) is the step function, which is 1 if \( r \) is not negative and zero otherwise. Substituting this integral into the obscurance formula we get

\[
W(\bar{s}) = \int_{\Omega} \int_0^R \frac{d\mu(r)}{dr} \cos^+ \theta \frac{1}{\pi} \epsilon(d-r)drd\omega
\]

Let us consider a ray of equation \( \bar{s} + \bar{\omega}r \) where shaded point is the origin, \( \bar{\omega} \) is the direction, and distance \( r \) is the ray parameter. If we assume that the ray intersects the surface at most once in the \( R \)-neighborhood, then \( \epsilon(d-r) \) is equivalent to the condition that the ray has not intersected the surface yet. If it has not intersected the surface, and other objects are far, then this condition is equivalent to the visibility of the sample point \( \bar{s} + \bar{\omega}r \), which can be checked using the content of the z-buffer [Mit07]. Note that this integral is a filtering scheme where we filter the 0, 1 values of visibility indicator \( \epsilon \) with the following kernel:

\[
W(r, \bar{\omega}) = \frac{d\mu(r)}{dr} \cos^+ \theta \frac{1}{\pi}
\]
This filter is the product of a density $d\mu(r)/dr$ of distances and density $\cos^\top\theta/\pi$ of ray directions.

To construct the non-uniform locations representing the filter kernel, we start with point set $[\tau, \upsilon, \xi]$ that is uniformly distributed in a 3D unit cube, and transform the uniform distribution to mimic the filtering kernel. As the first step of the transformation, two coordinates $[\tau, \upsilon]$ are transformed to a direction that has cosine distribution. Let us consider a unit radius sphere centered at the considered point. The sphere intersects the tangent plane $xy$ in a circle. If we take points in a unit radius circle uniformly, and then map up the point vertically in direction $z$ onto the sphere, we get samples with a cosine distribution. In order to generate uniformly distributed points in a unit radius circle, we take values $\tau, \upsilon$ and transform them linearly from $[0, 1]$ to $[-1, 1]$. The two scalars are considered as $x, y$ coordinates of a 2D point and it is checked whether the point is inside the unit radius circle. If not, this pair is rejected, and another pair is taken until we get a point that is really in the circle, and finally project it up to the unit sphere.

Having found a cosine distributed direction, distance $r$ is sampled with density $d\mu(r)/dr$. If we take the third coordinate $\xi$ of the uniform 3D point set, $r = \mu^{-1}(\xi)$ will have exactly the required density. Sample points generated this way are stored in a global array $OFFSET$. These points correspond to translations in the tangent space.

During run time, the post-processing filter computes obscurance from the depth buffer ($depthMap$) storing camera space $z$ values. The shader gets the current point’s texture coordinates ($Tex$) and its projection onto the first clipping plane ($Dir$). From the projection and the stored depth, the point is reconstructed in camera space. The shader program transforms the offsets of $OFFSET$ to camera space ($cOff$) and translates the camera space point with them obtaining $sampleCPos$ in camera space and then by projection transformation $sampleSPos$ in screen space. The visibility of the translated point is checked by projecting it onto the screen, and comparing its depth to the depth obtained in the direction of the translated point. If the sample point passes the depth test, then value 1 is added to the filtered variable, otherwise the variable is not changed. Note that to transform the offsets from tangent space to camera space we need the camera space normal vector of the current point. We can store these normals in the first three channels of the texture $depthMap$ storing depth values.

---

**Listing 4.1: Fragment shader of the post-processing filter**

```glsl
uniform sampler2D depthMap;

in vec2 Tex; // pixel coords in [0,1]
in vec4 Dir; // on front clipping plane
out vec4 Color;

void main()
{
    float depth = texture(depthMap, Tex).a; // camera space depth
    vec3 cPos = Dir.xyz * depth / Dir.z; // camera space location
    vec3 T, B, N;
    N = texture(depthMap, Tex).rgb; // camera space normal
    T = cross(N, vec3(0, 1, 0));
    B = cross(N, T);
    mat3x3 TangentToView = mat3x3(T, B, N);
    float occ = 0;
}
```
4.3 A new integration method for volumetric ambient occlusion

In order to find an efficient method for the evaluation of the ambient occlusion integral, we can also express it as a three-dimensional, i.e. a volumetric integral. First the fuzzy measure is written as the integral of its derivative:

$$
\mu(d) = \int_0^d \frac{d\mu(r)}{dr} dr.
$$

Substituting this integral into the ambient occlusion formula, we get

$$
O(\vec{s}) = \frac{1}{\pi} \int_{\Omega} \int_0^d \frac{d\mu(r)}{dr} (\vec{N}_s \cdot \vec{\omega}) dr d\omega,
$$

where $\vec{N}_s$ is the surface normal. Realizing that $r^2 dr d\omega = dV$ is a differential volume, the ambient occlusion can also be expressed as a volumetric integral

$$
O(\vec{s}) = \frac{1}{\pi} \int_S \frac{d\mu(r)}{dr} \frac{1}{r^2} (\vec{N}_s \cdot \vec{\omega}) dV,
$$

(4.6)

where $S$ contains those points of the hemisphere which are visible from the shaded point, and therefore also visible from the camera (Figure 4.4).

In our case, the occluder surface is a height field defined by the content of the depth-buffer. Thus a point $(x, y, z)$ belongs to the visible region if its $z$-coordinate is less than the value $z^*$ stored in the depth buffer for the same $(x, y)$ coordinates. We compute the volumetric integral with differential elements having $dz$ height and $dA = dx dy$ base area at the point $(x, y)$. The

```c
for(int k = 0; k < NUM_SAMPLES; k++) {
    // Transform offsets from tangent space to camera space
    vec3 c0ff = mul(OFFSET[k], xyz, TangentToView) * R;
    vec3 sampleCPos = cPos + c0ff; // pos in camera space
    
    // Compute screen coordinates
    vec4 sampleCPos = mul(vec4(sampleCPos, 1), projMatrix);
    vec2 sampleSPos = sampleCPos.xy / sampleCPos.w;
    sampleSPos = (sampleSPos + 1.0) * 0.5;

    // Read depth buffer
    float sampleDepth = texture(depthMap, sampleSPos).a;
    
    // Compare sample depth with depth buffer value
    if(sampleDepth >= sampleCPos.z) occ++;
    else if(sampleCPos.z - sampleDepth > R) occ++;
    occ = occ / NUM_SAMPLES; // Normalize
    Color = occ * tex2D(InputImage, Tex); // Compose with shaded image
}
```
Figure 4.3: Comparison of ambient occlusion without (left) and with interleaved sampling (right). The upper row evaluated with 1 sample per pixel, the middle row with 4 samples per pixel, the lower row with 8 samples per pixel.

disk \( \mathcal{C} \) of radius \( R \) and perpendicular to z-axis (Figure 4.4) as

\[
O(\vec{s}) = \frac{1}{\pi} \int_{x,y \in \mathcal{C}} \int_{z_{\min}}^{z_{\max}} \frac{d\mu(r)}{dr} \frac{1}{r^2} (\vec{N}_g \cdot \vec{\omega}) dz dy dz = \frac{1}{\pi} \int_{x,y \in \mathcal{C}} h(x,y,z_{\min}, z_{\max}) dz dy,
\]

where \( h \) is the integral over \( z \):

\[
h(x,y,z_{\min}, z_{\max}) = \int_{z_{\min}}^{z_{\max}} \frac{d\mu(r)}{dr} \frac{1}{r^2} (\vec{N}_g \cdot \vec{\omega}) dz = \int_{z_{\min}}^{z_{\max}} \frac{d\mu(r)}{dr} \frac{1}{r^2} \frac{(\vec{\sigma} - \vec{s}) \cdot \vec{N}_g}{r} dz.
\]
Recall that we are free to set the exponent of the fuzzy membership function. Classical ambient occlusion used a non-fuzzy measure, which corresponds to \( \alpha = \infty \). Mendez [MSC+05] examined several exponents and concluded that \( \alpha = 1/2 \) is a good choice. Our criterion of setting the exponent will be the ease of the evaluation of the ambient occlusion integral. This integral can be evaluated analytically if we define the fuzzy membership function as

\[
\mu(r) = \left(\frac{r}{R}\right)^\alpha
\]

with \( \alpha = 4 \). The center of the sphere is the shaded point whose coordinates are denoted by \((x_\bullet, y_\bullet, z_\bullet)\).

In this case,

\[
h(x, y, z_{\text{min}}, z_{\text{max}}) = \frac{4}{R^4} \int_{z_{\text{min}}}^{z_{\text{max}}} (\vec{\sigma} - \vec{s}) \cdot \vec{N}_s \, dz.
\]

The integral over the disk is evaluated with a numerical quadrature. The total volume of the visible part of the hemisphere is approximated by the sum of the volume of pipes. The axes of these pipes are parallel to axis \( z \). The pipes are inside the hemisphere and may be limited by the height field of the depth-buffer. To define the pipes, we sample \( n \) uniformly distributed points \((x_i, y_i)\) in the disk of radius \( R \). Thus, each pipe has the same cross section area \( \Delta A = R^2 \pi / n \).

A line crossing the \( i \)-th sample point and being parallel with the \( z \)-axis enters the sphere at

\[
z_i^{\text{min}} = z_\bullet - \sqrt{R^2 - (x_i - x_\bullet)^2 - (y_i - y_\bullet)^2},
\]

exits it at

\[
z_i^{\text{max}} = z_\bullet + \sqrt{R^2 - (x_i - x_\bullet)^2 - (y_i - y_\bullet)^2},
\]

and crosses the tangent plane of the shaded point at

\[
z_i^{\text{plane}} = \frac{N^x_{\vec{s}}(x_\bullet - x_i) + N^y_{\vec{s}} + N^z_{\vec{s}} Z_{\vec{s}}}{N^z_{\vec{s}}}.\]

The points on this line belong to the visible region when their \( z \)-coordinates are less than \( z_i^{\text{max}} = \min(z_i^{\text{exit}}, z_i^{\text{plane}}) \) and are greater than \( z_i^{\text{min}} \). The contribution of the pipes to the volumetric integral of the ambient occlusion is

\[
O(\vec{s}) = \frac{1}{\pi} \int_{x,y \in C} h(x, y, z_{\text{min}}, z_{\text{max}}) \, dx \, dy \approx \frac{R^2}{n} \sum_{i=1}^{n} h(x_i, y_i, z_i^{\text{min}}, \min(z_i^{\text{exit}}, z_i^{\text{plane}})).
\]
CHAPTER 4. APPROXIMATION OF AMBIENT LIGHTING

This quadrature is an approximation and its error decreases if new sample points are added. However, computing the formula with many sample points reduces rendering speed. Thus, we consider two techniques, including weighted uniform sampling and the already described interleaved sampling, that reduce the computation error without performance degradation.

