UNIVERSIDAD
REY JUAN CARLOS

COMPUTER GRAPHICS VIRTUAL
REALITY–GAMES MASTER

Academic Year 2017/2018

Master Thesis

GPU VOLUMETRIC PATH TRACING FOR CLOTH RENDERING

Author: F. Javier Fabre Herrando
Tutors: Jorge Lopez-Moreno
Carlos Aliaga

To
my parents and sister

Acknowledgements

I would like to thank all the people who have helped since the beginning to the end of this project.

First, thanks to Jorge, for introducing me to this awesome project. I would also want to thank Carlos. His help and dedication has been really valuable to me. I have not enough words to thank you two.

Thanks to all the people in Desilico Labs who have helped me. Without your help this project would not have been possible.

Last but not least, I would like to thank my parents, my sister and my friends, whose support and understanding during this year have been priceless to me.

Thanks to all of you.

Abstract

Over the last years, the computational capabilities of the Graphics Processing Units (GPUs) have increased drastically. While the performance of such hardware was progressively improving, the GPUs, traditionally used for computer graphics applications, started to be widely used to gain efficiency in computationally costly applications of many different disciplines out of computer graphics.

GPUs were born in the context of computer graphics from the requirements of real-time rendering for videogames. However, the entertainment industry and visual effects, which traditionally use Brute Force Monte Carlo techniques (off–line rendering) to render photorealistic imagery, still rely on the Central Processing Unit (CPUs), with few exceptions mostly concentrated in research purposes.

In this Master Thesis we propose a non–proprietary rendering engine that fully runs on the GPU. It has the rendering capabilites of any commercial engine, such as natural lighting and complex anisotropic materials. In addition, the engine has been designed to be specialist in rendering virtual garments. This involves many technical implications related mainly to the capability of running volumetric path tracing over thin, optically dense, and highly anisotropic heterogeneous volumes. Handling such features, strongly present in fabrics, constitute a very active and open research topic for the off-line rendering community.

Contents

1 Introduction **3**

2 Previous Work **7**
2.1 Path Tracing implementations 7
2.2 Scattering models for cloth rendering 8

3 Path Tracing on the GPU **9**
3.1 Path integral 10
3.2 Bidirectional Scattering Distribution Function (BSDF) 15
3.3 Camera model 16
3.4 Sampling and Reconstruction 17
3.4.1 Stratified jittered sampling 18
3.4.2 Reconstruction filter 18
3.5 Performance and Implementation Details 20
3.5.1 Acceleration Structures 20
3.5.2 Uniform Random Generation 22
3.5.3 State Saving 23

4 Materials **25**
4.1 Diffuse material 25
4.2 Disney principled BRDF 26
4.2.1 Importance Sampling 28
4.3 Measured Materials 29

5 Volumetric Path Tracing **31**
5.1 Volume Scattering Processes 31
5.1.1 Emission 32
5.1.2 Absorption 32
5.1.3 Scattering 33
5.2 Equation of Transfer 35
5.3 Phase Function 37

5.4 Homogeneous media 38
5.4.1 Sampling homogeneous media 38
5.5 Heterogeneous media 39
5.5.1 Sampling Heterogeneous Media 40
5.6 Performance and Implementation Details 41
5.6.1 Memory scheduling 42
5.6.2 Medium improvements 43
5.6.3 GPU grid implementation 44

6 Volumetric Materials: Cloth **45**
6.1 Cloth Volumes 45
6.2 Light Scattering for Textile Fibers 46

7 Results **49**
7.1 Disney BSDF 49
7.2 Measured materials 53
7.3 Cloth materials 53
7.4 Full garments 54

8 Conclusions and Future Work **61**
8.1 Improved woodcock tracking 61
8.2 Correlated media 62
8.3 Improved scheduling algorithms 63
8.4 Improve lighting 63
8.5 Conclusions 64

Bibliography **68**

Chapter 1

Introduction

Computer graphics imagery are ubiquitous in everyday life for a wide range of applications, from entertainment industry to marketing or industrial design, to name a few. When aiming for photo–realism, *Brute Force Monte Carlo Path Tracing* has proven to be a very effective method to generate outstanding images, accounting for complex lighting and material effects, since it relies on ray optics to simulate the light propagation in the scene. That is, it accounts for the trajectories of photons interacting with the surfaces present in the scene, what is known as global illumination.

(a) Clouds (b) Fire (c) Smoke

Figure 1.1: Examples of participating media in the real world.

However, most of the materials in nature do not behave like surfaces. In addition to canonic examples like smoke, fog or clouds, there are other common materials like wax, skin or cloth that present volumetric behaviors at different scales (Figure 1.1). These materials, usually known as *participating media*, have a common nature: they all are based on aggregates of smaller components or particles that interact with light. The way light interacts with the media ultimately depends on the optical properties of the material, like its optical thickness or its absorption and light scattering patterns. All these features will be further explained in Section 5.1. In the case of cloth, the overall appearance is ultimately defined by the smallest components, which

are the textile fibers.

Then, the original Path Tracing Algorithm is enhanced to handle participating media, in a method known as *Volumetric Path Tracing* (see Section 5). In order to make the problem tractable, a bunch of particles (fibers in our case of cloth) is treated as an statistical aggregate, by discretizing the media in 3D cubes (*voxels*) that store structural and/or optical properties of the material. Thus, in this Master Thesis we propose practical solutions to several problems. Namely:

- We propose a cross-platform architecture for path tracing in the GPU based on OpenGL Shading Language (GLSL). We discuss about typical CPU structures could be adapted to be implemented in GPU architectures and how GPU limitations affects this kind of algorithms.

- We propose a solution to produce photo–realistic renderings of cloth using Volumetric Path Tracing in the GPU. This constitutes a very hard problem to solve, since we need to handle very thin, optically-dense, heterogeneous participating media that might also present very anisotropic patterns of interaction with light.

- We propose a fiber scattering function that accurately reproduces the optical behavior of textile fibers, together with efficient importance sampling techniques to allow renderings.

- We propose a new screen–space optimization to reduce memory requirement when rendering volumetric media. We discuss how this optimization could be implemented using GLSL to avoid GPU memory limitations when rendering high resolution cloth models.

This document is structured as follows: In Chapter 2, we will review the main path tracing implementations used in research environments, some suitable render algorithms already implemented in the GPU, and the state of the art in light scattering models for cloth.

Chapter 3 explain the main mathematical theory behind our path tracing implementation as well as the technical details and implemented improvements. A complete explanation of the non–volumetric materials implemented is exposed in Chapter 4.

Chapters 5 and 6 cover our main contribution, explaining the full volumetric path tracing implementation on the GPU and also the particular

features of our cloth model.

Lastly, in Chapters 7 and 8, we show the results of the current state of the engine and discuss future avenues of work.

Chapter 2

Previous Work

2.1 Path Tracing implementations

To generate realistic renders of scenes including volumetric materials, centering our research in cloth materials represented using volumetric data, we have decided to implement Volumetric Path Tracing algorithm.

Among CPU implementations, two of the most renowned are the offline–renderer implementation given with the book *Physically Based Rendering from theory to implementation* [Pharr et al., 2016] (know as PBRT) and the rendering engine *Mitsuba* [Jakob, 2010].

PBRT has its full implementation of many render algorithms, including path tracing. All the source code explained in the book is available in the website of the book, allowing access to even a volumetric path tracing implementation supporting Heterogeneous media. Also many of the algorithms implemented support some kind of CPU parallelization techniques. However, PBRT does not implement specific phase functions modeling cloth fiber, although is possible to extend some of the phase function model already implemented to achieve some of the phase function that allow realistic cloth renderings.

Mitsuba implements a set of complex offline–render algorithms, including volumetric path tracing. Mitsuba also implements phase functions emulating cloth fibers using the Microflakes models presented by Jakob et al. [2010], and other implementations of fiber models [Khungurn et al., 2015, Aliaga et al., 2017] have been implemented over this render engine.

Mitsuba gives support, not only to thread parallelization, but also to clusterization, making this rendering engine one of favorites in the research

community. Unfortunately, none of this render engines support the use of GPUs to improve the performance of their offline–renders.

In the field of GPU Path Tracing implementations, less research has been made. Parker et al. [2010] implemented a full ray tracing engine on GPU, and many improvements have been made to their orginal work (developing the actual OptiX engine). However the implementation of this engine is made in CUDA, preventing it from running in AMD GPUs.

Although is not a Path Tracer implementation, Hachisuka [2015] implements a Photon Mapper using OpenGL [Khronos Group, 1992] and GLSL, which allows his implementation to run almost every GPU from the last decade. As far as Volumetric Path Tracing is concerned, all of the known methods presented in the literature lack a full GPU implementation.

In this work, we aim to develop and implement a full implementation of the Volumetric Path Tracing algorithm using OpenGL and GLSL languages, allowing our software to run in any kind of GPU.

2.2 Scattering models for cloth rendering

Previous approaches have used a variety of scattering models that go from microflakes [Jakob et al., 2010, Heitz et al., 2015] to fiber scattering models similar to previous hair rendering models [Kajiya and Kay, 1989, Marschner et al., 2003, Zinke and Weber, 2007, Yan et al., 2015]. However, fiber scattering models have shown to match real–world cloth appearance, while microflakes models fail in this task.

Following this idea Schröder and colleagues [citas] proposed to use a parametric BCSDF. Khungurn et al. [2015] proposed a fiber–based model (with some limitations) based on cylindrical fibers.

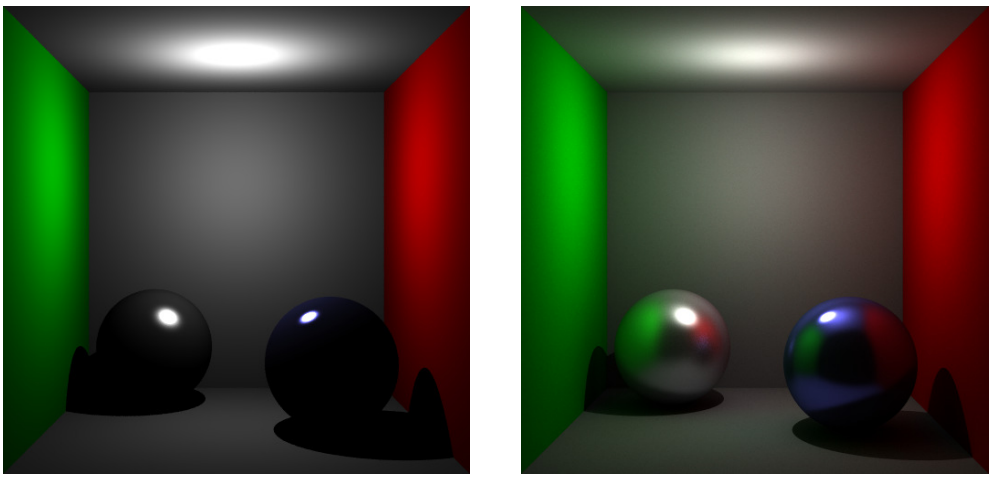
Later, Aliaga et al. [2017], presented a more complete model which aims to simulate the different shaped fibers instead of only cylinder, actually based on parameters used in the textile industry.

In this Master Thesis, we focus in the model presented by Khungurn et al. [2015] and aim to overcome some of its limitations by improving their actual scattering model.

Chapter 3

Path Tracing on the GPU

In a photograph, the value of each pixel is the result of the radiance reaching the sensor at that point, as a result of the integration of photons interacting with the media and bouncing in the scene. This is known as global illumination, and simulating such effects turns out to be crucial to produce accurate photo–realistic results (Figure 3.1).



(a) Local illumination (b) Global illumination

Figure 3.1: Comparative of the same scene rendered using only direct illumination (no light bounces) and taking into account global illumination.

Since Kajiya Kajiya [1986] proposed the Rendering Equation (or Light Transport Equation), many methods have been introduced in order solve it. Path tracing was developed in order to provide a numerical method to solve this problem by simulating the path of photons emitted from every light in

a scene that end up in the camera and contribute to the final image.

In this Master Thesis we will focus on Backward Path Tracing technique (Figure 3.2). This technique assumes that, thanks to the reciprocity of the light, we can simulate all the illumination launching the light paths from the camera instead of generating them on the light sources.

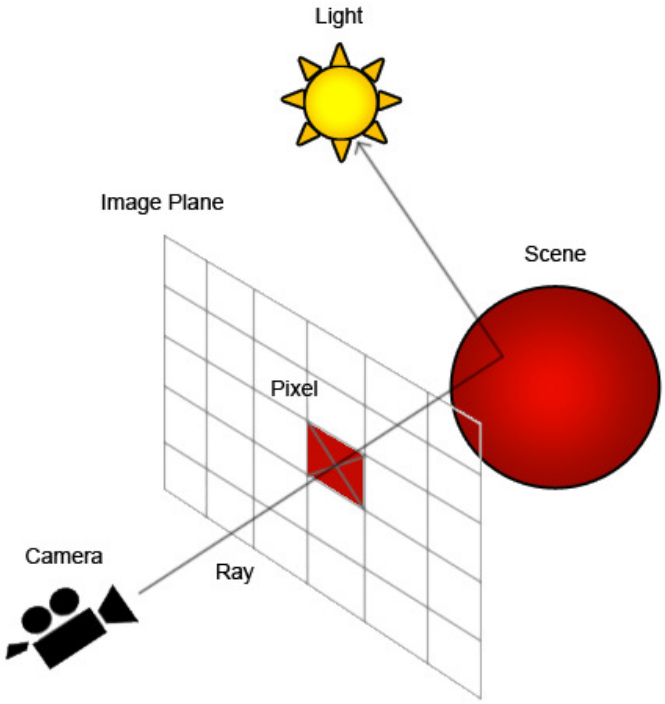


Figure 3.2: We generate ray starting from our camera assuming the path traveled is the same as if they were stated on the lights

3.1 Path integral

To simulate Global Illumination in a scene, we need to use the Rendering Equation, which is defined recursively, as we will show in the following section. Thus, if we want to precisely render a scene we would need to compute a excessive amount of recursive computations. However, [Veach, 1998, Section 8] proposed a different formulation to explain the light propagation phenomena. It is called *Path Integral*, and it reformulates the light propagation phenomena as only one integral instead of the recursion presented by the Rendering Equation.

