Light propagation simulation for the Wavelength-shifting Optical Module on CUDA GPUs

---

Florian Thomas

Johannes Gutenberg University Mainz

Faculty 08 - Physics, Mathematics and Computer Science
Institute of Physics
ETAP

Master Thesis - Master of Science in Computer Science

Light propagation simulation for the Wavelength-shifting Optical Module on CUDA GPUs

Florian Thomas

Primary Reviewer Prof. Dr. Elmar Schömer
Institute of Computer Science
Johannes Gutenberg University Mainz

Secondary Reviewer Prof. Dr. Sebastian Böser
Institute of Physics
Johannes Gutenberg University Mainz

October 25, 2019

Florian Thomas

Light propagation simulation for the Wavelength-shifting Optical Module on CUDA GPUs

Master Thesis - Master of Science in Computer Science, October 25, 2019
Reviewers: Prof. Dr. Elmar Schömer and Prof. Dr. Sebastian Böser

Johannes Gutenberg University Mainz
ETAP
Institute of Physics
Faculty 08 - Physics, Mathematics and Computer Science
Staudingerweg 7
55128 Mainz

Acknowledgements

First, I would like to thank Sebastian Böser and Elmar Schömer for giving me the opportunity to work on this project, for regular meetings, discussions and their sincere interest in my work. Furthermore, I would like to thank my supervisor Anna Steuer for helping me out whenever possible and reviewing an almost endless number of pages. Moreover, I would like to thank my group and the people in my office for the very nice atmosphere. Special thanks go to John Rack-Helleis for fruitful discussions, coffee breaks, classic cases of "angry looks" and the occasional "much more sophisticated".

Additionally, I would like to thank my friends Kati and Fabio, for enduring my constant complaints at times when my studies were tough. I would also like to thank my family and parents, not least because of the very important financial support that was willingly given for several years.

Finally, I would like to thank Sara for her endless support, her confidence in my abilities and all the patience during these final months when I was absent due to long working nights.

Abstract

This thesis presents the development, optimization and evaluation of a dedicated simulation tool for the Wavelength-shifting Optical Module (WOM). The WOM consists of a cylindrical quartz or plastic tube coated with wavelength shifting paint, light concentrators and photomultiplier tubes (PMTs). UV light striking the surface of the tube is absorbed, shifted towards longer wavelengths and guided to the PMT by total internal reflection within the tube wall. The PMTs convert the light signal into an electric signal, which is digitized and analyzed afterwards. The simulation presented in this thesis models the photon propagation within the tube wall and light concentrator and includes the simulation of photon interactions within the tube material, such as scattering and absorption. The simulation algorithm is based on a ray tracing approach and has been implemented for CUDA-enabled GPUs. It reaches a throughput of several million photons per second, which is three orders of magnitude faster than previous attempts using commercial software. The completed simulation has been used to analyze various properties of the WOM, which has confirmed theoretical models, such as a simplified model for the light capturing and guiding efficiency of the tube. Furthermore, the excellent performance enabled the application in a fit to experimental data.

Abstract (German)

Die vorliegende Arbeit beschäftigt sich mit der Entwicklung, Optimierung und Auswertung eines dedizierten Simulationsprogramms für das Wellenlängenschiebende Optische Modul (WOM). Dieses besteht aus einem mit wellenlängenschiebender Farbe beschichteten zylindrischen Rohr aus Quartz oder Plastik, Lichtkonzentratoren und Photomultipliern. Die beschichtete Oberfläche absorbiert UV Licht und verschiebt dessen Wellenlänge in den sichtbaren Bereich. Das wellenlängengeschobene Licht wird anschließend durch Totalreflexion innerhalb der Rohrwand zu den Photomultipliern geleitet. Letztere konvertieren das Lichtsignal in ein elektrisches

Signal, welches digitalisiert und danach ausgewertet wird. Die Simulation, die in dieser Arbeit vorgestellt wird, modelliert die Propagation von Photonen innerhalb der Rohrwand und innerhalb des Lichtkonzentrators. Sie beinhaltet auch die Simulierung von Wechselwirkungen der Photonen mit dem Rohrmaterial, wie etwa Streuung oder Absorption. Der Simulationsalgorithmus basiert auf einem Raytracing Ansatz und wurde auf CUDA unterstützenden Grafikprozessoren implementiert. Der Algorithmus erreicht eine Verarbeitungsmenge von mehreren Millionen Photonen pro Sekunde und ist damit drei Größenordnungen schneller als frühere Simulationen mit kommerzieller Software. Die fertiggestellte Simulation wurde verwendet um mehrere Eigenschaften des WOMs zu untersuchen, unter Anderem ein vereinfachtes Modell zur Beschreibung der Effizienz von Lichteinfang und Lichtleitung des Rohres. Des Weiteren hat die hohe Verarbeitungsmenge der Simulation einen Einsatz in einer Routine zur Kurvenanpassung an experimentelle Daten ermöglicht.

Contents

1 Introduction

1.1 Motivation and Problem Statement

The Wavelength-shifting Optical Module (WOM) is a novel ultraviolet (UV) sensitive single photon detector unit optimized for neutrino detection experiments. It has been developed in the context of IceCube-Gen2, the next generation of the IceCube Neutrino Observatory, which is a $1\,\mathrm{km}^3$ neutrino detector at the geographic South Pole.
Although the WOM has a development history of several years, its measurements have not yet been supplemented with extensive simulations. Therefore, this work aims at the development of a suitable simulation software for the WOM.

The need for a new simulation software dedicated to a single sensor might not be immediately apparent with other particle and photon tracking software – commercial and non-commercial – already existing. For instance, Geant4 [1] is a scientific toolkit commonly used for particle tracing. It provides a comprehensive simulation framework for particle-matter interactions including optics, has a long standing history since the 1970s and is one of the standard tools used in particle physics.
However, first attempts at implementing a simulation of the WOM with Geant4 revealed many pitfalls for setting up optical simulations and yielded inconsistent results [2]. Both can be attributed to Geant4 being mainly designed for the simulation of charged particles. Besides, it does not meet the performance requirements for the simulation of the WOM.
Apart from Geant4, efforts have been made [3] to simulate the WOM in the commercial package Fred Optical Engineering Software (FRED) [4]. Since FRED models much more sophisticated physical effects than needed in this specific use case, it reaches a simulation throughput of only 3000 photons per second [5], and can thus not provide the required statistics.

As Herb Sutter already pointed out in 2004, "The Free Lunch Is Over" [6], meaning that the physical limit for single-core processing performance has been reached. Instead, concurrency and parallelism of algorithms are crucial today in order to fully exploit the performance in multi-core environments. In recent years, the best possible parallel performance has been obtained with graphics processing units (GPU) for many scientific applications.

Even though the latest versions of Geant4 now support multi-threading [7], it still lacks GPU support and has a reputation for being the most time consuming part of simulation chains in particle physics experiments [8]. Similarly, the version of FRED used in previous attempts does not utilize the GPU.

A prototype simulation for the WOM implemented on GPUs was developed at JGU Mainz [9] and resulted in a significantly better performance. This work starts from scratch based on the knowledge gained from this prototype and aims at an overall improvement of the simulated physics, performance, usability and maintainability.

1.2 Neutrino experiments

The neutrino is an electrically neutral elementary particle in the Standard Model of particle physics. To date, there are many unanswered questions concerning neutrinos, making them an active field of modern particle physics research. For instance, the absolute mass and the mass ordering of the three flavors of neutrinos still have to be determined. Moreover, massive neutrinos and neutrino oscillations are not part of the Standard Model. Thus, their incorporation requires new theories, which have to be experimentally verified.

As neutral leptons, neutrinos only interact via the weak force and gravity. Interactions with matter have extremely small cross-sections, which makes them perfect messenger particles from astrophysical sources.
At the same time, the low interaction rate imposes an extreme challenge for their detection. As a result, acceptable event rates can only be achieved in large-volume detectors such as IceCube.

1.2.1 Cherenkov radiation

Interactions of neutrinos with matter produce charged leptons. If those are energetic enough, they can be detected via the so-called Cherenkov effect.
Cherenkov radiation is a result of highly energetic charged particles moving through a dielectric medium. A moving charged particle polarizes atoms in a medium. Subsequent de-excitation of atoms causes an emission of electromagnetic radiation (EMR). The charged particle can move faster than the EMR in that medium, since the phase velocity v_{ph} of EMR in a medium is reduced by the refractive index n of the medium:

$$v_{ph} = \frac{c}{n} \quad . \tag{1.1}$$

Here, c is the speed of light in vacuum. Hence, if the particle moves with $v > v_{ph}$, it is able to outpace the emerging waves. As a consequence, the emitted EM waves interfere constructively forming a conical wave front that can be detected.
The spectrum of Cherenkov radiation is given by the Frank-Tamm formula [10], which gives the number of emitted photons N per differential wavelength interval $\mathrm{d}\lambda$ and differential unit path length $\mathrm{d}x$:

$$\frac{\mathrm{d}^2 N}{\mathrm{d}x \mathrm{d}\lambda} = \frac{2\pi\alpha z^2}{\lambda^2}\left(1 - \frac{1}{(v/c)^2 n^2(\lambda)}\right) \quad . \tag{1.2}$$

Here, $z \cdot e$ denotes the charge of the particle, α denotes the fine-structure constant and $n(\lambda)$ is the wavelength dependent refractive index of the traversed medium.

1.2.2 IceCube

The IceCube Neutrino Observatory [11] uses $1\,\mathrm{km}^3$ of antarctic ice as detector material. The clear ice provides a medium of high transparency for optical photons. Figure 1.1 gives a schematic overview of the detector. It is instrumented with 5160

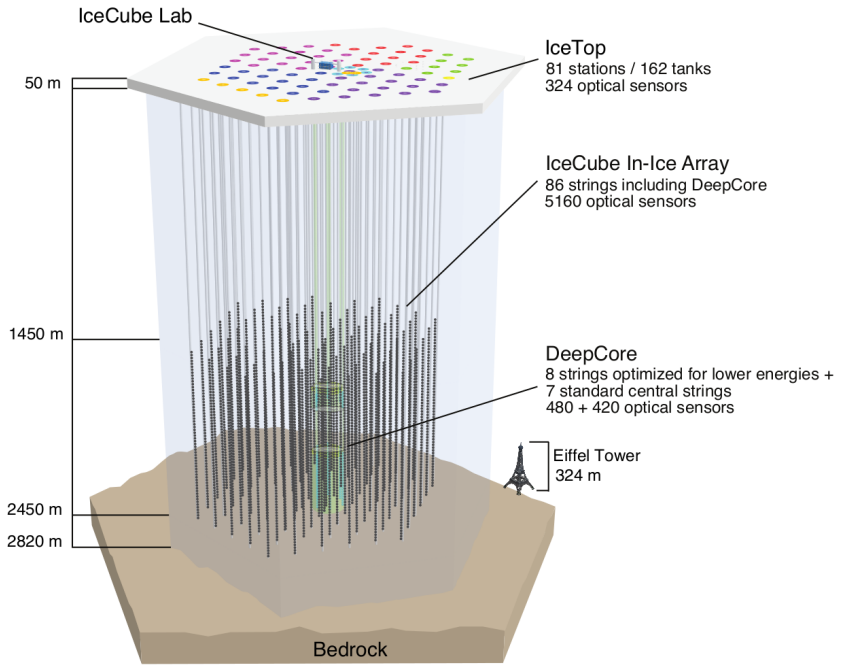


Fig. 1.1.: Overview of the IceCube Neutrino Observatory, including the DeepCore sub-array and the IceTop surface array. Source of picture: [11].

optical modules on 86 strings, which are arranged on a hexagonal grid with a spacing of $125\,\mathrm{m}$ between strings. On the strings, modules are deployed evenly spaced in a depth of 1450–$2450\,\mathrm{m}$ below the ice surface. Eight of the strings form the so-called DeepCore subarray. These strings are more densely instrumented with optical modules in the lower detector region, which results in a lower energy threshold, of

around $10\,\mathrm{GeV}$ neutrino energy. Above the surface, the IceTop array sits at roughly the same grid positions as the strings with two modules per point and serves as a cosmic-ray detector.

Digital Optical Module

Cherenkov light produced in neutrino interactions is detected by the Digital Optical Module (DOM). The DOM consists of a large PMT, its readout electronics and power supply; all enclosed in a spherical glass housing. The used borosilicate glass of the housing is only reasonably transparent (more than 50% transparency) for UV light above $340\,\mathrm{nm}$ [11]. The used Hamamatsu R7081-02 PMT is specified for the range of 300–$650\,\mathrm{nm}$, with peak quantum efficiency at around $420\,\mathrm{nm}$ [12].
However, the Cherenkov spectrum in ice (equation 1.2), which is shown in figure 1.2, has its peak photon yield far below that in the UV range. Therefore, the majority

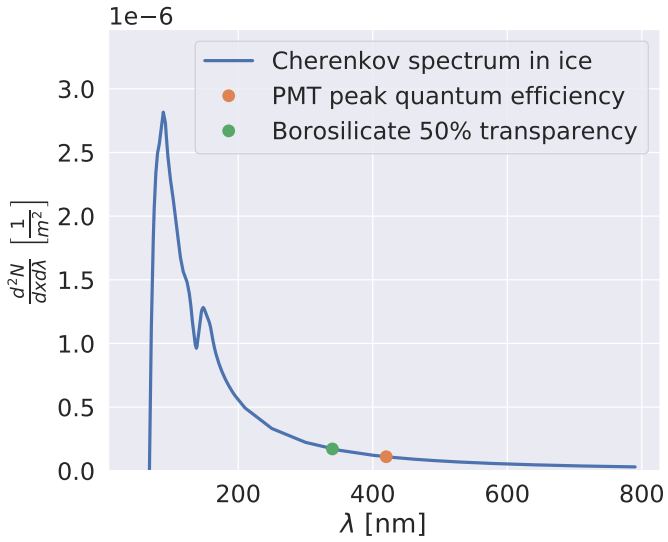


Fig. 1.2.: Cherenkov radiation photon yield per unit path length $\mathrm{d}x$ and per wavelength interval $\mathrm{d}\lambda$ plotted in blue assuming high energetic charged leptons. Thus, $z = 1$ and $v/c \approx 1$. $n(\lambda)$ uses data for ice at a temperature of $-7\,^{\circ}\mathrm{C}$ [13]. The wavelength-dependent photon attenuation length of the ice is not included here. The orange point marks the PMT peak quantum efficiency. The green point marks the border for 50% transparency of the borosilicate glass.

of Cherenkov photons in an event remains undetected.

1.3 The Wavelength-shifting Optical Module

The WOM [14] is schematically depicted in figure 1.3 . Its main component is a hollow cylinder made of quartz glass or Polymethyl methacrylate (PMMA). The cylin-

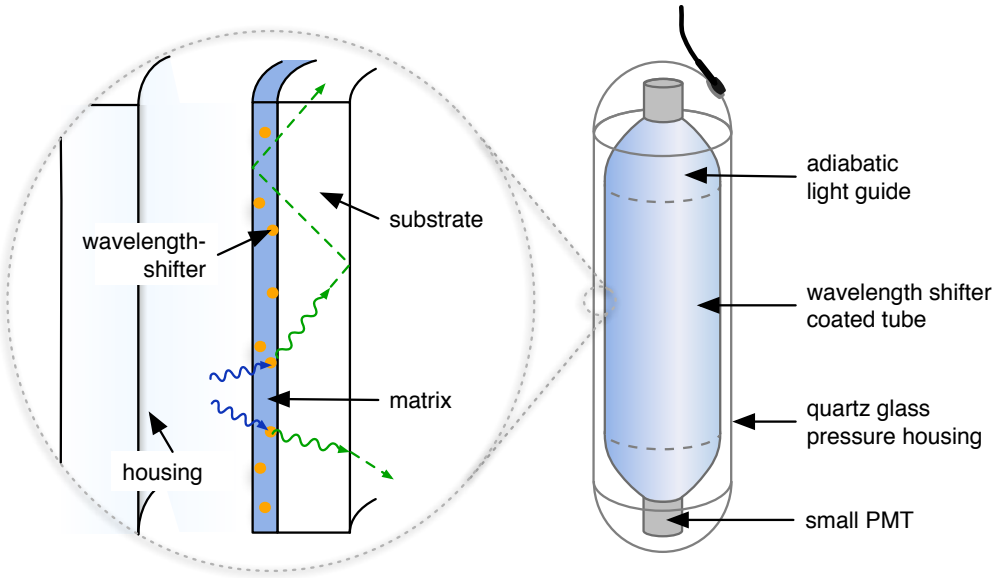


Fig. 1.3.: Schematic composition of the WOM. The closeup depicts the wall of the hollow cylinder and photon paths inside it.

der – or tube – is coated with a wavelength-shifting (WLS) paint. Photomultiplier tubes (PMTs) are attached at both ends via optical coupling. Optionally, so-called adiabatic lightguides (ALG) can be included between tube and PMT in order to concentrate the light for the utilization of smaller PMTs. The whole structure and the readout electronics for the PMTs are enclosed in a UV transparent quartz glass pressure vessel to provide protection in extreme conditions at possible deployment sites (e.g. pressures up to $690\,\mathrm{bar}$ during refreezing at IceCube).

1.3.1 Wavelength-shifting paint

The paint has been developed at the University of Bonn [15]. It contains a mixture of two wavelength-shifter molecules, where the wavelength-shift is a result of fluorescence: WLS molecules absorb photons of energy equal to the energy gap to an excited state $E_{abs} = \frac{hc}{\lambda_{abs}}$, where h denotes Planck's constant and λ_{abs} the photon wavelength. After some delay time, the excited molecule reverts to a lower energy state, emitting a photon with energy $E_{emit} = E_{abs} - \Delta E$. The energy difference ΔE arises from energy dissipated as heat or vibrations. Hence, the emitted photon has a wavelength $\lambda_{emit} > \lambda_{abs}$.
Figure 1.4 depicts the absorption and emission spectra of the paint used in the WOM. It is designed to absorb UV light and emit visible blue light with minimal overlap of both (large Stokes shift). The emission spectrum has its peak close to the peak of

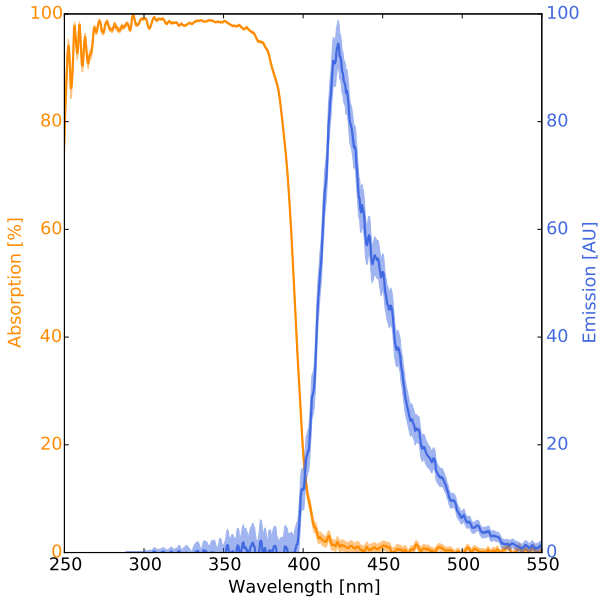


Fig. 1.4.: Absorption and emission spectrum of the WLS paint for the WOM.

the quantum efficiency of IceCube PMTs at $420\,\mathrm{nm}$. Therefore, employing this WLS paint facilitates UV sensitivity of the sensor down to $250\,\mathrm{nm}$ wavelengths.

1.3.2 Light guiding tube

For a scenario of a surrounding material with small refractive index, photons can be reflected at the quartz-air transition, the paint-air transition, or the paint-glass transition if the refractive indices of glass and paint differ. The emission of shifted photons is isotropic and photons have emission angle-dependent chance for total internal reflection (see chapter 2.1). The photons are captured in the tube wall and subsequently guided to the PMTs at both ends. More details on the tube geometry and its light guiding properties are discussed in chapter 3.5.
The effective area of PMT-based sensors like the DOM scales solely with the photocathode area of the PMT. However, the noise rate of a PMT is also proportional to its photocathode area. As a result, such modules have a nearly constant signal-to-noise-ratio (SNR). The WOM, on the other hand, increases the SNR by increasing the effective area via increasing the radius or the length of the passive tube, while keeping a small readout PMT. Due to the low event rate, a high SNR is an important property for neutrino experiments, since it facilitates lower energy detection thresholds with the same module density in the detector. A further increase in SNR can be achieved by using an ALG, which enables the utilization of even smaller PMTs.

To conclude, the WOM is an ideal candidate for the next generation of IceCube or other Cherenkov-based neutrino experiments due to its high SNR and the improved

UV sensitivity. Additionally, its small cylindric geometry allows for smaller drill hole diameters in IceCube. This saves time and costs in the installation of IceCube-Gen2.

1.4 Thesis Structure

Following this introduction, the thesis continues in chapter 2 with the theoretic backgrounds that are required for the simulation of light propagation and GPU computing. Next, chapter 3 covers the calculations for modelling the WOM geometry in the simulation. In chapter 4, the simulation algorithm and its implementation are discussed. Chapter 5 explains the performance optimization in order to maximize simulation throughput. Subsequently, chapter 6 presents results obtained from the final simulation. Last, Chapter 7 summarizes the thesis and provides an outlook to future work.

2 Theory

In this chapter we establish the theoretical background that is required for the simulation of the WOM. Section 2.1 treats the physical aspects of light propagation and its simulation, while section 2.2 deals with the technical aspects of computations on the GPU.