**Weighted uniform sampling** [PS66] exploits the fact that if there is no occlusion in the neighborhood sphere, then ambient occlusion should be equal to 1. If it is not, then the difference is due to the approximation error. So we compute not only the ambient occlusion from the samples but also the estimate of the factor assuming no occlusion at all. Ignoring occlusions, this factor is

\[
1 \approx \frac{R^2}{n} \sum_{i=1}^{n} h(x_i, y_i, z^\text{min}_i, \min(z^\text{exit}_i, z^\text{plane}_i)).
\]

Dividing the formula of ambient occlusion by this approximation, we can compensate the quadrature error, thus a better estimate of ambient occlusion is

\[
O(\vec{s}) \approx \frac{\sum_{i=1}^{n} h(x_i, y_i, z^\text{min}_i, \min(z^\text{exit}_i, z^\text{plane}_i))}{\sum_{i=1}^{n} h(x_i, y_i, z^\text{min}_i, \min(z^\text{exit}_i, z^\text{plane}_i))}.
\]

### 4.4 Approximation of indirect lighting

The solution of equation 4.4 requires some estimation of the radiance \( L^r(\vec{o}) \) reflected off the occluding points, which is responsible for indirect lighting. The first approach to extend the obscurances model with local indirect lighting was the *spectral obscurances method*, which took average spectral reflectivities of the neighborhood and the whole scene into account, thus even color bleeding effects could be cheaply simulated [MSC+05]. Bunnel [Bun05] extended ambient occlusion as well to incorporate color bleeding.

#### 4.4.1 Adding local indirect lighting to ambient occlusion

The formula of local indirect lighting (equation 4.4) can be traced back to the computation of ambient occlusion. Replacing the occluder radiance \( L^r(\vec{o}) \) by average \( \bar{L}^r(\vec{o}) \) of surface radiance values in the neighborhood of the shaded point, we can express the indirect lighting as:

\[
I(\vec{s}) = \frac{1}{\pi} \int_{\Omega} (1 - \mu(\vec{d})) \bar{L}^r(\vec{o}) \cos^+ \theta \, d\omega \approx \bar{L}^r(\vec{s})(1 - O(\vec{s})).
\]

The average surface radiance can be obtained as a filtered version of the color buffer.

#### 4.4.2 Local approximation of the ambient transfer function

The ambient transfer function can be written in the following form:

\[
W(\vec{s}) = \frac{1}{\pi} \int_{\Omega} \mu \cos^+ \theta \, d\omega + \frac{1}{\pi} \int_{\Omega} (1 - \mu) \frac{L^r(\vec{o})}{L^a(\vec{s})} \cos^+ \theta \, d\omega.
\]

The first term is the *obscurances value* [ZIK98, IKSZ03] of point \( \vec{s} \):

\[
O(\vec{s}) = \frac{1}{\pi} \int_{\Omega} \mu(\vec{d}(\vec{z})) \cos^+ \theta \, d\omega. \tag{4.7}
\]
If the ambient light is similar at $\vec{s}$ as at $\vec{o}$, the reflected radiance at $\vec{o}$ can be expressed by the ambient reflectivity as

$$L'(\vec{o}) = a(\vec{o}) \cdot W(\vec{o}) \cdot L'(\vec{s}),$$

thus we obtain a recursive formulation for the ambient reflectivity:

$$W(\vec{s}) = O(\vec{s}) + \frac{1}{\pi} \int_{\Omega} (1 - \mu) a(\vec{o}) W(\vec{o}) \cos \theta \, d\omega.$$

If the neighborhood is sufficiently small, then $W(\vec{o})$ is similar to $W(\vec{s})$, thus we can write:

$$W(\vec{s}) \approx O(\vec{s}) + \frac{1}{\pi} \int_{\Omega} (1 - \mu) a(\vec{o}) W(\vec{s}) \cos \theta \, d\omega,$$

from which the ambient transfer function is

$$W(\vec{s}) \approx \frac{O(\vec{s})}{1 - \frac{1}{\pi} \int_{\Omega} (1 - \mu) a(\vec{o}) \cos \theta \, d\omega}. \quad (4.8)$$

Finally, if the local approximation can also be used for the albedo, i.e. we can assume that $a(\vec{o}) \approx a(\vec{s})$, then the next approximation of the ambient transfer function is

$$W(\vec{s}) \approx \frac{O(\vec{s})}{1 - \frac{1}{\pi} \int_{\Omega} (1 - \mu) a(\vec{s}) \cos \theta \, d\omega} = \frac{O(\vec{s})}{1 - a(\vec{s})(1 - O(\vec{s}))}. \quad (4.9)$$

Let us interpret this formula. If the albedo is small, then the ambient transfer function is similar to the obscurances value. However, if the albedo is close to 1, then the ambient transfer function gets also close to 1, reducing the darkening of obscurances. For example, the corners of white rooms will not be darker than the wall itself, which corresponds to our everyday observations. This phenomenon is well known by skiers. In cloudy and foggy weather, the scratches and bumps of the high albedo snow become invisible.

Figure 4.5: Comparison of classic ambient occlusion (left) and the new ambient transfer function (right).
4.5 Implementation of ambient occlusion and indirect lighting

The discussed algorithm is implemented as a fragment program of the deferred shading pass. The program gets the fragment address in texture-space ($wPos$) and in 2D clipping-space ($hPos$), respectively. Looking up the texture map of normal vectors and depth values ($depthMapSampler$) with the texture-space address, we obtain the camera-space normal vector and the depth value. The 2D clipping-space coordinates are transformed back to camera-space using also the camera-space depth, which results in shaded point $\vec{s}$.

Then the numerator and denominator of the ambient occlusion formula and the irradiance are computed in variables $O$, $Denom$, and $I$ in the loop executed by $samplecount$ times. A single occluder sample $\vec{o}$ is generated from prepared 2D points uniformly distributed in the unit disk ($AO\_RAND\_k$). The depth map is fetched taking the direction of the occluder, which results in occluder depth $z_{start}$. This depth is compared to the entry depth of the sphere $z_{min}$, exit depth $z_{exit}$, and that of the intersection with the tangent plane $z_{plane}$, determining the $z_{min}$, $z_{max}$ interval where function $h$ is evaluated. In parallel, another integral is computed in $Denom$ that describes the unoccluded case. This integral will be used for error compensation. Note that we also check whether or not the occluder is much closer to the eye than the shaded point and ignore such occlusions, which would otherwise result in false silhouette edges (Figures 4.6 - 4.7).

If near occlusion happens and according to the occluder’s surface orientation it can illuminate the shaded point, then the occluder’s color is inserted into the average color used for indirect illumination. This fragment shader (Listing 4.2) returns with the average indirect illumination and the ambient occlusion of the fragment, which is then composited with the previously calculated direct illumination result.
CHAPTER 4. APPROXIMATION OF AMBIENT LIGHTING

Figure 4.7: The occlusion factor without (left) and with false silhouette elimination (right).

Listing 4.2: Implementation of ambient occlusion and indirect lighting

```c
uniform sampler2D colorMap;
uniform sampler2D depthMap;
uniform float pixelSize;
uniform int sampleCount;
uniform float A0_RAND[];
uniform mat4 projMatrixInverse;
in vec2 wPos;
in vec2 hPos;
out vec4 Color;

void main()
{
    wPos += pixelSize * 0.5;  // Texture coordinates in [0,1]

    vec3 N = texture(depthMap, wPos).xyz;  // Camera-space normal
    float depth = texture(depthMap, wPos).a;  // Camera-space depth

    // Compute camera-space position
    vec4 pCamera = mul(vec4(hPos, 0, 1), projMatrixInverse);
    pCamera.xyz /= pCamera.w;  // shaded point
    float S = 0;  // Enumerator of ambient occlusion
    float Denom = 0;  // Denominator of ambient occl.
    vec3 I = 0;  // Irradiance

    for (int sampleIdx = 0; sampleIdx < sampleCount; sampleIdx++)
    {
        vec2 sample = A0_RAND[sampleIdx].xy * R;
        vec3 s = s + vec3(sample.x, sample.y, 0);  // Occluder

        pCamera = mul(vec4(s, 1), projMatrixInverse);
        vec2 texCoord = pCamera.xy / pCamera.w;
        texCoord.y *= -1.0f;
        texCoord = (texCoord + 1) / 2;
        float zstar = texture(depthMap, texCoord).a;
        s.z = zstar;  // Occluder's depth

        float d = sqrt(R * R - dot(sample.xy, sample.xy));
        float zmin = s.z - d;
    }
```
As the compiler rolls out loops that contain `texture` calls, we may run out of registers when the sample number is high (it is greater than 12 in the specified hardware). If we need for more samples, the `texture` calls should be replaced by `textureLod`, which does not enforce loop rolling out. Surprisingly, this replacement does not degrade the performance.

### 4.6 Results

In order to demonstrate the advantages of the proposed non-uniform sample construction for containment test based screen-space ambient occlusion we implemented both the uniform and non-uniform sampling in obscurnce computation using square root membership function proposed by Mendez [MSC+05]. The left image in Figure 4.8 is the reference image which is obtained by uniform directions and distances. In order to show the differences, we turned interleaved sampling off and took 32 samples per pixel.

The new approach to evaluation of volumetric ambient occlusion is based on rewriting the ambient occlusion integral to evaluate the volume of the open part of the neighborhood hemisphere. The new integral can be evaluated more accurately with the same number of samples, thus, it is more appropriate in real-time systems where low noise results are needed with just a few cheap samples. The method does not require pre-processing and runs at high frame rates, since it only needs as few as 12-16 samples per pixel.

In Figure 4.9 we demonstrate the proposed ambient transfer function. It is compared to environment lighting and using obscurances. The average albedo of the texture of the carved object is \((0.6, 0.3, 0.3)\) on the wavelengths of red, green and blue, respectively. Note that the unrealistic darkening of the obscurances is eliminated by the application of the new ambient transfer function.