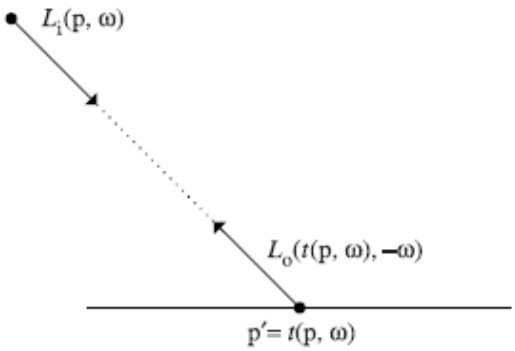


Figure 3.3: Radiance along a Ray is unchanged (if no participating media are present)

Starting from the main LTE:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{S^2} f(p, \omega_o, \omega_i) L_i(p, \omega_i) \, |\cos \theta_i| \, d\omega_i \tag{3.1}$$

If we assume (for now) that no participating media are present, radiance is constant along ray. Therefore, we can relate the incident radiance at p to the outgoing radiance at any other point of the space p' (Figure 3.3). Defining a *ray–casting function* $t(p, \omega)$ as a function computing the first surface point p' intersected by the ray casted from p in direction ω, the incident radiance at p can be rewritten as

$$L_i(p, \omega) = L_o(t(p, \omega), -\omega) \tag{3.2}$$

in terms of outgoing radiance at p.

To deal with the case where the ray does not intersect any object, we can define the *ray–casting function* to return a special value Δ, such as $L_o(\Delta, \omega) = 0$.

Using Equation 3.2 we can rewrite the LTE equation as (subscripts of L_o omitted for brevity):

$$L(p, \omega_o) = L_e(p, \omega_o) + \int_{S^2} f(p, \omega_o, \omega_i) L(t(p, \omega_i), \omega_i) \, |\cos \theta_i| \, d\omega_i \tag{3.3}$$

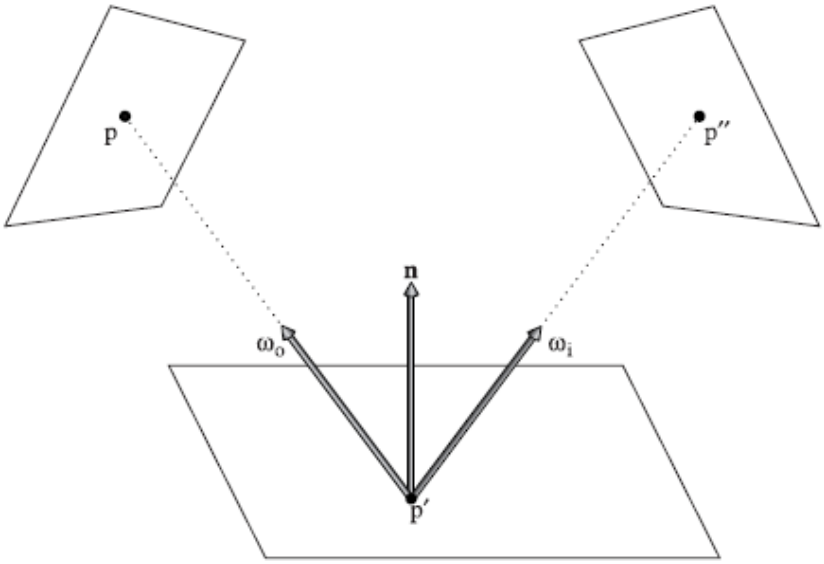


Figure 3.4: The three–point form of the LTE. Converts the integral from direction over the sphere, to be over a the domain of points on surfaces.

The main reason Equation 3.3 being complex is due to the fact that the relationship between geometry in a scene is implicit in the *ray–tracing* function $t(p, \omega)$. We can rewrite this equation as an integral over area instead of an integral over direction on the sphere to make the behavior of this function explicit in the integrand.

If we define the reflected radiance from a point p' to the point p as:

$$L(p' \to p) = L(p', \omega) \tag{3.4}$$

given that p' and p are mutually and $\omega_o = \widehat{p—p'}$.

The term $f(p', \omega_o, \omega_i)$ or Bidirectional Scattering Distribution Function (BSDF) (Section 3.2) at p' can be rewritten as

$$f(p'' \to p' \to p) = f(p', \omega_o, \omega_i), \tag{3.5}$$

where $\omega_i = \widehat{p''—p'}$ and $\omega_o = \widehat{p—p'}$ as we can see in Figure 3.4.

Although we have rewritten the LTE to be over the domain of points on surfaces in the scene, rather than over directions over the sphere, we still need to transform it from an integral over direction to one over surface area.

To do this, we need to multiply it by the Jacobian that relates solid angle to area.

This change–of–variable term, the original $|\cos\theta|$ term, and a binary visibility function V (with $V = 1$ if the point are mutually visible, and $V = 0$ otherwise) can be combined into a single geometry term:

$$G(p \leftrightarrow p') = V(p \leftrightarrow p')\frac{|\cos\theta|\,|\cos\theta'|}{\|p - p'\|^2}. \tag{3.6}$$

Substituting these equation into the LTE, and transforming it to an area integral:

$$L(p' \to p) = L_e(p' \to p) + \int_A f(p'' \to p' \to p)L(p'' \to p')G(p'' \leftrightarrow p' = dA(p''), \tag{3.7}$$

where A represent all the surfaces in the scene.

Now with Equation 3.7, instead of sampling a number of direction on the sphere and cast rays to evaluate the integrand (Equation 3.1), we could choose a number of points on surfaces of the scene and compute a coupling between those point to evaluate the integrand, tracing right so we can evaluate the visibility term $V(p \leftrightarrow p')$.

From this area integral from (Equation 3.7) we can derive the *path integral* formulation, which expresses radiance as an integral over paths (points in a high dimensional path space).

To reach this equation, we can start expanding the the *three–point* LTE, substituing the right–hand side of the equation into the $L(p'' \to p')$ term inside the integral. An example of the first few terms that give incidence radiance at point p_0 from another point p_1 (where p_1 is the first point on a surface along the ray starting in p_0 with direction $p_1 - p_0$) is expressed in the following equation:

$$\begin{aligned} L(p_1 \to p_0) =& L_e(p_1 \to p_0) \\ &+ \int_A L_e(p_2 \to p_1)f(p_2 \to p_1 \to p_0)G(p_2 \leftrightarrow p_1)dA(p_2) \\ &+ \int_A \int_A L_e(p_3 \to p_2 1)f(p_3 \to p_2 \to p_1)G(p_3 \leftrightarrow p_2) \\ &\qquad \times f(p_2 \to p_1 \to p_0)dA(p_3)dA(p_2) + ... \end{aligned} \tag{3.8}$$

where each term of the right side represent a path of increasing length.

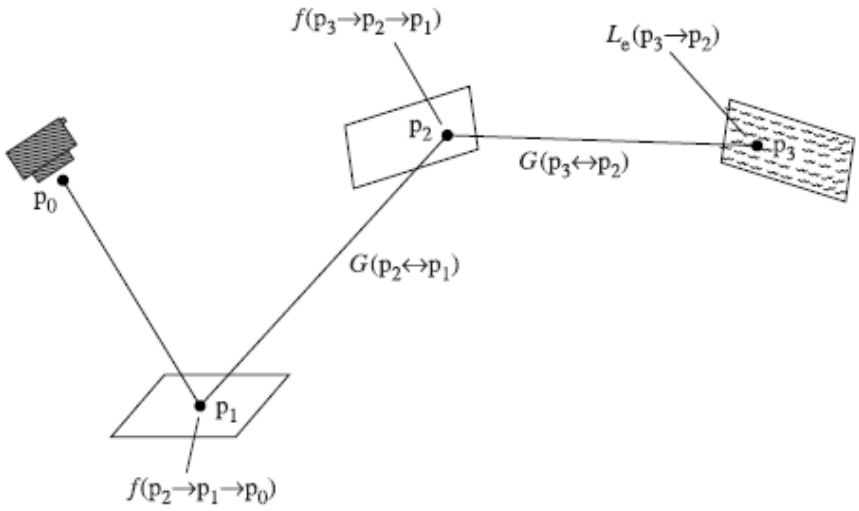


Figure 3.5: An example of a path of length 4 starting at the camera and ending at a light source.

The third term would de illustrated as in Figure 3.5. The total contribution of paths of length four (i.e. first vertex at the camera, 2 points at any surface of the scene, and a last vertex on a light source) is given by this term.

Here, the first two vertices of the path, p_0 and p_1, are determined based on the ray starting at the camera and the first point that it intersects, but p_2 and p_3 vary over all points on surfaces in the scene. The integral over all such p_2 and p_3 gives us the full contribution of paths of length four to arriving at the camera (p_0).

This infinite sum can be compacted as

$$L(p_1 \to p_0) = \sum_{n=1}^{\infty} P(\overline{p}_n). \tag{3.9}$$

$P(\overline{p}_n)$ gives us the amount of radiance scattered over a path $\overline{p}_n$ of $n+1$ vertices

$$\overline{p}_n = p_0, p_1, \ldots, p_n, \tag{3.10}$$

where p_0 is on the camera and p_n is on a light source.

Using Equation 3.9 (with a given length n), we can compute a Monte Carlo estimation of the radiance arriving at p_0 (the camera), due to path

of length n, sampling a set of vertices with an appropriate sampling density and evaluate an estimate of $P(\overline{p}_n)$ using those vertices.

3.2 Bidirectional Scattering Distribution Function (BSDF)

The term *Bidirectional Scattering Distribution Function* (BSDF) [Bartell et al., 1981] is not well standardized. However, it is often used to name the mathematical function describing the way the light is scattered by a surface. In practice, this phenomena is usually splitted into the reflected and the transmitted components, treated separately as *Bidirectional Reflectance Distribution Function* (BRDF) and *Bidirectiona Transmittance Distribution Function* (BTDF) respectively .

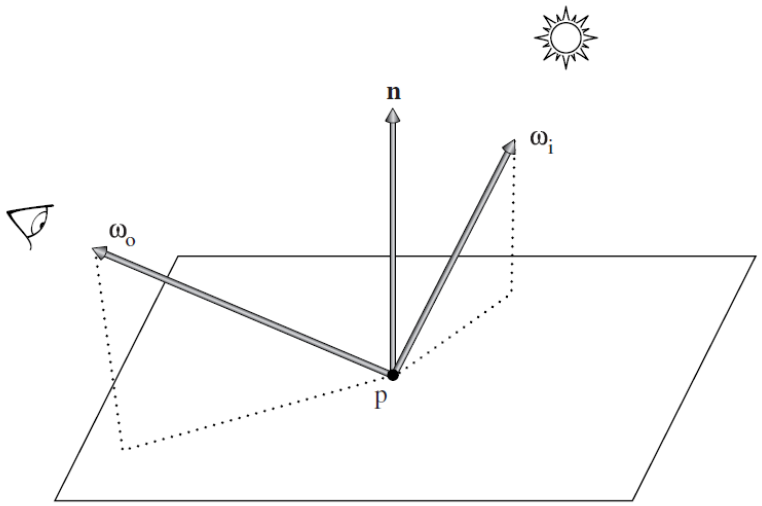


Figure 3.6: Example of the BRDF in a point of the space p

The BRDF function (usually written as $f_r(p, \omega_i, \omega_o)$) models the ratio of light reflected in a point given an incoming direction (ω_i) and a outgoing direction (ω_o) (Figure 3.6). Parameterizing each direction by azimuth angle Φ and zenith angle Θ we can understand the BRDF as a function of 4 variables.

Physically based BRDFs should have two important qualities:

- *Reciprocity:* For all pairs of directions ω_i and ω_o, $f_r(p, \omega_i, \omega_o) = f_r(p, \omega_o, \omega_i)$ (the value returned by the BRDF should be the same).

- *Energy conservation:* The total energy of the light reflected is less than or equal to the energy of the incident light. For all possible ω_o directions.

$$\int_{H^2(n)} f_r(p, \omega_o, \omega') \cos\theta' d\omega' \leq 1$$

The *Bidirectional Transmittance Distribution Function* (BTDF), which describes the distribution of transmitted light, can be defined in a similar way to that for the BRDF. Generally, the BTDF is denoted by $f_t(p, \omega_o, \omega_i)$ where ω_i and ω_o are defined in the opposite hemisphere around p than the ones in the BRDF. As an important remark, the BTDF does not obey the reciprocity defined above for the BRDF.

Our path tracer implements different BSDF models to cover a wide range of materials, and will be further discussed in Chapter 4.

3.3 Camera model

Our engine handles two camera models: **Orthographic** and **Perspective**. Note that all the renders shown in this document are rendered using the Perspective camera model. The first implementation made for both cameras supposes a pinhole camera model: a sealed box with one tiny hole allowing light to reach the sensor.

(a) Distortion effect

(b) Depth of Field effect

Figure 3.7: Different effect happening in real cameras due to the use of lens

Although this model is really simple to implement, it neglects many phenomena that happens in real cameras due to lens, such as *distortion* (Figure 3.7a) or *Depth of Field (DoF)* (Figure 3.7b) effects, that occur in real camera.

To achieve some of these effects that add realism to the final image, we have expanded our pinhole camera to account for the shape of the aperture (circular or polygonal with varying number of polygon sides), which naturally adds Depth of Field and Bokeh effects. An example render showing Depth of Field effect is shown in Figure 3.8

Figure 3.8: Standford dragon model rendered using our rendering engine with DoF effect

3.4 Sampling and Reconstruction

The final image captured by the camera is an array of pixels / color values, as a discretization of the underlying continuous function: the irradiance at the sensor. Thus, the way such continuous function is reconstructed from a set of discrete samples is critical for the quality of the final rendered image. In same way, if this process is performed wisely, we can reduce the number of distributed samples along the image, and consequently decrease the amount of computation needed to achieve high quality images.

To improve the sampling and reconstruction processes of our path tracer, we have implemented a **stratified jittered sampling** strategy to select the pixel real positions on camera space, and a **reconstruction filter** process that allow us to select the how we reconstruct the image given the pixel values generated by the path tracer.

3.4.1 Stratified jittered sampling

The stratified jittered sampling strategy combines both the jittered and stratified sampling strategies. Following the stratified sampling idea, we subdivide the pixel area in a 4x4 grid, an using a random generated number (Section 3.5.2), we select which one of the sub–pixels of the grid we will use to sampling.If we would have only implemented this sampling strategy, our final pixel position should be the center of the sub–pixel cell of the grid that we have selected using our random number as Figure 3.9b.

Using the jittered strategy, we would generate another random numbers to generate a uniform displacement in the pixel space coordinates. Implementing the sampling process using this strategy will lead us to select pixel positions with the pattern shown in Figure 3.9c

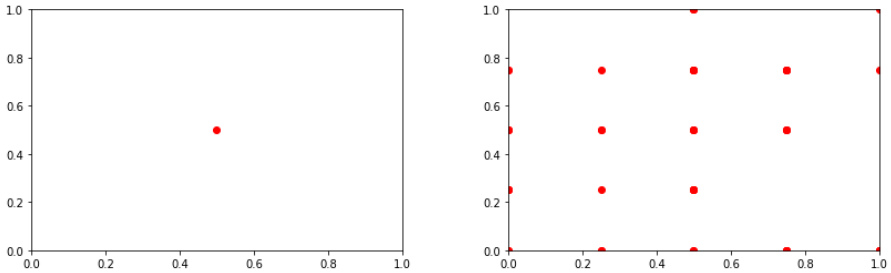
Finally, using with both strategies we could implement a sampling strategy where we first chose a sub–pixel from a 4x4 grid using the stratified strategy, and then we apply the jittered process in the sub–pixel space, generating the sampling pattern of Figure 3.9d.