2.1 Simulation of light propagation

In the classical limit light refers to electromagnetic (EM) radiation of all frequencies. Visible light is thus only the small portion of light that can be perceived by humans in the wavelength range 400–$700\,\mathrm{nm}$ of the EM spectrum. As EM radiation, light propagation is described by the electromagnetic wave equation

$$\left(v_{ph}^2 \nabla^2 - \frac{\partial^2}{\partial t^2}\right) \boldsymbol{F} = 0 \quad , \tag{2.1}$$

which is derived from Maxwell's equations. Here, $\boldsymbol{F}$ is either the vector of the electric field $\boldsymbol{E}$, or the vector of the magnetic field $\boldsymbol{B}$. Hence, light propagation is the result of oscillations of both fields forming a traverse wave.
The phase velocity of the wave $v_{ph} = \frac{1}{\sqrt{\epsilon\mu}}$ depends on the permeability μ and the permittivity ϵ of the traversed medium. In vacuum, the phase velocity is a fundamental physical constant $c \approx 3 \times 10^8\,\mathrm{m\,s^{-1}}$. The refractive index $n = \sqrt{\mu_r \epsilon_r}$, is defined via the relative permittivity and permeability of the medium and it relates the speed of light in vacuum, to the propagation speed in a medium:

$$v_{ph} = \frac{c}{n} \quad .$$

In quantum theory, on the other hand, light is quantized into photons. Photons are uncharged, massless particles which are the quanta of the electromagnetic field. The photon can be associated with a wave packet of probability that propagates light in discrete portions. The energy of a photon is determined by the light frequency ν and the Planck constant h:

$$E = h \cdot \nu \quad . \tag{2.2}$$

Nevertheless, in order to simulate light propagation on a macroscopic level in a geometry that has a size of many times the wavelength, we can use a simplified model of light employing the ray tracing algorithm.

2.1.1 Ray tracing

The ray tracing algorithm originates in 3D computer graphics. It was first introduced in 1968 [16] as a hidden-surface determination algorithm, which is required to solve the visibility problem that arises in image rendering. It is based on geometrical optics, thus describing light as lines that are perpendicular to the wavefronts of light waves.
To render an image, light rays are sent from an imaginary viewer through the pixels of an image plane (see figure 2.1). The pixel colors are determined by calculating

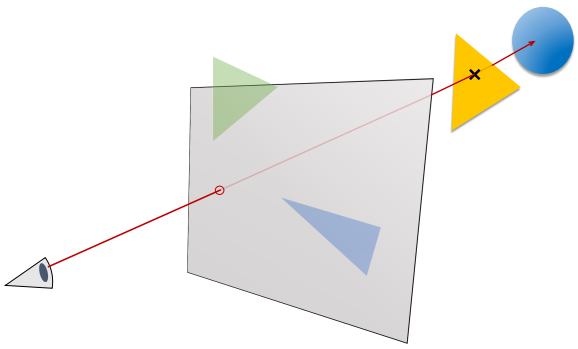


Fig. 2.1.: A viewer looking at a scene through a screen.

eventual intersection points of the rays and all objects in the scene. The closest existing intersection point on a ray is the one that is actually visible for the viewer. Compared to the real world, this is certainly the wrong way round. However, in the realm of geometrical optics the direction of the ray does not matter. Therefore, calculating only rays that can reach the viewer is a lot more efficient.
This simple core algorithm for solving the visibility problem can be extended to simulate various additional effects. For instance, recursively repeating the algorithm allows for the simulation of reflections, refractions and casting of shadows, which are the foundation for realistic rendering. All these effects are difficult to simulate using conventional rendering techniques. Thus, realistic renderers aiming at high quality images are usually based on the ray tracing algorithm.

The simulation of the light propagation for the WOM is also based upon the recursive ray tracing algorithm to simulate the paths of light. However, unlike renderers, it emits the rays from the light source instead of the detector and it does not calculate

pixel colors as result. Furthermore, the simulation uses extensions for physical correctness beyond geometrical optics where necessary.

2.1.2 Reflection and transmission at interfaces

Interfaces of two media with distinct refractive indices partially reflect incident light and partially transmit it. Both processes are quantitatively described by the Fresnel equations [17]. They determine the reflectance R, which is the ratio between the power of reflected light and the power of incident light depending on the incidence and transmission angles relative to the normal of the interface (see figure 2.2(a)). For the analogously defined transmittance it applies that $T = 1 - R$ due to conservation of energy. Depending on the polarization of light, the Fresnel equations yield two different expressions for the reflectance:

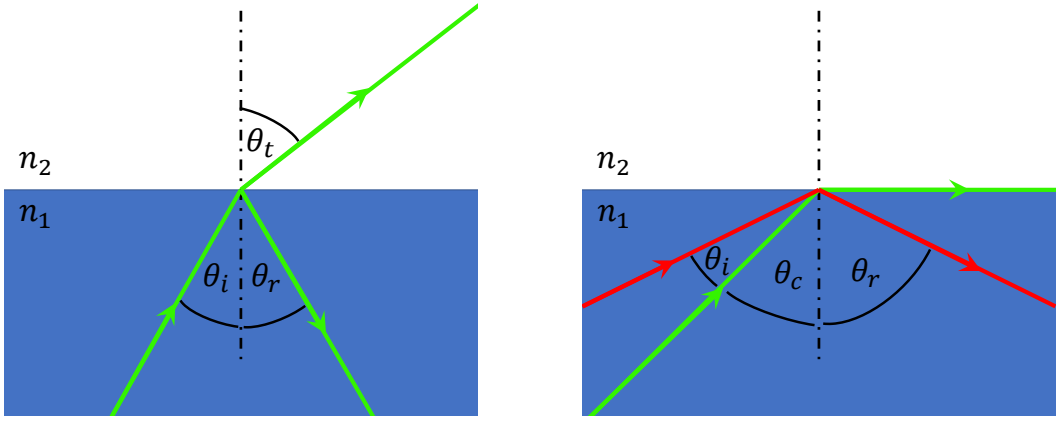
$$R_s = \left| \frac{n_1 \cos(\theta_i) - n_2 \cos(\theta_t)}{n_1 \cos(\theta_i) + n_2 \cos(\theta_t)} \right|^2 \tag{2.3}$$

$$R_p = \left| \frac{n_1 \cos(\theta_t) - n_2 \cos(\theta_i)}{n_1 \cos(\theta_t) + n_2 \cos(\theta_i)} \right|^2 \tag{2.4}$$

The transmitted part of light is refracted, causing a change of its angle relative to the surface normal (see figure 2.2(a)). The transmission angle θ_t after refraction is determined from the incidence angle θ_i and the refractive indices n_1 and n_2 of both media via Snell's law:

$$\sin(\theta_i) n_1 = \sin(\theta_t) n_2, \tag{2.5}$$

The reflection angle θ_r, on the other hand, is the same as the incidence angle θ_i.

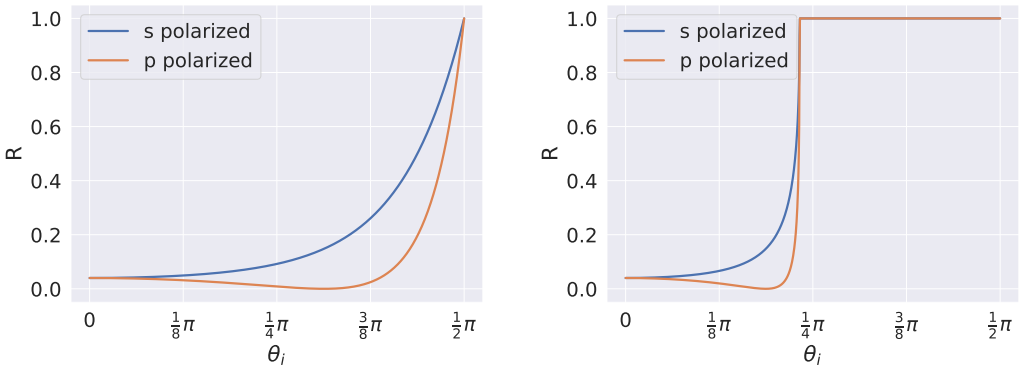


(a) Incident light reflected and transmitted at an interface with $n_1 > n_2$.

(b) Total internal reflection at an interface with $n_1 > n_2$.

Fig. 2.2.: Light reflection and transmission at an interface.

Figure 2.3 shows the reflectance of both polarizations according to equations 2.3 and 2.4 using Snell's law for θ_t for transitions at a glass-air interface. For small angles,



(a) $n_1 = 1.0$ and $n_2 = 1.5$.

(b) $n_1 = 1.5$ and $n_2 = 1.0$.

Fig. 2.3.: Reflectance of both polarizations.

the reflectance is typically less than 0.1 for transparent media, but it drastically increases shortly before the angle reaches a critical value. For angles above the critical angle θ_c the reflectance reaches a theoretical value of 1.
This phenomenon is called Total Internal Reflection (TIR). It occurs if the angle of the outgoing light θ_t is greater than $\frac{\pi}{2}$ (see figure 2.2(b)). In that case, the light only enters the second medium in the form of an evanescent wave with rapidly decreasing amplitude. Effectively, all the light has to return to medium one via reflection. Therefore, the critical angle θ_{crit} for total internal reflection is the value of θ_i for the case $\theta_t = \frac{\pi}{2}$. From equation 2.5 we get:

$$\theta_{crit} = \arcsin\left(\frac{n_2}{n_1}\right). \tag{2.6}$$

For transitions with $n_1 < n_2$ $\theta_t = \frac{\pi}{2}$ cannot be reached. Consequently, TIR does not exist in that case and a reflectance of 1 could only be reached for $\theta_i \to \frac{\pi}{2}$.

2.1.3 Light attenuation

Whenever a collimated beam of light traverses a medium, it gradually loses intensity. This is due to two effects, which are both caused by interactions of individual photons with the medium.
On the one hand, the photon can be absorbed by an atom. In that case it passes its energy on to the atom, which is in most cases converted to thermal energy of the medium.
On the other hand, the photon could scatter off electrons, atoms or molecules by one of several elastic or inelastic scattering types causing a change of direction of its movement. In general, the distribution of scattering directions depends on the

type of scattering. While the exact type of scattering in the WOM is unknown it is reasonable to assume an isotropic distribution as approximation.
However, for both scattering and absorption the photon is lost for an initially collimated beam. The Beer-Lambert law [17] describes the decrease of intensity in a medium due to both attenuation effects on a macroscopic scale. It is based on the assumption, that the intensity loss in the next differential segment $\mathrm{d}x$ is proportional to the intensity at the current position. This yields a typical differential equation for an exponential decay, $\frac{\mathrm{d}I}{\mathrm{d}x} = -\mu x$, with the solution:

$$I(x) = I_0\, e^{-\mu x}. \tag{2.7}$$

In this equation, x is the travelled distance, I_0 the initial light intensity of the beam and μ the attenuation coefficient, which is a characteristic property of the medium.

2.2 Processing on GPUs

This section gives a brief overview over the key aspects of CUDA and GPU computing that have to be considered in order to achieve high performance for the simulation. It is by no means a comprehensive treatment of the features of CUDA and GPU computing. References for further reading include: [18–20].

2.2.1 GPU hardware design

A graphics processing unit (GPU) is a processor specialized for images and 3D graphics. Both tasks require the independent application of the same instructions on large data sets (i.e. the pixels of a picture or vertices in a 3D scene) and are thus ideally suited for parallelization but intensive on memory. As a result, GPUs are designed as highly parallel manycore processors with high memory bandwidth.
Driven by the demands of the gaming industry and the recent advances in deep learning, GPUs have become a very powerful piece of hardware. They have consistently beaten central processing units (CPUs) in raw computing power and memory bandwidth in the last decade making them superior for many tasks [18].

A GPU is usually introduced to a system as a coprocessor that can be used as an accelerator to relieve the CPU of compute intensive tasks. It is not fit for entirely replacing a CPU as a primary processor since it is designed for very specific kinds of tasks. Figure 2.4 schematically compares the chip area usage of CPU and GPU cores. The CPU only uses a few arithmetic logic units (ALU), which are the actual execution units of calculations. The remaining chip area is taken by a sophisticated control unit (CU) and a large cache. The CU acts as the director of all calculations

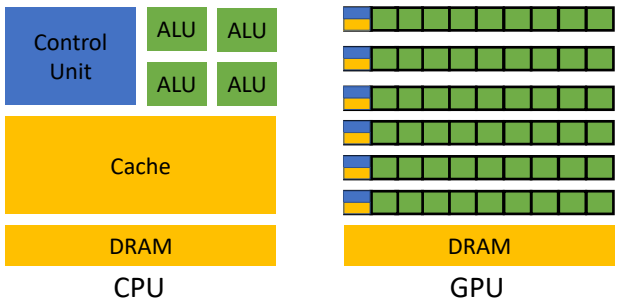


Fig. 2.4.: Schematic comparison of CPU and GPU chip area usage.

converting incoming external instructions into control signals for the ALUs. The cache memory reduces memory access latencies of complex programs by keeping frequently used data close by. A CPU is designed to be universal and as such its individual cores are optimized for executing more complex sequential tasks with low latency of a single thread.

A GPU on the other hand, accomplishes its goal by using a much larger chip area for ALUs instead of a cache and CUs. Using a great quantity of parallel threads, the long latencies due to the smaller caches can be accepted in individual threads, since another thread that is ready for execution can almost always be found. Contrary to the CPU, which minimizes the execution time of individual threads, the GPU design maximizes the throughput of threads.

Additionally, the GPU has its own dedicated high bandwidth Dynamic Random Access Memory (DRAM) separate from the CPU DRAM. Current high end models have memory transfer rates exceeding $500\,\mathrm{GB\,s^{-1}}$ [21], contrary to up to $25\,\mathrm{GB\,s^{-1}}$ of current DDR4 main memory available for the CPU [22]. However, in most cases the GPU is connected to the CPU via the much slower Peripheral Component Interconnect Express (PCIe) v3 expansion bus, providing transfer rates of at most $16\,\mathrm{GB\,s^{-1}}$ [23]. Hence, data transfers between CPU and GPU have to be reduced to a minimum.

Despite the high bandwidth, the GPU DRAM is still regarded as slow compared to the GPU computing performance. Current high end GPUs like the NVIDIA Tesla V100 reach a theoretical peak performance beyond $10\,\mathrm{TFLOP/s}$ [21]. Hence, such a GPU could perform roughly 20 single precision floating point operations in the time it has to wait for a single byte to be read from memory. That makes a high compute-to-global-memory-access[1] ratio important in GPU programming, hence avoiding the "slow" DRAM as much as possible.

[1]The CUDA programming model calls the DRAM *global memory* since it is accessible by all threads on the GPU unlike other types of memory in CUDA.

2.2.2 CUDA

General-purpose computing on GPUs (GPGPU) first became possible in the early 2000s with the support of programmable shaders. This enabled the usage of the rendering pipeline and its parallel processing capabilities for the data-heavy tasks found in scientific computing. With the release of CUDA (Compute Unified Device Architecture) in 2007, this became more practical as programmers were given direct access to the GPU via an application programming interface (API) for Fortran and C/C++. For the implementation of the simulation the CUDA C++ API is used with CUDA 8.0.

Thread-blocks

In CUDA the GPU can be programmed by declaring so-called *kernel* functions. When a kernel function is called the number of thread-blocks B and the block size T have to be stated, resulting in the invocation of $B \cdot T$ threads.
Organizing the whole grid of threads in such blocks has a few advantages. A block can comprise up to 1024 threads, which are able to cooperate with each other via synchronization and via access to a faster but smaller *shared memory*. Blocks themselves are independent of each other and there is no mechanism for grid-wide synchronisation, since it would be too wasteful to have the whole GPU wait for individual blocks.
The blocks are scheduled by the CUDA runtime for execution on the actual hardware processors of the GPU, which are called Streaming Multiprocessors (SM) by NVIDIA. In general each SM can run several thread-blocks in parallel. How many depends on the block size and the resource consumption of the kernel function. The independence of blocks allows the runtime scheduler to distribute blocks in any order to all available SMs. Remaining blocks are queued up and get assigned to an SM once another block finishes. In that way CUDA applications automatically scale with the executing GPU models, which mostly differ in the number of SMs in the same hardware generation.
Each block and each thread of a block has a builtin identifier, yielding a global id number for the thread `id = blockDim.x*blockIdx.x + threadIdx.x`.

SIMT execution model

On an SM the threads of a block are executed in a model called Single Instruction Multiple Thread (SIMT). In groups of 32, which are called warps, threads are processed simultaneously in lockstep similar to Single Instruction Multiple Data

(SIMD) vector units. That means that all threads in a warp generally have to perform the exact same instruction. However, contrary to SIMD, it is possible to set individual threads of a warp as inactive during an instruction. This allows threads inside a warp to still take different branches of the code. Nevertheless, diverging branches have to be processed serially, slowing down the computation. Therefore, branch divergence should be prevented if possible.
Obviously, the block size should always be a multiple of the warp size. Usually it is also beneficial to fit several warps into a block, even more than the SM can physically process in one clock cycle. This helps hiding latency of memory accesses by switching between warps resident on the SM while keeping the data of the threads in registers. Ultimately, the size of the block should be chosen such that the number of warps resident on an SM is maximized within the hardware limitations given the register and shared memory consumption of the kernel function.

Due to the simultaneous execution of warps, each memory access in the code results in 32 memory requests at once. Following a memory request, data from several consecutive memory addresses is also loaded to the small cache. Subsequent memory requests for one of the following data elements can then be served faster from the cache. Therefore, the threads in a warp should always access contiguous memory positions in order to avoid permanent cache misses. Memory access patterns that allow threads in a warp to access contiguous memory are called *coalesced* and should be employed whenever possible.
Coalesced memory access patterns on the GPU are different from their CPU counterparts since on the CPU contiguous memory is read by the same thread in subsequent memory accesses rather than several threads in one access.

It is quite common that structured data has to be processed instead of simple one-dimensional arrays. For instance, the data could be a set of N three-dimensional vectors. Naturally, this leads to packing of data in an Array of Structures (AoS) layout which is demonstrated in figure 2.5(a), contrary to a Structure of Arrays layout in figure 2.5(b). The first one stores the three coordinates of each vector in

```
struct Vec3D {
    float x;
    float y;
    float z;
};
struct Vec3D positions[N];
positions[i].x;
```

(a) Array of Structures data layout.

```
struct Vec3DSoA {
    float x[N];
    float y[N];
    float z[N];
};
struct Vec3DSoA positions;
positions.x[i];
```

(b) Structure of Arrays data layout.

Fig. 2.5.: Data layouts written in C.

the array contiguously in memory, which is the preferred layout on the CPU. The second one has distinct arrays for each of the coordinates that are physically apart in memory. For subsequent vectors, however, the respective coordinates are stored contiguously. Hence, the SoA layout yields coalesced memory access on the GPU and is the preferred one.

3 Simulation geometry

> *Mathematics is fun!*
>
> **— C. Hundt**
> (Deep Learning expert)

The most fundamental task for ray tracing is the determination of intersection points of rays with the geometric objects in the simulated universe. Consequently, the ray tracer needs to solve equations, which are derived from the mathematical description of the geometries in use. For arbitrarily complex geometries, this is a non-trivial task. A common practice is the approximation of complex geometries with simpler surfaces (e.g. triangles and spheres), which can be solved easily. This chapter deals with the derivation of solutions for the surface equations used to model the WOM in the simulation and discusses theoretical properties of its geometry.

3.1 Quadric surfaces

A Quadric [24] is a (D-1)-dimensional hypersurface in a D-dimensional space. It is defined as the set of points Q, with

$$Q = \{(x_1, \ldots, x_D) \in \mathbb{R}^D \,|\, q(x_1, \ldots, x_D) = 0\} \quad , \tag{3.1}$$

where

$$q(x_1, \ldots, x_D) = \sum_{i,j=1}^{D} a_{ij} x_i x_j + 2 \sum_{i=1}^{D} b_i x_i + c \quad .$$

The parameters a_{ij}, b_i and c are arbitrary real numbers and $a_{ij} \neq 0$ for at least one pair of i and j. This defines the quadric surface as the solution set of a general quadratic equation in multiple variables. Several well known geometric objects (e.g. ellipsoids and cylinders) are examples of quadrics. Obviously, quadrics are very useful for ray tracing due to the numeric simplicity of quadratic equations. Using matrix vector multiplication, equation 3.1 can be written as:

$$Q = \{\boldsymbol{x} \in \mathbb{R}^D \,|\, \boldsymbol{x}^T \mathbf{A} \boldsymbol{x} + 2\boldsymbol{b}^T \boldsymbol{x} + c = 0\} \quad ,$$

where $\mathbf{A} \in \mathbb{R}^{D\times D}$ is a symmetric matrix, $\boldsymbol{b} \in \mathbb{R}^D$ is a vector and $c \in \mathbb{R}$. Using homogenous coordinates $\bar{\boldsymbol{x}} = \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix}$ and a matrix $\bar{\mathbf{A}} = \begin{pmatrix} \mathbf{A} & \boldsymbol{b} \\ \boldsymbol{b}^T & c \end{pmatrix} = (\bar{a}_{ij})$, this can be written even more compactly:

$$Q = \{\boldsymbol{x} \in \mathbb{R}^D \,|\, \bar{\boldsymbol{x}}^T \, \bar{\mathbf{A}} \, \bar{\boldsymbol{x}} = 0\} \quad . \tag{3.2}$$

3.1.1 Intersection of ray and quadric

Equation 3.2 is the starting point for the derivation of the ray-quadric intersection points. Let $D = 3$ and let $\bar{\boldsymbol{v}}(k) = \bar{\boldsymbol{v_0}} + k\,\bar{\boldsymbol{d}}$, with $k \in \mathbb{R}$ and $\|\bar{\boldsymbol{d}}\| = 1$, define a three-dimensional ray in homogeneous coordinates (i.e. $\bar{\boldsymbol{v_0}} = \begin{pmatrix} \boldsymbol{v_0} \\ 1 \end{pmatrix}$ and $\bar{\boldsymbol{d}} = \begin{pmatrix} \boldsymbol{d} \\ 0 \end{pmatrix}$). The intersection points are the points of $\bar{\boldsymbol{v}}(k)$, which fulfill the condition of equation 3.2. Thus, substituting $\bar{\boldsymbol{v}}(k)$ and using $\bar{\mathbf{A}}^T = \bar{\mathbf{A}}$, yields a quadratic equation of the form $\alpha k^2 + 2\beta k + \gamma = 0$. For the coefficients we find:

$$\alpha = \bar{\boldsymbol{d}}^T \, \bar{\mathbf{A}} \, \bar{\boldsymbol{d}} \tag{3.3}$$

$$\beta = \bar{\boldsymbol{d}}^T \, \bar{\mathbf{A}} \, \bar{\boldsymbol{v_0}} \tag{3.4}$$

$$\gamma = \bar{\boldsymbol{v_0}}^T \, \bar{\mathbf{A}} \, \bar{\boldsymbol{v_0}} \tag{3.5}$$

Once the coefficients α, β and γ have been calculated from the ray and quadric parameters, the two solutions k_1 and k_2 of the quadratic equation can be calculated by:

$$k_{1,2} = -\frac{\beta}{\alpha} \pm \sqrt{\left(\frac{\beta}{\alpha}\right)^2 - \frac{\gamma}{\alpha}} \tag{3.6}$$

In conclusion, the intersection routine for a ray and any quadric works like this:

1. Calculation of equations 3.3, 3.4 and 3.5 for the given ray $\bar{\boldsymbol{v}}(k)$ and quadric $\bar{\mathbf{A}}$.

2. Solving the corresponding quadratic equation yields two possible solutions for k.

3. The actual intersection point of the ray is the closest one in forward direction. Since $\|\bar{\boldsymbol{d}}\| = 1$, it follows the condition $k_{\text{hit}} > \epsilon$ and $k_{\text{hit}} < k_{\text{no\_hit}}$. [1]

[1] Rays starting on the surface have $k = 0$ as one solution, which is not in forward direction. However, due to floating point imprecision the actual number is likely a very small positive or negative number. In the case of a positive result, the closest intersection in forward direction would wrongly turn out to be the solution for $k = 0$. The addition of the small positive constant ϵ helps preventing such errors.