The proposed ambient transfer function can capture not only the local occlusions but also multiple scattering and diffuse interreflections between neighboring surface elements. In Fig-
CHAPTER 4. APPROXIMATION OF AMBIENT LIGHTING

Figure 4.8: Comparison of uniform sampling (left) and importance sampling in the obscurance method (right).

Figure 4.9: The effects of modulating the environment lighting with the obscurances, $O(\vec{x})$, and with the proposed ambient transfer function, $W(\vec{x})$. The ambient transfer function is also shown on the right.

Figure 4.10 we demonstrate this diffuse interreflection.

4.7 Conclusions

In this chapter we presented an efficient sample distribution scheme to construct non-uniform sample locations for containment test based screen-space ambient occlusion methods.

For the evaluation of volumetric ambient occlusion we proposed a new integration scheme, which takes into account the local slope of the surfaces, thus provides a more accurate approximation of the occlusion term than previous approaches. The new method provides smoother results with the same number of samples and has the same performance as classic methods.

To approximate the indirect lighting, classical ambient occlusion methods take into account the geometry of the scene and scale the indirect illumination according the openness of the surface. We proposed an extension to the occlusion factor, which takes into account the albedo of the surface. The proposed new ambient transfer function can be extended to approximate the
indirect lighting between diffuse surfaces. As we calculate the ambient occlusion with random point samples in the neighborhood of the shaded point, the color of the frame buffer at these points can be used to obtain the average reflected radiance from given direction. By inspecting the camera-space normal, we can also check whether the surface is oriented toward shaded point, and ignore it in the average otherwise. The new ambient transfer functions can be implemented with minimal overhead.
Chapter 5

Scattering simulation in participating media

This chapter presents a fast parallel Monte Carlo method to solve the radiative transport equation in inhomogeneous participating media for light photons. Real-time graphics applications are speed critical since we need to render more than 20 images per second in order to provide the illusion of continuous motion. Fast simulation is also important in interactive systems like radiotherapy [NFML88] or physical experiment design where the user is allowed to place the source and expects an instant feedback about the resulting radiation distribution. The speed of multiple scattering simulation is also crucial in iterative tomography reconstruction where the scattered radiation is estimated from the actually reconstructed data, removed from the measurements, and reconstruction is continued for the residual that is assumed to represent only the unscattered component.

5.1 Previous work

Computing multiple scattering and rendering inhomogeneous participating media in a realistic way are challenging problems [RT87, Rus94, LBC95, CPP05, Fat09]. The most accurate approaches are based on Monte Carlo quadrature and trace photons or importons (i.e. visibility rays) randomly in the medium [JC98, PKK00, QXFN07].

The conditional probability density that a photon collides with the particles of the material provided that the photon arrived at this point is defined by the extinction coefficient $\sigma_t(\vec{x})$, which may depend on the photon’s frequency and also on the location $\vec{x}$ if the particle density is not homogeneous. Upon collision, the photon may get reflected or absorbed. The probability of reflection is called the albedo and is denoted by $a$.

The random reflection direction is characterized by two spherical coordinates, scattering angle $\theta$ and azimuth angle $\phi$ (Figure 5.1). The probability density of scattering angle $\theta$ that is between the original and the new directions may depend on the frequency and is described by the physical model. We use the Rayleigh scattering [Enc13b] model for light photons:

$$P(\cos \theta) = \frac{3}{16\pi}(1 + \cos^2 \theta).$$

The probability density of the azimuth angle is uniform due to the rotational symmetry.

In computer simulation, the photon is traced according to the basic laws mentioned so far. The inputs of the simulation are the density of the participating medium, which is defined by
CHAPTER 5. MULTIPLE-SCATTERING IN PARTICIPATING MEDIA

Figure 5.1: Photon scattering in participating media. The incident photon collides with a particle of the material. Due to this collision the photon’s direction may change. The new direction is specified by scattering angle θ and azimuth angle φ.

A 3D scalar density field, the frequency dependent albedo function, and the probability density of the scattering angle. The density field is assumed to be available in a discrete form as a 3D texture of voxels. The frequency dependent albedo and scattering density functions may be defined either by tables stored in textures or by algebraic formulae.

In order to simulate the photon transport process, the collision, the absorption, and the scattering direction should be generated with the probability densities defined by the physical model.

Generating a single step of the random path involves the sampling of the free path traveled by the photon before scattering, deciding whether or not absorption happens, and finally sampling the new scattering direction. Absorption handling and the determination of the new scattering direction are based on the local properties of the medium, thus these tasks can be solved by evaluating relatively simple formulae that need just a few variables, which are either globally available or can be determined from the actual position.

Free path sampling is also simple when the medium is homogeneous since the next scattering point will depend just on the constant extinction coefficient of the medium. However, in inhomogeneous media having position dependent density, free path sampling becomes complicated and will dominate the total computation time since we should gather information about the continuously changing extinction coefficient along the way of the photon [CSI09].

5.1.1 Light attenuation and free path sampling

In participating media radiance $L$ is attenuated along ray $\vec{p}(s) = \vec{p}_{\text{start}} + \vec{ω}S$ of origin $\vec{p}_{\text{start}}$ and direction $\vec{ω}$ by absorption and out-scattering, which results in an exponential decay:

$$L(S) = L(0) \cdot \exp \left(-\int_{0}^{S} \sigma_t(\vec{p}(S'))dS'\right)$$  \hspace{1cm} (5.1)

where $\sigma_t(\vec{p})$ is the extinction coefficient that defines the probability density that photon-particle collision happens in $\vec{p}$ provided that the photon arrived at this point.

The integral of the extinction coefficient is also called the optical depth of this interval and
is denoted by $\tau$:

$$
\tau(S_0, S_1) = \int_{S_0}^{S_1} \sigma_t(\vec{p}(S'))dS'.
$$

(5.2)

Monte Carlo methods generate discrete light path samples and approximate integral quadratures by the weighted sum of the contribution of these paths. The error of the approximation can be reduced by *importance sampling* that places the discrete samples with a frequency that is proportional to the integrand.

Sampling proportionally to a prescribed function can be done by the *inversion method*. The inversion method first calculates the probability density as the normalization of the original function, then obtains the desired *cumulative probability distribution* (CDF) as the integral of the probability density, and finally generates the discrete samples by inverting the CDF for values that are uniformly distributed in the unit interval.

The CDF of the free path length $S$ along ray $\vec{p}(S)$ is

$$
P(S) = 1 - \exp(-\tau(0, S)).
$$

(5.3)

Thus, free path length $S$ corresponding to a uniform random number $r$ is the solution of the following equation:

$$
r = P(S) \iff -\log(1 - r) = \tau(0, S).
$$

(5.4)

When the medium is inhomogeneous, the extinction coefficient is not constant but is represented by a voxel grid or by other finite elements. In this case, the usual approach is *ray marching* that takes small steps $\Delta S$ along the ray and checks when the Riemann sum approximation of the optical depth gets larger than $-\log(1 - r)$:

$$
\sum_{i=0}^{n-1} \sigma_t(\vec{p}(i\Delta S))\Delta S \leq -\log(1 - r) < \sum_{i=0}^{n} \sigma_t(\vec{p}(i\Delta S))\Delta S.
$$

(5.5)

Unfortunately, this algorithm is biased [RSK08] and requires a lot of voxel array fetches, especially when the voxel array is large and the average extinction is small.

*Woodcock tracking* [WMHL65] (also called as fictitious interaction tracking, pseudo-scattering, hole-tracking, self-scattering or delta-tracking) provides an unbiased alternative to *ray marching*, and samples the free path with the following randomized algorithm:

1. Generate tentative path length $S$ using the maximum extinction coefficient $\sigma_{\text{max}}$ in the volume.
2. Accept the tentative collision point with probability $\sigma_t(\vec{p}(S))/\sigma_{\text{max}}$.
3. If the collision is rejected, then the particle’s direction is not altered and a similar sampling step is repeated from the tentative collision point.

We do not repeat the proof [Col68] that this method is unbiased here since our new sampling strategy will also contain Woodcock tracking as a special case, and we shall prove the unbiasedness for the general case.

Woodcock tracking becomes very inefficient when the maximum extinction coefficient is much larger than the extinction coefficient in a particular domain of the volume, since here the acceptance probability, $\sigma_t(\vec{p}(S))/\sigma_{\text{max}}$, will be very small, which requires the sampling of a lot of tentative scattering points [Lep10]. In the *Hole geometry package* [Ans], the application developer should decompose the geometry to roughly homogeneous regions where Woodcock tracking
can be executed with different maximum extinction parameters. However, this approach requires the inclusion of a fictitious scattering at each crossed boundary of different regions, where a new random sample needs to be generated and the sample process repeated, which degrades performance when the path crosses many regions.

In the following sections we propose a new free path sampling method which is an extension of the Woodcock tracking method.

5.2 Free path sampling with virtual particles

Free path sampling is equivalent to the solution of equation 5.4 for path length $S$. If the extinction coefficient and consequently the optical depth are available in a simple algebraic form defined by a few parameters, then the solution is straightforward and requires just the fetching of these parameters from the main memory. However, if the optical depth can only be computed from many data, which happens when the extinction coefficient is specified by a high-resolution voxel array, then the sampling process will be slow. To solve this problem, we modify the volume by adding virtual “material” or particles in a way that the total density will follow a simple function. One might think that modifying the material density would also change the light radiance inside the volume resulting in a distorted rendering solution, which is obviously not desired. Fortunately, this is not necessarily the case if the other two free properties of the virtual material, namely the albedo and the phase function are appropriately defined.

Virtual particles do not alter the radiance inside the medium if they do not change the energy and the direction of photons during scattering. This requirement is met if the virtual particle has albedo 1, and its phase function is a Dirac-delta, since in this case the collision with a virtual particle alters neither the photon’s energy nor its direction with probability 1, so the virtual material does not affect the light’s radiance (Figure 5.2).