This combined strategy is implemented in our rendering engine to generate a pixel position that we use as a starting point of the camera rays. The stratified jittered strategy allows us to reduce the aliasing in our final image and avoid other unwanted artifacts.

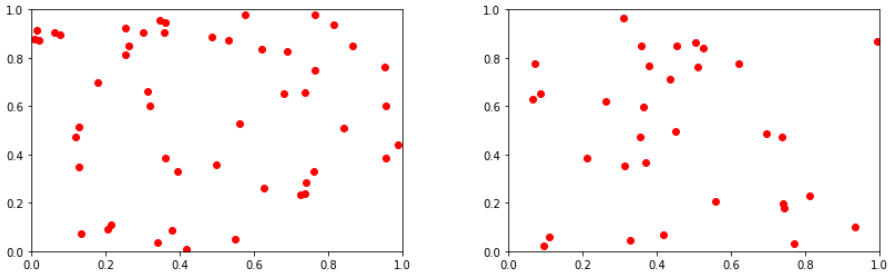
3.4.2 Reconstruction filter

To generate the final image from the radiance samples samples distributed randomly over the image plane, we have implemented a filter structure that allow us to implement different reconstruction filters to apply when we add radiance samples to final image.

Due to the parallel nature of the GPU, the filter is applied each time we generate one sample per pixel in the image, right before adding such new pixel values to the accumulated radiance image. For this, we need to save the real position of each pixel in camera space, which is not usually computed in GPU applications, and use it to evaluate the filter to obtain a weight for each sample.

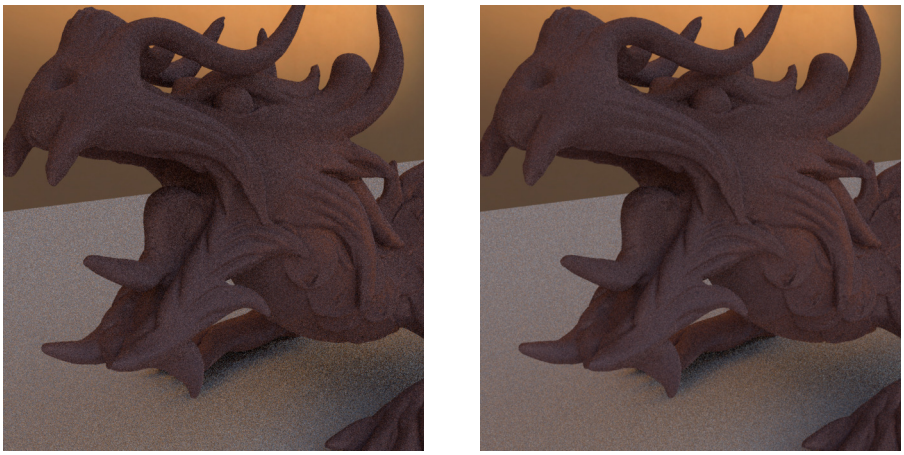


(a) Naive sampling (b) Stratified sampling



(c) jittered sampling (d) Stratified jittered sampling

Figure 3.9: Comparison of the different sampling strategies implemented in our path tracer



(a) Reference (b) Lanczos filter

Figure 3.10: We use Lanczos filter for reconstruction to reduce the overall noise in our rendered scenes

In our final path tracer implementation we have developed 2 filter aside from the box filter: **Gaussian filter** and **Lanczos filter**. After empirical tests, we decided to use the Lanczos filter implementation using parameters

$radius = 1.2$ and $\tau = 3$. (Pharr et al. [2016] and Jakob [2010] use those as default parameters) to generate the majority of renders shown in this Master Thesis, because it provides the best results in practice (Figure 3.10) and it also avoids aliasing between consecutive frames in videos.

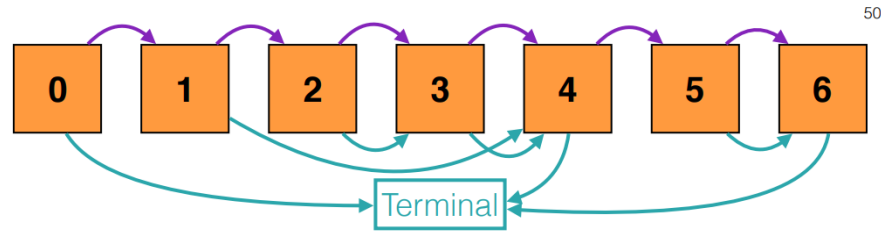
3.5 Performance and Implementation Details

Along this section we will discuss the proposed solutions to improve the performance of the rendering engine, as well as the most relevant implementation details, tightly related to the constraints that the GPU architecture imposes.

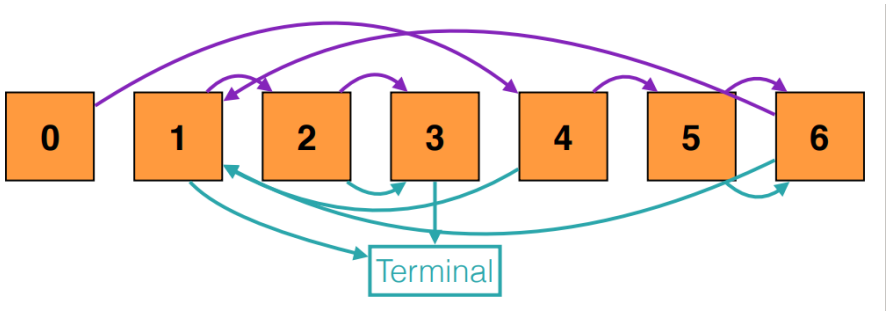
3.5.1 Acceleration Structures

One of the first things to take into account when implementing Path Tracing algorithms are acceleration structures to improve the speed of the ray intersection routine. The main reason to do this is because each successive interaction, with the geometry defining the scene to render, need to know which primitive will be hited first. If we reduce the time to compute this operation, the performance of our algorithm will dramatically increase.

Due to this, every efficient Path Tracer implements acceleration structures, which is usually some kind of tree that allows to quickly access any 3D point in the space, such as KD-trees [Jakob, 2010] or BVHs [Pharr et al., 2016]. In particular, our Path Tracer uses a GPU efficient Bounding Volume Hierarchies (BVH) implementation based on Hachisuka [2015] BVH for his GLSL photon mapper, converting the typical pointer-based BVH tree structure, generated in CPU with a Surface Area Heuristic (SAH), into a flat structure we can fit in OpenGL buffers. This allows us, both cache-friendly access to this structure and a stackless implementation of the traversal algorithm to query ray-triangle intersection into this BVH.



(a) Example of $+X$ face precomputation



(b) Example of $-X$ face precomputation

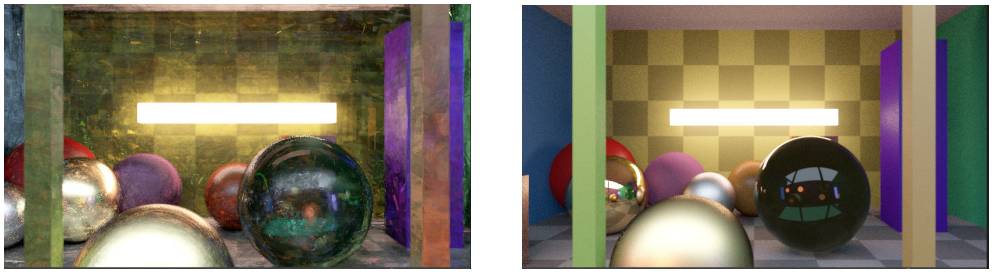
Figure 3.11: Comparison of different BVH faces computation using the same nodes. (Purple represent following node if hit was made, green if we missed the node)

As said, our structure is basically the same as the one presented by Hachisuka. We precompute the different exploration paths that could happen while doing the traversal in a the BVH structure after building it, and store each next node that the algorithm would have to visit depending on hitting the node or missing it as an explicit jump. The main change in our implementation is that we decided to aim for a stack–less structure as the one Pharr et al. [2016] presents, which seems more cache-friendly, while maintaining a different structure for the AABB using a cache-friendly organization only for storing the indexes we need to refer the different bounding boxes.

Hachisuka suggested to make this pre–computation for each of the different efficient trees that we could made for each of the sides of the Axis Aligned Bounding Box of the whole scene. We implement this idea while maintaining our cache–friendly structure, improving the ray–primitive intersection time without relocating the AABBs and triangle data in GPU memory, only generating the hit and miss structure per each side of the main AABB (Figure 3.11).

3.5.2 Uniform Random Generation

Random Number Generation (RNG) is really important when dealing with offline rendering implementations due to the continuous use of random number to generate events and different decisions in each collision. One of the main disadvantages of using the GPU for statistic computation is the generation of good quality random numbers, which turns important when implementing algorithms such as Path Tracing.



(a) Bad RNG (b) Good RNG

Figure 3.12: Comparison of the same scene with 2 different RNG algorithms

While many GPGPU applications have developed efficient pseudo-random number generation algorithms based on sine and cosine functions, the numbers generated heavily depend on current clock time, and have some stochastic correlation between adjacent threads, making them non desirable for Path Tracing. Thus, higher quality number generation is needed.

In our implementation, we have developed a RNG based on cryptographic hash [Tzeng and Wei, 2008] to generate floating point pseudo-random numbers. Using different random numbers generated previously in CPU as a starting seed for each pixel computation gives us the needed variability among pixels. It also provides enough random quality, as we need due to perform the different random–based functions that different materials need.

In our implementation, we use the MD5 algorithm mentioned above to generate random numbers in groups of 4 floating point numbers which are stored for further use (integer numbers and vectors are generated using this floating point numbers). With this preventive storage we avoid to execute the algorithm each time we need a random number. Instead, we generate 4 numbers at once using less computational resources than the ones we would need to perform the algorithm four times.

Also, we have decided to implement the hashing algorithm iterating 15 times the mD5 implementation, to avoid excessive compute time, because, as the reference papers demonstrates, the numbers generated using this amount of iterations are good enough for our implementations. In practice, these 15 iterations are implemented unrolled to avoid further computations costs.

3.5.3 State Saving

Unlike CUDA kernels, OpenGL shaders cannot spent an unlimited time doing computation on the GPU, since the hardware imposes time restrictions by design, to be used in real-time graphics. Because of this limitation, not only our implementation has to had each sample separated in different shader executions, but also each depth step of the same path, since long paths could spent enough computation time to cause the GPU driver to timeout, specially when the amount of triangle primitives increase drastically.

As a result of this depth subdivision, some kind structure to preserve the information of the path is needed, in order for the Path Tracer to continue the path correctly in the next shader execution. The data needed to recover the same state include:

- Ray origin.
- Ray direction.
- Accumulated luminance.
- Accumulated throughput.
- Depth of the path in last interaction.
- Last medium visited.
- Length (t) of last woodcock tracking (if not finished)

Table 3.1: Data from one to the next shader execution, compressed in 4 GLSL **vec4**. $[v0 \dots v3]$

	Scructure	Size	Position
Ray origin	vec3	12 bytes	v0.xyz
Ray direction	(θ, ϕ)	8 bytes	v1.xy
Luminance	vec3	12 bytes	v2.xyz
Throughput	vec3	12 bytes	v3.xyz
Path depth	float	4 bytes	v1.w
Last medium index	uint	4 bytes	v0.w
Length t	float	4 bytes	v1.z

With all this information we can recover the path as it was before the shader finished its execution, and so generate the same result as if we executed the algorithm without any interruption. The memory requirement and subdivision of this data is specified in Table 3.1. This information is stored textures using the Image Load Store OpenGL 4 feature that allows us to arbitrarily read from and write to texture, instead of the regular Render Target method used in previous OpenGL versions. This reduces the need of more shader passes to store data.

Chapter 4

Materials

Typically BSDFs and materials are implemented using inheritance present in programming languages such as C++. Due to the fact that inheritance is not possible in GLSL, we have implemented a system that allows us to refer materials using an unique numeric ID to select the correspondent function to be applied in each case. This allows us to have the typical structure for materials in the CPU part of our Path Tracer, while on the GPU each materials is only represented by this unique ID that any triangle on the scene has as a property. Using this ID we could decide the function to use whenever we want to shade some triangle, get the output direction of the ray after hitting some material, etc.

4.1 Diffuse material

The simplest material we have implemented is the smooth diffuse material (also known as Lambertian), allowing to create ideally diffuse surfaces by specifying a constant albedo value for a whole surface or a spatially varying albedo using a RGBA texture. We have implemented a cosine-weighted importance sampling for this material to improve the render time, and be able to use Multiple Importance Sampling [Veach, 1998]. This cosine-weighted importance sampling ends up giving us the following PDF:

$$PDF_{diffuse}(\theta_i, \theta_o) = \frac{\cos\theta_o}{\pi} \tag{4.1}$$

Just like in other materials, we have implemented the possibility of use albedo textures, so the final albedo can be calculated using both a base

albedo and the texture value as follows:

$$albedo = baseAlbedo * texture(UV)$$

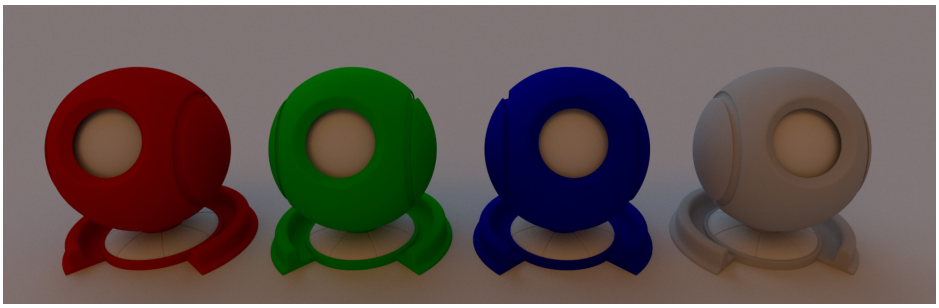


Figure 4.1: Rendered image using diffuse materials varying the albedo parameter

4.2 Disney principled BRDF

One of the complex BSDF that we have implemented is the Disney BRDF model explained by Burley and Studios [2012].This BRDF model, while not strictly physically correct, allows an art oriented specification of the materials because it uses intuitive parameters rather than physical ones.