4. Return k_{hit} or return -1, if the quadratic equation has no real-valued solution, which means that no intersection exists. If it exists, the intersection point can be obtained by calculating the ray position $\bar{\boldsymbol{v}}(k_{hit})$.

For best performance, computational overheads from matrix multiplications are avoided. Hence, the scalar results of equations 3.3, 3.4 and 3.5 are hard-coded in coordinate form (see appendix A.1).
Additionally, for the use of specific quadrics, the results in coordinate form are further simplified to avoid multiplications by zero from quadric parameters that are always zero for that geometry.[2]

3.1.2 Coordinate transformations

Let Q be a quadric defined by a matrix $\bar{\mathbf{A}}$ similar to equation 3.2. We apply a bijective linear transformation to the set of points $\bar{\boldsymbol{x}} = \left(\begin{smallmatrix} \boldsymbol{x} \\ 1 \end{smallmatrix}\right)$ in homogeneous coordinates with $x \in Q$. The bijective linear transformation can be written as $\bar{\boldsymbol{x}}' = \mathbf{M}\,\bar{\boldsymbol{x}}$, where $\mathbf{M} \in \mathbb{R}^{(D+1)\times(D+1)}$ is an invertible matrix and thus $\bar{\boldsymbol{x}} = \mathbf{M}^{-1}\,\bar{\boldsymbol{x}}'$. Starting from the defining condition of Q:

$$\begin{aligned} 0 &= \bar{\boldsymbol{x}}^T\,\bar{\mathbf{A}}\,\bar{\boldsymbol{x}} \\ &= (\mathbf{M}^{-1}\bar{\boldsymbol{x}}')^T\,\bar{\mathbf{A}}\,(\mathbf{M}^{-1}\bar{\boldsymbol{x}}') \\ &= \bar{\boldsymbol{x}}'^T{\mathbf{M}^{-1}}^T\,\bar{\mathbf{A}}\,\mathbf{M}^{-1}\bar{\boldsymbol{x}}', \end{aligned}$$

we define the transformed matrix

$$\bar{\mathbf{A}}' = {\mathbf{M}^{-1}}^T\,\bar{\mathbf{A}}\,\mathbf{M}^{-1}. \tag{3.7}$$

Obviously, $\bar{\mathbf{A}}'^T = \bar{\mathbf{A}}'$ and $\bar{\mathbf{A}}'$ thus still defines a quadric Q', which is the transformation of Q.
As a consequence, it is possible to describe a general coordinate transformation of a specific quadric surface via an affine transformation applied as linear transformation in homogeneous coordinates, using equation 3.7. Restricting the affine transformation to positions and orientations, the transformation matrix is a combination of a rotation followed by a translation: $\mathbf{M} = \mathbf{M_{trans}} \cdot \mathbf{M_{rot}} = \left(\begin{smallmatrix} \mathbf{R} & \boldsymbol{t} \\ 0 & 1 \end{smallmatrix}\right)$, where $\boldsymbol{t} \in \mathbb{R}^D$ is a translation vector and $\mathbf{R} \in \mathbb{R}^{D\times D}$ is an orthogonal matrix with $\det(\mathbf{R}) = 1$.

[2] The compiler will not optimize this on its own, since in floating point arithmetic $0 \cdot x = 0$ is not true in general due to the existence of NaN, ± 0 and $\pm inf$.

3.1.3 Normal vectors

The simulation of a reflection at a surface requires the surface's normal vector at the position of the reflection. In the derivation of the normal vector of a quadric, we use the fact that gradients of scalar fields are always perpendicular to their isosurfaces. Since the quadric defined by 3.1 is the isosurface of the scalar field $q(x_1, \ldots, x_D)$ for the value 0, the gradient of q is a normal vector of the quadric. Starting from the partial derivative for a single coordinate

$$\begin{aligned}
\frac{\partial}{\partial \bar{x}_l}(\boldsymbol{x}^T \bar{\mathbf{A}} \boldsymbol{x}) &= \frac{\partial}{\partial \bar{x}_l} \sum_{i=0}^{3} \bar{x}_i \sum_{j=0}^{3} \bar{a}_{ij} \bar{x}_j \\
&= \sum_{i=0}^{3} \sum_{j=0}^{3} (\delta_{il} \bar{a}_{ij} \bar{x}_j + \delta_{jl} \bar{x}_i \bar{a}_{ij}) \\
&= \sum_{j=0}^{3} \bar{a}_{lj} \bar{x}_j + \sum_{i=0}^{3} \bar{x}_i \bar{a}_{il} \\
&= \sum_{j=0}^{3} \bar{a}_{lj} \bar{x}_j + \sum_{i=0}^{3} \bar{a}_{li} \bar{x}_i \\
&= 2(\bar{\mathbf{A}} \bar{\boldsymbol{x}})_l,
\end{aligned}$$

we find the gradient

$$\nabla \bar{\boldsymbol{x}}^T \bar{\mathbf{A}} \bar{\boldsymbol{x}} = 2 \bar{\mathbf{A}} \bar{\boldsymbol{x}} \quad . \tag{3.8}$$

3.2 Surfaces of revolution

A surface of revolution is a two-dimensional surface, constructed by the rotation of a curve around a given axis. Let $f[z_0, z_n] \to \mathbb{R}$ be an arbitrary function which generates a surface of revolution via rotation around the z-axis. We define the surface of revolution as the set of points with distance to the z-axis always equal to the value of $f(z)$:

$$S_R = \{(x, y, z) \in \mathbb{R}^3 \mid f^2(z) = x^2 + y^2\} \quad . \tag{3.9}$$

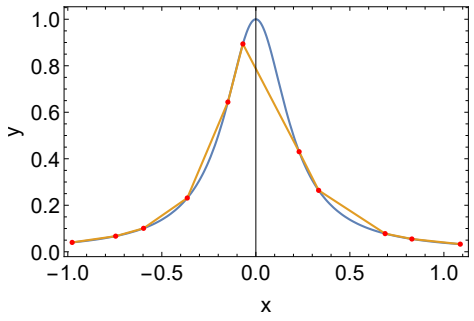
Unfortunately, we cannot find a general solution for the intersection point of S_R with an arbitrary function f and a ray $\boldsymbol{v}(k) = \boldsymbol{v_0} + k\,\boldsymbol{d}$. For an exact solution, we have to derive the solution individually for each given function f by solving

$$f^2(v_{0z} + kd_z) - (v_{0x} + kd_x)^2 - (v_{0y} + kd_y)^2 = 0$$

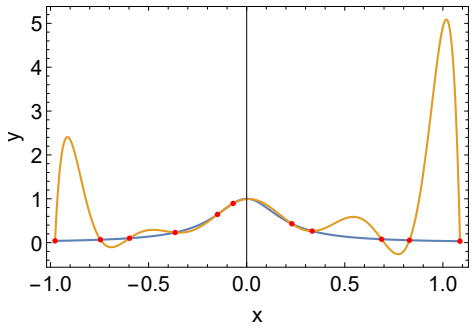
for k. Depending on the choice of f, an exact solution might or might not exist. In this section, we derive a general solution for any choice of f, based on an approximation using splines.

3.2.1 Spline interpolation

In numerical analysis, interpolation is a method for the determination of a function, which describes a set of sample data points. This is useful e.g. for sampling new data points inside the range of known data points. In contrast to a fit, interpolation requires the function to exactly run through the data points.
The interpolation problem is the task of determining the $n+1$ parameters $a_0, \ldots, a_n$ of an interpolation function $\Theta(x, a_0, \ldots, a_n)$, given a set of $n+1$ sample data points (x_i, f_i), such that $\Theta(x_i, a_0, \ldots, a_n) = f_i$ for all $i = 0, \ldots, n$. Interpolation can also be used to approximate a complicated function f by sampling data points $(x_i, f(x_i))$.
One of the simplest approaches is linear interpolation, which connects adjacent data points by a straight line. This method yields rather poor accuracy, due to the limited flexibility of the model. Additionally, the interpolation function is not smooth. A smooth solution can be obtained from polynomial interpolation. Here, the interpolation function $\Theta(x, a_0, \ldots, a_n) = \sum_{i=0}^{n} a_i x^i$ is a polynomial of degree n. Using classical single polynomial interpolation has the disadvantage of the required degree n for $n+1$ sample data points, as this makes the function very flexible for high values of n. A polynomial of degree n can have up to $n-1$ extrema and, as a result, high degree polynomials tend to oscillate in between data points. Figure 3.1 exemplifies the shortcomings of the two approaches.



(a) A linear interpolation function exhibits significant deviations from the real function.



(b) A polynomial interpolation function exhibits large oscillations between data points.

Fig. 3.1.: Two examples of poor interpolations (orange) for function approximation. The red data points are samples from the blue curve.

In spline interpolation the interpolating function is called a *spline*. The spline S, with $S : [x_0, x_n] \to \mathbb{R}$, is a piecewise-defined function of polynomials $s_i : [x_i, x_{i+1}] \to \mathbb{R}$ of degree d for all $i = 0 \ldots n-1$. Using low degree polynomials prevents the oscillations seen in usual polynomial interpolation. If we want S to be smooth, it has to be

two times continuously differentiable. This requires that $s_i'(x_{i+1}) = s_{i+1}'(x_{i+1})$ and $s_i''(x_{i+1}) = s_{i+1}''(x_{i+1})$. Additionally, we require $S(x_i) = f_i$ to solve the interpolation problem itself and thus $s_i(x_i) = f_i$ and $s_i(x_{i+1}) = f_{i+1}$. This yields four unique equations for all s_i but the last one, which is missing a partner for the two equations from the derivative conditions. Adding appropriate boundary conditions, we can get four equations in total per polynomial piece, now including the last one. Hence, a smooth spline requires polynomials with $d = 3$, since polynomials with $d \leq 2$ yield an overdetermined system of equations. In conclusion, solving the whole system of $(n-1) \cdot (d+1)$ linear equations determines the spline.
Efficient solutions build upon a derivation of the equations, which already eliminates some of the variables that are the same for all sets of data points. As a consequence, the system is reduced to $n+1$ linear equations and parameters. Additionally, for the cubic spline the system matrix is non-zero only on the main diagonal and the first two minor diagonals [25].[3]
Using splines, $d = 1$ is equivalent to linear interpolation; $d > 3$ is an option if conditions for higher order derivatives are included; although it is only continuously differentiable once, $d = 2$ is of particular interest for us, because the spline is a piecewise quadric.

3.2.2 Quadric approximation of surfaces of revolution

Using a sufficient number of spline pieces $s_i : [z_i, z_{i+1}] \to \mathbb{R}$, we approximate the square of any function $f^2(z)$ with acceptable accuracy using quadratic splines. We find $f^2(z) \approx s_i(z) = \alpha_i z^2 + \beta_i z + \gamma_i$ for $z \in [z_i, z_{i+1}]$. Similar to equation 3.9, we define surfaces of revolution for the individual pieces of the spline via

$$\begin{aligned} \alpha_i z^2 + \beta_i z + \gamma_i &= x^2 + y^2 \\ \alpha_i z^2 + \beta_i z + \gamma_i - x^2 - y^2 &= 0 \quad , \end{aligned} \tag{3.10}$$

which is a quadric with

$$\bar{\mathbf{A}}_i = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & \alpha_i & \frac{\beta_i}{2} \\ 0 & 0 & \frac{\beta_i}{2} & \gamma_i \end{bmatrix} . \tag{3.11}$$

As a result, any surface of revolution can be approximated by quadrics using this approach as long as the spline describes the original function accurately enough and the curvature does not have to be strictly continuous (the quadratic spline is not smooth). Once again, we can utilize equation 3.7 to get a transformed quadric (At

[3] This can be solved efficiently by the tridiagonal matrix algorithm in $\mathcal{O}(n)$ instead of $\mathcal{O}(n^3)$ for Gaussian elimination.

least arbitrary positions will be necessary in the simulation). For performance reasons, the implementation uses the pre-calculated result matrix of this transformation (see equation A.5 in the appendix).

Each of the approximating quadrics has two potential points of intersection, thus it is important again to determine the correct intersection for the combined surface. Earlier, the ray-quadric-intersection returned only the smallest positive solution k. Now, when both solutions are positive, both have to be considered. It is not yet clear if the smaller positive k is valid at all:

Consider a ray starting at the very left in figure 3.2 (dashed line). In that case, the ray-quadric-intersection of the dashed line and the orange line s_0 gives two valid solutions with $0 < k_{01} < k_{02}$. Substituting into the ray equation yields $z(k_{01}), z(k_{02}) \notin [z_0, z_1]$. Likewise, the intersection of the dashed and green line s_1 yields solutions with $0 < k_{11} < k_{12}$. This time, we have $z(k_{11}) \notin [z_1, z_2]$, but $z(k_{12}) \in [z_1, z_2]$. Therefore, $z(k_{01})$ and $z(k_{11})$ would indeed be the correct intersection points on the respective quadrics, but the quadrics would not be a valid approximation of the surface at these points. The correct intersection with the surface is $z(k_{12})$ on the green line. In conclusion, for all quadrics both potential points of intersection have to be examined. The correct one then is the closest one to the starting point from those, which are located inside the valid approximation range of their respective quadric.

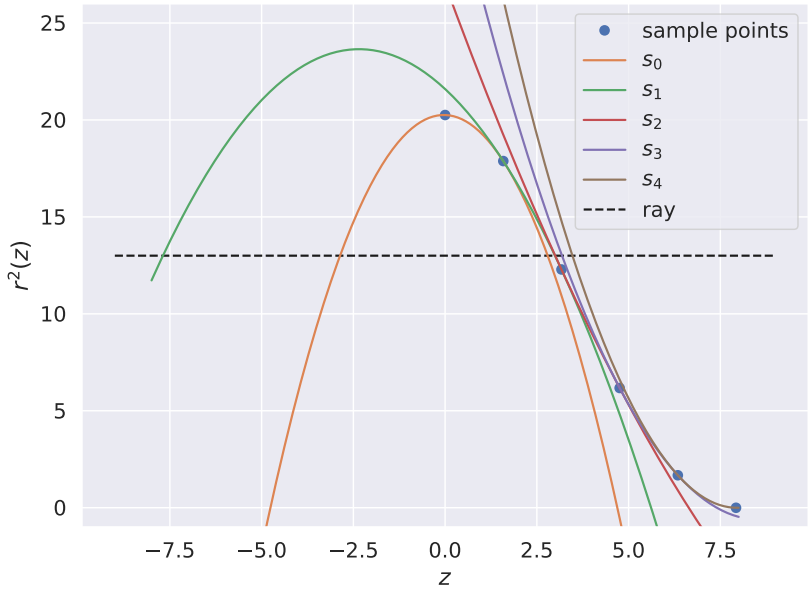


Fig. 3.2.: A ray intersecting the polynomial pieces of a spline. The spline approximates a function from the blue sample points. Each spline piece s_i approximates the function on the interval $[z_i, z_{i+1}]$. The intersection routine has to find the intersection with the correct spline piece.

The simulation program utilizes pre-calculated splines. For this, the Python library SciPy [26] is used to approximate the generating function $f^2(z)$ using $n + 1$

sample points $\left(z_i^{\text{sample}}, f^2(z_i^{\text{sample}})\right)$ and the `splrep` function from the module scipy.interpolate. Additionally, the piecewise polynomial object `PPoly` is required to obtain the actual polynomial coefficients.
The function `splrep` employs algorithms [27–30] that differ from the approach discussed in section 3.2.1. Most importantly, for quadratic splines these algorithms use only $n-1$ pieces s_i instead of n. Therefore no boundary conditions are required in order to obtain the spline. This is accomplished by determining n transition points z_i that differ from the actual sample points z_i^{sample}. Moreover, it is important that the piecewise polynomial from SciPy defines a piece s_i between z_i and z_{i+1} relative to the transition points:

$$s_i(z) = \sum_{j=0}^{d} c_{ij} \cdot (z - z_i)^{(d-j)} \quad .$$

In other words, the individual pieces in each case have a shift of z_i along the z-axis, which we have to incorporate using equation A.5 in the appendix with $x_0 = 0$, $y_0 = 0$ and $z_0 = z_i$. The set of corrected coefficients from the matrices $\bar{\mathbf{A}}_i{}'$ and the transition points serve as the actual input for the simulation program.

3.3 Geometry of the WOM

In theory, the WOM tube is a hollow cylinder, with a cross section described by two perfect concentric circles. In practice, however, a real manufactured WOM tube is subject to imprecisions.
To model imprecisions to some extent, while keeping a simple setup, a cylinder with elliptic cross section is a suitable model geometry. In that case, the hollow cylinder features a cross section of two ellipses with arbitrary misalignment between each other. The ellipses can be rotated against each other and do not necessarily share the same center point. The case of only two ellipses models the scenario where the refractive indices of tube and paint are identical, hence the outer surface is the interface of the paint and the surrounding material.
For a more comprehensive model, additional cylinders can be used to model the paint layer on the tube separately and even the other layers of the sensor. Thus, the full WOM geometry should be considered as layers of $n \geq 2$ non-overlapping elliptic cylinders. The ray tracer has to calculate the intersection points with all layers and find the correct one.

3.3.1 Intersection of the elliptic cylinder

The elliptic cylinder is a special case of a quadric surface with $D = 3$, which is defined by

$$E = \left\{ (x\, y\, z)^T \in \mathbb{R}^3 \,|\, \frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 = 0 \right\}. \tag{3.12}$$

Equation 3.12 corresponds to the following matrix for the quadric representation:

$$\bar{\mathbf{A}} = \begin{bmatrix} \frac{1}{a^2} & 0 & 0 & 0 \\ 0 & \frac{1}{b^2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}. \tag{3.13}$$

In this definition, the two semi-axes are aligned with the x- and y-axis, the cylinder axis is aligned with the z-axis and the ellipse center sits in the origin of the coordinate system. Thus, rotations of the cylinder about the z-axis can be used to rotate the parts of the WOM against each other. The shift of the center is obtained by applying a translation $\boldsymbol{t}$ in the x-y-plane: $\boldsymbol{t} = (x_0\, y_0\, 0)^T$. This yields the combined transformation matrix

$$\mathbf{M} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & x_0 \\ \sin(\alpha) & \cos(\alpha) & 0 & y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.14}$$

The matrix $\bar{\mathbf{A}}'$ of the quadric describing a general elliptic cylinder with its axis aligned to the z-axis can then be obtained using equation 3.7. Again, the implementation uses the pre-calculated result matrix of this transformation for better performance (see equation A.4 in the appendix). Using matrix $\bar{\mathbf{A}}'$ of the elliptic cylinder in the ray-quadric intersection routine described in section 3.1.1 gives the intersection point.

For the actual point of intersection with the entire WOM geometry with $n \geq 2$ elliptic cylinders, the correct surface has to be determined. If k_i is the result of the intersection with the i-th surface, $k_i \leq 0$ means the i-th surface was not hit and the correct solution is:

$$hit = \arg\min_i \left(\{ k_i \,|\, i \in [0, n],\, k_i > 0 \} \right) \quad . \tag{3.15}$$

Substituting the solution k_{hit} into $\bar{\boldsymbol{v}}(k)$ yields the intersection point. Figure 3.3 illustrates the choice of the point.

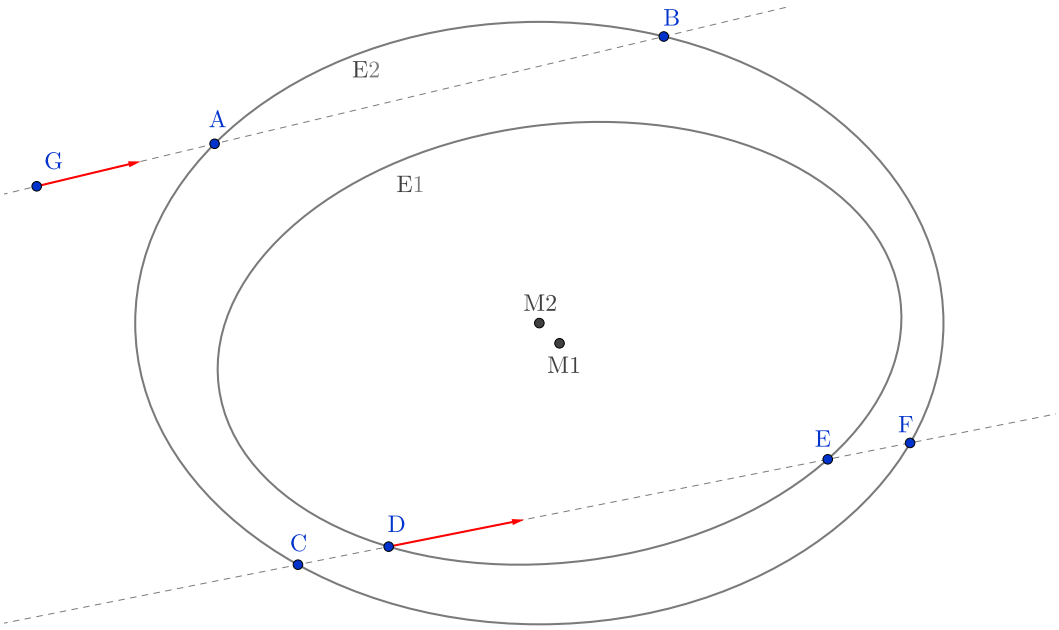


Fig. 3.3.: Intersections of rays with the WOM geometry: A ray starting at G returns no real-valued k for E1. For E2 both k_A and k_B are positive, but the correct hit point is A, since k_A is the smaller one. A ray starting at D returns the solution k_E for E1, since $k_D \approx 0$. For E2 the solution k_F is returned, since k_C is negative. Since $k_E < k_F$ the correct point is E.