![Figure 5.2: Virtual particles modify the density but not the radiance since their albedo is 1 and their phase function is a Dirac-delta.](image)

More formally, we handle heterogeneous volumes by mixing additional virtual particles into
the medium to augment the extinction coefficient to a simpler upper-bounding function $\sigma_{\text{max}}(\vec{p})$. For original extinction coefficient $\sigma_{t}(\vec{p})$, we have to find the extinction coefficient $\sigma_{v}(\vec{p})$ of the virtual particles, so that in the \textit{combined medium} of real and virtual particles the extinction coefficient is $\sigma_{\text{max}}(\vec{p}) = \sigma_{t}(\vec{p}) + \sigma_{v}(\vec{p})$. During scattering we have to determine whether it happened on a real or on a virtual particle. As sampling is required to generate random points with a prescribed probability density, it is enough to solve this problem randomly with the proper probabilities. As the extinction parameters define the probability density of scattering, ratios $\sigma_{t}(\vec{p})/\sigma_{\text{max}}(\vec{p})$ and $\sigma_{v}(\vec{p})/\sigma_{\text{max}}(\vec{p})$ give us the probabilities whether scattering happened on a real or on a virtual particle, respectively.

Having added the virtual particles, free path sampling is executed in the following steps:

1. Generate path length $S$ using the upper-bounding extinction coefficient function $\sigma_{\text{max}}(\vec{p}(S))$.

2. When a potential scattering point $\vec{p}$ is identified, we decide randomly with probability $\sigma_{t}(\vec{p})/\sigma_{\text{max}}(\vec{p})$ whether scattering happened on a real or on a virtual particle. If only virtual scattering occurred, then the particle's direction is not altered and a similar sampling step is repeated from the scattering point.

Note that this concept is quite similar to that of Woodcock tracking. The key difference is that we do not require the maximum extinction coefficient be a global constant, but allow arbitrary non-negative virtual particle density $\sigma_{v}(\vec{p})$.

### 5.2.1 Upper-bounding extinction coefficient function

Free path length $S$ in the combined medium of the real and virtual particles is obtained by solving the following equation, which is the adaptation of equation 5.4 for the medium containing both the real and the virtual particles:

$$\log(1 - r) = \int_{0}^{S} \sigma_{\text{max}}(\vec{p}(S'))dS'.$$

If we have a simple representation for $\sigma_{\text{max}}(\vec{p})$, then the scattering point in the combined medium can be found in a simpler way than identifying the scattering point of the medium containing only the real particles.

For the sake of simplicity we assume that the upper-bounding $\sigma_{\text{max}}(\vec{p})$ is a piece-wise polynomial function defined in a low-resolution grid. The voxels of this low-resolution grid are much larger than the actual voxels defining the true extinction coefficient $\sigma_{t}(\vec{p})$, and are called super-voxels (Figure 5.3). We note that other upper-bounding functions might also be used if the solution of equation 5.6 is straightforward for them.

We execute a 3D DDA like voxel traversal [FTK86, AW87] on the super-voxels and find the super-voxel that contains the root of equation 5.6 (Figure 5.4). The 3D DDA algorithm is based on the recognition that the boundaries of the cells are on three collections of parallel planes, which are orthogonal to the $x$, $y$, and $z$ axes, respectively. The algorithm maintains three ray parameters representing the next intersection points with these plane collections. The minimum of the three ray parameters represents the exit point of the current cell. To step onto the next cell, an increment is added to this ray parameter. The increments corresponding to the three plane collections are constants for a given ray.

As super-voxels are visited one after the other, we check whether the root of equation 5.6 is in this super-voxel. The inequalities selecting the super-voxel $n$ that contains the scattering
\[ \sum_{i=0}^{n-2} \tau_{\text{max}}(S_i, S_{i+1}) \leq -\log(1 - r) < \sum_{i=0}^{n-1} \tau_{\text{max}}(S_i, S_{i+1}) \]  

(5.7)

where

\[ \tau_{\text{max}}(S_i, S_{i+1}) = \int_{S_i}^{S_{i+1}} \sigma_{\text{max}}(\vec{p}(S'))dS' \]

is the optical depth of the ray segment intersecting the \( i \)th super-voxel, computed for the upper-bounding extinction coefficient.

The important differences of ray marching and the proposed approach are that steps \( \Delta S_i = S_{i+1} - S_i \) are not constant but are obtained as the length of the intersection of the ray and the super-voxel, and sample points \( S_i \) are consequently on super-voxel boundaries.
When in step \( n \) the inequalities are first satisfied, the super-voxel of the scattering point is located. The actual scattering point is computed by solving the following equation for \( S \):

\[
\tau(S_{n-1}, S) = -\log(1 - r) - \sum_{i=0}^{n-2} \tau_{\text{max}}(S_i, S_{i+1})
\]  

(5.8)

where the right side is computed from the uniform sample \( r \) and the optical depth of previously visited super-voxels.

To find the super-voxel containing the scattering point and to locate the scattering point inside this super-voxel, we need to compute the optical depth along the ray segments intersecting the super-voxels. Let us consider a single super-voxel and use normalized coordinates \( X, Y, Z \) that are all zeros at the left-front-lower corner of the super-voxel and all ones at its right-back-upper corner (Figure 5.5). The optical depth integral of a single super-voxel can be computed from the extinction coefficients \( \sigma_{000}, \ldots, \sigma_{111} \) of the corners, where the first, second, and third indices denote the \( X, Y, Z \) coordinates, respectively.

The ray enters the super-voxel at entry point \( \vec{e} \) and leaves it at exit point \( \vec{o} \). Points \( \vec{p} \) of the ray segment that are inside the super-voxel satisfy the following equation:

\[
\vec{p}(S) = (1 - t(S))\vec{e} + t(S)\vec{o}
\]

where \( t(S) = (S - S_i)/\Delta S_i \in [0, 1] \). As extinction coefficient \( \sigma_{\text{max}}(\vec{p}) \) is a tri-variate polynomial of the Cartesian coordinates, which are linear functions of parameter \( t \), the extinction coefficient will be a polynomial of parameter \( t \):

\[
\sigma_{\text{max}}(\vec{p}(S(t))) = \sum_{d=0}^{M} c_d t^d,
\]  

(5.9)

where \( (c_0, \ldots, c_M) \) are the coefficients of the polynomial. Thus, the optical depth between entering this super-voxel and parameter \( S \) is:
\[
\tau_{\text{max}}(S_i, S) = \int_{S_i}^{S} \sigma_{\text{max}}(\vec{p}(S'))dS'
\]

\[
= \Delta S_i \int_{0}^{t} \sigma_{\text{max}}(\vec{p}(S(t')))dt'
\]

\[
= \Delta S_i \sum_{d=0}^{M} c_d\frac{t^{d+1}}{d+1}.
\] (5.10)

The total optical depth \(\tau(S_i, S_{i+1})\) in super-voxel \(i\) can be obtained by substituting the end point, i.e. \(S = S_{i+1}\) or \(t = 1\):

\[
\tau_{\text{max}}(S_i, S_{i+1}) = \Delta S_i \sum_{d=0}^{M} c_d\frac{1}{d+1}.
\] (5.11)

Obtaining the upper-bounding extinction function

Measured or computed volumes are often defined in a discretized form, as a 3D regular grid of voxel values, which can have large resolution. If we use ray marching, then the computational complexity of free path sampling will be linear in terms of the linear voxel grid resolution. In order to use the proposed method, we need an upper-bounding functional representation defined on a low-resolution grid. An upper-bounding zero-order polynomial, i.e. a single maximum value of voxels in each super-voxel is easy to find. For tri-linear or higher-order functional upper-bounds, we use the following algorithm.

First the original voxel values are considered only at the super-voxel corners and a function is fitted to these samples. Then the original voxel values inside each super-voxel are compared to the initial functional representation, and the maximum difference between them is obtained. Finally, this maximum difference is added to the functional upper-bound. Note that using this method, neighboring super-voxels may have different values at the shared super-voxel corners. The increased storage is small since the super-voxel grid has a low-resolution. However, should it pose problems, the maximum value at each shared corner is computed, and assigned to the corner as the value of every super-voxel sharing it.

5.3 The photon transport algorithm

Photons travel in space independently of each other and thus can be simulated in parallel, which reveals the inherent parallelism of the physical simulation (Nature is a massively parallel machine). The algorithm of simulating the path of a single photon is a nested loop. Both the outer and the inner loops are dynamic and their number of executions cannot be predicted. The body of the outer loop is responsible for finding the next scattering point, then sampling absorption and the new scattering direction in case of survival. This loop is terminated when the photon is absorbed or it leaves the simulation volume. The number of scattering events of
different photons will range on a large interval, so will the number of outer loop executions. The inner loop visits voxels until the next scattering point is located, which depends on the drawn random number and also on the density of medium along the ray of the photon path. Both the random number and the density vary, thus again the number of visited voxels may be different for different samples (Figure 5.6).

Figure 5.6: Causes of control path divergence in photon transport simulation. A photon path has random number of scattering points and between two scattering points there may be different number of voxels.

The problem of executing different number of iterations in parallel threads becomes crucial on the GPU. If we assigned a photon to a computational thread of the GPU, than the longest photon path and the longest ray of all paths in each warp will determine the computational speed, which is unacceptable. In order to maintain warp coherence, the execution lengths of the threads must be similar and the threads are preferred to always execute the same instruction sequence. To attack these problems, we restructure the transport simulation according to the requirements of efficient GPU execution, i.e. we develop algorithms that can be implemented with minimal conditional instructions [LSK10].

To solve the problem of different number of scattering events, we assign a computational thread not to a complete photon path, but only to a single step of a photon, including the identification of the next scattering point and the decision on termination. Then, another kernel samples the new direction if the photon survived the collision, otherwise generates the direction of a new photon at the source, keeping all threads busy at all times.

After this restructuring, threads may still diverge. The determination whether or not the photon is absorbed and sampling the new scattering direction are based on the local properties of the medium, thus these tasks require the solution of an equation that contains just a few variables, which can be determined from the actual position. Even this simple problem becomes difficult if the CDF cannot be analytically computed and symbolically inverted. In these cases, numerical methods should be used. Free path sampling is even worse as it requires the exploration of a larger part of the medium, which corresponds to a lot of memory accesses being relatively slow on current GPUs. Both iterative numerical root finding and the exploration of the volume in regions of different density lead to thread divergence and eventually poor GPU utilization.