Regarding the implementation details, Disney principled BRDF is composed by 5 simple components (diffuse, subsurface, anisotropic microfacets, clearcoat and sheen) which mixed using the weights that the model parameters specify create the whole material.

Diffuse component The diffuse model in this BRDF uses an empirical model, instead of using the default Lambert or the Oren and Nayar [1994] reflection model. This implementation reduces the diffuse reflectance by 0.5 at grazing angles for smooth surfaces, increasing the reflectance up to 2.5 for rough surfaces. The empirical model is defined as follows:

$$f_d(\theta_i, \theta_o) = \frac{baseAlbedo}{\pi}(1 + (F_{D90} - 1)F(\theta_i))(1 + (F_{D90} - 1)F(\theta_i)) \quad (4.2)$$

$$F_{D90} = 0.5 + \cos\theta^2 roughness \quad (4.3)$$

For the fresnel term, instead of using the full fresnel equations, the Disney BRDF uses the Schlick [1994] appoximation, defined as:

$$F(\theta) = (1 - \cos\theta)^5 \tag{4.4}$$

Subsurface component The Disney BRDF blends between the values of the diffuse model and a Hanrahan and Krueger [1993] based model to emulate subsurface scattering.

Due to this model being a very limited approximation, it only works for very short mean free paths.

Specular/Clearcoat component The specular lobe of the Disney BRDf uses a standard microfacet model defined as:

$$f_s(\theta_i, \theta_o) = \frac{D(\theta_i)F(\theta_i)G(\theta_i, \theta_o)}{4\cos\theta_i \cos\theta_o} \tag{4.5}$$

where D is the normal distribution, F is the fresnel term (Equation 4.4) and G is the shadowing term for the normal distribution, where the Disney BRDF uses the Generalized-Trowbridge-Reitz (GTR) [Trowbridge and Reitz, 1975]:

$$D_{GTR}(\theta_h) = \frac{c}{(\alpha_2 \cos^2\theta_h + \sin^2\theta_h)^\gamma} \tag{4.6}$$

Here, c is the normalization constant and α is the roughness value.

In the implementation, 2 specular lobes ares used, the main specular lobe with $\gamma = 2$ and the clearcoat lobe using $\gamma = 1$.

The BRDF does not specify the IOR explicitly, instead uses the **specular** parameter in the range [0..0.8] which maps to the IOR range [1..1.8]. For the clearcoat lobe the Disney BRDF uses a fixed IOR value of 1.5, and its strength is defined by the **clearcoat** parameter in range [0..0.25]. The roughness value of the lobes is defined as $\alpha = roughness^2$ for the main lobe, and $\alpha = lerp(clearcoatGLoss, 0.2, 0.001)$ for the clearcoat lobe. Finally, for the shadowing term G this BRDF uses a standard smith shadowing term [Smith, 1967], however it is modified due to artistic reasons.

Sheen component In order to add in *sheen*, observed in materials such as cloth, the following term is used:

$$f_{sheen} = F(\theta_h) \cdot sheen \cdot lerp(sheenTint, 1, baseAlbedo) \tag{4.7}$$

4.2.1 Importance Sampling

Aside from analytic evaluation, we have implemented importance sampling of this material using 3 importance sampling methods (diffuse, anisotropic microfacet and clearcoat). This 3 methods are sampled using the following samplable distributions:

- Cosine weighted hemisphere.
- GTR1 normal distribution.
- GTR2 normal distribution.

We select the distribution to sample taking into account the following ratios:

$$w'_{diffuse} = \frac{1 - metallic}{2},$$

where *metallic* represents the input parameter of the BRDF in range $[0 \dots 1]$, and

$$w'_{GTR2} = \frac{1}{1 + clearcoat},$$

where *clearcoat* is also a parameter of the BRDF in range $[0 \dots 1]$.

Here $w'_{diffuse}$ represents the ratio between the diffuse component and the 2 specular lobes and w'_{GTR2} the ratio between the first specular lobe and the clearcoat lobe.

Knowing this ratio we can compute the overall ratios of the 3 lobes to take into account when doing importance sampling and weighting the 3 PDFs as:

$$\begin{aligned} w_{diffuse} &= w'_{diffuse} \\ w_{GTR2} &= w'_{diffuse} + w_{GTR2} - (w_{GTR2} \cdot w'_{diffuse}) \\ w_{GTR1} &= 1 - w_{diffuse} - w_{GTR2} \end{aligned} \tag{4.8}$$

Some example renders of different materials using this BRDF could be seen in Figure 4.2.

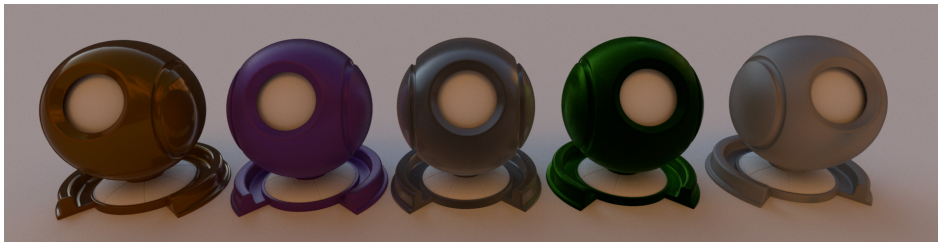


Figure 4.2: Comparison of multiple materials rendered using our implementation of the Disney principled BRDF

4.3 Measured Materials

Although in previous sections we have only talked about analytical models, another solution to represent a BRDF model is to explicit storage measured data from real materials.

Previous publications [Matusik et al., 2003, Ngan et al., 2005] have measured different real materials, and stored the data showing that is possible to generate renderized images similar to the original materials. We have decided to implement the materials measured by Matusik et al. [2003] due to the large dataset of different materials available to use from the measurement of their original work, including metals, plastics, different types of paints, etc.

In order to have importance sampling we calculate the tabulate CDF of each material we plan to use, uploading it to the GPU as bindless textures so we could perform a binary search through the data to select the proper output direction given a random number previously generated. The original publication found that specular peaks were difficult to represent using the natural coordinate system $(\theta_i, \theta_o, \phi_{diff})$ even using a denser representation, so they decided to use an alternative representation.

Due to this materials being BRDFs, and thus only being defined in the hemisphere, they can use an alternative representation based on the angles with respect to the half-angle of the incoming and outgoing directions, allowing them to vary the sampling density near the specular highligh.

While this representation reduces the size of the dataset, it differs from the actual coordinate system implemented in our GPU path tracer, so we considered to transform the data to our coordinate system to simplify the tabulated CDF construction and evaluation, but we ended using the original representation to avoid bigger memory occupancy because, as we observed, the main calculation overhead falls in the tabulated CDF calculation, which

we only calculate once per material in CPU.

Chapter 5

Volumetric Path Tracing

In Chapter 3 we talked about *Path tracing*, assuming that our scenes are made up of multiple surfaces in a perfect vacuum. This assumption means that radiance is constant along rays between surfaces. However, in many real–world situations this assumptions is inaccurate. Effects such as fog, smoke and even the color of the sky cannot be explained with this assumption. Due to how the light is affected as it passes through *Participating Media*, many effects as the ones mentioned above are generated.

In this chapter we will expose how this process is modelled, extending our *Path Tracer* model to *Volumetric Path Tracer*, and how we have implemented it on the GPU.

5.1 Volume Scattering Processes

Before introducing the *Equation of Transfer* in Section 5.2, we must know the different processes affecting the distribution of radiance in participating media. These three, are processes that affect the distribution of radiance in an environment with *participating media*:

- *Emission*: Radiance added to the environment coming from luminous particles.
- *Absorption*: Radiance reduction due to light converted into another form of energy, such as heat.
- *Scattering*: Radiance heading in one direction scattered to another direction as a result of the collision with particles.

In the following subsections, we will explain in more detail how each of them affect the radiance of a ray traveling through a medium.

5.1.1 Emission

As we mentioned earlier, emission increases the amount of radiance along a ray as it passes through a medium, this can be caused by chemical, thermal, or even nuclear processes converting energy into light. In Figure 5.1 we show the effect of emission in a ray, where $L_e(p, \omega)$ denotes the emitted radiance added to the ray (per unit distance) at point p in direction ω.

The following differential equation denotes the change of radiance due to emission, assuming that L_e (the emitted light) does not depend on the incoming light L_i [1]:

$$dL_o(p, \omega) = L_E(p, \omega)dt. \tag{5.1}$$

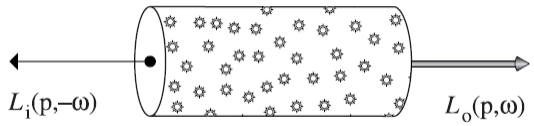


Figure 5.1: The emission process increases the radiance along the ray as it passes through a differential volume with emissive particles.

5.1.2 Absorption

While emission increases the amount of radiance along a ray passing through a medium, absorption decreases that radiance due to the particles in the participating media.

Absorption is described by the medium's *absorption cross section* (σ_a), this coefficient represent the probability density that light is absorbed per unit distance traveled in the medium. Generally, σ_a can change depending on both position p and direction ω, although normally only depends on the position (Also it is a spectral varying variable). The units of this *absorption*

[1] The assumption mentioned earlier, while not always true, remains valid under the linear optics assumptions made by many offline–render implementations (included our path tracer).

cross section are reciprocal distance (m^{-1}).

Figure 5.2 shows the absorption process along a very short segment of a ray. If some amount of radiance $L_i(p,\omega)$ arrives at point p, the exitant radiance $L_o(p,\omega)$ after absorption in a supposed differential volume by a ray follows the following expression:

$$L_o(p,\omega) - L_i(p,-\omega) = dL_o(p,\omega) = -\sigma_a(p,\omega)L_i(p,-\omega)dt.$$

This equation express that the differential reduction in radiance along the ray, is a linear function [2] of the radiance carried when arriving at point p.

We can solve this differential equation to obtain the integral expresing the total fraction of light absorbed by the ray

$$e^{\int_0^d \sigma_a(p+t\omega,\omega)dt},$$

assuming the ray travels a distance d in a direction ω, starting at point p.

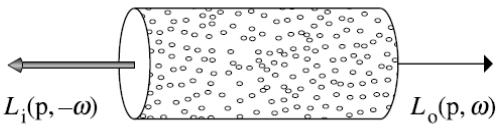


Figure 5.2: The absorption process, reduces the radiance along a ray through a participating media.

5.1.3 Scattering

As a ray passes through a medium, it may collide and be scattered in different directions, this effect is called *scattering*.

This process has two different effect on the radiance carried by the beam. First, due to some of the radiance being reflected to different directions, it effectively reduces the radiance existing in a differential region of the ray. This process is named *out–scattering* (Figure 5.3). Also, as result of this first effect, radiance from other beams can be scattered into the current ray. This effect is called *in–scattering*(Figure 5.4).

[2] This is other assumption. The fraction of light absorbed does not vary depending on the ray's radiance, instead it is always a fixed fraction

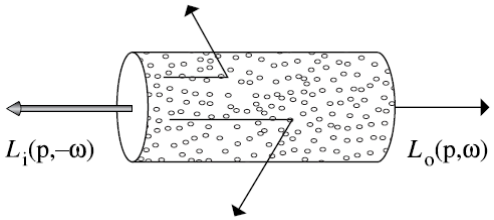


Figure 5.3: The out–scattering process reduces the radiance along a ray as well as absorption, but contrary to absorption, light that hits particles may be scattered in another directions.

Out–scattering

The probability of an out–scattering event happening per unit distances defined by the scattering coefficient (σ_s). This phenomenon is similar to absorption, because it reduces the radiance of the ray.

The reduction along a differential length dt due to this effect is:

$$dL_o(p,\omega) = -\omega_s(p,\omega)L_i(p,-\omega)dt$$

The combined effect of absorption and out–scattering is called *attenuation* or *extinction*, and is denoted as σ_t:

$$\sigma_t(p,\omega) = \sigma_a(p,\omega) + \sigma_s(p,\omega).$$

Related to this coefficients we can define the *scattering albedo* as

$$\rho = \frac{\sigma_s}{\sigma_t},$$

and describes the probability of scattering (instead of being absorbed) at a scattering event.

We can also define the *mean free path*, $1/\sigma_t$, that gives the average distance that the ray can travel before interacting with a particle of the medium.

In–scattering

As we have defined, opposed to the out–scattering effect, in–scattering increases the radiance due to the scatterig from toehr directions (Figure 5.4).

To better explain this process, we will asumme that the separation between particles is (at least) a few times the lengths of their radii, which allows us to ignore inter–particle interaction when we are describing scattering. If we follow this assumption, the *phase function* $p(\omega, \omega')$ (Section 5.3), which is the analog to the volumetric analog of the BSDF (Section 3.2), defines the angular distribution of scattered radiance at a point p.

However, the analogy is not exact. As an example, phase functions have the following constraint, for all ω

$$\int_{S^2} p(\omega, \omega') d\omega' = 1$$

must be true. With this constraint, phase functions actually define probability distributions for scattering in a particular direction.

The total radiance per unit distance caused by in–scatering is given by what we call the *source term* L_s:

$$dL_o(p, \omega) = L_s(p, \omega) dt.$$

This term account for both volume emission and in–scattering in the following way:

$$L_s(\mathrm{p}, \omega) = L_e(\mathrm{p}, \omega) + \sigma_s(\mathrm{p}, \omega) \int_{S^2} p(\mathrm{p}, \omega_i, \omega) L_i(\mathrm{p}, \omega_i) d\omega_i.$$

If we assume that there are no surfaces in the scene

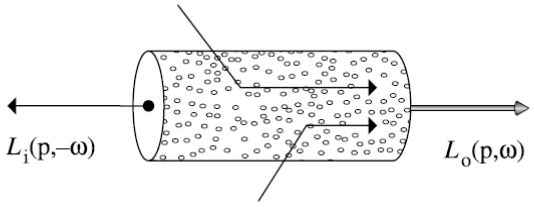


Figure 5.4: In–scattering process increases the radiance along the beam due to scattering of light from other directions.

5.2 Equation of Transfer

The *Equation of Transfer* is the fundamental equation that governs the nature of light in a medium that **emits**, **absorbs**, and **scatters** radiation.

The *Light Transport Equation* (Section 3.1) is, indeed, a special case of the *Equation of Transfer* with no *participating media* and specialized scattering from surfaces.

From the source term L_s introduced in Section 5.1.3 and the attenuation coefficient, $\sigma_t(p, \omega)$, introduced in Section 5.1.3, we can get the integro–differential[3] form of the equation of transfer:

$$\frac{\delta}{\delta t} L_o(\mathrm{p} + t\omega, \omega) = -\sigma_t(\mathrm{p}, \omega) L_i(\mathrm{p}, -\omega) + L_s(\mathrm{p}, \omega). \tag{5.2}$$

If we assume that rays are never blocked and have infinite length (there are no surfaces in the scene), we can rewrite the previous equation as a pure integral:

$$L_i(\mathrm{p}, \omega) = \int_0^\infty T_r(\mathrm{p}' \to \mathrm{p}) L_s(\mathrm{p}', -\omega) dt. \tag{5.3}$$

Here, the point $\mathrm{p}' = \mathrm{p} + t\omega$.