3.4 Geometry of the adiabatic lightguide

The manufactured adiabatic lightguide (ALG) has been developed by Falke [31] in a semi-analytical approach. The shape of the lightguide was derived by a ray tracing simulation, which implements two common techniques for the development of light concentrators from nonimaging optics [32], namely the string method and conservation of etendue. The conservation of etendue ensures that all light stays captured in total internal reflection and does not travel back. This approach yields a curve for the cross section along the symmetry axis, which is close to the lower branch of the hyperbola defined by

$$\frac{(r-a-r_0)^2}{a^2} - \frac{z^2}{b^2} = 1 \quad . \tag{3.16}$$

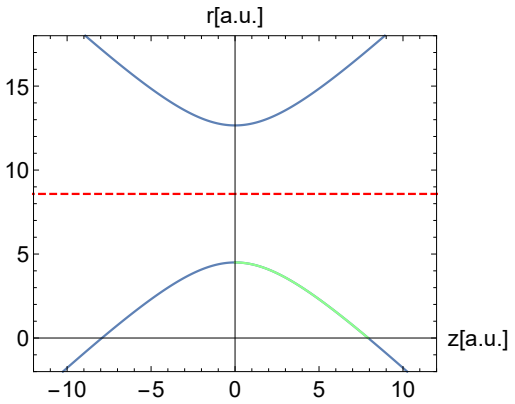
A hyperbola in these coordinates is a standard hyperbola with its conjugate axis along the z-axis. Additionally, a shift of $a + r_0$ in positive r-direction changes the vertex of the lower branch from $(-a, 0)$ to $(r_0, 0)$ (see figure 3.4(a)). Solving for r yields a function of z for the curve of the lower branch

$$r(z) = r_0 + a - \sqrt{a^2 + \frac{a^2}{b^2} z^2} \quad . \tag{3.17}$$

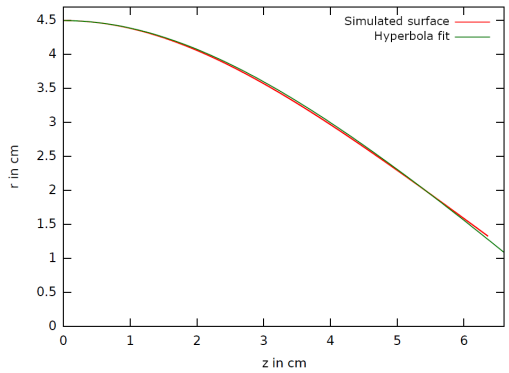
Using the eccentricity ϵ and the semi-latus rectum p as hyperbola parameters with $a = \frac{p}{\epsilon^2-1}$ and $b = \frac{p}{\sqrt{\epsilon^2-1}}$ the equation becomes

$$r(z) = r_0 + \frac{p}{\epsilon^2 - 1} - \sqrt{\left(\frac{p}{\epsilon^2 - 1}\right)^2 + \frac{1}{\epsilon^2 - 1} z^2} \quad . \tag{3.18}$$

Figure 3.4(b) shows a fit of equation 3.18 to the simulated curve.



(a) Hyperbola from equation 3.16. The red dashed line is the shifted symmetry axis at $r = a + r_0$. The green part of the hyperbola resembles the curve of the ALG.



(b) Cross section of the simulated shape with a hyperbola fit. $p = R_{\text{outer}}^{\text{WOM}}$ and the offset $r_0 = R_{\text{outer}}^{\text{WOM}}$ are given by the outer WOM radius. The fit yields $\epsilon = 1.4502$.

Fig. 3.4.: Hyperbolic cross section of the ALG.

The shape of the ALG in three dimensions is the surface of revolution obtained by rotating $r(z)$ around the z-axis on the interval $[0, z_0]$. Here z_0 is either the positive root of $r(z)$ or a smaller desired cutoff length.

In general, hyperbolas in two dimensions are quadrics.[4] We will now show, however, that the ALG surface is not a quadric.
For the surface of revolution around the z-axis the property $r(z) = \sqrt{x^2 + y^2}$ holds for all possible values of x, y and z. Rearranging and squaring equation 3.17 yields

$$\begin{aligned} a^2 \left(1 + \frac{1}{b^2} z^2\right) &= (r - r_0 - a)^2 \\ \Leftrightarrow a^2 \left(1 + \frac{1}{b^2} z^2\right) &= r^2 - 2(r_0 + a)r + (r_0 + a)^2. \end{aligned}$$

Substituting $r = \sqrt{x^2 + y^2}$ gives the surface of revolution:

$$a^2 \left(1 + \frac{1}{b^2} z^2\right) = (x^2 + y^2) - 2(r_0 + a)\sqrt{x^2 + y^2} + (r_0 + a)^2.$$

[4] Even in three dimensions the hyperboloids, which are surfaces of revolution of two dimensional hyperbolas around their symmetry axes, are quadrics.

Rearranging and squaring again results in

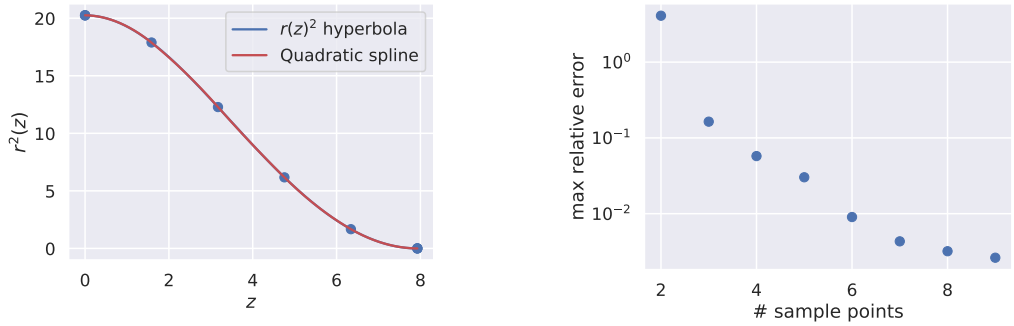
$$4(r_0+a)^2\left(x^2+y^2\right) = a^4\left(\left(1+\frac{1}{b^2}z^2\right)-\left(x^2+y^2\right)-(r_0+a)^2\right)^2 \quad . \tag{3.19}$$

This is obviously a 4-th order polynomial in x, y and z and thus not a quadric. This is due to the existing offset $r_0 + a$, which shifts the symmetry axis away from the rotation axis. Indeed, with the choice $r_0 + a = 0$ in equation 3.17 and after squaring it, the surface of revolution is defined by

$$x^2 + y^2 = a^2 + \frac{a^2}{b^2}z^2,$$

which is a quadric as stated earlier. Yet, figure 3.4(a) shows that the offset is essential for obtaining a narrowing surface after rotation.
In fact, there exist well known solution methods for equations like equation 3.19, for instance Ferrari's [33] or Euler's solution [34]. Nevertheless, solely expanding the equation after substituting a ray for (x, y, z) results in long equations for the individual quartic coefficients which are tough to handle. Overall, the exact solution requires more code and is less elegant than the previous solutions for quadrics. On the other hand, the equation for the ALG never was an exact solution of the problem in the first place. Instead, we could try to employ the approximation approach with quadrics. This avoids the implementation of a general solver for 4-th order equations.

Figure 3.5(a) shows an approximation of the square of equation 3.17 with a quadratic spline. It works very well, as there is almost no visible difference between the actual curve and its approximation. Figure 3.5(b) shows that only six sample points are sufficient to get a maximum relative error of less than one percent.



(a) Approximation of $r^2(z)$ with a quadratic spline.

(b) Relative error of the spline approximation.

Fig. 3.5.: Quality of the spline approximation for the ALG curve.

The curve $r_{\text{inner}}(z)$ for the inner surface of the manufactured ALG is defined by two conditions:

- The inner surface connects to the inner surface of the WOM tube: $r_{\text{inner}}(0) = R_{\text{inner}}^{\text{WOM}}$.

- The cross sectional area of the ALG is conserved along the z-axis: $A(z) = A(0)$.

The two conditions yield

$$\begin{aligned} \pi \left((R_{\text{outer}}^{\text{WOM}})^2 - (R_{\text{inner}}^{\text{WOM}})^2 \right) &= \pi \left(r_{\text{outer}}^2(z) - r_{\text{inner}}^2(z) \right) \\ \Leftrightarrow r_{\text{inner}}^2(z) &= r_{\text{outer}}^2(z) - (R_{\text{outer}}^{\text{WOM}})^2 + (R_{\text{inner}}^{\text{WOM}})^2 \quad . \end{aligned}$$

Hence, the inner surface can use the same spline as the outer surface for a quadric approximation according to equation 3.11, but we have to correct the respective values for the γ_i of the inner quadrics to

$$\gamma_{inner_i} = \gamma_{outer_i} - (R_{\text{outer}}^{\text{WOM}})^2 + (R_{\text{inner}}^{\text{WOM}})^2 \quad . \tag{3.20}$$

3.5 Theoretical properties of the tube

In this section, we want to establish some theoretical expectations for the idealized geometry of the WOM tube to compare the simulation against. Idealized means that we consider the tube to consist of two perfect concentric cylinders with inner radius R_i and outer radius R_o.

3.5.1 Reflection angles

Consider a ray that starts on the outer surface at Q at an angle α (see figure 3.6). The inner surface reflects it at P and it reaches the outer surface again at Q'. Due

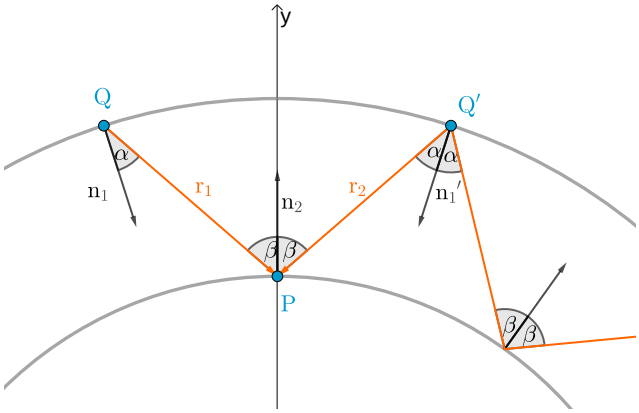


Fig. 3.6.: Light reflection inside the geometry of a perfectly circular WOM with inner radius R_i and outer radius R_o.

to rotational symmetry, we can always pick a coordinate system in which the y-axis is aligned to $\boldsymbol{n_2}$ and the x-axis is perpendicular to that. In a two dimensional orthogonal projection, we have $Q = (-x_0, y_0)$ and $P = (0, R_i)$. The reflection of Q at the y-axis yields $Q' = (x_0, y_0)$. With the origin of the coordinate system at the center of the circles, the position vector of each point on one of the circles also is a normal vector to that circle (in two dimensions). Thus, we obtain for the vectors:

$$\boldsymbol{n_1} = \begin{pmatrix} x_0 \\ -y_0 \end{pmatrix}, \quad \boldsymbol{n_1'} = \begin{pmatrix} -x_0 \\ -y_0 \end{pmatrix}, \quad \boldsymbol{n_2} = \begin{pmatrix} 0 \\ R_i \end{pmatrix},$$

and

$$\boldsymbol{r_1} = P - Q = \begin{pmatrix} x_0 \\ R_i - y_0 \end{pmatrix}, \quad \boldsymbol{r_2} = P - Q' \begin{pmatrix} -x_0 \\ R_i - y_0 \end{pmatrix}.$$

Since $\|\boldsymbol{n_1}\| = \|\boldsymbol{n_1'}\|$, $\|\boldsymbol{r_1}\| = \|\boldsymbol{r_2}\|$ and $\boldsymbol{n_1}^T \boldsymbol{r_1} = \boldsymbol{n_1'}^T \boldsymbol{r_2}$, it follows that both the angles $\measuredangle(\boldsymbol{n_1}, \boldsymbol{r_1})$ and $\measuredangle(\boldsymbol{n_1'}, \boldsymbol{r_2})$ are the same angle α with

$$\cos(\alpha) = \frac{R_o^2 - R_i y_0}{\|r_1\| R_o} \quad .$$

For the angle $\beta = \measuredangle(-\boldsymbol{r_1}, \boldsymbol{n_2})$ on the other hand, we find

$$\cos(\beta) = \frac{y_0 - R_i}{\|r_1\|} \quad .$$

The proof that β is still the same after the reflection at Q' is analogous. This result also generalizes to the three dimensional case: the z component of the normal vectors is 0 for the cylinders, hence the z components of rays do not contribute to the scalar products and the angles α and β are the same in three dimensions.

Next, assuming that $\cos(\beta) \leq \cos(\alpha)$, yields

$$\begin{aligned} \frac{y_0 - R_i}{\|r_1\| R_i} &\leq \frac{R_o^2 - R_i y_0}{\|r_1\| R_o} \\ \Leftrightarrow y_0 R_o - R_i R_o &\leq R_o^2 - y_0 R_i \\ \Leftrightarrow y_0 (R_o + R_i) &\leq R_o (R_o + R_i). \end{aligned}$$

In order to actually reach P, it is required that $R_i \leq y_0 \leq R_o$, and therefore the last inequality is true. That, in turn, confirms the assumption that $\cos(\beta) \leq \cos(\alpha)$. Since $x_1 \leq x_2$ implies that $\arccos(x_1) \geq \arccos(x_2)$, we come to the conclusion that $\alpha \leq \beta$ in all cases.
In other words, they are indeed different angles, unless $y_0 = R_o$ (which yields $\alpha = \beta = 0$). The fact that $\alpha < \beta$, implicates that one successful reflection at the inner surface does not guarantee that light is reflected again at the outer surface. After a successful reflection at the outer surface, however, a successful reflection at

the inner surface is certain.
In summary, this result clarifies that – for a circular WOM – all light, which is reflected via total internal reflection at least twice, remains captured inside the wall.

Finally, there is another case to discuss. Under certain emission angles it is possible that rays do not hit the inner surface at all and circulate around the tube being only reflected at the outer surface. Figure 3.7 illustrates the borderline case with rays that are tangential to the inner surface. Consider a ray starting at Q under an angle α, which reaches the outer surface again at P. Due to the symmetry (dashed symmetry axis s) the angle of incidence at P has to be the same angle α. Therefore, the reflection angle never changes if reflections only occur at the outer surface.
Let now $Y_0 = (0, y_0)$ define the light emission point. Via trigonometry we find for the angle enclosed between the dotted line and a ray from Y_0 to P that $\gamma = \arccos\left(\frac{R_i}{y_0}\right)$. Due to the symmetry, it is clear that all rays, which have an enclosed angle of γ with the dotted line yield a similar border case, even those which reflect at the outer surface first before touching the inner surface. Thus, a ray reflects only at the outer surface when the following condition for the x-y-plane polar angle of its direction is fulfilled:

$$\varphi \in [0, \gamma] \cup [\pi - \gamma, \pi + \gamma] \cup [2\pi - \gamma, 2\pi) \tag{3.21}$$

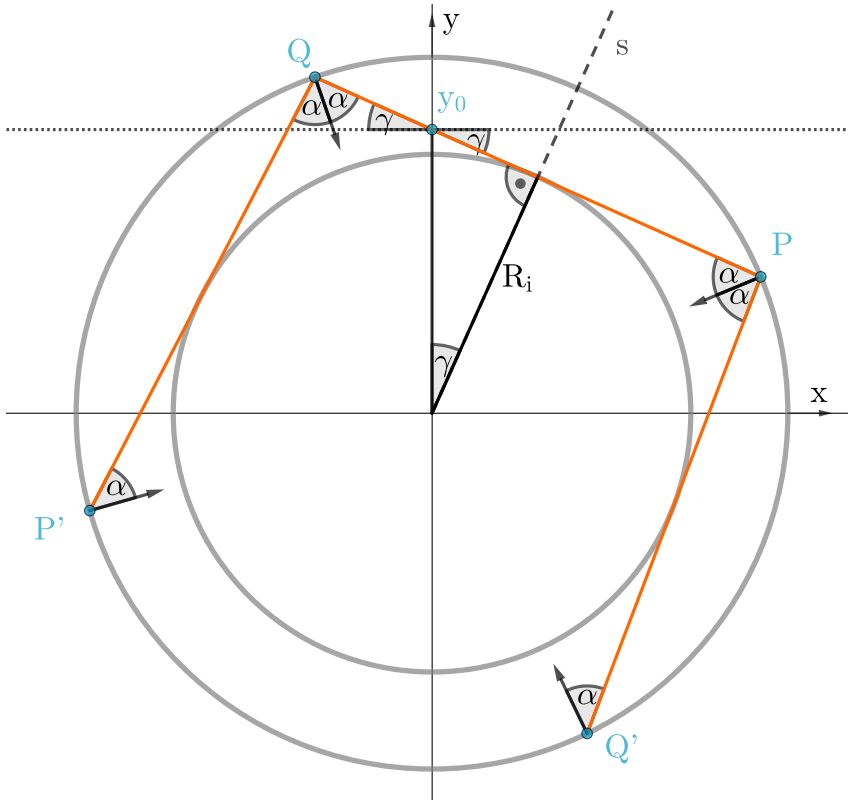


Fig. 3.7.: Light being reflected only at the outer surface of the tube.

3.5.2 Condition for capturing light

Next, we want to exploit the geometry to work out the exact starting conditions which result in rays being captured via total internal reflection.
Let $\boldsymbol{v_0} = (0, y_0, 0)$ define the light emission point, with $R_i < y_0 < R_o$ and let $\boldsymbol{d} = (d_x, d_y, d_z)$ define a random direction, with $\|\boldsymbol{d}\| = 1$. The intersection point of the ray $\boldsymbol{v}(k) = \boldsymbol{v_0} + k\,\boldsymbol{d}$ and the outer surface can be found by inserting it into the equation of a circle and solving for k:

$$\begin{aligned} R_o^2 &= x^2 + y^2 \\ \Leftrightarrow R_o^2 &= k^2 d_x^2 + (y0 + k\,d_y)^2. \end{aligned}$$

For the solution in forward direction (the positive solution) this yields:

$$k_1 = \frac{-d_y y_0 + \sqrt{R_o^2(d_x^2 + d_y^2) - d_x^2 y_0^2}}{d_x^2 + d_y^2} \quad .$$

A normal at a point r on the outer surface is given by:

$$\boldsymbol{n}(\boldsymbol{r}) = \frac{1}{R_o} \begin{pmatrix} r_x \\ r_y \\ 0 \end{pmatrix}.$$

Thus, the angle of incidence of the ray on the outer surface is determined by:

$$\begin{aligned} \cos(\alpha) = \boldsymbol{d}^T \boldsymbol{n}(\boldsymbol{v_0} + k_1 \boldsymbol{d}) &= \frac{k_1 d_x^2 + d_y(k_1 d_y + y_0)}{R_o} \\ &= \frac{k_1(d_x^2 + d_y^2) + d_y y_0}{R_o} \quad . \end{aligned}$$

Subistituting k_1 yields:

$$\cos(\alpha) = \frac{\sqrt{R_o^2(d_x^2 + d_y^2) - d_x^2 y_0^2}}{R_o} \quad . \tag{3.22}$$

Using spherical coordinates for $\boldsymbol{d}$ in equation 3.22 yields:

$$\cos(\alpha) = \frac{\sqrt{R_o^2(\cos^2(\varphi)\sin^2(\theta) + \sin^2(\varphi)\sin^2(\theta)) - \cos^2(\varphi)\sin^2(\theta) y_0^2}}{R_o}$$

$$\Leftrightarrow \cos(\alpha) = \frac{\sin(\theta)\sqrt{R_o^2 - \cos^2(\varphi) y_0^2}}{R_o} \quad . \tag{3.23}$$

A visualization of the resulting reflection angle depending on the two starting angles is given in figure 3.8 as contour plots for different values of y_0. In these plots, the

critical angle for total internal reflection appears as a single isoline. As an example, the isoline for a common glass-air transition ($n = 1.5$ to $n = 1.0$) is marked in red. Total internal reflection occurs if $cos(\alpha) < cos(\theta_c)$. Therefore, according to these plots, the majority of emission angles is captured by the WOM, except for two areas inside the red isoline, which are centered at $\left(\frac{1}{2}\pi, \frac{1}{2}\pi\right)$ and $\left(\frac{3}{2}\pi, \frac{1}{2}\pi\right)$.