As different photons are traced randomly, they visit different parts of the simulated volume, resulting in parallel accesses of far parts of the data memory. This kind of memory accesses degrades cache utilization. To solve this problem, one might propose the assignment of those photons to a warp which possibly have similar paths. As random paths are computed from
sequences of uniformly distributed random numbers, this would mean the clustering of vectors of random numbers. A complete path may be associated by many, e.g. more than 20 random numbers, so the clustering operation should work in very high dimensional spaces, which is not feasible in practice. So, photons are sorted according to only the first random number, and assigned to warps in the sorted sequence. This provides little initial data access coherence, but sooner or later, threads will inevitably fetch different parts of the volume.

5.3.1 Free path sampling

The first step of simulating a photon of current location $\vec{x}$ and direction $\vec{ω}$ is the sampling of the free path that the photon can fly without collision, i.e. the determination of the next interaction point along the ray $\vec{p}(S) = \vec{x} + S \vec{ω}$ of its path. The proposed algorithm to find the next scattering point is a nested loop which takes the ray of origin $\vec{p}_{\text{start}}$ and direction $\vec{ω}$. The outer loop is responsible for random decisions whether a virtual or a real particle is hit. The inner loop is a 3D DDA voxel traversal of the super-voxel grid.

```
function SuperVoxelTraversal( $\vec{p}_{\text{start}}, \vec{ω} \Rightarrow \vec{p}$)
    $\tau_{\text{sample}} = -\log(1 - \text{rnd}());$
    $\vec{σ} = \vec{p}_{\text{start}}, \vec{s}_{\vec{σ}} = 0, \tau_{\vec{σ}} = 0;$
    while $\tau_{\vec{σ}} < \tau_{\text{sample}}$
        $\vec{e} = \vec{σ}, \vec{s}_{\vec{e}} = \vec{s}_{\vec{σ}, \vec{ω}} = \tau_{\vec{σ}};$
        if (out of volume) return "No scattering";
        $\vec{σ} = \vec{p}_{\text{end}};$
        PolyCoeff($\vec{e}, \vec{σ} \Rightarrow c_0, \ldots, c_M$);
        $\tau_{\vec{σ}} = \tau_{\vec{e}} + (\vec{s}_{\vec{σ}} - \vec{s}_{\vec{e}}) \cdot \sum_{d=0}^{M} c_d/(d + 1);$  // Next exit point
    endwhile
    SolvePolyInt($c_0, \ldots, c_M, \vec{s}_{\vec{e}}, \vec{s}_{\vec{σ}}, \tau_{\vec{e}}, \tau_{\vec{σ}}, \tau_{\text{sample}} \Rightarrow S, t);$  // Max extinction coefficient
    $\sigma_{\text{max}} = \sum_{d=0}^{M} c_d t^d;$
    $\vec{p} = \vec{p}_{\text{start}} + \vec{s}_{\vec{σ}};$
    $P_{\text{real}} = \sigma_t(\vec{p})/\sigma_{\text{max}};$
    while (rnd() > $P_{\text{real}})$
        return "Scattering";  // Is real or virtual?
    endwhile
    return "Scattering";  // Real scattering is found
end
```

The outer loop starts with sampling the optical depth $\tau_{\text{sample}}$ by transforming a random value uniformly distributed in the unit interval, generated by the $\text{rnd}()$ function. Then, the inner loop visits super-voxels, maintaining entry point $\vec{e}$ with its ray parameter $S_{\vec{e}}$ and optical depth $\tau_{\vec{e}}$, as well as exit point $\vec{σ}$ with its ray parameter $S_{\vec{σ}}$ and optical depth $\tau_{\vec{σ}}$. Initially, the entry point is the origin of the ray with zero ray parameter and optical depth, then the parameters of the entry point are updated with the parameters of the exit point of the previous super-voxel at each DDA step. The ray parameter of the exit point $S_{\vec{σ}}$ is determined by the 3D DDA algorithm, and the location of the exit point is obtained by inserting the ray parameter into the equation of the ray. Function $\text{PolyCoeff}$ computes the coefficients of the polynomial of the upper-bounding optical depth in this super-voxel. The optical depth of the exit point is computed as the sum of that of the entry point and the integral of the polynomial in this super-voxel. The inner loop is executed until the optical depth of the exit point becomes larger than the sampled optical depth, i.e. when the super-voxel contains the sample point. Exiting the inner loop, we have to identify the exact location of the scattering point, which is the solution of the sampling equation by calling
SolvePolyInt, which returns ray parameter $S$ and its normalized version $t$. The upper-bounding extinction coefficient $\sigma_{\text{max}}$ is computed by substituting the ray parameter into the polynomial, and the location of the scattering by inserting the ray parameter into the equation of the ray. At the end of the outer loop, we decide randomly whether this point corresponds to a real or a virtual scattering having fetched extinction $\sigma_t(\vec{p})$ from the original volume. The important differences of ray marching and the proposed approach are that steps are not constant but are obtained as the length of the intersection of the ray and the macrocells.

The computation of the polynomial coefficients by PolyCoeff and the solution of the sampling equation by SolvePolyInt depend on the actual form of the upper-bounding function. In the following subsections, we provide the implementations for the piece-wise constant and tri-linear cases.

**Piece-wise constant upper-bound**

If the upper-bounding $\sigma_{\text{max}}(\vec{p})$ is constant in a super-voxel, then $c_0 = \sigma_{\text{max}}(\vec{p})$ in this super-voxel and all other coefficients are zero. The optical depth will be a linear function of ray parameter $t$, thus equation $\tau(S_1, S) = \tau_{\text{sample}} - \tau_e$ can be solved directly:

$$
\text{SolvePolyInt}(c_0, S_e, S_o, \tau_e, \tau_{\text{sample}} \Rightarrow S, t) \\
t = (\tau_{\text{sample}} - \tau_e)/(S_o - S_e)/c_0; \\
S = S_e + (S_o - S_e) \cdot t. 
$$

**Tri-linearly interpolated upper-bound**

In the case of tri-linear upper-bound, the polynomial of $\sigma_{\text{max}}(\vec{p}(t))$ is cubic with coefficients $(c_0, c_1, c_2, c_3)$. The coefficients are computed from the extinction values $\sigma_{000}, \ldots, \sigma_{111}$ in the eight corners and from the end points of the ray segment. First, we transform the entry and exit points into normalized coordinates, where the left-front-lower corner of the super-voxel is the origin and the right-back-upper corner has coordinates $(1,1,1)$, resulting in transformed entry point $\vec{E}$ and exit point $\vec{O}$. In this space, the extinction coefficient is the following function of normalized coordinates $X, Y, Z$:

$$
\sigma_{\text{max}}(X, Y, Z) = \sigma_{000}XY\bar{Z} + \sigma_{100}X\bar{Y}Z + \sigma_{010}\bar{X}Y\bar{Z} + \sigma_{001}\bar{X}\bar{Y}Z + \sigma_{110}XY\bar{Z} + \sigma_{101}X\bar{Y}\bar{Z} + \sigma_{011}\bar{X}Y\bar{Z} + \sigma_{111}XY\bar{Z}
$$

where we used the shorthand notation $\bar{A} = 1 - A$. Substituting the ray of equation $(X(t), Y(t), Z(t)) = (1 - t)\vec{E} + t\vec{O}$ into this equation, the maximum coefficient is expressed as a cubic polynomial of $t$, where the similar powers of $t$ are grouped and the coefficients are expressed. The computation is summarized in the following function:

$$
\text{PolyCoeff}(\vec{e}, \vec{\sigma} \Rightarrow c_0, c_1, c_2, c_3) \\
\vec{E} = \text{Transform entry point } \vec{e} \text{ to the unit voxel cube}; \\
\vec{O} = \text{Transform exit point } \vec{e} \text{ to the unit voxel cube}; \\
(\Delta x, \Delta y, \Delta z) = (\vec{O}_x - \vec{E}_x, \vec{O}_y - \vec{E}_y, \vec{O}_z - \vec{E}_z);
$$
\[ d_{xyz} = \sigma_{111} - \sigma_{011} - \sigma_{101} - \sigma_{110} + \sigma_{100} + \sigma_{010} + \sigma_{001} - \sigma_{000}; \]
\[ d_{xy} = \sigma_{000} - \sigma_{100} - \sigma_{010} + \sigma_{110}; \]
\[ d_{xz} = \sigma_{000} - \sigma_{100} - \sigma_{010} + \sigma_{110}; \]
\[ d_{yz} = \sigma_{000} - \sigma_{100} - \sigma_{010} + \sigma_{110}; \]
\[ d_x = \sigma_{100} - \sigma_{010}; \]
\[ d_y = \sigma_{010} - \sigma_{001}; \]
\[ d_z = \sigma_{001} - \sigma_{000}; \]
\[ c_3 = d_{xyz} \Delta x \Delta y \Delta z; \]
\[ c_2 = (E_z \Delta x \Delta y + \bar{E}_y \Delta y \Delta z + \bar{E}_x \Delta x \Delta z) d_{xy} + d_{yz} \Delta x \Delta y + d_{xz} \Delta x \Delta z + d_{yz} \Delta y \Delta z; \]
\[ c_1 = (E_z E_x \Delta x + \bar{E}_x \bar{E}_y \Delta y + \bar{E}_x \bar{E}_y \Delta z) d_{xy} + d_x \Delta x + d_y \Delta y + d_z \Delta z + \]
\[ (\bar{E}_y \Delta x + \bar{E}_y \Delta y) d_{xy} + (\bar{E}_x \Delta x + \bar{E}_z \Delta z) d_{xz} + (E_z \Delta y + \bar{E}_y \Delta z) d_{yz}; \]
\[ c_0 = \bar{E}_x \bar{E}_y \bar{E}_z d_{xyz} + \bar{E}_x \bar{E}_y \bar{E}_z d_{xy} + \bar{E}_x \bar{E}_y \bar{E}_z d_{xz} + \bar{E}_y \bar{E}_y d_{yz} + \bar{E}_y \bar{E}_z d_{yz} + \bar{E}_x \bar{E}_z d_{xz} + \bar{E}_y \bar{E}_z d_{yz} + \sigma_{000}; \]