In a more generic way, if we have surfaces in the scene (reflecting and/or emtting light) rays do not necessarilly have infinite light and the surfaces hitted by the ray affect the radiance of the ray, adding o subtracting. If a ray coming from point p with direction ω hits a surface at point p_0 after a distance t, then the integral equation fo transfer is

$$L_i(\mathrm{p}, \omega) = T_r(\mathrm{p}_0 \to \mathrm{p}) L_o(\mathrm{p}_0, -\omega) + \int_0^t T_r(\mathrm{p}' \to \mathrm{p}) L_s(\mathrm{p}', -\omega) dt, \tag{5.4}$$

where $\mathrm{p}_0 = \mathrm{p} + t\omega$ is the point on the surface and $\mathrm{p}' = \mathrm{p} + t'\omega$ represent the points along the ray (Figure 5.5)

This equation describe all the effect that contribute to the radiance along the ray. The L_o term give us the emitted and reflected radiance from the surface, that can be attenuated (the ray transmitted account for this). The second term of the equation describes the radiance added along the beam as result of emission and scattering until the ray hits the surface.

[3]Due to to the integral over the sphere in the source term

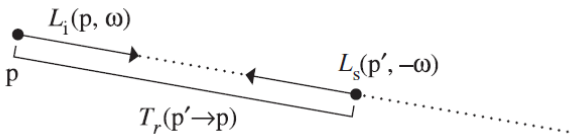


Figure 5.5: The incidence radiance is equal to the outgoing radiance coming from the surface time the transmittance to the surface added to the whole radiance from all the point in the ray from p to p_0.

But, to use in our Volumetric Path Tracer implementation, just as we did with the LTE in Section 3.1, we want to express the Equation of Transfer as a sum over paths of scattering events.

Using a similar approach, we can derive a medium–aware path integral, but the derivation is laborious, so we refer to [Jakob, 2013, Chapter 3] and Pauly et al. [2000] for derivation.

5.3 Phase Function

In Section 3.2 we introduced the BSDF concept to explain how light interacts with surfaces. Here we introduce the concept of phase function, which have been developed to explain how light interacts with the particles in media.

In most naturally occurring media, this *phase functions*can be defined by the angle θ between the two direction ω_i and ω_o (making them a 1D function, usually written as $p(\cos\theta)$).

We call this type of media *isotropic* due to their response to incident illumination being locally invariant to rotations. Aside from being normalized, the phase functions (as the BRDFs) should satisfy that they are reciprocal. In the isotropic case this is trivial because $\cos(-\theta) = \cos(\theta)$.

On the other hand, *anisotropic* media consist of particles arranged in a coherent structure. The phase function in this type of media can be a 4D function of the 2 directions (ω_i and ω_o). Examples of this are media made of coherently oriented fibers (as we will see in Chapter 6) or even crystals.

Aside from media, *phase functions* themselves can be isotropic or anisotropic. Thus is possible to have an anisotropic phase function in a isotropic medium.

An isotropic *phase function* describes equal scattering in all direction, making it independent of the two directions. Due to being normalized, there is only one function fulfilling this premise:

$$p(\omega_i, \omega_o) = \frac{1}{4\pi}$$

We have implemented this *phase function* as well as an anisotropic one for cloth fibers. Our anisotropic *phase function* model is fully explained in Section 6.2.

5.4 Homogeneous media

We use the term *Homogeneous media* to refer those media whose properties remain the same in their whole domain. Due to this the properties of the media stay the same in the whole domain so we onlye nee to store them once.

In practice, the limits of and homogeneous media domain are represented using some kind of boundary geometry, which we will refer as *stencil*. In our path tracer, this stencil is implemented as a tagged triangle geometry which allows us to change from regular path tracing (where we suppose vacuum) to volumetric path tracing. Also intersecting this stencil geometry allow us to obtain the maximum distance that a ray can travel through the media.

5.4.1 Sampling homogeneous media

When sampling an homogeneous media we must know that this kind of media follows an exponential behavior. Thus, we define the sampling methods for an exponential distribution defined over $[0, \infty)$. For $f(t) = e^{-\sigma_t t}$, it is

$$t = -\frac{\ln(1-\xi)}{\sigma_t}, \tag{5.5}$$

with PDF defined as

$$p_t(t) = \sigma_t e^{-\sigma_t t}. \tag{5.6}$$

However, σ_t (the attenuation coefficient) varies depending of the channel (varies by wavelength), and sampling multiple points in the medium is not

desirable, so a uniform sample is used to select a channel first. Then the corresponding scalar σ_t^i is used to sample a distance along the distribution

$$\hat{p}_t^i(t) = \sigma_t^i e^{-\sigma_t^i t}, \tag{5.7}$$

using the technique in Equation 5.5. The resulting sampling density is the average of the individual strategies p_t^i:

$$\hat{p}_t(t) = \frac{1}{n} \sum_{i=1}^{n} \sigma_t^i e^{-\sigma_t^i t}. \tag{5.8}$$

The probability of sampling an intersection at $t = t_{max}$ (the output stencil) is the complement of the probability of generating a medium scattering event between $t = 0$and $t = t_{max}$. This works out to a probability equal to the average transmittance over all the channel representing out Spectrum:

$$p_{surf} = 1 - \int_0^{t_{max}} \hat{p}_t(t)dt = \frac{1}{n} \sum_{i=1}^{n} e^{-\sigma_t^i t_{max}}. \tag{5.9}$$

Our implementation draws a sample based on this equations. If the sampled distance is before the next intersected primitive, a medium scattering event is generated a processed. Otherwise, medium interactions are ignored, and the next intersection event is processed.

5.5 Heterogeneous media

Although we have defined *Homogeneous media*, many *Patitipating media* such as cloth (Chapter 6), clouds and smoke cannot be modeled using this approach. Due to this, media where the properties vary among the space are needed to represent such effects.

In practice, this media are represented in a different way. Although we define the boundaries using the same stencil methods that we use in *homogeneous media*, we need a way to represent how the parameter vary. To represent this, we use a grid based representation of the parameters, storing values in the space occupied by the media using a 3D grid of a certain resolution, where each cell (or voxel) store a value and represent an homogeneous region of the space (Figure 5.6a).

Depending on the media, a grid with more resolution (more voxels to store information) will be need in order to a void artifact while rendering.

(a) Voxel representation (b) Final render

Figure 5.6: Example of a grid representation of an Heterogeneous volume, where each voxel store different values and the final volumetric render using this representation.

5.5.1 Sampling Heterogeneous Media

When dealing with heterogeneous media, extra effort is needed to deal with the mediums nature. First we will present the more basic way of sampling heterogeneous media stored in voxel representation, to talk after of another implementation that improve the performance of the sampling process.

Regular tracking

When the spatial variation of the *heterogeneous media* can be decomposed into uniform regions, a technique called *regular tracking* solves the heterogeneous media sampling problem by by applying standard *homogeneous medium* techniques to the voxels individually. This technique has the disadvantage that it becomes costly when the medium is stored using many voxels.

Ray marching

Other techniques build on a straightforward generalization of the homogeneous sampling PDF from Equation 5.6 using a spatially varying attenuation

coefficient:

$$p_t(t) = \sigma_t(t) e^{-\int_0^t \sigma_t(t')dt'}, \tag{5.10}$$

where $\sigma_t(t) = \sigma_t(p + t\omega)$ evaluates the attenuation t along the ray.

The most commonly used method for importance sampling Equation 5.10 is know as *Ray marching*.

This method approximate the cumulative distribution by subdividing the range $[0, t_{max}]$ into a number of subintervals, numerically approximating the integral in each interval, inverting this discrete representation.

Woodcock tracking

Instead of using a ray marching process to sampling the volumetric data, we have also implemented the Woodcock tracking algorithm [Woodcock et al., 1965] (also known as delta tracking) to improve performance, which allow us to adjust the ray marching step proportionally depending on the densities found among the path through the volume.

Assuming $\sigma_{t,max}$ (the maximum extinction throughout the medium), each woodcock tracking iteration i performs a standard exponential step through the uniform medium:

$$t_i = t_{i-1} - \frac{\ln(1 - \xi_{2i})}{\sigma_{t,max}} \tag{5.11}$$

starting with $t_0 = t_{min}$. These step process is repeated until we satisfy one of the two stooping criteria, $t_i > t_{max}$ (we have left the medium without any interaction) or the loop terminate with probability $\sigma_t(t_i)/\sigma_{t,max}$, the local fraction of "real" particles. Deciding between this two criteria consumes ξ_{2i}, the second random generated per iteration i of the algorithm.

Figure 5.7 shows the difference between delta tracking and the methods presented earlier.

5.6 Performance and Implementation Details

In this section we explain some implementation details about volumetric path tracing, focusing in *Heterogeneous media*.

We also specify some performance details about memory management on the GPU to store volumetric data.

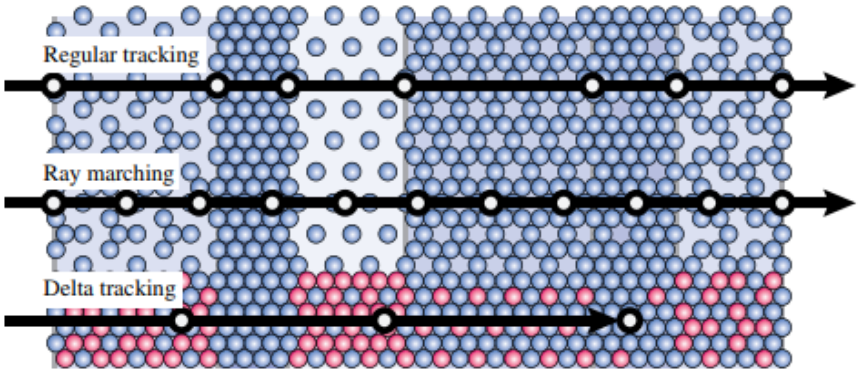


Figure 5.7: (top) Regular tracking partitions the medium into a number of homogeneous sub-spaces, relying on standard techniques for dealing with the regions supposing homogeneous media. (middle) Ray marching partitions the ray into a number of discrete segments and approximates the transmittance through each one. (bottom) Woodcock tracking considers a medium "filled" with additional "virtual" particles (in red) until it reaches uniform density. Image from Novák et al. [2014]

5.6.1 Memory scheduling

Rendering heterogeneous media requires a high amount of memory, due to the data storage of the grid containing the different data of each of the voxels of the volume. This problem gets worse if the volume is represented in a high resolution grid of voxels and the media has a high occupancy. Due to the memory limitations of the GPU we have mentioned earlier, storing all the volumetric data in the GPU at the same time becomes impossible in some cases.

To avoid this limitation, we have developed a scheduling system that subdivides the render process in different **RenderBatches**, that we can render separately. Each of this batches will only need to upload a subset of the volume data that will fit in the GPU memory. In order to get this batches, we perform an adaptative subdivision in screen space, checking the volume needed for each subdivision. If the memory requirement is bigger than we can handle the subdivision is split in 2 new ones and the memory requirement for each one is calculated.

To perform the adaptive subdivision we use a priority queue structure, ordering the **RenderBatches** by memory requirement so that we could perform the subdivision process before in those whith bigger memory occupancy. We execute this algorithm starting with the whole screen, so can end with a

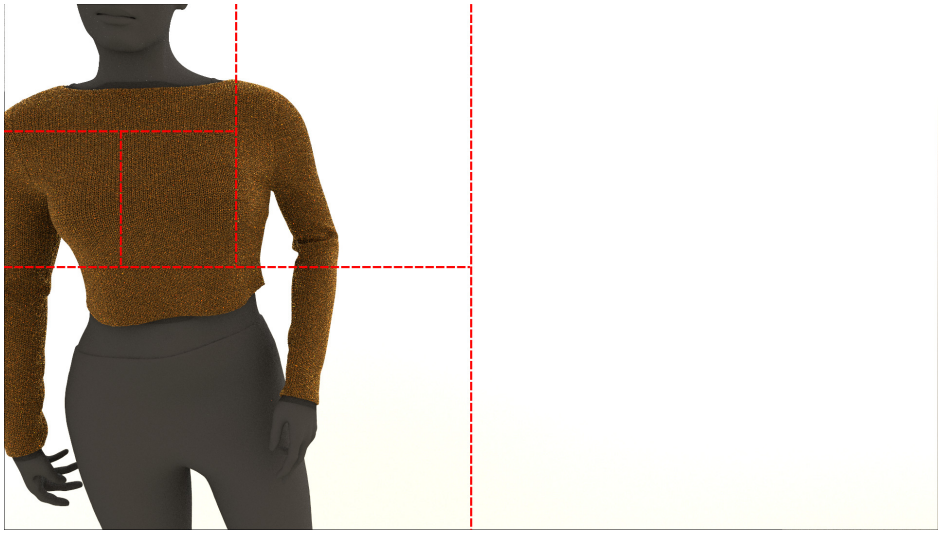


Figure 5.8: Example of the subdivision process for a example scene.

set of batches that we will render in sequence, resulting in the same output image as the one we could obtain doing al the render at the same time.

Although this subdivision process solves the memory limitation problem, it generates a whole new problem. When the ray traveling in the medium goes outside of the view frustum of the ScreenBatch, we do not have any volumetric information to use. Also, we do not know if we should have this information, or the empty voxels are correct.
To solve this problem we have developed a shading method that checks when paths traveling inside volumetric mediums exit the frustum of their respective ScreenBatch, we use the adjusted stencil (representing the cloth volume and its bounds) and simulate an homogeneous media with similar properties as the heterogeneous media. We repeat this process every-time we left the view frustum and return to use the heterogeneous model when we come back to view frustum.

Comparisons between the result generated without performing this solution and renders where this problem is solved are shown in the result section (Section 7).

5.6.2 Medium improvements

We also allow multiple medium render, and intersection of different mediums by selecting proportionally to their respective densities which one of the

intersected mediums will be sampled in the woodcock tracking process in order to obtain a density value.

Due to the limitations in execution time of the GLSL shaders, we cannot allow our algorithm to iterate indefinitely, but the implementation of the Woodcock tracking process can result in such type of execution in high density volumes media. To avoid GPU-driver timeout (and therefore unrecoverable crashes) we use the state saving process presented earlier to preserve the path state after a given number of iteration of the algorithm to avoid crashing. In the next shader step we can recover all the parameters to continue the algorithm, reaching the same result as if the algorithm was done in only one shader execution.