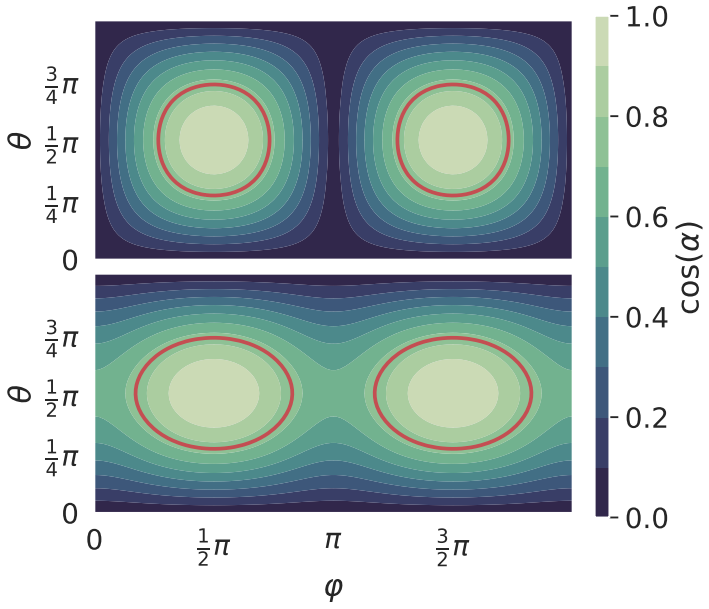


Fig. 3.8.: Two contour plots for equation 3.23. Both plots use $R_o = 4.5$. The top one uses $y_0 = 4.49$, while the bottom one uses $y_0 = 3.5$

3.5.3 Light capture efficiency

The bottom plot in figure 3.8 shows that moving the emission point y_0 away from the outer surface results in a deformation and increased size of the transmission area, which is caused only by the φ angle.
This can be explained by figure 3.9: Moving y_0 towards the origin along the y-axis, while still maintaining the same emission angle φ, gives the ray more space to travel before reaching the surface. This results in the intersection point moving away from the y-axis. Thus, the inclination of the normal vector increases and the angle between ray and normal vector gets smaller. In the extreme case of $y_0 = 0$, the normal vectors are perfectly aligned with all possible directions and total internal reflection becomes impossible. Overall, this means that we expect the highest capture efficiency with an emission point as close to the outer surface as possible.

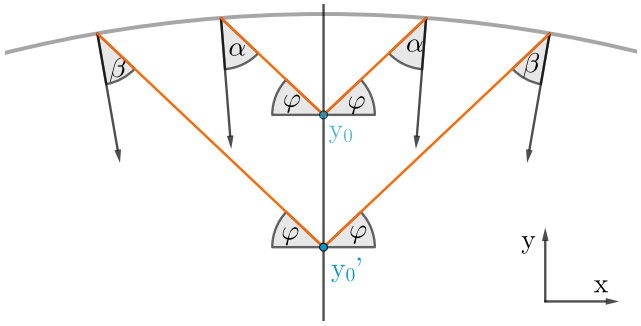


Fig. 3.9.: Light emission under angle φ for two different emission points.

For a WOM coated with paint, we can assume that the isotropic emission indeed takes place at $y_0 \approx R_o$[5]. In that case, equation 3.22 simplifies to

$$\cos(\alpha) = d_y. \tag{3.24}$$

For the critical angle, the intersection of the associated plane $d_y = \cos(\theta_c)$ and the unit sphere is a small circle of the sphere that is perpendicular to the y-axis. All rays emitted to points of the unit sphere above that plane are lost and form a loss cone through that circle (see figure 3.10). Using any point $\boldsymbol{d'}$ on the circle, the half opening angle γ of this cone can be calculated via

$$\cos(\gamma) = \hat{\boldsymbol{y}}^T \boldsymbol{d'} = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} d'_x \\ \cos(\theta_c) \\ d'_z \end{pmatrix} = \cos(\theta_c).$$

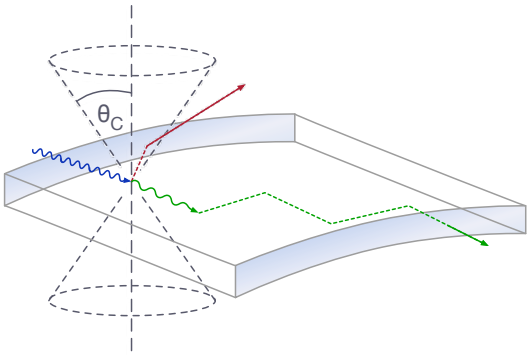


Fig. 3.10.: Illustration of the loss cone defined by the critical angle θ_c. Rays emitted inside the loss cone reach the outer wall at an angle smaller than the critical angle.

We can use this information to get a theoretical maximum capture efficiency of the circular WOM geometry. Correctly, this also should include reflections outside of total internal reflection, but the detection probability of such photons decreases

[5] It is not $y_0 = R_o$, because the paint does not absorb all light immediately at the boundary.

exponentially with the number of reflections. Therefore, we give a lower bound instead, which only includes total internal reflection. It can be derived from the solid angle covered by the loss cone:

$$\Omega = 2\int_0^{2\pi'} \mathrm{d}\varphi' \int_0^{\theta_c} \sin(\theta')\mathrm{d}\theta' = 4\pi(1-\cos(\theta_c)) \quad .^{6}$$

Light travelling towards the inner surface forms a second loss cone. Its opening angle is the same as the other one, since we have proven earlier that the reflection at the outer surface is more conservative. For this, consider figure 3.6 again and imagine the starting point on r_1 somewhere close to the outer surface. If $\alpha = \theta_c$ at Q, then it is the same at Q', even with a successful reflection in between. Considering both loss cones then yields as lower bound of the maximum capture efficiency:

$$\epsilon_c = 1 - \frac{2\Omega}{4\pi} = \cos(\theta_c) \quad . \tag{3.25}$$

Remarkably, the thickness of the tube wall and the radius of the tube are irrelevant for this result. In fact, the inner surface has no effect on the efficiency at all and it would be the same for a solid cylinder. $y_0 \approx R_o$ is the only assumption that was made in order to use the approximation in equation 3.24, which is viable as long as the distance of the light emission point to the outer surface is small compared to the radius.

[6] Breaking with earlier conventions, the spherical coordinate system used for the integral measures the angle θ' to the y-axis.

4 Implementation and validation of the simulation algorithm

The simulation is implemented in CUDA C++ in order to run it on GPUs. Thus, all the main functionality of the simulation is implemented in a number of CUDA kernel functions, while the rest of the simulation program only serves as a wrapper to these functions which provides a user interface. This chapter discusses the simulation algorithm for a single photon, its basic implementation and validations. All aspects of the actual parallel GPU implementation and performance optimizations are topic of chapter 5. The user interface is documented in a public Git repository [35].

4.1 Initialization

The simulation starts with a separate kernel function dedicated to the initialization process. This includes all the work which has to be done before the simulation enters the main simulation loop. From the given starting parameters, it generates initial light rays, which hit the WOM geometry for the first time. Subsequently, it executes the wavelength-shift and the isotropic redistribution of light. It was desirable to have the initialization in its own kernel function to enable different kinds of initialization separate from the main simulation loop. For example, another kind of initialization takes light from a file as input without performing the wavelength-shift. This can be used to run simulations with light that was generated as output from a previous simulation run. Currently, this is the approach to analyze the efficiency of the ALG.

4.1.1 Light source

In the first kind of initialization, we implement an extended light source positioned somewhere outside the WOM geometry with a given orientation to the surface. From there, it draws starting positions and directions from two exchangeable probability distributions, which can be useful to model the shape of the light source in the experiment and its light output.
Despite the possibility of having different distributions, so far only one is included in the implementation. It is a symmetrical 2D Gaussian distribution for the starting positions in a plane above the WOM and parallel to the x-y-plane together with

a constant distribution of light directions which are only facing downwards onto the surface. A single parameter σ controls the width of the light source in both x and y. Setting $\sigma = 0$ results in a point light source, which was used in most of the simulations.

4.1.2 Isotropically distributed light directions

The WLS paint emits shifted light isotropically. For the simulation, this means that it generates rays in random directions where each possible direction has the same probability. This section demonstrates how to algorithmically generate a three-dimensional isotropic distribution. We look on ray directions as points on the unit sphere in spherical coordinates with $\varphi \in [0, 2\pi]$ and $\theta \in [0, \pi]$:

$$\boldsymbol{d}(\varphi, \theta) = \begin{pmatrix} \sin(\theta)\cos(\varphi) \\ \sin(\theta)\sin(\varphi) \\ \cos(\theta) \end{pmatrix} \quad . \tag{4.1}$$

We need a sampling scheme for φ and θ that uniformly covers the sphere surface. The naive approach would be to uniformly sample both in their respective domain of definition, which unfortunately is incorrect. To comprehend this, we consider uniformly sampling n values of φ for a fixed value $\theta = \frac{\pi}{2}$. We end up with n points spread over a full great circle of the sphere. Now we do the same for values of θ which are closer to 0 or π. Those are still n points on circles, but the circles are getting smaller and thus the point density on a circle increases. Therefore, uniformly sampling θ as well, results in an overall higher point density (i.e. a higher probability density of the distribution) towards the upper and lower bounds of θ. If we keep a uniform distribution of φ, then we have to sample θ from a distribution which has maximum probability density at $\theta = \frac{\pi}{2}$ and symmetrically decreases towards both bounds.

We assert that the correct distribution is uniform for $z \in [-1, 1]$. To prove this, consider the unit sphere being cut in m slices of equal thickness along the z-axis. For the i-th slice we have $z \in [\frac{2i}{m} - 1, \frac{2(i+1)}{m} - 1]$. We can calculate the outer surface area of the i-th slice (excluding the area of the slice planes) by solving the surface integral with z constrained within the slice. In spherical coordinates we have $z = \cos(\theta)$ and thus the integral becomes:

$$\begin{aligned} A_i &= \int_0^{2\pi} \mathrm{d}\varphi \int_{\arccos\left(\frac{2(i+1)}{m} - 1\right)}^{\arccos\left(\frac{2i}{m} - 1\right)} \sin(\theta)\mathrm{d}\theta \\ &= 2\pi \left(\cos\left(\arccos\left(\frac{2(i+1)}{m} - 1 \right) \right) - \cos\left(\arccos\left(\frac{2i}{m} - 1 \right) \right) \right) \\ &= \frac{4\pi}{m} = \frac{A}{m} \quad . \end{aligned}$$

In other words, each of the slices has a fraction $\frac{1}{m}$ of the total surface area of the sphere. That means sampling n points on the outer surfaces results in equal point densities for each slice and we get a uniform probability density for the whole sphere surface by uniformly sampling $\varphi \in [0, 2\pi]$ and $z \in [-1, 1]$. Actually, this means we are calculating $\theta = \arccos(1 - 2U)$ with $U \in [0, 1]$ uniform. According to the inverse transformation method [36], we are thus sampling θ from the distribution $F_\theta(\theta) = \frac{1}{2}(1 - \cos(\theta))$. The probability density function of this distribution is $f_\theta(\theta) = \frac{1}{2}\sin(\theta)$, which has the above mentioned features.

Using $\sin(\theta) = \sqrt{1 - \cos^2(\theta)}$, the actual vector of the direction is

$$\boldsymbol{d}(\varphi, \theta) = \begin{pmatrix} \sin(\theta)\cos(\varphi) \\ \sin(\theta)\sin(\varphi) \\ \cos(\theta) \end{pmatrix} = \begin{pmatrix} \sqrt{1 - z^2}\cos(\varphi) \\ \sqrt{1 - z^2}\sin(\varphi) \\ z \end{pmatrix} \quad . \tag{4.2}$$

4.2 Main simulation loop

The main part of the simulation contains a single loop for each photon, which is implemented in a kernel function, to run for multiple photons in parallel on the GPU. Algorithm 1 conceptually demonstrates the loop in pseudocode. With the procedure

Algorithm 1 Main simulation loop

```
1:  procedure SIMULATE_PHOTON(pos, dir, *params)
2:      out ← false
3:      d_t ← 0
4:      while not out do
5:          d_part ← 0
6:          INTERSECT_WOM(pos, out, normal, d_part, dir)
7:          d_t ← d_t + d_part
8:          interaction ← false
9:          APPLY_MEDIUM_INTERACTION(pos, dir, out, d_t, interaction, *params)
10:         if not out and not interaction then
11:             APPLY_SURFACE_INTERACTION(dir, normal, out, *params)
12:         end if
13:     end while
14: end procedure
```

INTERSECT_WOM, the loop repeatedly propagates a photon from one surface to the next one by determining the closest intersection point of its ray (chapter 3) and updating the photon position accordingly. The procedure switches the loop control parameter out to true, if the ray reaches the geometry specific detection area or if it does not find an intersection. Additionally, it returns the $normal$ of the surface at the new position and the distance travelled d_{part}.

Afterwards, via **APPLY_MEDIUM_INTERACTION**, each step enables a belated interaction of the photon with the medium before reaching the next surface. These processes depend on the distance travelled d_{part} and other interaction parameters. Interactions with the medium can cause an update of the position and the direction, which means the procedure can also overwrite the previous outcome of out.
Finally and only if the photon has not left the simulation geometry and did not interact with the medium before reaching the surface, it gets an interaction with the surface. **APPLY_SURFACE_INTERACTION** then changes the direction depending on the surface $normal$.

Due to the logical separation of intersection, interaction with the medium and interaction with the surface, we can extend or exchange the individual parts without breaking the main simulation structure.
INTERSECT_WOM, for instance, is an abstract intersection procedure to calculate the correct intersection point of the ray and the entirety of all simulated surfaces. At this position in the loop, it only requires the ray parameters and is completely independent of the actual geometry. In the kernel function, this abstract function is implemented as a C++ template parameter. This has the advantage that we can use the same unmodified code for the main loop to simulate different assemblies of geometries via template specialization with the appropriate intersection routine. We have to pass this template parameter as a lambda function, which captures the required geometry parameters and exposes the specific intersection routine to the main loop. The compiler is able to optimize the loop entirely during template specialization, which results in an abstraction without performance penalty.
APPLY_MEDIUM_INTERACTION implements absorption and scattering of photons, while **APPLY_SURFACE_INTERACTION** currently only applys total internal reflection at the surface. An extension to simulate, for example, correct reflection and transmission behaviour outside of total internal reflection should be straight forward using equations 2.3 and 2.4 for the decision.

4.2.1 Applying interactions with the medium

To model interactions with the medium, we have to implement the Beer-Lambert law, which describes the intensity loss of a light beam in a medium. The intensity is given by equation 2.7, but we have to translate it to a probability for individual photons.
The intensity I is the energy transfer E of the light beam per unit area A and unit time t and is given by:

$$I = \frac{E}{A\,t} \quad .$$

Substituting the energy contributions of N individual photons for the total energy E yields:

$$I = \frac{\sum_{i=0}^{N-1} h\nu_i}{A\,t} \quad ,$$

where h is Planck's constant and ν is the photon energy. In the case of monochromatic light this simplifies to:

$$I = \frac{h\nu N}{A\,t} \quad .$$

Therefore, we find that $I \sim N$. As a consequence, we can write for the probability that a single photon is still part of the beam at a distance x:

$$\Pr(\overline{I}; x) = \frac{N(x)}{N(0)} = \exp\left(-\frac{x}{\lambda}\right). \tag{4.3}$$

It follows that $\Pr(I; x) = 1 - \exp\left(-\frac{x}{\lambda}\right)$, which is the cumulative distribution function (CDF) for the probability distribution of photon interaction distances in the medium. Here we use the more convenient parameter $\lambda = \frac{1}{\mu}$, which is often called *mean free path* because it is the expectation value $\langle x \rangle$ of the distribution. Using the inverse transform method [36] and a uniformly sampled $y \in [0, 1)$, we can sample from this distribution by calculating

$$x = -\lambda \log(1 - y).^{1} \tag{4.4}$$

Since attenuation includes absorption and scattering, which are independent events, the relationship

$$\Pr(\overline{I}; x) = \Pr(\overline{A} \cap \overline{S}; x) = \Pr(\overline{A}; x) \cdot \Pr(\overline{S}; x)$$

holds. Substituting this into equation 4.3, we can deduce that

$$\frac{1}{\lambda} = \frac{1}{\lambda_a} + \frac{1}{\lambda_s}, \tag{4.5}$$

$$\Pr(\overline{A}; x) = \exp\left(-\frac{x}{\lambda_a}\right)$$

and

$$\Pr(\overline{S}; x) = \exp\left(-\frac{x}{\lambda_s}\right).$$

Hence, the individual, independent probabilities for absorption or scattering are similar to the probability of attenuation itself, but with their own interaction lengths λ_a and λ_s.

To conclude, we simulate absorption and scattering by sampling two distances x_a and x_s for a photon using equation 4.4 with the respective interaction lengths λ_a and λ_s. From those, we calculate an absorption and a scattering distance $d_{a/s} = x_{a/s} + d_t$.

[1] The actual implementation calculates the equivalent version $x = -\frac{1}{\mu}\log(y)$ to avoid $\log(0)$ due to the used uniform sampling in the interval $(0, 1]$.

Here, d_t is the absolute accumulated travel distance of the photon up to that point. Every time the photon transitions to another medium, d_a and d_s have to be updated. Right now, this only happens once before the main loop as transmission at a surface is not yet implemented (see section 4.2.2). With proper simulation of transmission, d_a and d_s will be updated in **APPLY_SURFACE_INTERACTION**.
The actual interaction is simulated in **APPLY_MEDIUM_INTERACTION**, if the total travel distance of the photon exceeds d_a or d_s. Algorithm 2 demonstrates the procedure. At this point in the main loop, the position has already been updated by

Algorithm 2 Interaction with the medium

```
1:  procedure APPLY_MEDIUM_INTERACTION(pos, dir, out, d_t, interaction, d_a, d_s, λ_s)
2:      if d_t ≥ d_s then
3:          out ← false
4:          interaction ← true
5:          pos ← pos − (d_t − d_s) · dir
6:          d_t ← d_s
7:      end if
8:      if d_t ≥ d_a then
9:          out ← true
10:         interaction ← true
11:         pos ← pos − (d_t − d_a) · dir
12:         d_t ← d_a
13:     end if
14:     if not out and interaction then
15:         d_s ← d_s + SAMPLE_INTERACTION_DISTANCE(λ_s)
16:         dir ← SAMPLE_DIR
17:     end if
18: end procedure
```

the intersection routine to the position on the next surface (see algorithm 1). That means that if any interaction occurred on the way to this position, we have to correct the position by travelling backwards on the ray (lines 5 and 11). Additionally, we set the variable $interaction$ to **true** (lines 4 and 10), which prevents the surface interaction later, since the surface has not been reached yet.
The first two parts of the procedure test if scattering and absorption occurred in the last segment. Scattering sets the loop control switch out to **false** in order to reconsider the previous decision from the intersection routine. Absorption on the other hand sets it to **true**, since the photon then definitely stops. In the first part, scattering does not yet assign a new direction. This takes place in the last part (line 15) together with the calculation of a new scattering distance, but only if the photon was not absorbed before.
The order of the three parts in algorithm 2 is essential to correctly handle the cases $d_s < d_a < d_t$ and $d_a < d_s < d_t$ (the cases for $d_t < d_{s/a}$ and $d_{s/a} < d_t < d_{a/s}$ are trivial). In the first one, the algorithm begins with correcting the photon position to the distance d_s, then the absorption fails, and finally the direction changes. Only

in the next iteration, it tries the absorption again and the correct order of events is ensured. For the case that $d_a < d_s < d_t$ on the other hand, the absorption does not fail and the photon undergoes a second position update before stopping. If the scattering assigned a new direction prior to that, the second update would end up at the wrong position.
As a final remark, the implementation of **SAMPLE_DIR** samples from an isotropic distribution (see section 4.1.2) to keep the model simple. This is reasonable, because the exact scattering behaviour in the WOM is so far unknown. In principle, the isotropic distribution in **SAMPLE_DIR** can be replaced by any other more sophisticated scattering distribution.

4.2.2 Applying surface interactions

According to Fermat's principle, the angle between surface normal and reflected light ray is equal to the angle between surface normal and incident light ray. Using this, figure 4.1 demonstrates the construction of the reflected ray's direction $\boldsymbol{dir}'$ from the incident ray's direction $\boldsymbol{dir}$, with $\|\boldsymbol{dir}\| = 1$ and the surface normal $\boldsymbol{n}$, with $\|\boldsymbol{n}\| = 1$.

$$\begin{aligned}
\boldsymbol{dir}' &= \boldsymbol{dir} + \boldsymbol{t} \\
&= \boldsymbol{dir} + 2\|\boldsymbol{d}\| \cdot \boldsymbol{n} \\
&= \boldsymbol{dir} + 2\cos(\alpha) \cdot \boldsymbol{n} \\
&= \boldsymbol{dir} + 2(-\boldsymbol{n}^T \boldsymbol{dir}) \cdot \boldsymbol{n} \\
\boldsymbol{dir}' &= \boldsymbol{dir} - 2(\boldsymbol{n}^T \boldsymbol{dir}) \cdot \boldsymbol{n} \qquad (4.6)
\end{aligned}$$

For this solution the orientation of the surface normal does not matter, since substituting $\boldsymbol{n} = -\boldsymbol{n}$ in equation 4.6 does not change the result. The current implementation of **APPLY_SURFACE_INTERACTION** first calculates the angle enclosed between $\boldsymbol{n}$ and $\boldsymbol{dir}$:

$$\alpha_e = \arccos(\boldsymbol{n}^T \boldsymbol{dir}).$$

Applying a correction for the orientation of the normal yields the reflection angle $\alpha = \max(\alpha_e, \pi - \alpha_e)$. Whether a ray is reflected or not is decided based on the total internal reflection criterion in equation 2.6 (instead of considering the Fresnel equations). A reflection occurs if $\alpha > \theta_{crit}$. Then, the direction is changed according to equation 4.6, otherwise the photon simulation terminates. The latter of course is incorrect, but the simulation does not yet support transmission with correct refraction, because it is expected to only have a minor effect on the properties that are currently analyzed. It can be implemented by calculating a new direction according to Snell's law. This is demonstrated in appendix B.1.

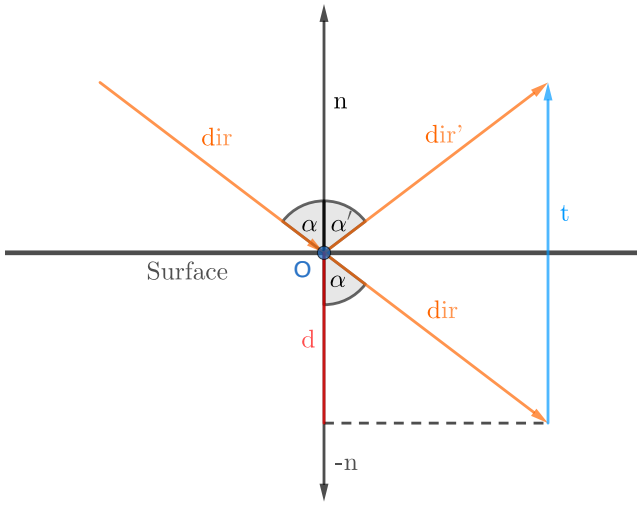


Fig. 4.1.: Reflection of a light ray dir at the point O of a surface with normal n and construction of the reflected ray dir'.