Function \texttt{SolvePolyInt} finds the solution of \( \tau(S_i, S) = \tau_{\text{sample}} - \tau_0 \) by the \textit{false position method} (regula falsi):

\[
\texttt{SolvePolyInt}(c_0, \ldots, c_M, S_t, S_B, \tau_0, \tau_{\text{sample}} \Rightarrow S, t) \\
\texttt{t}_{\text{low}} = 0; \quad \texttt{t}_{\text{high}} = 1; \\
\texttt{\tau}_{\text{low}} = \tau_0; \quad \texttt{\tau}_{\text{high}} = \tau_{\text{sample}}; \\
\texttt{while} (|\texttt{t}_{\text{high}} - \texttt{t}_{\text{low}}| > \texttt{EPS}) \\
\quad \quad \texttt{t} = \texttt{t}_{\text{low}} + (\texttt{t}_{\text{high}} - \texttt{t}_{\text{low}}) \cdot (\texttt{\tau}_{\text{sample}} - \texttt{\tau}_{\text{low}})/(\texttt{\tau}_{\text{high}} - \texttt{\tau}_{\text{low}}); \\
\quad \quad \texttt{\tau} = \texttt{\tau}_0 + (S_t - S_B) \cdot \sum_{d=0}^{M} c_d t^{d+1}/(d+1); \\
\quad \quad \texttt{if} (\texttt{\tau} < \texttt{\tau}_{\text{sample}}) \texttt{t}_{\text{low}} = \texttt{t}, \quad \texttt{\tau}_{\text{low}} = \texttt{\tau}; \\
\quad \quad \texttt{else} \quad \texttt{t}_{\text{high}} = \texttt{t}, \quad \texttt{\tau}_{\text{high}} = \texttt{\tau}; \\
\texttt{endwhile} \\
\quad S = (S_B - S_t) \cdot \texttt{t} + S_t; \\
\texttt{end}
\]

5.3.2 Absorption sampling

When the photon collides with a real particle, we should determine whether or not absorption happened and also the new direction if the photon survived the collision. The probability of survival is defined by the albedo, thus this random decision is made by checking whether or not the albedo is larger than a random number distributed in the unit interval:

\[ r < a. \]

5.3.3 Scattering direction

If the photon survives the collision, a new continuation direction needs to be sampled, which is defined by scattering angle \( \theta \) and azimuth angle \( \phi \).

The azimuth angle \( \phi \) is uniformly distributed, thus it is generated as \( \phi = 2\pi r_1 \) where \( r_1 \) is a uniform random number in the unit interval. The scattering angle is sampled by solving the sampling equation

\[ P_{\cos \theta} (\cos \theta) = r_2 \]

where \( P_{\cos \theta} (\cos \theta) \) is the cumulative probability distribution of the cosine of the scattering angle on a given frequency, and \( r_2 \) is another uniform random number. The probability density of the scattering angle is usually defined by the physical model and is a moderately complex algebraic expression. We use Rayleigh scattering for light photons from which the cumulative distribution can be obtained by integration. Other phase functions can also be handled in a similar way.
5.4 Attenuation calculation with virtual particles

In both physically based and simplified simulations, we often have to determine the total radiance attenuation between two points. If scattering is ignored, the total attenuation between the source and the camera will determine the image [DCH88]. In volume shadowing, the attenuation between the source and the current sample point is needed to find the shadowing factor. In Monte Carlo simulation, paths are usually completed by deterministic connections [DBB03], and the attenuation along this connection is included into the estimator. If the path is started in the source (photon tracing), then scattering points are connected to the eye. Alternatively, in path tracing, scattering points are usually connected to the sources.

In all these calculations the integral of the optical depth should be computed for the line segment. The optical depth can be obtained by ray marching at the expense of many texture fetches. Another possibility is the direct application of free path sampling. We decide randomly whether or not the free path is longer than the line segment. If the random free path is longer, the radiance at the start point is added to the end without attenuation. If the random free path is shorter, then no radiance is added. Although the expected value of this estimator provides correct results, its variance is too high.

Thus, we apply the concept of virtual particles in a better way. The optical depth is decomposed to two terms corresponding to the upper-bounding extinction and the negative extinction of the virtual particles:

\[
\int_{0}^{S} \sigma_t(\vec{p}(S')) dS' = \int_{0}^{S} \sigma_{\text{max}}(\vec{p}(S')) dS' - \int_{0}^{S} \sigma_v(\vec{p}(S')) dS'.
\] (5.13)

The first term can be analytically computed. The second term is estimated as a numerical quadrature. Note that the difference of the maximum and the actual extinction coefficients (the extinction of the virtual particles) has relatively low variation, thus few samples can provide acceptable accuracy. This idea, its advantages and drawbacks are similar to those of the separation of the main part used in Monte Carlo approaches. If the upper-bounding extinction coefficient is a tight and thus the analytically integrated term is already a good approximation for the optical depth, then accurate results can be expected even from just a few samples. However, if the upper-bound is a very crude approximation and is farther from the extinction coefficient than the constant zero function, then this method is not worth using.

5.5 Parallel random number generation

All discussed sampling steps transform random numbers that are uniformly distributed in the unit interval. Free path sampling needs as many random numbers as virtual scattering events happen until the first real scattering point. Absorption sampling requires just one additional random variable, but direction sampling uses two. As different steps of the photon path are statistically independent, every step should use a new set of random numbers. Furthermore, since different photons behave statistically independently, the simulation of every photon should be based on a new random number sequence. Clearly, if we used the same random number generator in all parallel threads simulating different photons, then we would unnecessarily repeat the very same computation many times.

The typical solution of this problem is the allocation a private random number generator for each parallel thread. To guarantee that their sequences are independent, the private random number generators are initialized with random seeds. The private random number generators run
on the GPU and are assigned to parallel threads, while their random initial seeds are generated on the CPU using the random number generator of the C library.

We note that the mathematical analysis and the construction of robust parallel random number generators [Knú81, Deá89] (and even non-parallel ones) are hard problems, which are out of the scope of this dissertation. We use a CUDA implementation [HT09].

### 5.6 Evaluation of the photon transport algorithm

The discussed method has been integrated into a photon mapping global illumination application, which decomposes the rendering into a shooting and a gathering phase. During shooting, multiple scattering of photons are calculated in the volume, registering the collision points, photon powers, and incident directions. The powers of photon hits are accumulated in cells defined by a 3D array called the illumination buffer. The gathering phase visualizes the illumination buffer by standard alpha blending.

The inputs of the simulation system are the position of the isotropic radiation source, the position of the virtual camera, and the three dimensional texture of the density field.

The shooting phase iteratively updates three arrays (Figure 5.7):

1. The one dimensional seed buffer that stores the seeds of the parallel random number generators. This buffer is initialized by the CPU and is updated when a new random number is generated by its respective thread.

2. The one dimensional photon buffer representing the currently simulated photons with their current position, direction, and energy.

3. The three dimensional illumination buffer.

The photon buffer and the seed buffer have the same size, which is equal to the number of concurrently simulated photons. The photon buffer is managed by three kernels that are started in a cyclic manner:

1. The Initialization kernel puts newly emitted photons into the places of a dead or uninitialized photons in the photon buffer, and samples the scattering direction of an existing and living ones. A thread is assigned to each item of the photon buffer, and checks whether or not the photon is alive. If it is dead, then a new photon is generated at the position of the source with uniformly distributed random direction and assigning the source's energy level to the photon. If the photon already exists and is alive, then its position is not altered, but a new scattering direction is sampled. The random direction sampling requires two new uniform random numbers in the unit interval. These random numbers are obtained by calling the random number generator assigned to this thread, which will also update the respective seed in order to generate a different random number when it is called the next time.

2. The Shooting kernel simulates one step of the photons stored in the photon buffer. A thread is responsible for a photon, and uses its photon buffer item both for input and output. The thread calculates the next scattering event of its own photon executing free path sampling, then it decides whether absorption occurs. If the photon leaves the volume without collision or the absorption check tells us that the photon is absorbed, then its status is set to “dead”, to get the next Initialization operation to generate a new emitted photon in its place. The thread finally writes out the modified location and the status of the photon.
CHAPTER 5. MULTIPLE-SCATTERING IN PARTICIPATING MEDIA

Figure 5.7: The architecture of the simulation program. The seed buffer contains the seeds of the random number generators, which are initialized by the CPU and are updated by the Initialization and the Shooting kernels. The Initialization kernel processes the photon buffer and executes two operations depending on whether or not the photon exists and is alive. It samples the scattering direction of already existing and living photons, and sets the initial position and direction of newly born ones that replace the dead photons. The Shooting kernel finds the next collision location and decides on termination. A thread in the Initialization and Shooting kernels is assigned to a respective item of the seed buffer and the photon buffer, so no synchronization is needed here. The Accumulation kernel adds the power of the photon to a cell of the illumination buffer selected by the photon’s location.
3. The Accumulation kernel gathers the incident photon powers from the photon buffer and
adds them to the illumination buffer. The illumination buffer is a 3D grid, where an item
represents the total irradiance arriving at the respective grid cell. The quantized Cartesian
coordinates of the photon’s actual position serves as an index into the accumulation buffer
where the incident power is added. Absorbed photons are also considered since they also
arrived at this point. Note that this is the only place in our algorithm where different
threads may write to the same memory location, and synchronization problems might occur.
The correct implementation requires semaphores or atomic add operations to guarantee that
no photon power gets lost. If we remove the mutual exclusion, then the power of some
photons may not be added to the buffer. As the number of active threads is far smaller
than the number of data items in the illumination buffer, the probability of such missing
photons is rather small. Moreover, the simulation process will trace millions of photons,
so a few missing photons cause just a negligible error.

For the visualization of the simulation results, the illumination buffer is rendered from the
camera accumulating the attenuation and the radiance in gathering super-voxels that are along
the view ray.

In a gathering super-voxel at point \( \vec{x} \), the in-scattered radiance \( L(\vec{x}, \vec{\omega}') \) is obtained from the
in-scattered flux [JC98]:

\[
L(\vec{x}, \vec{\omega}') = \frac{1}{\sigma_t(\vec{x})} \frac{d^2 \Phi(\vec{x}, \vec{\omega}')}{d\omega dV} \tag{5.14}
\]

where \( \Phi(\vec{x}, \vec{\omega}') \) is the in-scattered flux from direction \( \vec{\omega}' \). The in-scattered flux is obtained from
the photon hits stored in the gathering super-voxel. The radiance scattered towards the eye
being in direction \( \vec{\omega} \) is

\[
L^s(\vec{x}, \vec{\omega}) = a(\vec{x}) \sum_i \frac{\Delta \Phi_i P(\vec{\omega}_i', \vec{\omega})}{\Delta V} \tag{5.15}
\]

where \( a(\vec{x}) \) is the albedo of the volume, \( P(\vec{\omega}_i', \vec{\omega}) \) is the phase function, \( \Phi_i \) and \( \vec{\omega}_i' \) are the power
and the incident direction of the \( i \)th photon hit, respectively, and \( \Delta V \) is the volume of the
gathering super-voxel where photon hits are considered.