5.6.3 GPU grid implementation

As we mentioned earlier, we cannot implement a sparse second level for the grid in OpenGL using conventional CPU structures. Due to this, we need the OpenGL *Bindless Textures* to implement a sort of pointers to data in GPU.

Our structure consist on first dense level stored using *Shader Storage Buffer Objects* (SSBO) storing the bindless texture handle of 3D textures representing the second grid level.

Using this implementation we can reach a sparse representation by tagging empty bindless handle value (0 represented in integers of 64 bits), avoiding to generate the 3D texture representing an empty space.

Chapter 6

Volumetric Materials: Cloth

As many papers have discussed before, to render cloth in a proper way, a volumetric representation that allows us to have yarn level detail (at least) is needed. Due to this fact, implementing volumetric path tracing is mandatory if we want simulate the appearance of cloth in a realistic way.

To difference the volumetric interactions from regular Path Tracing, where we suppose perfect vacuum light traveling, between the different surface interactions we use a geometry wrapping the volume space that will not be rendered, but will be use to know when we need to start a ray marching process to sampling the medium.

6.1 Cloth Volumes

To storage the volumetric data representing the cloths we want to render using volumetric path tracing, we have implemented a 2 level grid storage in a similar way that the one implemented in Jakob [2010] to storage heterogeneous media.

This type of storage allow low memory consumption compared to a classical grid avoiding unnecessary storage due to low occupation.

The original 2 level grid used in this work is implemented in CPU, which have force us to redesign this type of structure to be used in GPU. A more detailed description of the implementation is presented in Section 5.6.3.

We use this grid stricture to storage the different parameters needed by our Fiber Scattering Model (Section 6.2), **fiber orientation** and **optical density**, so we can access with low computing cost to the parameters corresponding a 3D point of the space inside a cloth material.

To generate the volumetric data needed to represent real cloth, we have used a procedural generation model similar to the one presented by Zhao et al. [2016], starting from spline data generated using realistic cloth simulations.

This data is later voxelized (using a GPU voxelization method based on Lopez-Moreno et al. [2017]) to produce real volumetric data that we can store using the 3 level grid structure.

All these techniques combined with our scattering model (Section 6.2) allow us to have realistic volumetric data of enough resolution to execute volumetric path tracing achieving realistic results, as we show in our result section (Section 7)

6.2 Light Scattering for Textile Fibers

Previous papers have proposed phase function models to represent cloth light interactions. Zhao et al. [2011] proposed a microflake phase function to render cloth, but it does not represent scattering correctly. Schröder et al. [2014] presented a phase function derived from a previous one for hair fiber, which results in non-optimal behaviour, and Aliaga et al. [2017] modeled light scattering in a more detailed way, including non–cylindrical fibers.

We present a specialized phase function for cloth fiber rendering based on the phase function model proposed by Khungurn et al. [2015]. As the original one, our phase function provides an analytic evaluation as well as an analytic sampling, but we provide a 3 lobe representation instead of 2 lobe, allowing a more accurate representation of the real phenomena.

This 3 different lobes represent the 3 main sequences of light interactions happening in fibers:

R lobe Light reflected off the fiber surface. Specular reflection.

TT lobe Light transmitted into the fiber and then immediately out.

TRT lobe Light transmitted into the fiber and reflected internally before being transmitted out.

Each lobe have different parameters to adjust the base albedo and the shape of the lobe. Additionally, we can specify the energy transmitted from one lobe to the next set of lobes.

We use the same parameters as Khungurn et al. [2015], plus additional parameters to define the extra lobe that we have implemented.

C_R The base albedo of the R lobe.

C_{TT} The base albedo of the TT lobe.

C_{TRT} The base albedo of the TRT lobe.

$Fresnel_R$ The specular reflectance at normal incidence.

$Fresnel_{TT}$ Percentage of light transmitted again, once it is transmitted for the first time.

β_R Longitudinal width of the R lobe.

β_{TT} Longitudinal width of the TT lobe.

γ_{TT} Azimuthal width of the TT lobe.

β_{TRT} Longitudinal width of the TRT lobe

To ensure energy conservation in our phase function, we sample the function to calculate the integral for a given parameter combination. Using this integral, we can normalize the value computed analytically, so the value returned ensures energy conservation.

Although the integral computation have some significant cost, it only needs to be computed once the parameters have been chosen, allowing us to avoid further calculations when we are evaluating/sampling the phase function.

Table 6.1: Different phase function parameter sets tested.

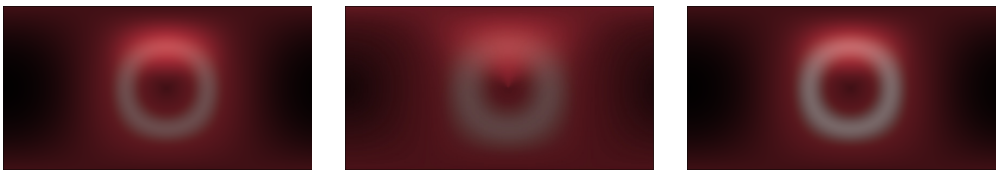
	β_R	β_{TT}	γ_{TT}	β_{TRT}	$Fresnel_{TT}$	$Fresnel_R$
Reference	6.0º	12.0º	220.0º	24.0º	0.85	0.01
Rougher	10.0º	20.0º	220.0º	40.0º	0.85	0.01
Shinier	6.0º	12.0º	220.0º	24.0º	0.85	0.04

For the sampling process we sample each lobe with a probability equal to the relative weight of that lobe, taking a outgoing direction in one of the 3 possible lobes of the phase function.

Although we cannot analytically sample each lobe, due to the fact that the Von-Mises distribution does not have an analytic CDF, we decided to fit a Gaussian distribution (with analytic CDF) to the Von-Mises, so we can

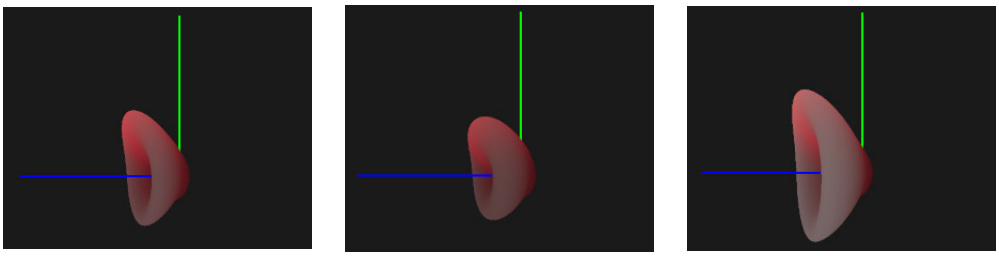
sample the CDF analytically and obtain the values of the Von-Mises distribution for each sample taken. This way we avoid to generate tabulated CDFs that will reduce the performance and result in a bigger memory requirement for the Path Tracer.

As an example of the expressiveness of our Phase Function we show 3 different configurations of our parameters, changing only the shape parameters (base albedos stay the same):



(a) Reference (b) Rougher (c) Shinier

Figure 6.1: Slice comparison of the output light at 45º, 0º configuration (θ, ϕ) for the parameters in Table 6.1.



(a) Reference (b) Rougher (c) Shinier

Figure 6.2: 3D comparison of the output light at 45º, 0º configuration (θ, ϕ) for the parameters in Table 6.1.

Chapter 7

Results

In this chapter we show the results of a set of different test scenes rendered using our volumetric path tracer. The scenes contain different lighting conditions, as well as a wide range of materials, ranging from analytical models, to measured BRDFs and complex heterogeneous volumes of cloth, also exploring the differences between some of the techniques implemented. All the renders shown in this chapter were generated using a computer equipped with a Intel(R) Core(TM) i7-7700k CPU with 4 cores (3.4 GHz), 32 GB of RAM and a NVIDIA GeForce GTX 1080 Ti GPU.

7.1 Disney BSDF

As stated in previous chapters, we have implemented the *Disney Principled BRDF*, as well as the analytic importance sampling techniques needed to quickly draw random samples from the this BRDF's distribution. The *Disney Principled BRDF* is currently used in production path tracing of the animated movies at *Walt Disney Animation Studios*, due to its capability to mimic many material appearances in a physically plausible way. It is a very expressive model, due to its 10 parameters, so that we generated several renders with varying combinations of such parameters to test our implementation.

Figures 7.1, 7.2 and 7.3 show the main variations of these parameters and how they affect the whole appearance of the material. Each column keeps all the parameters fixed while varying one in isolation subsurface, roughness, metallic, specular, specularTint, clearcoat, clearcoatGloss, anisotropic, sheen and sheenTint.

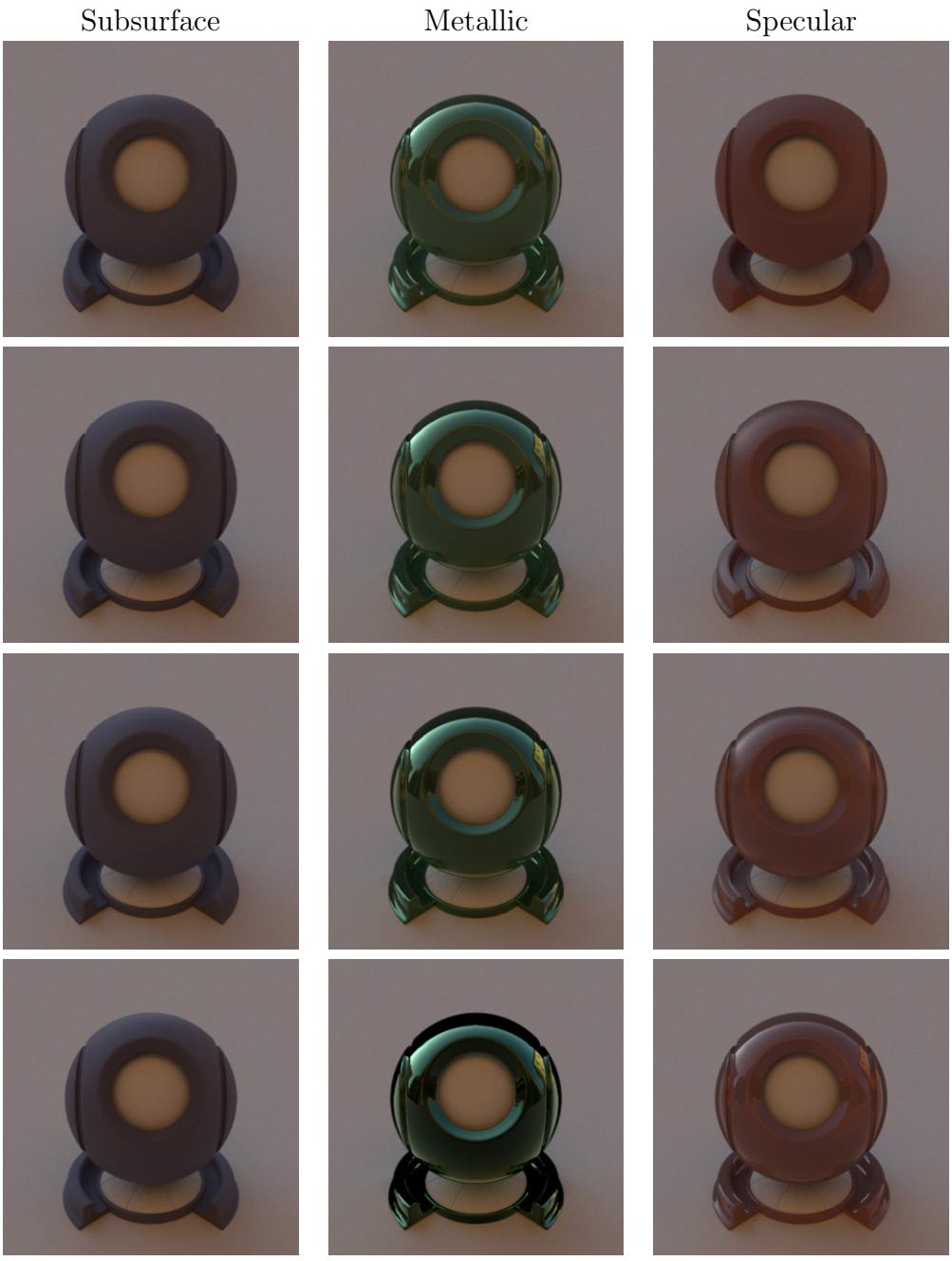


Figure 7.1: Rendered examples of the Disney Principled BRDF varying its *Subsurface*, *Metallic* and *Specular* parameters

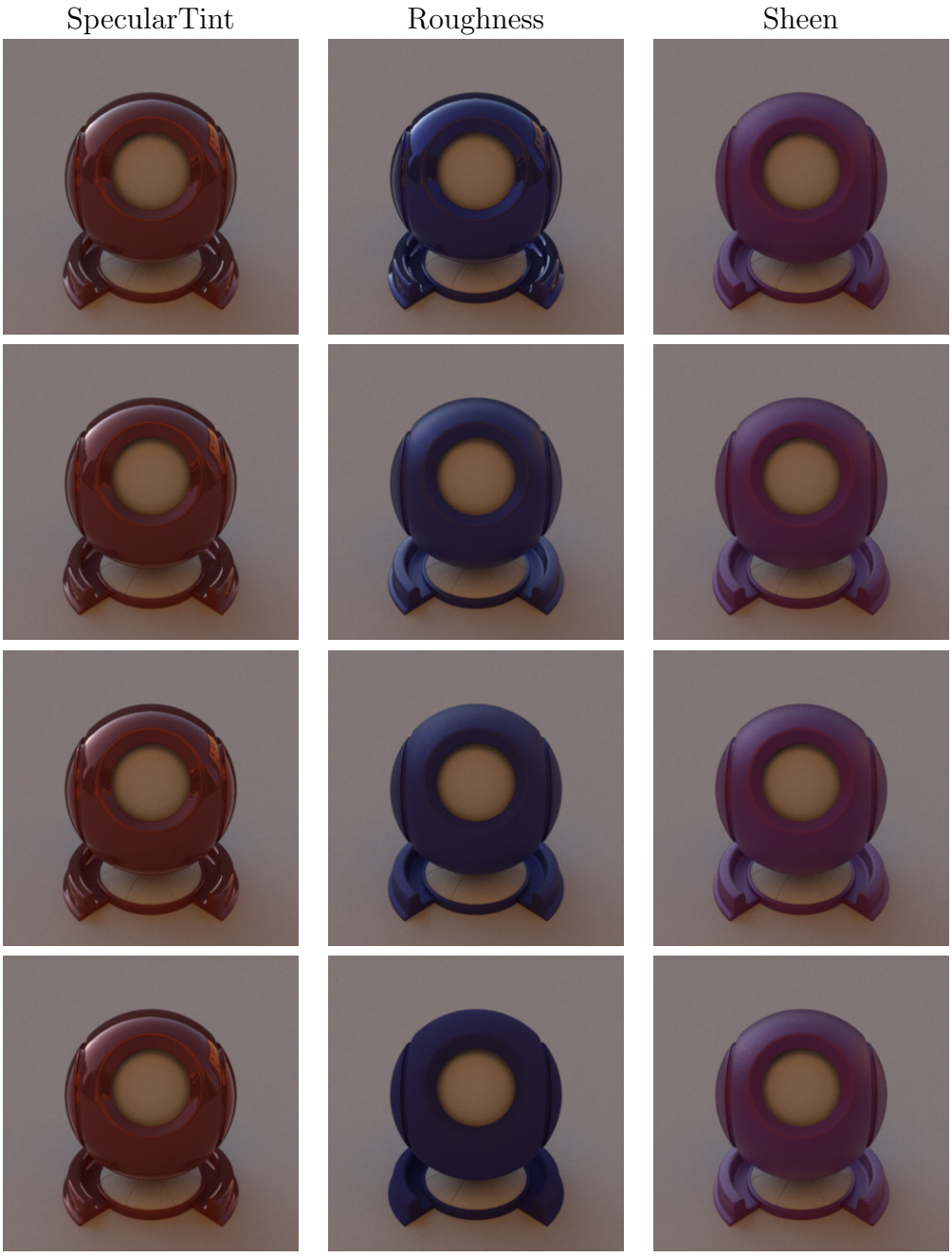


Figure 7.2: Rendered examples of the Disney Principled BRDF varying its *SpecularTint*, *Roughness* and *Sheen* parameters