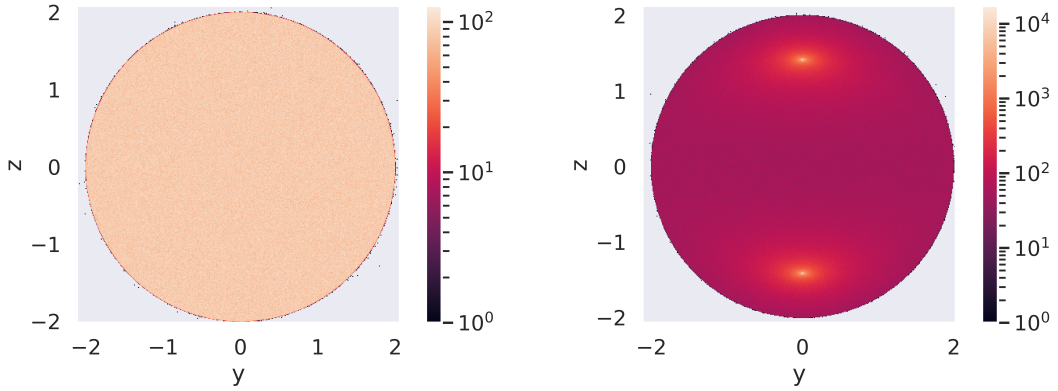
4.3 Validation of the implementation

This section presents validation tests for the implementation, by comparing simulation results to the theoretical expectations from chapter 3.5 and chapter 2.1.
This section and the following chapters will repeatedly mention simulation setups for the WOM tube. Therefore, the relevant input parameters of the simulation are introduced and defined here:

- Parameters for the tube geometry. These include the full set of quadric parameters of both cylinders. However, most relevant are the outer and inner radius R_o and R_i and the length L.
- Number of simulated photons N.
- Refractive index of the tube n_1 and its surrounding material n_2.
- Absorption length λ_a and scattering length λ_s.
- Position of the light source and the light emission depth of the wavelength-shifter. Usually, the effective light emission point inside the tube wall is the listed input parameter for this defined by $x_0 = 0$, y_0 and z_0.

4.3.1 Verification of the isotropic light distribution

At first, we test whether the isotropic distribution works as intended. Figure 4.2(a) shows the two dimensional distribution of 10^7 rays generated as described in section 4.1.2. For illustration purposes, the histogram uses a Lambert azimuthal equal-area projection [37]. This kind of projection applies the transformation $(y', z') = \left(\sqrt{\frac{2}{1-x}}y, \sqrt{\frac{2}{1-x}}z\right)$ to the points (x, y, z) on the unit sphere, which represent the three dimensional rays. A Lambert projection portrays a sphere surface on a disk of radius 2, with the center representing the point (-1,0,0) and the entire circular boundary representing the point (1,0,0). All points on the disk with $r < \sqrt{2}$ represent points with $x < 0$, while those with $r > \sqrt{2}$ represent points with $x > 0$. Apart from small expected random noise, the result is highly uniform across the whole surface of the sphere. What looks like a low density corona at the edge is an artifact of the projection, since the whole boundary represents a single point. The same projection using for instance (x', z'), does not reveal anything conspicuous for the point $(1, 0, 0)$ In contrast, figure 4.2(b) shows the result of uniform sampling of both angles with the same projection. As expected, it features two areas of high density, which are located around $\theta = 0$ or $(0, 0, 1)$ and $\theta = \pi$ or $(0, 0, -1)$.



(a) Isotropic distribution.

(b) Uniform distribution of both angles.

Fig. 4.2.: Comparison of distributions of 10^7 rays in a two-dimensional Lambertian projection.

4.3.2 Visual inspection

A first possible assessment can be made via simple visual inspection of photon tracks. For this purpose, an OpenGL based tool has been developed to render the tracks together with the simulation geometry in an interactive view. Since it would be infeasible to save all intermediate positions of every photon, this has to be enabled as an option for a set of photons.

Figure 4.3 depicts a collage of screenshots from photon tracks in the rendering tool. First, they show that the photons are always hitting the first surface in their path. Thus, we can conclude that the intersection routines from chapter 3 indeed are correct both for tubes with arbitrary elliptic geometry and the ALG. Second, the change of direction at a surface appears as expected from a reflection. In addition to this, some photon tracks prematurely end at surfaces, when the total internal reflection criterion is not fulfilled (bottom left). Finally, some tracks prematurely end or suddenly change their direction at random positions between surfaces, which is the result of absorption (bottom middle) and scattering (bottom right).

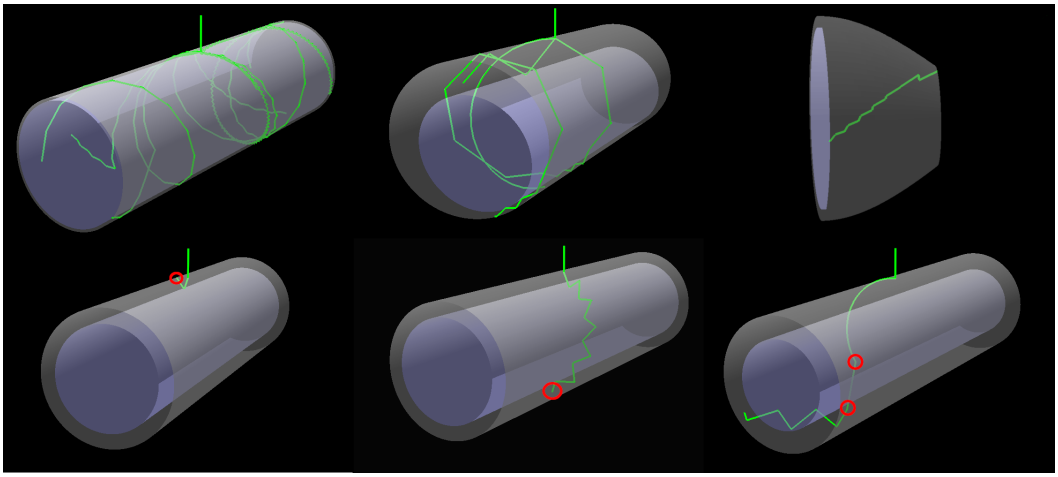
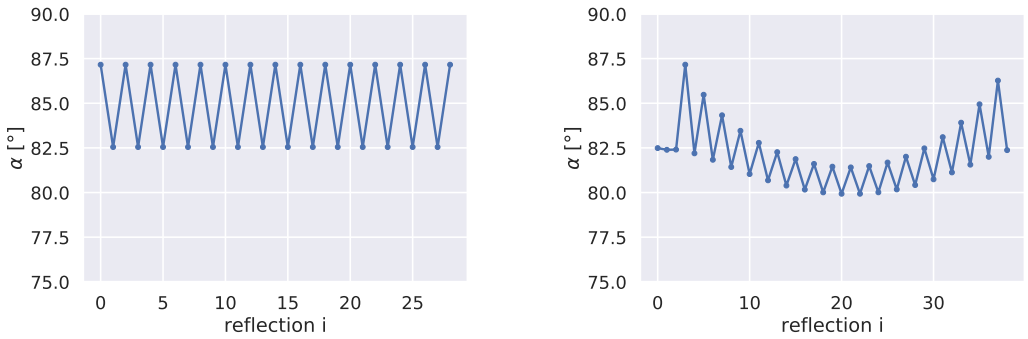


Fig. 4.3.: Visualization of photon tracks for selected simulation geometries in an OpenGL based renderer.

4.3.3 Verification of the reflection angles

Next, we examine the reflection angles. Figure 4.4(a) shows a measurement of reflection angles for a single ray inside a circular WOM. The angles oscillate between two discrete values, which meets our expectations from chapter 3.5.1.
Figure 4.4(b), on the other hand, depicts the measurement for the same ray in a slightly different elliptic geometry. In that case, a deviation of only $\pm 1\%$ relative to the circular geometry was applied to both radii in x and y direction together with a shift of the center of 1% relative to the outer radius. This is already sufficient to cause a significant deviation in the measured angles. This indicates that a more accurate simulation of an elliptic geometry indeed requires the full reflection simulation including the Fresnel equations. Otherwise photons with reflection angles close to the critical angle drop out immediately after the first time they fall below the critical angle, although there is a good chance they could survive a few reflections before fulfilling the TIR criterion again.



(a) Circular WOM geometry.

(b) WOM with slightly elliptic cross section.

Fig. 4.4.: Reflection angles of a ray at the i-th reflection for different geometries.

4.3.4 Verification of the theoretical capture efficiency

In a next step, we investigate the capture behaviour of the idealized WOM geometry. To do so, we track the number of reflections that each photon accomplishes and their initial emission angles φ and θ in spherical coordinates (φ measured from the positive x-axis and θ measured from the positive z-axis). We set up a simulation based on the parameters in table 4.1.

Tab. 4.1.: Simulation parameters for the analysis of the number of reflections depending on the emission angles.

N	R_o[cm]	R_i[cm]	L[cm]	n_1	n_2	λ_a[cm]	λ_s[cm]	z_0[cm]	y_0[cm]
5×10^6	4.5	3.5	40	1.5	1.0	5×10^5	∞	20	y_0

At this point, we are only interested in reproducing the geometric aspects, thus scattering is turned off and only little absorption is used. [2]

Figure 4.5 visualizes the result for $y_0 = 4.4955\,\mathrm{cm}$. Each point (φ, θ) in the plot represents the initial angles of a photon in the simulation and its color represents the number of reflections it took the photon before it was terminated. The color bar translates this number to a log scale. Photons without successful reflections are excluded for better distinction from those with one reflection. The two circle-like areas that stand out represent the photons that are not trapped by total internal reflection. The left one corresponds to photons emitted in positive y-direction, hence colliding with the outer surface first. The right one represents photons in the opposite direction, which can reach the inner surface first. A fraction of them stands out with exactly one successful reflection, which has to take place at the inner surface.

In parts, the plot resembles the contour plots for the angle of incidence in figure

[2] Entirely disabling absorption is technically impossible, since the implementation relies on it to terminate photons without momentum in z-direction.

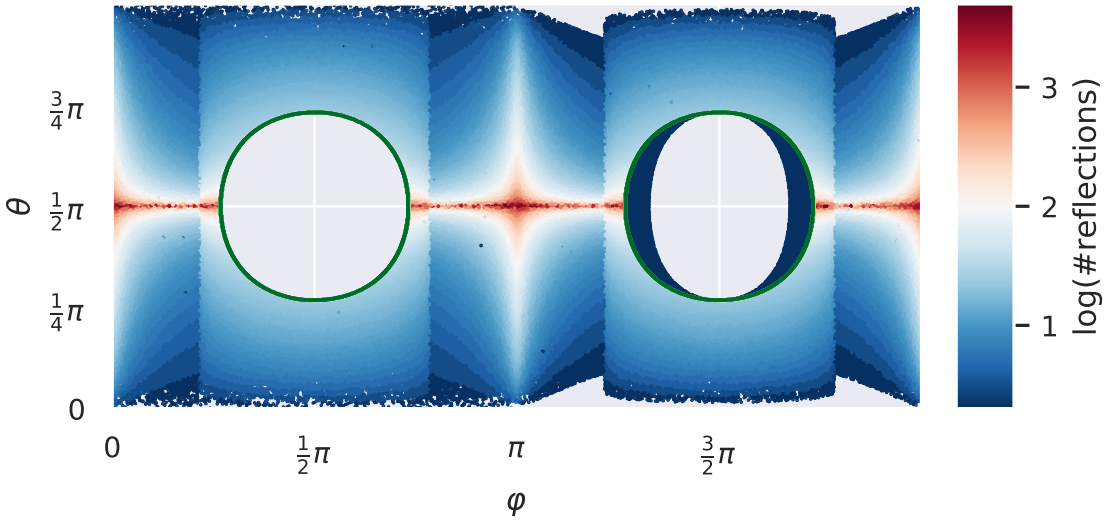


Fig. 4.5.: Two-dimensional map of the number of reflections depending on the emission angles φ and θ in spherical coordinates of the photon. Photons without successful reflections are excluded.

3.8. This is understandable, as photons with larger angles of incidence travel larger distances with each reflection and thus require less reflections to reach the end, as long as the reflection takes place between both surfaces and unless $\theta \to \frac{\pi}{2}$. Furthermore, the green lines mark the pairs of angles that are on the isoline of $\alpha = \theta_c$ in equation 3.23 with the used simulation parameters. The line exactly separates trapped photons from non-trapped photons.

The second feature that stands out is the uneven transition and the area around $\varphi = \pi$ and $\varphi = 0, 2\pi$. Those areas represent photons that are only reflected at the outer surface. This is due to φ angles with starting directions that are nearly tangential to the outer surface. The edge locations are exactly as predicted by equation 3.21 at $\gamma = 0.6784 \approx \frac{\pi}{5}$. Certainly, the contours in these areas do not match figure 3.8 at all. This is due to the fact that an increase of the angle of incidence here actually results in a smaller travel distance. In that case the outer surface is reached again even sooner, hence more reflections are required. The extreme case occurs at exactly $\varphi = 0$, $\varphi = \pi$ and $\varphi = 2\pi$, when the starting direction is tangential to the outer surface. Nevertheless, such an extreme case only occurs for emission close to the outer surface.

$\theta = \frac{\pi}{2}$ is a similar case, because the photons start without any movement in z-direction and therefore endlessly circle around in the x-y-plane. Those are the reason why some small absorption is always required in the simulation.

Last, four other small triangular areas at the outer border in the right part stand out. Those areas are missing a color because the photons actually reach the end on a direct path without reflection. The areas only become substantial in size if the wall thickness is not insignificant compared to the distance to the end in z-direction. In

this case, they appear in the right half because the light entry point y_0 was very close to the outer surface and thus only photons initially facing "down" had enough space to travel prior to their first reflection.
In appendix C.1, more of these plots are included for different sets of parameters. Most of the discussed areas appear there as well with varying shapes. They all have in common that they agree with the theoretical considerations and the equations from chapter 3.5.2. Moreover, as expected, the size of the loss area increases with y_0 moving away from the outer surface.

Subsequently, we want to analyze the consequences for the capture efficiency of the tube. For that purpose we set up a new simulation using the parameters in table 4.2

Tab. 4.2.: Paramters for the simulation of the capture efficiency.

N	R_o[cm]	R_i[cm]	L[cm]	n_1	n_2	λ_a[cm]	λ_s[cm]	z_0[cm]	y_0[cm]
5×10^6	4.5	0	60	1.5	1.0	5×10^5	∞	30	y_0

We define the two-sided efficiency $\epsilon_{two} = \frac{N'}{N}$, where N' is the number of photons, which is detected by both ends of the tube. Figure 4.6 displays simulation results for the efficiency with varying y_0. The orange line in figure 4.6 marks the outer surface.

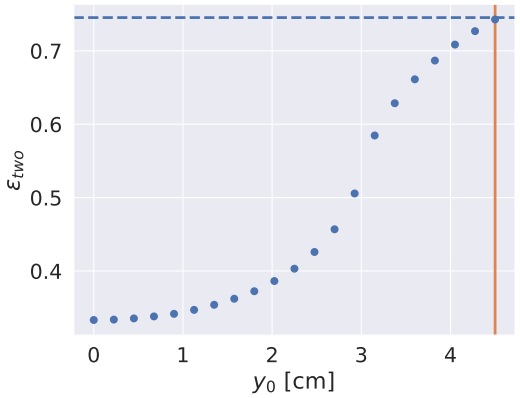


Fig. 4.6.: Two-sided efficiency for varying light emission point. The orange line marks the outer surface. The dashed line marks the theoretical maximum efficiency.

This is indeed where the efficiency reaches its maximum value. The dashed line marks the theoretical value of the maximum capture efficiency according to equation 3.25. For the used simulation parameters the equation yields $\epsilon_c = 0.7454$.
Overall, the implementation exactly reproduces all theoretical expectations for the tube geometry. This indicates light propagation works correctly.

4.3.5 Verification of the Beer-Lambert law

Last, we confirm the correct implementation of the Beer-Lambert law. This can be done by counting the number of photons that neither have left the tube wall nor have been absorbed or scattered before reaching a distance d. This means that the distance d is compared to the individual photon path lengths since the photons do not travel on a straight path in the tube geometry.
For the measurement, it is necessary to temporarily add the line $out \leftarrow$ `false` after line 16 in algorithm 2. In that way, a scattered photon behaves like an absorbed one and does not contribute to the intensity of a collimated light beam anymore. Running the simulation with λ_a for absorption and λ_s for scattering then should recreate the Beer-Lambert law with λ consistent with equation 4.5.
We record the total travel distances $d[j]$ of each photon j in a simulation with the parameters defined in table 4.3. The tube is long enough for all photons to interact

Tab. 4.3.: Simulation parameters for the verification of the Beer-Lambert law.

N	R_o[cm]	R_i[cm]	L[cm]	n_1	n_2	λ_a[cm]	λ_s[cm]	z_0[cm]	y_0[cm]
10^7	4.5	4.3	10^5	1.58	1.0	λ_a	λ_s	5×10^4	4.499

before they reach the end. To get the measurement described above, all photons that are not captured by TIR have to be filtered out. The remaining number then is the initial number of photons N_0 in the Beer-Lambert law.
We count the number of remaining photons at 100 distance sample points $d_i \in [0, d_{max}]$, i.e. $N(d_i) = \sum_{j=0}^{N_0-1} \Theta(d[j] - d_i)$.[3] Figure 4.7 shows the result of such a measurement and another one can be found in figure C.6 in the appendix. Additionally, the plots display the theoretical curve according to Beer-Lambert.

That confirms, that algorithm 2 correctly applies absorption and scattering at distances compliant with the Beer-Lambert law. The actual scattering then uses the sampling of isotropic directions from section 4.1.2.
In summary, this section demonstrated that the simulation algorithm reproduces all expectations for light propagation in the implemented physical framework.

[3] Θ is the Heaviside function

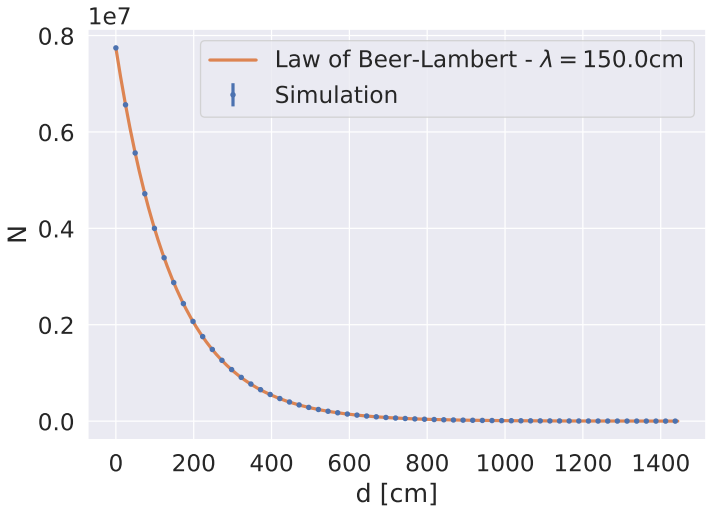


Fig. 4.7.: Simulation of absorption and scattering and theoretical curve. The simulation used the parameters $N_0 = 7\,743\,659$, $\lambda_a = 600\,\mathrm{cm}$ and $\lambda_s = 200\,\mathrm{cm}$, which corresponds to $\lambda = 150\,\mathrm{cm}$ for the theoretical curve. The statistic uncertainties of the measurements with $\Delta N = \sqrt{N}$ are not displayed here, as they are too small.

5 Parallelization and optimization on the GPU

> *The Free Lunch Is Over.*
>
> **— Herb Sutter**
> (ISO C++ committee convener)

The main objective of this work is to create a fast simulation software for light propagation in the WOM geometry, aiming at a simulation throughput of several millions of photons per second. To accomplish this goal the processing power of modern GPUs is utilized. In this chapter, the parallelization and performance optimization of the time consuming parts of the simulation for this hardware platform are addressed.

5.1 Parallelization

5.1.1 Random number generation

We start with a discussion of the choice of a parallel pseudo-random number generator (PRNG), considering random number quality as well as generation performance. The PRNG is an important part of the simulation, which can have a significant impact on the simulation results. The prototype simulation [9], for instance, has been reported to generate incorrect event distributions at the detection plane when using GPU generated random numbers.
As a workaround the prototype instead uses pre-calculated random numbers from a CPU implementation of the Mersenne Twister MT19937 [38], which is famous for high-quality random numbers. However, the approach is rather slow compared to a pure GPU solution, owing to the high level of parallelism on the GPU and the fact that the CPU implementation involves data transfers over the slower PCIe to the GPU. Hence, we prefer the pure GPU solution, but only if we can ensure that the random numbers are of sufficient quality without patterns.

We use the cuRAND library [39], which provides several parallel PRNGs for the GPU. The prototype uses the MTGP32 generator, a GPU variant of the Mersenne Twister

from the cuRAND library [40], which produces the faulty results mentioned above. We employ the cuRAND implementation of Marsaglia's XORWOW generator [41] instead. An NVIDIA CUDA performance report [42] shows that it samples faster than the MTGP32. Furthermore, being a Mersenne Twister, the MTGP32 requires a very large state, while the XORWOW only requires five variables to maintain its state, which reduces its memory footprint. It has a period length $> 2^{190}$, which is less than the MTGP32 with its enormous 2^{11214}, but still long enough for all our needs.
In order to properly use the parallel XORWOW generator with statistically uncorrelated values, it requires an individual state for each thread. Each state is based on the same seed and therefore part of the same sequence of numbers. More precisely, the individual states are set up by skipping ahead 2^{67} entries in the whole sequence defined by the seed. This results in yet again smaller subsequences per thread. However, even with billions of photons per thread, each of those photons can consume 10^{11} random numbers before subsequences of the threads start overlapping.