In medical simulation, we are interested in the radiation dose of the body tissues, which
requires another visualization scheme. Instead of the scattered radiance, the power in-scattered
from all directions per unit volume is calculated:

\[
I(\vec{x}) = \sum \frac{\Delta \Phi_i}{\Delta V} \tag{5.16}
\]

Then, assigning a pseudo-color to the radiation dose by an appropriate transfer function, the
volume is rendered with standard alpha blending.

### 5.7 Results

The implementation of the discussed method is based on CUDA and runs entirely on the GPU.
We tested the system on an NVIDIA GeForce 480 GTX graphics card. The current version
assumes a single isotropic point source and runs on a single GPU, but the extension to more
general sources and the exploitation of multi-GPU cards and GPU clusters are straightforward.
In our application the radiation source can be placed interactively in the volume represented by
a 3D texture fitting into the GPU texture memory.

To measure the performance, we took the standard Visible Human data set and resampled
it at 512^3 resolution to fit it into the GPU memory, simulated multiple scattering and computed
Figure 5.8: Radiation dose calculation in the Visible Human data set of $512^3$ resolution voxel array. The power density is shown with pseudo-colors obtained with the transfer function of the right bar, and is superimposed onto the density field visualized with the transfer function of the left bar. Times refer to building the photon map containing 32 million hits.
the radiation dose with four different free-path sampling methods: (a) classical ray marching, (b) Woodcock tracking, (c) the new method using piece-wise constant upper-bound, and (d) with piece-wise tri-linear upper bound. The resolution of the super-voxel grid was $16^3$. The images obtained with 32 million photons are shown by Figure 5.8. Note that the visual quality is similar since all methods use the same probability density and the same number of the samples, but the photon shooting times significantly differ since the cost of finding the scattering point is method dependent.

<table>
<thead>
<tr>
<th>Upper-bound</th>
<th>$1^3$</th>
<th>$2^3$</th>
<th>$4^3$</th>
<th>$8^3$</th>
<th>$16^3$</th>
<th>$32^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>5</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>18</td>
<td>15</td>
</tr>
<tr>
<td>Linear</td>
<td>5</td>
<td>11</td>
<td>14</td>
<td>19</td>
<td>22</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 5.1: The effect of the super-voxel grid resolution on the performance, which is expressed as million rays per second traced in the Visible Human data set.

The dependence of the performance on the super-voxel grid resolution is shown by Table 5.1. At higher super-voxel resolution, rays cross more super-voxels, which requires slightly more computation. On the other hand, increasing the resolution makes the upper-bound tighter, thus it reduces the probability of virtual collisions and consequently the number of texture fetches from the large voxel array. This effect is strong if the extinction coefficient has high variation, thus the optimal tradeoff depends on this factor.

To demonstrate the efficiency of the proposed free path sampling scheme, we tested our system on different volumetric datasets (Figures 5.9, 5.10 and 5.11).

We tested the proposed attenuation calculation algorithm in an X-ray like scenario (Figure 5.12). Assuming a constant radiance planar source behind the body, the pixel intensity is proportional to the exponential of the optical depth between the source and the camera along the ray of the pixel. Here, the images are obtained by ray marching and by the proposed attenuation estimation method taking 1, 2, or 4 samples in each super-voxel. The step size of ray marching is equal to the linear size of the voxel in the original $512^3$ resolution array. The resolution of the super-voxel grid is $32^3$. Note that the image rendered with as few as 4 samples per super-voxel is very similar to the image computed with more effort due to the fact that the upper-bounding function is a good approximation and is integrated analytically.

## 5.8 Conclusions

In this chapter, we presented an effective method to sample free path length in inhomogeneous participating media, and we used this method to find the global illumination solution of volumetric models defined by high-resolution voxel arrays. The proposed free path sampling method causes significantly smaller warp divergence than ray marching or Woodcock tracking.

The strength of the method is that its computational complexity depends just on the resolution of the super-voxel data and is independent of the number of additional scales, thus volumes of very large resolution can be rendered with multiple scattering effects in seconds. This kind of scalability is due to the randomization of the volume by adding virtual particles and deciding randomly whether a collision happened on these or on real particles. Randomization may have overhead, which is small if the super-voxels absorb the major variations of the original data, and thus the scales above the super-voxels have small amplitude. This is fortunately the case of many natural phenomena having smoother large shapes and small random fluctuations.
Figure 5.9: Radiation dose calculation in the Visible Human data set of $512^3$ resolution voxel array. The power density is shown with pseudo-colors obtained with the transfer function of the right bar, and is superimposed onto the density field visualized with the transfer function of the left bar.
Figure 5.10: Radiation dose calculation in the Engine data set of $256^3$ resolution voxel array. The power density is shown with pseudo-colors obtained with the transfer function of the right bar, and is superimposed onto the density field visualized with the transfer function of the left bar.
Figure 5.11: Radiation dose calculation in the Head data set of $128^3$ resolution voxel array. The power density is shown with pseudo-colors obtained with the transfer function of the right bar, and is superimposed onto the density field visualized with the transfer function of the left bar.
Figure 5.12: Attenuation calculation for the Visible Human, with ray marching visiting voxels, and with the proposed scheme taking 1, 2, and 4 samples in each super-voxel. Rendering times are measured at $600 \times 600$ image resolution.
Unfortunately measured voxel arrays should be stored into the GPU memory, which limits their resolution. As our proposed method accesses the high resolution array just to decide whether or not a real collision happened, and otherwise uses only the low-resolution array of super-voxels, it is a natural extension to store only the low-resolution array in the GPU memory and copy the real extinction coefficients from the CPU memory only when they are needed. This extension would allow practically arbitrarily high resolution models to be simulated, which is badly needed in medical applications. However, the data access pattern of the high resolution array is very incoherent, so realizing a caching scheme for it in the GPU memory is not simple and is considered as a future work.
Chapter 6

Summary

This thesis work concentrated on indirect lighting effects and their efficient rendering using Monte Carlo methods and exploiting the features of the GPU.

6.1 Thesis group 1: Efficient sampling for screen-space algorithms

Texture filtering with importance sampling

Texture filtering is a critical part in many rendering and post-processing algorithms. In case of typical filter based approaches, the applied shader program samples not only the shaded point but its neighborhood too. Thus, if we do this sampling naively, the shader needs to access texture memory proportionally with the applied filter size. On the modern graphics hardware the texture fetching is at least an order of magnitude slower than the arithmetic operations, so it is crucial to minimize it for real-time rendering.

I proposed a new sampling scheme to reduce the necessary texture fetch operations without sacrificing the accuracy. The new sampling method is built upon importance sampling and exploits the interpolation capability of the modern graphics hardware. I demonstrated the applicability of the method on typical screen-space algorithms like tone-mapping, depth of field and ambient occlusion [B1, C2, C3].

Screen space approximation of volumetric single scattering

Taking into account the volumetric scattering effect in participating media can greatly improve the overall realism of rendered images. The common methods (i.e. Monte Carlo quadrature, volumetric photon maps) require expensive computations which prohibit their application in real-time rendering. In many cases the physically correct multiple scattering model does not contribute too much to the overall appearance of the final image, and a simplified single scattering model can capture the main characteristics of light transport in the participating media.

I proposed a new single scattering simulation model, which can be applied as post-processing in real-time. The new technique uses the sampled representation of the scene geometry and a modified shadow mapping algorithm to approximate the light transport and applies interleaved sampling to improve performance [C9, D23].
CHAPTER 6. SUMMARY

6.2 Thesis group 2: Screen-space approximation of the ambient lighting

Non-uniform sample distribution scheme for obscurances calculation

The obscurance model approximates the shadowing of the ambient light. The evaluation of directional integral of the obscurance formula requires rays to be traced in many directions, which is rather costly. With the assumption that the neighborhood of the shaded point and the variance of the surface are sufficiently small, the expensive ray tracing operation can be replaced by a simple containment test. This approach samples the occlusion at discrete sample points. The distribution of the sample points along the rays is crucial.

I proposed an efficient sample distribution scheme based on importance sampling to construct non-uniform sample locations for containment test based screen-space ambient occlusion methods [B1].

A new volumetric integration method for volumetric ambient occlusion

The screen-space ambient occlusion method calculates the openness of surfaces by sampling the discrete representation of the scene (i.e. depth buffer). The accuracy of this approximation is proportional to the number of samples taken. More samples provide better results at the cost of higher computation complexity.

I proposed a new integration method for the volumetric ambient occlusion calculation, which takes into account the local slope of the surfaces, thus provides a more accurate approximation of the occlusion term than previous approaches. The new method provides smoother results with the same number of samples and has the same performance as classic methods [C7, J6, D22].

Approximation of local indirect lighting

If we can assume that the ambient light is similar in the neighborhood of the shaded surface point, then we can extend the ambient transfer function with the interreflection between diffuse surfaces.

I proposed a new ambient transfer function for the local approximation of the indirect lighting between diffuse surfaces:

\[ W(\vec{s}) \approx \frac{O(\vec{s})}{1 - \frac{1}{2\pi} \int_{\Omega} (1 - \mu(d)) a(\vec{d}) \cos \theta \, d\omega} , \]

where \( O(\vec{s}) \) is the ambient occlusion of the shaded point \( \vec{s} \), \( a(\vec{d}) \) is the albedo of the occluder point \( \vec{d} \), \( \mu(d) \) is the fuzzy measure and \( \Omega \) is the directional sphere. When this local approximation can also be used for the albedo of the occluder we can simplify the ambient transfer function to

\[ W(\vec{s}) \approx \frac{O(\vec{s})}{1 - \frac{1}{2\pi} \int_{\Omega} (1 - \mu) a(\vec{d}) \cos \theta \, d\omega} = \frac{O(\vec{s})}{1 - a(\vec{s}) (1 - O(\vec{s}))} . \]

This function approximates the ambient lighting better than the classical obscurances when the albedo of the surface is high [B2, J2, J3].
6.3 Thesis group 3: Multiple scattering simulation in participating media

Homogenization of inhomogeneous participating media with virtual particles

Simulating multiple scattering and rendering inhomogeneous participating media can be achieved by solving the volumetric rendering equation. The most accurate approaches are based on Monte Carlo quadrature and trace photons or importons (i.e. visibility rays) randomly in the medium.