Figure 7.3: Rendered examples of the Disney Principled BRDF varying its *SheenTint*, *Clearcoat* and *ClearcoatGloss* parameters

7.2 Measured materials

Another BSDF implemented our rendering engine is the measured BSDF presented by Matusik et al. [2003]. This BSDF allows to realistically replicate the appearance of the measured real–life counterpart. However, it implies great memory since the data for each of the material have been measured from the original materials with great accuracy. As an example of the realism that this BSDF could reach we show an example render using the **Gold Metallic Paint** material data provided among the original publication in Figure 7.4.

Figure 7.4: Example of the Standford dragon using the **Gold Metallic Paint** MERL BRDF.

7.3 Cloth materials

One of the main goals of this work was to achieve realistic renderings of cloth materials thanks to the use volumetric a representation of fabrics. Also, we aimed to use a expressive model for light scattering in cloth fibers. In this section we show how our volumetric path tracing can achieve high quality results when rendering scenes with complex cloth materials. As an example, in Figure 7.5 we show a 3 different result using three functions with similar parameters as the one shown in Section 6.2, but using spatially

varying albedo parameter for each lobe.

But, not only phase function model affects the appearance of cloth. We have tested different procedural cloth patterns, using the same volumetric model and the same phase function parameters. As Figure 7.7 shows, the cloth structure really affects the final appearance, although the optical properties of the fibers (phase function) remains the same. Also, different illuminations create great differences in the resulting image. We have generated different scenes using the same volumetric model, but changing the environment map used to obtain different illuminations, obtaining really different results (Figure 7.8).

(a) Backward (b) Isotropic (c) Forward

Figure 7.5: Example cloth scene rendered using the same light configuration changing the scattering parameters of our phase function implementation

7.4 Full garments

As we commented in Section 5.6.1, we have implemented a memory scheduler that allows us to subdivide the render process to be able to handle bigger data volumes. We have also commented the need of performing some process to avoid rendering artifacts when rays travel outside the view frustum. In such situations, we cannot correctly sample the whole volume due to the lack of information. In Figure 7.6 we show the result obtained with and without this improvement.

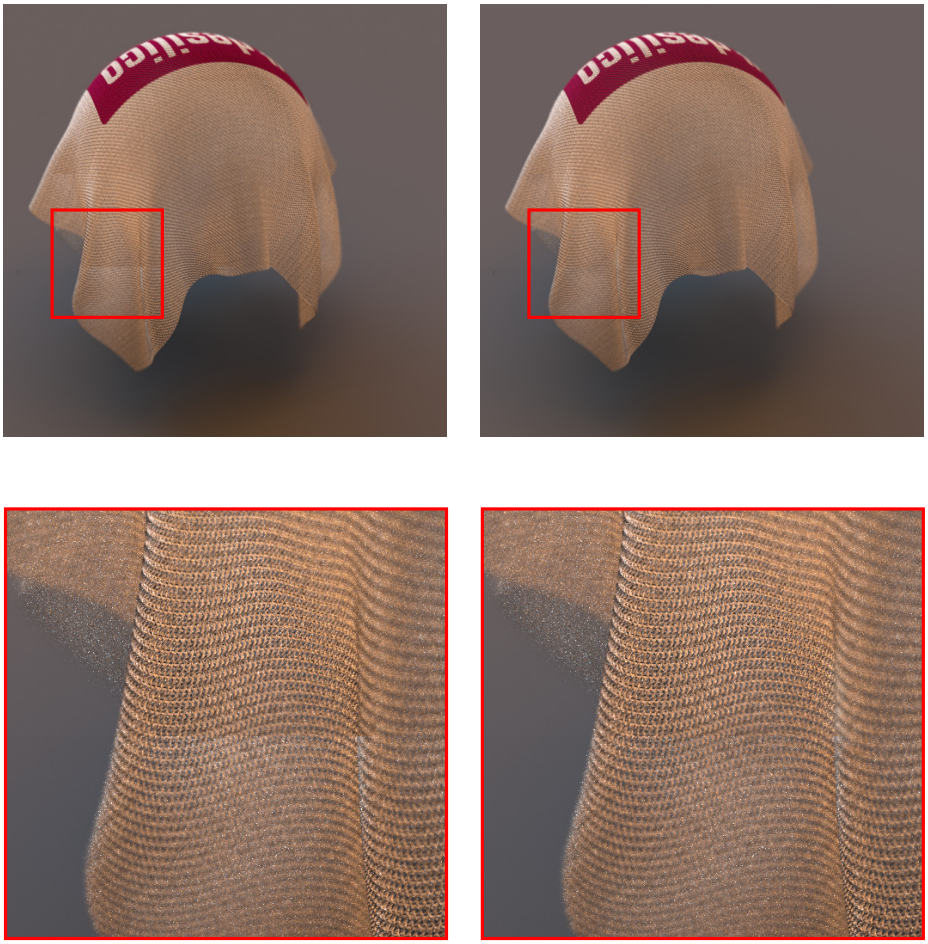


Figure 7.6: Comparison of renders with and without the screen–space subdivision. Left shows the naive subdivision, which creates artifacts due to incoherences in transmission along the volume.

Once this problem is solved we can render full garment scenes without artifacts related to memory limitation problems. We have tested multiple cloth scenes that do not fit entirely in the GPU if treated naively and that work perfectly when applying this new subdivision method. We have also tested the robustness of the system by checking if garments that fit in GPU without this screen–space subdivision are rendered correctly, and the results are indistinguishable compared to when rendered using our method. Most of the results of this test scenes are shown in Figures 7.9, 7.10, 7.11 and 7.12.

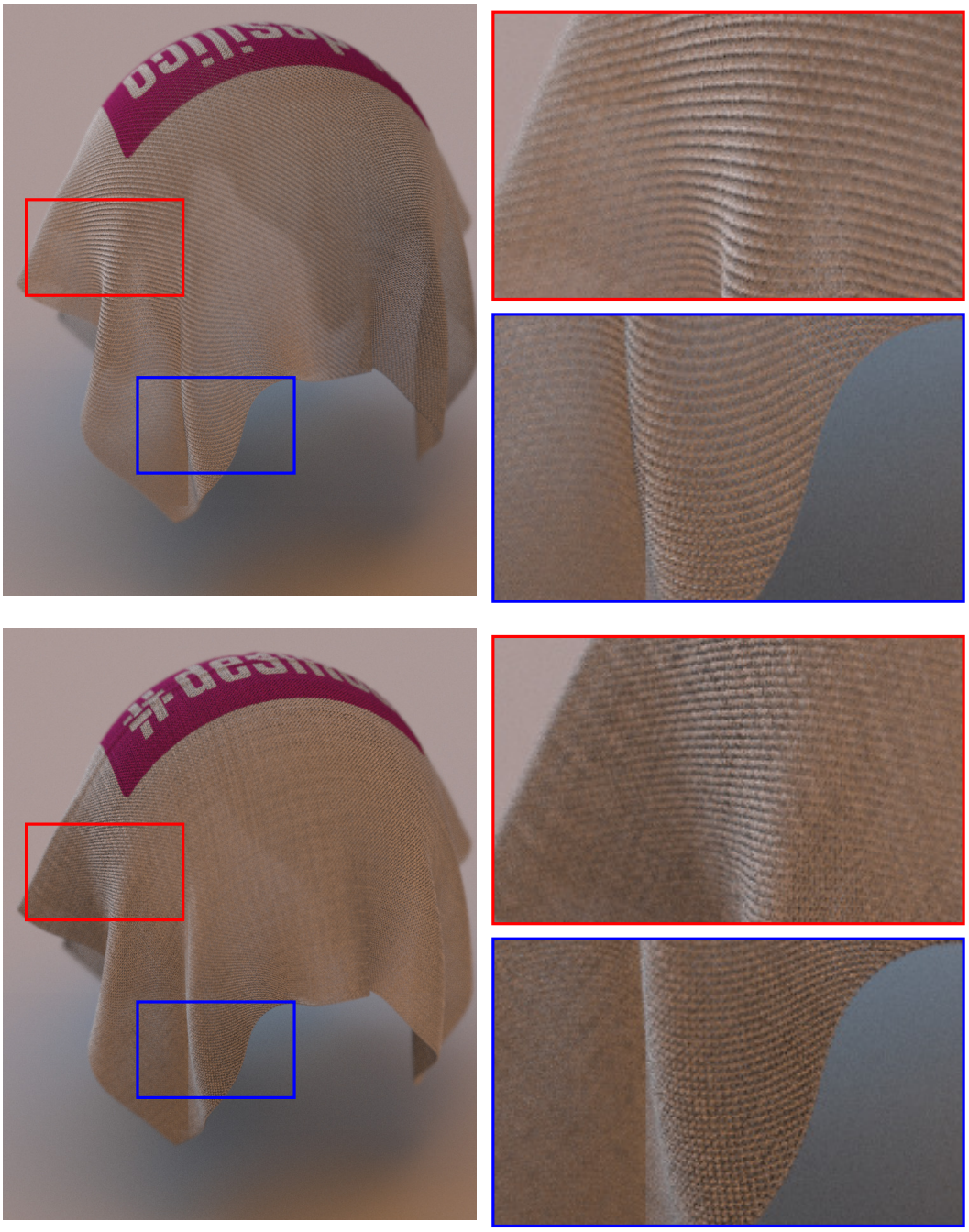


Figure 7.7: We have tested how different cloth patterns change the whole cloth appearance although the yarn material remains the same. Here we show how 2 different twill patterns (up and down) differ when rendered using the same scattering parameters.

Figure 7.8: Same volumetric model rendered using 2 different environment maps.

Figure 7.9: Red jersey close–up image generated using our rendering engin

Figure 7.10: Red shirt volumetric render using the model presented in this Thesis

Figure 7.11: Yellow jersey rendered using our volumetric path tracer implementation close–up render

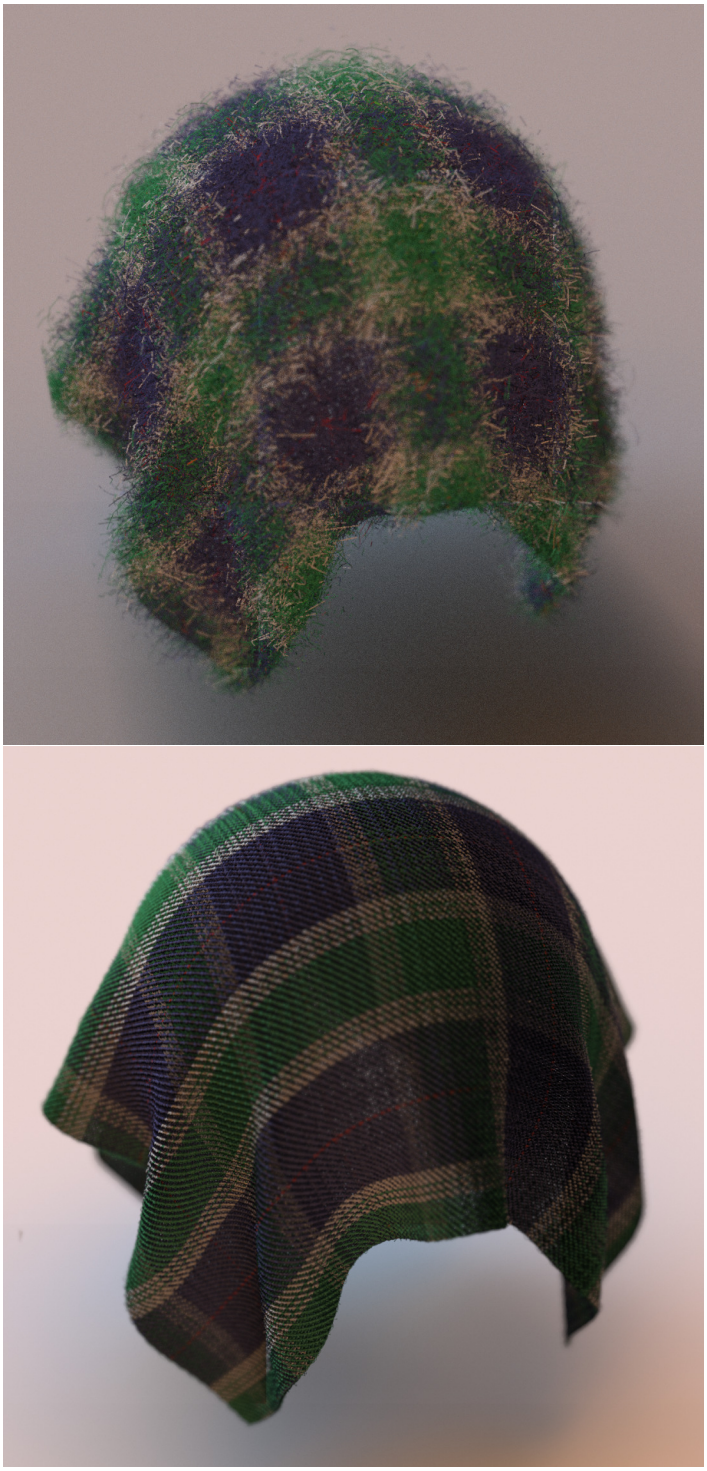


Figure 7.12: Cloth model with (top) and without (bottom) increased amount and length of fly–out fibers.