The cuRAND documentation includes reports on rigorous statistical tests for the randomness of the generators. Such tests are hypothesis tests for patterns in the distributions generated from the PRNG with significance levels $< 0.1\%$. XORWOW has been tested with "BigCrush" from [43] and the full set of NIST tests [44]. It passes all of those tests with occasional failures of individual tests in single runs. Additionally, it has been tested in the context of the development of this simulation with the "Dieharder" tests from [45], in which it passed 111 out of 114 tests.
Note that only very few generators pass all of those tests. For instance, the famous Mersenne Twister in its standard implementation consistently fails two of the "BigCrush" tests, and also its MTGP32 variant from cuRAND occasionally fails some of the tests.
To sum up, the quality of the random numbers from the chosen PRNG is mostly excellent and is not expected to affect the results. It is likely that the flawed results in the prototype were only due to an incorrect usage of the MTGP32 generator and not a result of the PRNG quality itself.
In fact, all validations shown in the previous chapter that involve random numbers used the XORWOW from cuRAND and no suspicious artifacts were observed.

5.1.2 Initialization and main loop

The parallelization of the initialization and the main loop can be handled in a straight forward manner using data parallelism. Since the calculations for the individual photons do not depend on other photons, each can be calculated from start to end in any order. With data parallelism we distribute the data of N photons equally across any available number p of compute nodes, which run through the calculations in parallel.

For our case of a CUDA GPU, we can only specify the number of threads in each block and the number of independent blocks while the CUDA runtime scheduler takes care of distributing those to actual compute nodes. Due to the low overhead for thread creation, achieving pure data parallelism is then as simple as creating in total N threads for the N photons. Scaling the number of threads with the problem size like this is a usual approach to CUDA, since this way the runtime scheduler ensures the automatic scalability for different hardware.
Conceptually, using this simple parallelization scheme, the simulation structure is given by:

1. Use $T \in [0, 1024]$ [1] threads per block and $B = N/T$ blocks[2].
2. Allocate memory for the data arrays ($positions$, $directions$, $results$, ...) and the array of PRNG states with each holding $T \cdot B$ elements.
3. Setup the PRNG states with $T \cdot B$ threads.
4. Call the initialization with $T \cdot B$ threads.
5. Call the main loop with $T \cdot B$ threads.
6. Copy the results to the CPU.
7. Write the results to an HDF5 file.

Steps 3, 4 and 5 call separate CUDA kernel functions with the specified number of blocks and threads which are committed to the CUDA runtime scheduler. In each of the kernel functions a thread uses its unique global id number to identify its piece of data to work on. In step 3 each thread sets up its own PRNG state based on the same global seed and saves it in global memory. In step 4 each thread initializes a single photon and saves the starting position and direction after the wavelength shift in global memory. In step 5 each thread loads the starting position and direction of its photon, runs it through the main loop and saves the results in global memory.

The data arrays such as $positions$ and $results$, which hold multi-dimensional values, are implemented using a Structure of Arrays layout (see chapter 2.2.2). This is sufficient to get coalesced memory accesses throughout all of the kernel functions. Moreover, the kernel functions load data from global memory to registers only once at the beginning and write it back once at the end. This results in a very good compute-to-global-memory-access ratio for the algorithm. Finally, step 6 is the only data transfer over the slow PCIe.

[1] T has to be tuned depending on the resource consumption of the code.
[2] Precautions for the case $(N \bmod T) \neq 0$ required.

5.2 Performance optimization

After establishing a simple parallelization in the previous section, this section investigates performance issues and possible optimizations.

5.2.1 Evaluation of the performance

In order to evaluate the simulation performance, a benchmark setup is defined in table 5.1.

Tab. 5.1.: Simulation setup for the benchmarks.

N	R_o[cm]	R_i[cm]	L[cm]	n_1	n_2	λ_a[cm]	λ_s[cm]	z_0[cm]	y_0[cm]
N	4.5	4.3	100	1.5	1.0	600	600	90	4.4955

Ideally, the run time of the simulation scales linearly with the number of simulated photons N. Apart from the number of photons, major factors for the run time are: the thickness of the wall, since thinner walls increase the number of reflections; the length of the tube L; and the absorption and scattering length λ_a and λ_s. The parameters for the benchmark are chosen such that they are similar to the actual experiment: with the latest measurements [46], attenuation lengths of $\lambda_{att} \approx 300\,\mathrm{cm}$ are expected and with $L = 100\,\mathrm{cm}$ even the longest tubes are covered.
Using this setup, a benchmark consisting of 20 repetitions for a range of values N was conducted. With timers inside the code, the run times of the individual parts of the simulation are obtained by calculating the mean of all iterations. Figure 5.1 presents the benchmark result on a double logarithmic scale. All benchmarks are executed on a system equipped with an Nvidia GeForce GTX 1050Ti GPU and an Intel Core i7-3770 CPU. The individual lines correspond to the steps 3 to 7 above, complemented by the total run time. However, it has to be noted that the total run time is not the sum of all displayed run times. For example, overheads for program startup, reading and interpreting the input file and cleanup are not measured by the timers. Those overheads appear in the plot as initially smaller slope in the total run time.
The result immediately reveals a major issue with this implementation: The setup of the PRNG states does not scale linearly with N. Starting from $N = 5 \times 10^5$ photons it even dominates the total run time. The reason for this is that each thread sets up a state using the same seed in order to avoid statistical correlations with different seeds. As stated in section 5.1.1 this is achieved by each thread skipping to a subsequence 2^{67} entries ahead of its predecessor. This skipping is expensive and using N threads, the last one has to skip $N - 1$ subsequences.

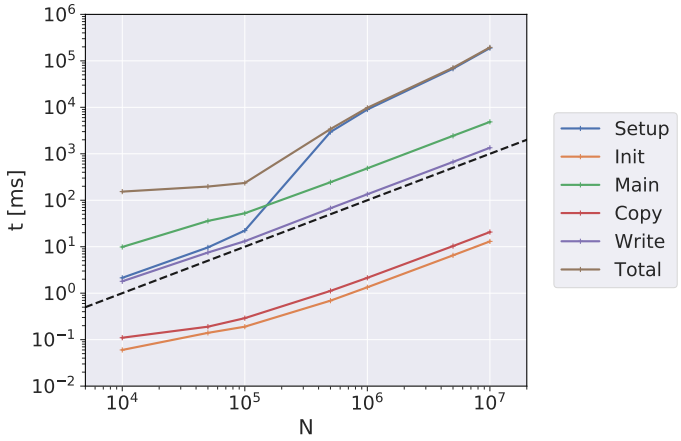


Fig. 5.1.: Benchmark result for the simple parallelization. The dashed black line serves as optical guidance for linearity in the double log scale.

Apart from that, the remaining run times scale fairly linearly as expected. Moreover, the performance of the main simulation loop is already very good with this simple parallelization, as it achieves a throughput of more than one million photons per second.

5.2.2 Optimization of the initialization

There are two possible solutions to optimize the initialization of the PRNG.
First, we could use a different seed for each thread, thus avoiding skipping of subsequences. Unfortunately, this has the disadvantage that there is no guarantee for obtaining statistically uncorrelated values if different seeds are used, while this can be guaranteed for different subsequences of the same seed.
As a second solution, a constant number p of PRNG states with $p \ll N$ can be used. This seems reasonable, considering that when using arbitrarily big values of N, the scheduler never assigns N threads simultaneously to the GPU's Streaming Multiprocessors (SMs) anyway. Hence, setting up only as many individual PRNG states as there are threads assigned to SMs is sufficient. In that way, the initialization time does not scale with N any more and becomes insignificant for small p. However, since we have no control over the thread assignment of the runtime scheduler, we could only safely share PRNG states between threads in a block using synchronization, which is undesirable. Therefore, we have to use a constant number of threads as well and assign the tasks manually to the threads.

Nonetheless, the second solution is the preferred one. Implementing this requires so-called grid-stride-loops for the kernels of the initialization and the main loop. Algorithm 3 demonstrates the structure of the kernels using a grid-stride-loop. The function in line 3 represents the bodies of the previous kernel functions of step 4 and

Algorithm 3 Grid-stride-loop kernel

1: $PRNGState \leftarrow states[thread\_id]$
2: **for** $i \leftarrow thread\_id, i < N$ **do**
3: FUNCTION$(PRNGState, i, *params)$
4: $i \leftarrow i + p$
5: **end for**
6: $states[thread\_id] \leftarrow PRNGState$

5 with their respective parameters $*params$. This kernel is now launched with only p threads and p PRNG states in the $states$ array. Each thread starts with a different loop index i, determined by its unique id, performs the simulation of the i-th photon and continues in the loop with a stride of p. In this loop the j-th photon is processed exactly once by thread $(j \bmod p)$.
The only part left for this solution is the choice of p in order to get the full workload on the GPU. The maximum number of active threads depends on the number of SMs of the GPU model. Depending on the resource consumption of the kernel functions (not the GPU model), all recent GPUs can have at maximum 2048 active threads per SM. Roughly speaking, the blocksize T and the number of blocks B have to be chosen such that each SM of the target GPU reaches at least the maximum possible number of active threads for the kernel function. Using too many blocks is not a problem, as the GPU can still queue up blocks.
That concludes the optimization. Conducting the benchmark again with the same setup yields the run times shown in figure 5.2.

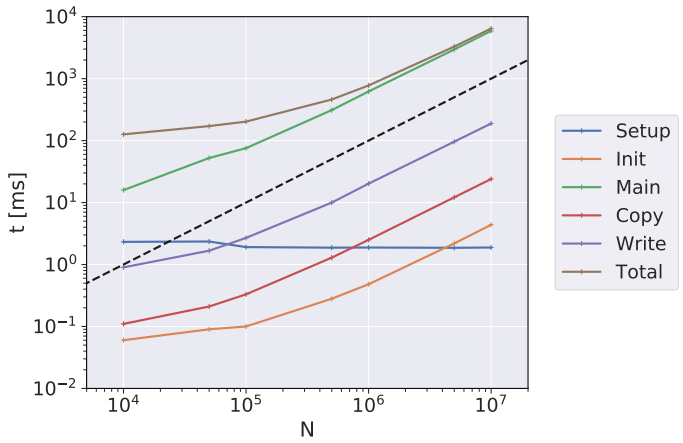


Fig. 5.2.: Benchmark result for the grid-stride-loop parallelization. The dashed black line serves as optical guidance for linearity in the double log scale.

As expected, the initialization does not scale with N any more and the run time is dominated by the main loop. Examining the simulation throughput of the benchmark results in figure 5.3 shows that apart from the constant initialization all timed components converge to a more or less constant rate for high N. For smaller values

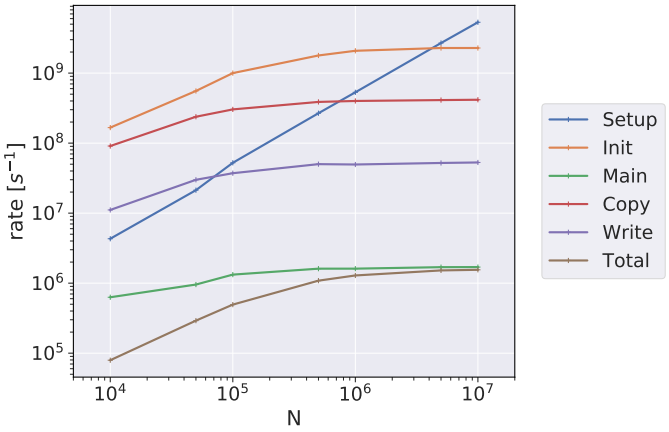


Fig. 5.3.: Simulation throughput for the grid-stride-loop parallelization. Higher values are more desirable.

of N the simulation is obviously less efficient, due to all overheads contributing to the run time in a higher proportion.

5.2.3 Further performance issues

Although the simulation is already very fast after the optimization from the previous section, there is still room for improvement.
The rather simple parallelization presented above suffers from branch divergence as discussed in chapter 2.2.2 and perhaps also from load imbalance. The problem arises from the irregular work load of the simulation of photons. Depending on a photon's initial direction of emission, the number of reflections it undergoes in the tube wall can vary greatly. In addition to that, scattering and absorption also influence the number of reflections. Figure 5.4 depicts a distribution for the number of reflections of photons using the benchmark setup with $N = 10^7$. According to this distribution, it is highly likely that most threads in one warp simulate only very few reflections while very few threads in the same warp have to simulate many. This causes the branch divergence of the threads during the main simulation loop. While some threads are already done with the loop, they have to remain inactive due to the nature of warps. Only when the last one is finished, all threads reunite, write their results to global memory and load the next photon into registers in unison. Considering that there are few photons in the distribution that take an order of magnitude more reflections than others it can occur that only a single thread in a warp is active for a long time.

In order to analyze the significance of branch divergence caused in that way, we inspect the number of reflections n_i of all photons $i \in [0, N-1]$ using the benchmark setup with $N = 5 \times 10^6$. Since algorithm 3 distributes the photons to threads and

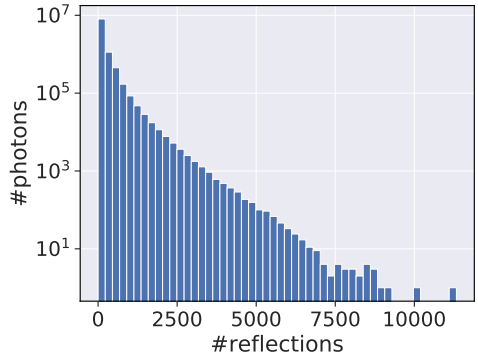


Fig. 5.4.: Distribution of the loop iterations for all photons on a log scale. Very few photons accomplish more than 7000 reflections due to the low probability of long travel distances caused by attenuation. This results in few bins being occupied by less than ten photons or only one photon.

therefore warps in a deterministic way, we can easily identify photons that were processed in the same warp by rearranging the n_i in groups of 32.
The result n_{jk} is the number of reflections that a photon experienced, as calculated by the k-th thread of warp j[3] with $k \in [0, 31]$. Since all threads in a warp have to wait for the last one to finish, the load of warp j is effectively determined by the highest reflection count of all photons:

$$load_j = \max_k(n_{jk}) \quad .$$

Using the load, we define the divergence of thread k from the load of warp j:

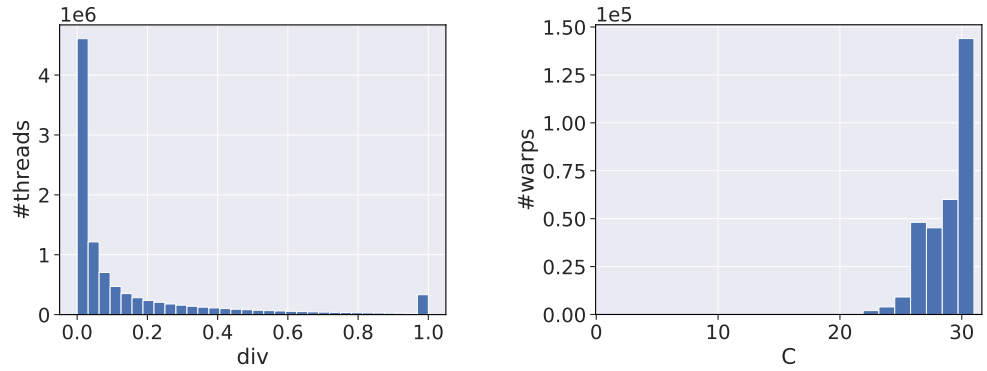
$$div_{jk} = \frac{n_{jk}}{load_j} \quad . \tag{5.1}$$

The number $div_{jk} \in [0, 1]$ can be interpreted as the fraction of time that thread k in warp j is active compared to the longest running thread in warp j. Ideally, it would be close to 1 for all j and k, which means that all threads in all warps execute without branch divergence most of the time.
We can also count the number of threads that are inactive in a warp for at least a fraction s of the total time:

$$C_j(s) = \sum_{k=0}^{31} \Theta(s - div_{jk}) \quad . \tag{5.2}$$

Figure 5.5 shows the flat distribution of div_{jk} and the distribution of $C_j\left(\frac{1}{2}\right)$ for the benchmark. Unfortunately, the value of div_{jk} is close to 0 for the majority of threads in all warps. The rightmost bin is only slightly higher than $\frac{N}{32}$, hence it almost purely

[3] For the sake of simplicity we will continuously refer to warp j. Correctly it would be the j-th group of 32 and warp $(j \mod \frac{p}{32})$, since there are only p threads and $\frac{p}{32}$ warps.



(a) Flat distribution of the div parameter of all threads.

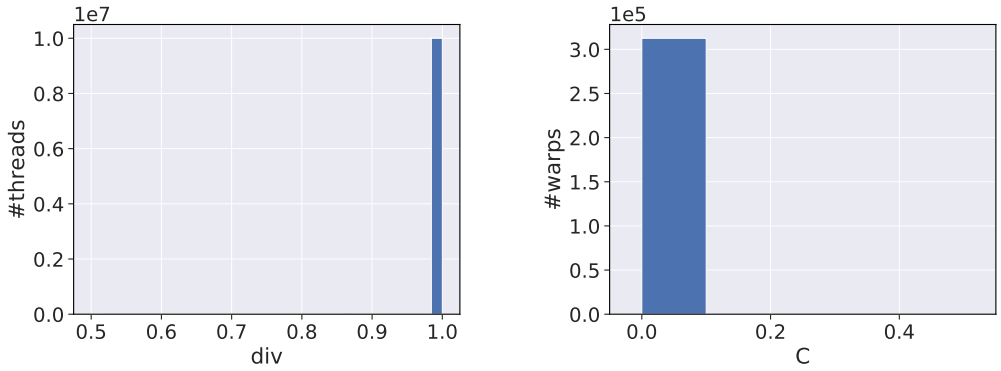
(b) Distribution of the number $C\left(\frac{1}{2}\right)$ of inactive threads in the warps.

Fig. 5.5.: Distributions to showcase branch divergence in the simulation.

consists of the maxima in the warps. In addition, figure 5.5(b) shows that in the majority of cases almost the entire warp indeed is idle for at least half the time. To summarize, the algorithm so far acts rather inefficiently concerning branch divergence and needs improvement.

Simplified solution

Theoretically, the problem could be solved in a simple way by sorting the photons according to their number of reflections. In that way, the threads in each warp would always have a similar workload, reducing the idle time to a minimum. Figure 5.6 shows the results for divergence and idle threads after sorting the number of reflections n_i from the same benchmark run. Clearly, the issue of diverging branches



(a) Flat distribution of the div parameter of all threads.

(b) Distribution of the number C of inactive threads in the warps.

Fig. 5.6.: Distributions for divergence and idle threads in the simulation after sorting n.

disappears.

In order to verify that the presented results affect the performance also in practice, a simplified replication of the parallel GPU algorithm was implemented in CUDA. This approach provides more control over the assignment of workloads than an analysis in the actual simulation. Algorithm 4 demonstrates the simplified version. Basically,

Algorithm 4 Simplified algorithm

```
1: PRNGState ← states[thread_id]
2: for i ← thread_id, i < N do
3:     iterations ← n[i]
4:     res ← 0
5:     for j ← 0, i < iterations do
6:         res ← res+ sum(X random numbers)
7:     end for
8:     i ← i + p
9: end for
10: states[thread_id] ← PRNGState
```

the main simulation loop is replaced by a loop that calculates a sum of random numbers. The number of iterations for that loop is determined by $\boldsymbol{n}$, the number of reflections of the photons in a real simulation. The variable X is used to tune the workload of a single iteration of the loop. It should not be too small in order to get a good compute-to-global-memory-access ratio.
The simplified program was executed with the simulation result $\boldsymbol{n}$ of the benchmark setup with $N = 10^7$ and a value of $X = 100$. We examine the execution times of using the original array $\boldsymbol{n}$, a sorted version of it and one where each entry is replaced by its mean $\langle \boldsymbol{n} \rangle$.
Although this algorithm has nothing to do with the simulation of light, the workload distribution is similar to the distribution of the simulation that created $\boldsymbol{n}$. Moreover, it has the same problem of branch divergence. Thus, the execution time of the first case corresponds to the real simulation, the second one to the time of a simulation with input sorted according to the individual work load per photon, and the third one to a simulation with equal work load for each photon and no branch divergence. Table 5.2 shows the results for taking the mean of 100 repetitions.

Tab. 5.2.: Run time and speedup comparison for the simplified program.

method	time	speedup
Original	10 432 ms	–
Sorted	1359 ms	7.7
Mean	1340 ms	7.8

Consequently, we can conclude three things. First, the irregular workload indeed affects the performance. Second, eliminating branch divergence and load imbalance entirely gives a speedup of 7.8. Third, sorting the input yields almost the same

performance as the divergence-free case, proving that this is an acceptable solution to minimize branch divergence.
Nevertheless, so far we have been looking at the number of reflections in retrospect. In order to improve the performance, however, we have to be able to sort them beforehand. In the next section, we will explore four possible optimizations to minimize the branch divergence in the real simulation.