I proposed a new sampling scheme by adding virtual particles in a way that the total density will follow a simple function appropriate for free path sampling. The added material has special properties not to alter the radiance inside the medium. With this extension the free path can be sampled with conventional methods for homogeneous media, and during scattering we have to determine whether it happened on a real or on a virtual particle. The collision with virtual particle changes neither the energy nor the direction of the traced photon [B3, D27].

Efficient free path sampling with virtual particles

Conventional free path sampling methods becomes very inefficient if the maximum extinction coefficient of the simulated participating media is much larger than the extinction coefficient in a particular domain. In these parts the density of the virtual particles will be much higher, therefore the acceptance probability will be very small, which requires the sampling of a lot of tentative scattering points.

I proposed a new sampling method, which is a generalization of the Woodcock tracking. With an appropriate upper-bound function the free path sampling can be decomposed. The extinction coefficient upper-bound function can be represented as a low-resolution voxel grid, and the rough estimation of the place of scattering can be computed with a 3D DDA like algorithm. With this rough estimation the accurate scattering point calculation can be narrowed to small region of the original voxel array [C10, J7].
Own publications


Bibliography


Index

3D DDA, 51, 56
absorption, 3, 4, 48, 58
absorption coefficient, 4
accessibility shading, 30
albedo, 4, 7, 10, 24, 30, 35, 50
ambient light, 7
ambient lighting, 7
ambient occlusion, 29–31, 37, 40
ambient transfer function, 40, 44
attenuation, 22, 59
attenuation coefficient, 4
bi-linear filtering, 13
blending, 9
blending operation, 9
BRDF, 3, 30
camera model, 3
camera space, 17
camera transformation, 8
CDF, 13, 14, 49
CIE XYZ transform, 15
circle of confusion, 17–19
color bleeding, 40
cross sections, 4
CUDA, 60, 62
depth of field, 16, 18–20
Direct3D, 7
extinction coefficient, 4, 47, 48, 59
filter kernel, 11, 13, 14
focal distance, 17
fragment shader, 9, 14, 19
free path sampling, 48, 49, 56
fuzzy measure, 30, 35
Gaussian filter, 11, 14–16, 18
global illumination, 2, 60
GLSL, 9
graphics processing unit, 2, 29, 55
height-field, 32, 33
high dynamic range, 15
HLSL, 9
importance sampling, 6, 12, 13, 15, 18, 20, 49
in-scattering, 4
incident radiance, 30
incremental pipeline, 6
indirect lighting, 31
input assembler, 8
integral equation, 5
interleaved sampling, 25, 26, 37
irradiance, 2, 3, 31, 62
key, 16
local illumination model, 29
local indirect lighting, 29
low dynamic range, 15
luminance, 15, 16
model space, 7
Monte Carlo integration, 5, 12
Monte Carlo quadrature, 6
Neumann series, 5
non-uniform sampling, 14
normal distribution, 14
normal vector, 7
obscurances, 29–31, 35, 36
Ogre3D, 19
OpenGL, 7, 26
optical depth, 48, 52, 59
out scattering, 4
output merge, 9
participating media, 4, 47, 48
perspective projection, 8
phase function, 4, 22, 24, 50
pinhole camera, 3, 16
Poisson disc distribution, 15
post-processing, 11, 20, 24, 25, 36
probability density, 13
probability density function, 12
probability distribution, 12
radiance, 2-5, 21, 24, 30, 48, 59
rasterization, 7
rasterizer, 9
ray marching, 23, 24, 33, 49
Rayleigh scattering, 47
reflected radiance, 3, 7, 30
reflection, 3
refraction, 3
relative luminance, 16
render calls, 7
render states, 7
rendering equation, 3, 6
sampling, 11
scattering, 3, 48, 55
scattering coefficient, 4
screen space, 17
screen-space, 9, 32, 44
separable filters, 11
shader program, 9, 11
shadow mapping, 23
spatial integral, 11
standard deviation, 11
super-voxel, 51, 53, 54, 57, 62
surface point, 7
tessellation control, 8
tessellation evaluation, 9
tessellator, 8, 9
texture coordinate, 7
texture filtering, 11
texture memory, 11
tone mapping, 15
triangle meshes, 7
uniform distribution, 15
uniform sampling, 14
variance, 6, 11, 12
vertex color, 7
vertex position, 7
vertex shader, 8
viewport transformation, 8
virtual particles, 50, 59
volumetric ambient occlusion, 31, 44
volumetric rendering equation, 5, 22
weak perspective approximation, 33
weighted uniform sampling, 40
Woodcock tracking, 49
world space, 7, 8
z-buffer, 9
z-buffer algorithm, 7
## Nomenclature

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu$</td>
<td>fuzzy measure function</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>domain of directions</td>
</tr>
<tr>
<td>$\phi$</td>
<td>azimuth angle</td>
</tr>
<tr>
<td>$\sigma_a$</td>
<td>absorption coefficient</td>
</tr>
<tr>
<td>$\sigma_s$</td>
<td>scattering coefficient</td>
</tr>
<tr>
<td>$\sigma_t$</td>
<td>extinction coefficient</td>
</tr>
<tr>
<td>$\tau$</td>
<td>optical depth</td>
</tr>
<tr>
<td>$\theta$</td>
<td>scattering angle</td>
</tr>
<tr>
<td>$\theta'$</td>
<td>incident angle</td>
</tr>
<tr>
<td>$\vec{\omega}$</td>
<td>outgoing direction</td>
</tr>
<tr>
<td>$\vec{\omega}'$</td>
<td>incoming direction</td>
</tr>
<tr>
<td>$\vec{o}$</td>
<td>occluder point</td>
</tr>
<tr>
<td>$\vec{s}$</td>
<td>shaded point</td>
</tr>
<tr>
<td>$\vec{x}$</td>
<td>surface point</td>
</tr>
<tr>
<td>$a$</td>
<td>albedo</td>
</tr>
<tr>
<td>$f_r$</td>
<td>bidirectional reflection distribution function (BRDF)</td>
</tr>
<tr>
<td>$h$</td>
<td>height</td>
</tr>
<tr>
<td>$I(\vec{s})$</td>
<td>irradiance</td>
</tr>
<tr>
<td>$L$</td>
<td>radiance</td>
</tr>
<tr>
<td>$L^a$</td>
<td>ambient radiance</td>
</tr>
<tr>
<td>$L^e$</td>
<td>emitted radiance</td>
</tr>
<tr>
<td>$L^i$</td>
<td>incident radiance</td>
</tr>
<tr>
<td>$L^i$</td>
<td>in-scattered radiance</td>
</tr>
<tr>
<td>$L^r$</td>
<td>reflected radiance</td>
</tr>
<tr>
<td>$L^s$</td>
<td>scattered radiance</td>
</tr>
<tr>
<td>$O(\vec{s})$</td>
<td>ambient occlusion</td>
</tr>
<tr>
<td>$P(\vec{x}, \vec{\omega}')$</td>
<td>phase function</td>
</tr>
<tr>
<td>$r$</td>
<td>distance</td>
</tr>
</tbody>
</table>

$T$ tangent sphere

$V$ volume

$v(\vec{x})$ visibility function

$W(\vec{x})$ obscurance
## List of Figures

1.1 Radiance change in participating media. .................................................. 4  
1.2 Architecture of the modern rendering pipeline. ........................................ 8  
2.1 The kernel of the Gaussian filter and its cumulative distribution function. ........ 14  
2.2 The computation of the circle of confusion. ........................................... 17  
2.3 The Moria scene with tone mapping and glow. ........................................ 19  
2.4 The Moria scene without and with depth of field. .................................... 20  
3.1 Taking into account the opaque geometry during ray marching. .................... 23  
3.2 Computation of the scattered component by ray marching. .......................... 23  
3.3 Single scatter as post-processing pass. ................................................... 24  
3.4 Compositing the final image. ...................................................................... 25  
3.5 Scheme of interleaved sampling. ............................................................... 25  
3.6 Comparison of scatter image generated with 8,128 and 8 interleaved samples. .... 26  
3.7 Sample scene without and with volumetric scattering. ............................... 26  
3.8 Sample scene without and with volumetric scattering. ............................... 27  
4.1 Surfaces represented as height field and by the depth buffer. ....................... 33  
4.2 Replacing ray tracing by containment tests. .............................................. 34  
4.3 Comparison of ambient occlusion without and with interleaved sampling. ....... 38  
4.4 Evaluation of the volumetric integral in the visible part of the neighborhood sphere. 39  
4.5 Comparison of classic ambient occlusion and the new ambient transfer function. 41  
4.6 False silhouette elimination. ................................................................. 42  
4.7 The occlusion factor without and with false silhouette elimination. ............... 43  
4.8 Comparison of uniform sampling and importance sampling in the obscurance method. 45  
4.9 Modulating the environment lighting with the obscurances and with the new ambient transfer function. ................................................................. 45  
4.10 Sample scene without and with ambient occlusion, and only the new ambient transfer function. ................................................................. 46  
4.11 Comparison of a GI solution, classic AO and the new ambient transfer function. 46  
5.1 Photon scattering in participating media. .................................................. 48  
5.2 Virtual particles modify the density but not the radiance. ............................ 50  
5.3 Super-voxel grid that encodes upper-bounding density. .............................. 52  
5.4 3D DDA algorithm to visit super-voxels. .................................................. 52  
5.5 Processing of a single super-voxel. ......................................................... 53  
5.6 Causes of control path divergence in photon transport simulation. ............... 55  
5.7 The architecture of the simulation program. ............................................. 61  
5.8 Radiation dose calculation in the Visible Human data set. ......................... 63  
5.9 Radiation dose calculation in the Visible Human data set. ......................... 65  
5.10 Radiation dose calculation in the Engine data set. .................................. 66  
5.11 Radiation dose calculation in the Head data set. .................................... 67  
5.12 Attenuation calculation for the Visible Human data set. ........................... 68