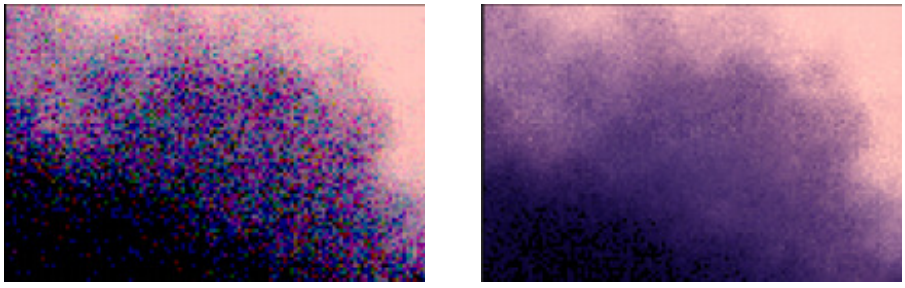
Chapter 8

Conclusions and Future Work

In this Master Thesis we have developed a volumetric path tracing implementation based on GPU, specialized in cloth rendering. Despite the fact that we can simulate very accurate renderings of both volumetric and non–volumetric materials, many improvements could be made to reduce render times and increase the quality of the results.
In this chapter we present some of those improvements, and explain how they would affect our work.

8.1 Improved woodcock tracking

Although we have implemented the Woodcock tracking algorithm, Novák et al. [2014] presented a new tracking technique based on the calculation of majorant values. Figure 8.1 show the differences when using their method instead of classic delta tracking.



(a) Delta tracking (b) Residual ratio tracking

Figure 8.1: Cloud rendered using Woodcock tracking algorithm and residual ratio tracking.

Their tracking technique could be really useful when performing importance sampling of light on participating media due to their method to evaluate transmittance in heterogeneous media. Also, this approach shows good result compared to classic delta tracking when dealing with complex heterogeneous media (such as cloth), so implementing this algorithm could drastically reduce the noise when sampling cloth materials, specially in scenes with different clothes.

8.2 Correlated media

Classic volumetric render, assume uncorrelated media with a uniform, random local distribution of particles. Jarabo et al. [2018] have proposed a framework to render homogeneous media taking into account the correlation between the scattering particles of the media (Figure 8.2, achieving result that cannot be rendered with uncorrelated media even changing the parameters of the media to try to adjust the model.

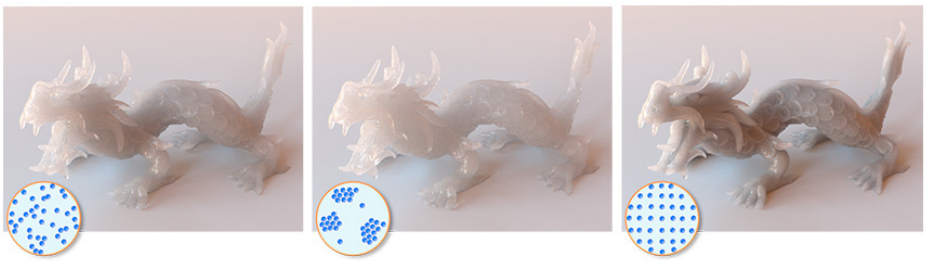


Figure 8.2: Jarabo et al. [2018] renders proving that the correlation between media particles drastically affect the final appearance of homogeneous materials.

Although this framework does not treat this problem in heterogeneous media, researching about the possibilities of this model to render heterogeneous media as correlated as cloth could end in more realistic render for many volumetric render of cloth that cannot be modeled using the uncorrelated model.

8.3 Improved scheduling algorithms

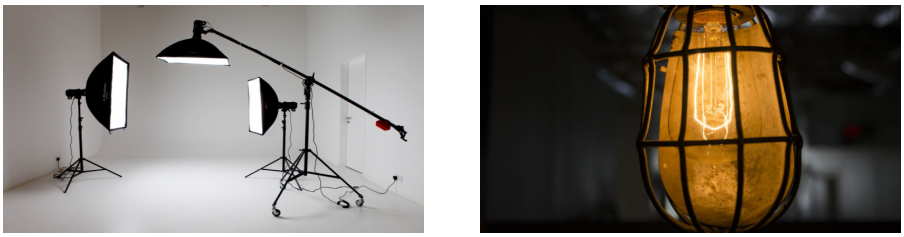
Although we present some scheduling process to be able to render high resolution volumetric materials that will not fit at once in GPU memory we do not perform any scheduling while doing the ray traversal process through the BVH structure. Implementing some kind of scheduling or ray-packaging, as many CPU render algorithms do, could improve the performance even more in GPU due to threads executing similar instructions. This type of scheduling could also improve the performance of the intersection process, which usually becomes the bottleneck of the computation for scenes with high resolution meshes.

However, complex scheduling algorithms cannot be implemented in GLSL as efficiently as in CUDA, so in order to implement such scheduling systems we will need to re-implement the path tracer algorithm using CUDA, which will lead to incompatibility with non NVIDIA systems.

8.4 Improve lighting

Our actual implementation only implements **Directional** and **Environment map** lights (with importance sampling).

Although environment light allow path tracing to generate very realistic light scenarios, some real–life scenarios imply other kinds of lights (Figure 8.3) that cannot be achieved using environment maps.



(a) Lights illuminating a studio (b) Light bulb

Figure 8.3: Different real–life light types that are not implemented in our path tracer

Implementing other light types such as triangles (and thus triangle meshes), quad lights or even disk lights [Guillén et al., 2017] would improve the lighting situations that we can simulate.

8.5 Conclusions

In this Master thesis we have proved, how GPU can be used to implement offline–render algorithms that have been classically implemented using CPU, improving their performance thanks to their parallel nature.

Although currently GPU languages are not really optimized to implement complex algorithms of such nature, technology is evolving to the extent that allow this kind of processes run faster on the GPU, while new standards for offline render are appearing for the use of GPUs in offline rendering, such as OptiX. Due to this, we should expect a progressive evolution of this kind of hardware facilitate the implementation of complex algorithms, aside from the ones used in real–time rendering.

On the other side, we have proven that using scattering model for fiber in cloth rendering allows renderings to achieve a high degree of realism as we show with our results. It is reasonable to assume that, in the future, more complex models to describe this types of materials will be researched using this kind approach producing even realistic results.

Bibliography

Carlos Aliaga, Carlos Castillo, Diego Gutierrez, Miguel A Otaduy, Jorge Lopez-Moreno, and Adrian Jarabo. An appearance model for textile fibers. In *Computer Graphics Forum*, volume 36, pages 35–45. Wiley Online Library, 2017. 7, 8, 46

FO Bartell, EL Dereniak, and WL Wolfe. The theory and measurement of bidirectional reflectance distribution function (brdf) and bidirectional transmittance distribution function (btdf). In *Radiation scattering in optical systems*, volume 257, pages 154–161. International Society for Optics and Photonics, 1981. 15

Brent Burley and Walt Disney Animation Studios. Physically-based shading at disney. In *ACM SIGGRAPH*, volume 2012, pages 1–7, 2012. 26

Ibón Guillén, Carlos Ureña, Alan King, Marcos Fajardo, Iliyan Georgiev, Jorge López-Moreno, and Adrian Jarabo. Area-preserving parameterizations for spherical ellipses. *Comput. Graph. Forum*, 36(4):179–187, July 2017. ISSN 0167-7055. doi: 10.1111/cgf.13234. URL https://doi.org/10.1111/cgf.13234. 63

T. Hachisuka. Implementing a Photorealistic Rendering System using GLSL. *ArXiv e-prints*, May 2015. 8, 20, 21

Pat Hanrahan and Wolfgang Krueger. Reflection from layered surfaces due to subsurface scattering. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 165–174, New York, NY, USA, 1993. ACM. ISBN 0-89791-601-8. doi: 10.1145/166117.166139. URL http://doi.acm.org/10.1145/166117.166139. 27

Eric Heitz, Jonathan Dupuy, Cyril Crassin, and Carsten Dachsbacher. The sggx microflake distribution. *ACM Trans. Graph.*, 34(4):48:1–48:11, July 2015. ISSN 0730-0301. doi: 10.1145/2766988. URL http://doi.acm.org/10.1145/2766988. 8

Wenzel Jakob. Mitsuba renderer, 2010. http://www.mitsuba-renderer.org. 7, 20, 45

Wenzel Jakob. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, 2013. 37

Wenzel Jakob, Adam Arbree, Jonathan T. Moon, Kavita Bala, and Steve Marschner. A radiative transfer framework for rendering materials with anisotropic structure. *ACM Trans. Graph.*, 29(4):53:1–53:13, July 2010. ISSN 0730-0301. doi: 10.1145/1778765.1778790. URL http://doi.acm.org/10.1145/1778765.1778790. 7, 8

Adrian Jarabo, Carlos Aliaga, and Diego Gutierrez. A radiative transfer framework for spatially-correlated materials. *ACM Transactions on Graphics*, 37(4), 2018. 62

J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. *SIGGRAPH Comput. Graph.*, 23(3):271–280, July 1989. ISSN 0097-8930. doi: 10.1145/74334.74361. URL http://doi.acm.org/10.1145/74334.74361. 8

James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986. ISSN 0097-8930. doi: 10.1145/15886.15902. URL http://doi.acm.org/10.1145/15886.15902. 9

Khronos Group. OpenGL, 1992. https://www.opengl.org/. 8

Pramook Khungurn, Daniel Schroeder, Shuang Zhao, Kavita Bala, and Steve Marschner. Matching real fabrics with micro-appearance models. *ACM Trans. Graph.*, 35(1):1:1–1:26, December 2015. ISSN 0730-0301. doi: 10.1145/2818648. URL http://doi.acm.org/10.1145/2818648. 7, 8, 46

Jorge Lopez-Moreno, David Miraut, Gabriel Cirio, and Miguel A. Otaduy. Sparse gpu voxelization of yarn-level cloth. *Comput. Graph. Forum*, 36 (1):22–34, January 2017. ISSN 0167-7055. doi: 10.1111/cgf.12782. URL https://doi.org/10.1111/cgf.12782. 46

Stephen R. Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan. Light scattering from human hair fibers. *ACM Trans. Graph.*, 22(3):780–791, July 2003. ISSN 0730-0301. doi: 10.1145/882262.882345. URL http://doi.acm.org/10.1145/882262.882345. 8

Wojciech Matusik, Hanspeter Pfister, Matt Brand, and Leonard McMillan. A data-driven reflectance model. *ACM Transactions on Graphics*, 22(3): 759–769, July 2003. 29, 53

Addy Ngan, Frédo Durand, and Wojciech Matusik. Experimental analysis of brdf models. In *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques*, EGSR '05, pages 117–126, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association. ISBN 3-905673-23-1. doi: 10.2312/EGWR/EGSR05/117-126. URL http://dx.doi.org/10.2312/EGWR/EGSR05/117-126. 29

Jan Novák, Andrew Selle, and Wojciech Jarosz. Residual ratio tracking for estimating attenuation in participating media. *ACM Trans. Graph.*, 33 (6):179:1–179:11, November 2014. ISSN 0730-0301. doi: 10.1145/2661229.2661292. URL http://doi.acm.org/10.1145/2661229.2661292. 42, 61

Michael Oren and Shree K. Nayar. Generalization of lambert's reflectance model. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 239–246, New York, NY, USA, 1994. ACM. ISBN 0-89791-667-0. doi: 10.1145/192161.192213. URL http://doi.acm.org/10.1145/192161.192213. 26

Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics*, August 2010. 8

Mark Pauly, Thomas Kollig, and Alexander Keller. Metropolis light transport for participating media. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 11–22, London, UK, UK, 2000. Springer-Verlag. ISBN 3-211-83535-0. URL http://dl.acm.org/citation.cfm?id=647652.732117. 37

Matt Pharr, Wenzel Jakob, and Greg Humphreys. Physically based rendering, 2016. http://www.pbrt.org/. 7, 20, 21

Christophe Schlick. An inexpensive brdf model for physically-based rendering. *Computer Graphics Forum*, 13:233–246, 1994. 27

Kai Schröder, Arno Zinke, and Reinhard Klein. Image-based reverse engineering and visual prototyping of woven cloth. *IEEE Transactions on Visualization and Computer Graphics*, PP(99), 2014. ISSN 1077-2626. doi: 10.1109/TVCG.2014.2339831. To be presented at Pacific Graphics 2014. 46

B Smith. Geometrical shadowing of a random rough surface. *IEEE transactions on antennas and propagation*, 15(5):668–671, 1967. 27

TS Trowbridge and Karl P Reitz. Average irregularity representation of a rough surface for ray reflection. *JOSA*, 65(5):531–536, 1975. 27

Stanley Tzeng and Li-Yi Wei. Parallel white noise generation on a gpu via cryptographic hash. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, I3D '08, pages 79–87, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-983-8. doi: 10.1145/1342250.1342263. URL http://doi.acm.org/10.1145/1342250.1342263. 22

Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford, CA, USA, 1998. AAI9837162. 10, 25

E Woodcock, T Murphy, P Hemmings, and S Longworth. Techniques used in the gem code for monte carlo neutronics calculations in reactors and other systems of complex geometry. In *Proc. Conf. Applications of Computing Methods to Reactor Problems*, volume 557, 1965. 41

Ling-Qi Yan, Chi-Wei Tseng, Henrik Wann Jensen, and Ravi Ramamoorthi. Physically-accurate fur reflectance: Modeling, measurement and rendering. *ACM Trans. Graph.*, 34(6):185:1–185:13, October 2015. ISSN 0730-0301. doi: 10.1145/2816795.2818080. URL http://doi.acm.org/10.1145/2816795.2818080. 8

Shuang Zhao, Wenzel Jakob, Steve Marschner, and Kavita Bala. Building volumetric appearance models of fabric using micro ct imaging. *ACM Trans. Graph.*, 30(4):44:1–44:10, July 2011. ISSN 0730-0301. doi: 10.1145/2010324.1964939. URL http://doi.acm.org/10.1145/2010324.1964939. 46

Shuang Zhao, Fujun Luan, and Kavita Bala. Fitting procedural yarn models for realistic cloth rendering. *ACM Trans. Graph.*, 35(4):51:1–51:11, July 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925932. URL http://doi.acm.org/10.1145/2897824.2925932. 46

Arno Zinke and Andreas Weber. Light scattering from filaments. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):342–356, March 2007. ISSN 1077-2626. doi: 10.1109/TVCG.2007.43. URL http://dx.doi.org/10.1109/TVCG.2007.43. 8