5.2.4 Optimization of the branch divergence

Sorting the photons

In chapter 4.3.4 we analyzed the number of reflections that a photon accomplished in the space of emission angles φ and θ using a color map. Figure 5.7 shows such a map for a simulation without attenuation. In this instance the used tube parameters are close to some of the manufactured tubes. Those results are useful for obtaining a

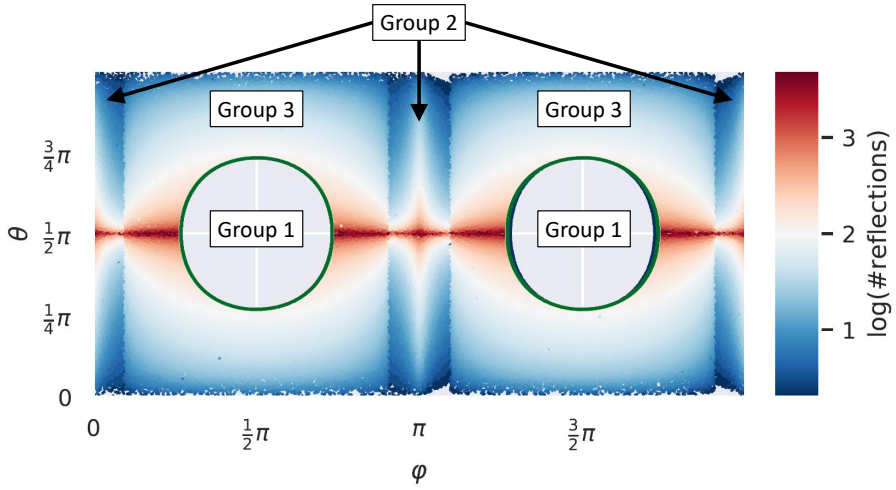


Fig. 5.7.: Map of the number of reflections in the space of emission angles for the parameters: $R_o = 4.5\,\mathrm{cm}$; $R_i = 4.3\,\mathrm{cm}$; $y_0 = 4.4955\,\mathrm{cm}$; $L = 40\,\mathrm{cm}$; $z_0 = 20\,\mathrm{cm}$. The annotations mark three groups of photons. Group one is presumably not captured in TIR. Group two is only reflected at the outer surface. Group three is reflected at both surfaces.

scheme for sorting the photons without prior knowledge of the number of reflections, which we obviously do not have.
Based on theoretical considerations from chapter 3.5 we were able to discriminate three regions in the space of emission angles. Using those criteria yields a coarse sorting of the photons in three groups. The first one is based on photons that are presumably not captured in TIR. The condition for this group is given by $\cos(\alpha) > \cos(\theta_c)$ with α being the angle of incidence at the outer surface determined by equation 3.23. In the second group we have all photons that are only reflected at the

outer surface. Equation 3.21 gives a condition for the boundaries of φ that separates those rays from rays reflected between both surfaces. The remaining rays, which are reflected between both walls, are in the last group.
The first group already has an even workload with no significant branch divergence The two remaining groups, however, still have a large variance in the number of reflections according to figure 5.7 and thus require further sorting. With only a single sorting criterion, the angle θ could be a good choice for the third group. It does not yield a perfect sorting everywhere but it still looks like an overall good approximation that separates the photons with high reflection count at $\theta \to \frac{\pi}{2}$ from those with few at the upper and lower edges. The second group on the other hand looks more complicated. The isolines bear some resemblance to branches of hyperbolas, but there is no obvious equation that describes them. Using the angles alone as possible sorting criteria, θ again looks like the better choice, since the variance for the number of reflections is definitely smaller for a fixed θ.

Algorithm 5 summarizes the sorting approach in pseudo code. Here, the function `ArgSort(x)` returns the index array that would sort x. The algorithm has to be applied to the photon output of the initialization kernel. The sorted result is used as input to the main kernel.

Algorithm 5 Sorting of photons

$\gamma \leftarrow \arccos\left(\frac{R_i}{y_0}\right)$
for $i \leftarrow 0, i < N$ **do**
    $\cos(\alpha) \leftarrow \left(\sin(\theta[i])\sqrt{R_o^2 - \cos^2(\varphi[i])y_0^2}\right)/R_o$
    if $\cos(\alpha) > \cos(\theta_c)$ **then**
        Add i to $G1$
    else if $\varphi[i] \in ([0,\gamma] \cup [\pi-\gamma, \pi+\gamma] \cup [2\pi-\gamma, 2\pi))$ **then**
        Add i to $G2$
    else
        Add i to $G3$
    end if
end for
$ind2 \leftarrow$ ArgSort$(\theta[G2])$
$ind3 \leftarrow$ ArgSort$(\theta[G3])$
$G2 \leftarrow G2[ind2]$
$G3 \leftarrow G3[ind3]$
$photons \leftarrow$ Concatenate$(photons[G1], photons[G2], photons[G3])$

However, there are two issues with this optimization. First, it requires additional time for preprocessing which diminishes any potential overall speedup. Second, it only evens out the static load imbalance in each warp due to the starting direction while scattering and absorption can still randomly change the number of reflections causing branch divergence again.

Manipulating the isotropic sampling

Instead of sorting the photons after the ray generation, it might be better to generate rays in a more controlled way. The idea is to divide the sampling space for the isotropic emission in $\frac{N}{32}$ equal sub spaces. For each group of 32 photons that are going to end up in the same warp, the directions are sampled from only one of the sub spaces.
The isotropic directions so far are defined by uniform sampling of $\varphi \in [0, 2\pi]$ and $z \in [-1, 1]$, using equation 4.2 for the final direction. In order to get the sub spaces we first have to define a two-dimensional grid in the sampling space with $\frac{N}{32}$ cells in total. Length and width of the grid are defined by two integers N_1, N_2 with $N_1 \geq N_2$ via:

$$\frac{N}{32} = N_1 \cdot N_2 \quad .$$

The i-th photon is sampled from cell $j = \left\lfloor \frac{i}{32} \right\rfloor$. The grid coordinates (l, k) of the cell are defined as:

$$(l, k) = \left(\left\lfloor \frac{j}{N_1} \right\rfloor, j \bmod N_1 \right) \quad .$$

Finally, the width $(\Delta\varphi, \Delta z)$ is defined as:

$$(\Delta\varphi, \Delta z) = \left(\frac{2\pi}{N_1}, \frac{2}{N_2} \right) \quad .$$

Thus, the direction for the i-th photon is obtained using equation 4.2 after uniform sampling of $\varphi \in [\Delta\varphi \cdot l, \Delta\varphi \cdot (l+1)]$ and $z \in [\Delta z \cdot k - 1, \Delta z \cdot (k+1) - 1]$.

The advantage of this procedure is that it does not require additional preprocessing time and it might get more even workloads in the warps, since the sorting of the previous approach is not perfect.
The drawback of this method is that it could harm the quality of the random directions. Applying a sampling scheme like this fixes the number of rays generated in each cell to a constant number. For big values N and actual isotropic sampling that might be approximately true for all cells, but never exactly. This could be fixed, by adding random offsets to the cell bounds. However, section 5.2.5 shows that the performance gain of this approach is not worth to spend more time on improvements.

Restructuring the simulation loop

Unfortunately, both approaches presented so far have disadvantages and cannot compensate branch divergence that is caused by changes of the number of reflections due to scattering or absorption. Both occur at random and thus with the used simu-

lation algorithm the number of actual reflections becomes unpredictable. In order to regard absorption and scattering it is therefore not sufficient to only manipulate the input.
The branch divergence in the algorithm arises because each thread processes a photon from start to end. It turns out that this can be changed by restructuring the simulation loop, such that each thread gets the opportunity to move on to another photon after each simulation step.
The restructured loop is presented in algorithm 6. In this approach the grid-stride-

Algorithm 6 Restructured simulation loop

```
1:  PRNGState ← states[thread_id]
2:  out ← false
3:  d_t ← 0
4:  i ← thread_id
5:  PREPARE_DATA(i, pos, dir, *params)
6:  while i < N do
7:      SIMULATE_STEP(PRNGState, pos, out, d_t, dir, *params)
8:      if out then
9:          SAVE_SESULT(pos, dir)
10:         i ← i + p
11:         PREPARE_DATA(i, pos, dir, *params)
12:     end if
13: end while
14: states[thread_id] ← PRNGState
```

loop and the main simulation loop for a single photon collapse to one loop. In each iteration, this loop performs one simulation step for the i-th photon in line 7, which is the loop body of algorithm 1. Subsequently, if the photon terminates, it writes the result to memory and updates i. **PREPARE_DATA** loads and prepares the data of the new photon i for the next iteration.
Using this new structure, all threads in a warp always perform an update step of the simulation in unison. Branch divergence can still occur at line 8. However, after any diverging threads have finished that short extra path, they immediately reunite and continue in the next iteration with a new photon.

Adding dynamic scheduling

However, this solution actually does not fix an eventually existing load imbalance between the p threads. A load imbalance may occur due to the fact that the number and indices of photons that are assigned for processing to a thread are statically determined. Therefore, the total workload of the threads as sum of the workload for processing all assigned photons varies. Nevertheless, for $p \ll N$ the different work loads of individual photons even out when summed up. Hence, aiming at $N \geq 10^6$

and with $p \leq 12\,288$ for the used GTX 1050Ti[4] a major load imbalance cannot exist. We can still reduce any existing load imbalance by allowing dynamic assignment of photons to threads. This can be accomplished by maintaining a single counter variable in global memory, which is accessible by all threads and initialized to p. The counter always remembers the highest photon id number that has not been assigned to a thread yet. Therefore it acts as the manager of a queue of photons. Once a thread has finished with a photon, it does not statically increment its index i by p anymore. Instead, it looks up the value of the global counter in order to determine the new value for i. Subsequently, the counter is incremented. To prevent race conditions with several threads trying to access and update the counter at the same time, this has to be performed with an atomic operation in line 10 of algorithm 6. Using the counter has the benefit that threads calculating mostly photons with fewer reflections can calculate more photons, which reduces the workload for threads with long running photons.

5.2.5 Final performance

Table 5.3 compares the final performance of the main simulation kernel for the different methods. For the results the mean of 100 repetitions per method was taken

Tab. 5.3.: Run time and speedup comparison of the different optimization methods.

method	time	speedup
Original	5912 ms	–
Sorting input	2503 ms	2.4
Cell sampling	2201 ms	2.7
Restructured	931 ms	6.4
Dynamic	824 ms	7.2

using the previously defined benchmark setup with $N = 10^7$. We observe that the imperfect sorting concept yields the weakest performance, closely followed by the cell sampling. What makes the sorting approach even worse is that it requires a preprocessing step that is not included in this run time. Apparently, both methods are not suited to achieve groups of 32 with similar work loads for all photons. After all, they do not incorporate scattering and absorption and cannot reduce the branch divergence as much as sorting the photons in retrospect.
Good results are obtained with the restructured algorithm with a speedup of 6.4 w.r.t the non-optimized version. This is much closer to the speedup of 7.8 that was observed for an even workload in the simplified program. Adding the dynamic photon assignment pushes the speedup further to 7.2, which indicates that the summed load of the threads indeed was slightly imbalanced otherwise. The still

[4] This GPU has 6 SMs with up to 2048 active threads per SM. Hence, there are at most 12 288 active threads.

existing small discrepancy is likely caused by two factors. First, as stated earlier, in the restructured algorithm branch divergence still occurs for a few instructions every time a photon terminates. Second, another access to global memory was added for the counter which is slower than a usual one, since it is an atomic operation. Furthermore, if many threads try to update the counter at the same time, they have to wait for each other. In conclusion, this is probably the closest we can get to the performance of the even workload.

Figure 5.8 depicts the throughput of the final optimized simulation using the restructured algorithm and the dynamic photon assignment. The part for the main

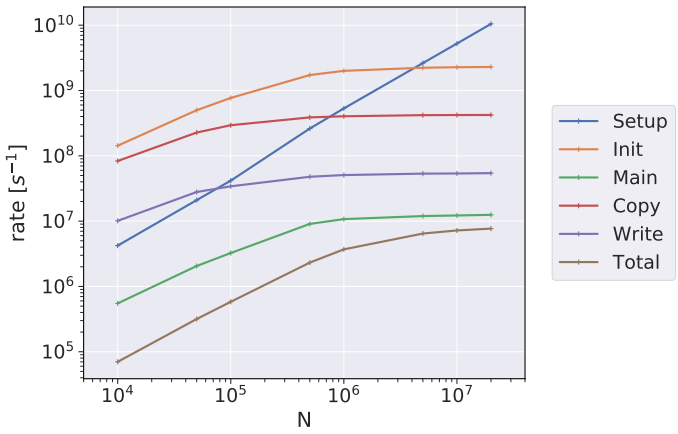


Fig. 5.8.: Simulation throughput for the final improved simulation. Higher values are more desirable.

loop alone now exceeds 10^7 photons per second. Also including the total program run time the simulation still reaches around 7.5×10^6 photons per second. After applying all optimizations the simulation truly achieves excellent performance. It is unlikely that there is much room for further improvements. Therefore, we want to close this chapter.

6 Simulation results

Each simulation run saves results in an output file in HDF5 file format [47]. In the implemented format, an output file is organized in numbered datasets. Each dataset is a result of a single simulation run. If an existing file is given as output file for a simulation run, the new dataset is appended to the file. A dataset consists of metadata in the form of HDF5 attributes for all simulation parameters of the run and the results for the individual photons. For each photon, the coordinates of the final position x, y and z, two angles φ and θ, which define the final direction in spherical coordinates and the total distance travelled d are saved. Additionally, a variable $exit\_code$ saves the cause of the photon termination (detected, absorbed, left tube) encoded as an integer value. In this chapter, the simulation results are used to analyze various aspects of light propagation in the WOM.

6.1 Light distribution in detection plane

First, we want to visualize the light distribution in the detection plane. We run simulations with varying position of the light entry point z_0 and the setup in table 6.1. We read out the values of x and y of photons that reached the detection plane at

Tab. 6.1.: Parameters for the simulation of light distributions.

N	R_o[cm]	R_i[cm]	L[cm]	n_1	n_2	λ_a[cm]	λ_s[cm]	z_0[cm]	y_0[cm]
2×10^7	4.5	3.5	60	1.5	1.0	100	100	z_0	4.4955

$z = 0$. Figure 6.1 shows two-dimensional distributions for several light entry points. They all share the same color bar, which is standardized to the overall highest bin in the figure. This standardization immediately visualizes the decrease of intensity for greater light entry distances due to light attenuation. To get a more detailed view of the distributions, we transform the data points to polar coordinates:

$$r = \sqrt{x^2 + y^2} \quad ,$$
$$\varphi_R = \mathrm{atan2}(x, y) \quad .$$

Here, $\mathrm{atan2}$ denotes the two-argument $\arctan$. Contrary to the regular $\arctan\left(\frac{y}{x}\right)$ it takes the quadrant of x and y into account and gives a result $\in [0, 2\pi]$.

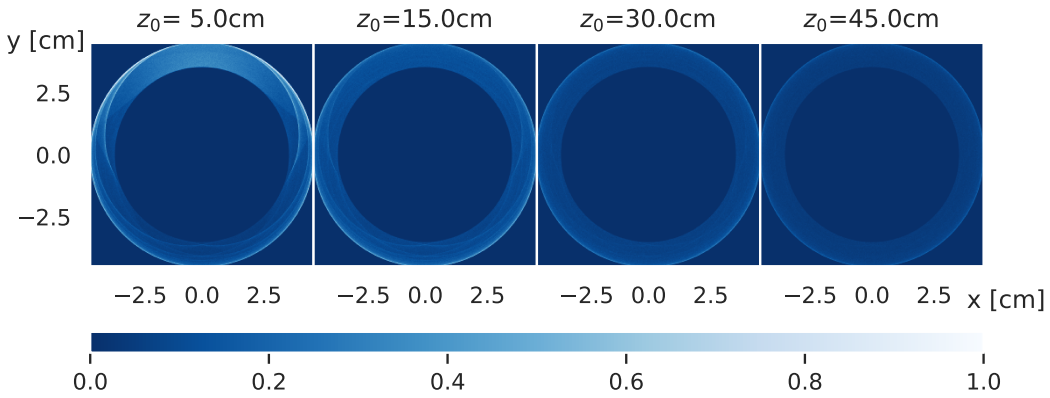


Fig. 6.1.: Two-dimensional histograms of the light distribution in the detection plane.

Figure 6.2 shows a histogram in these coordinates for the leftmost plot in figure 6.1. The light distribution is clearly non-uniform in φ_R with a peak at $\varphi_R = 0$, which

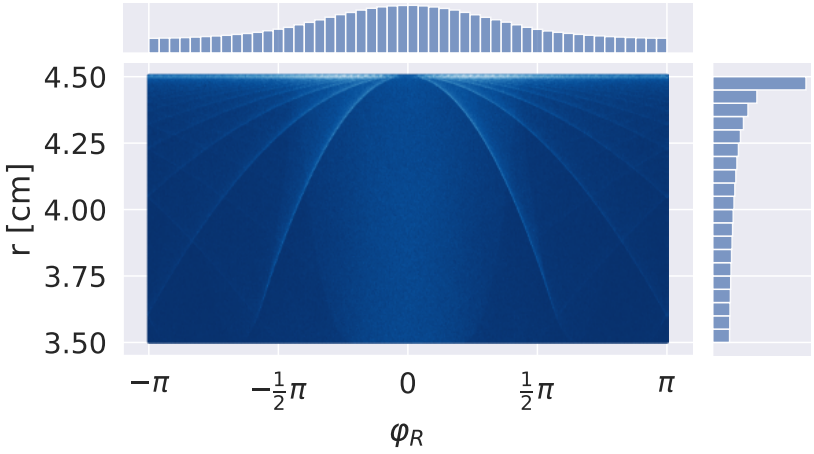
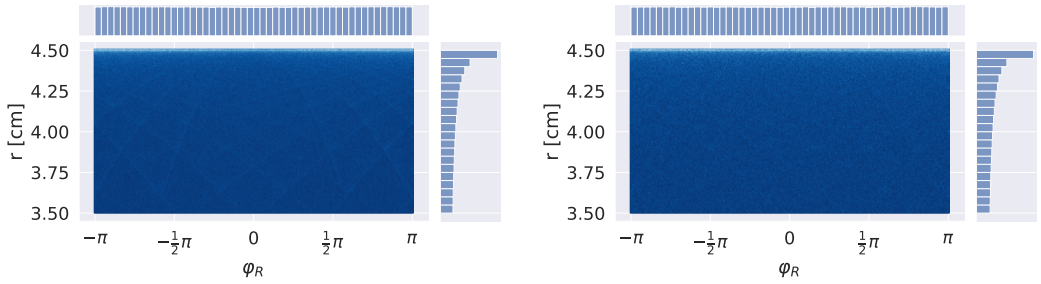


Fig. 6.2.: Two-dimensional histogram of the light distribution in polar coordinates for $z_0 = 5\,\mathrm{cm}$. At the top and to the right, the marginal distributions of φ_R and r are included. The patterns in the joined distribution are discussed in detail in the text.

corresponds to the position of the light source on the outer surface. Figures 6.3(a) and 6.3(b) demonstrate, however, that the distribution of φ_R becomes uniform for higher distances of light emission. This is expected, since the longer travel distance gives photons the opportunity to spread out more evenly.

The distribution of φ_R has been measured at DESY Zeuthen [48]. For a WOM tube with $R_o = 2.24\,\mathrm{cm}$ and $R_i = 1.89\,\mathrm{cm}$, the measured distribution appears to be reasonably uniform for $z_0 \geq (5.0 \pm 1.0)\,\mathrm{cm}$. A qualitative comparison to a simulation with similar parameters yields a consistent result (see figure C.10 in the appendix).

The distribution of r, on the other hand, has a distinct peak at $r = R_o$ and continuously decreases towards $r = R_i$. This peak exists even for high distances of light



(a) $z_0 = 30\,\mathrm{cm}$. (b) $z_0 = 55\,\mathrm{cm}$.

Fig. 6.3.: Two-dimensional histograms of the light distribution in polar coordinates.

emission. Experimental studies of the light distribution using a camera [49] yielded pictures that are vaguely similar.

Furthermore, the two-dimensional joint distributions feature distinct patterns that are not visible in the marginal distributions. The patterns appear in the shape of parabolas that blur for higher values of z_0 before vanishing completely. The position of the patterns along the r-axis and especially the peak in the r distribution is a direct result of the light emission depth of the wavelength-shifter. Figures 6.4, 6.5(a) and 6.5(b) show results of a similar simulation setup with y_0 shifted to the middle of the tube wall. As a result, the patterns and the peak also move to the middle of

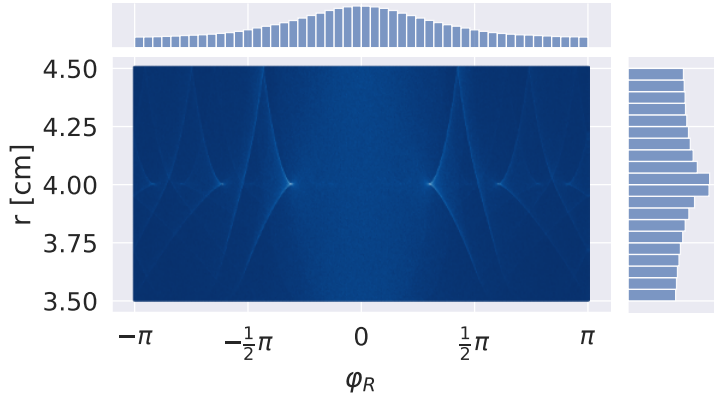
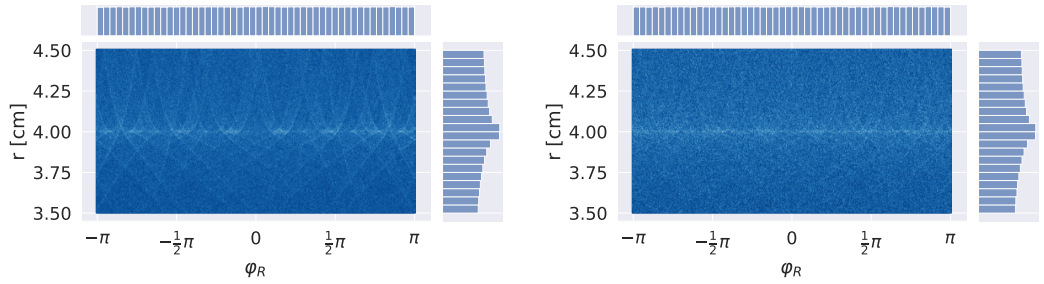


Fig. 6.4.: Two-dimensional histogram of the light distribution in polar coordinates with $z_0 = 5\,\mathrm{cm}$ and $y_0 = 4.0\,\mathrm{cm}$.

the wall.
Repeating the simulations with different parameter sets reveals that the displayed distributions are characteristic for the geometry. The patterns and the peak only disappear for very strong scattering (see figure 6.6).

Some insights about the patterns can be gained from inspections in the interactive renderer. The majority of rays that terminate at one of the peak points in the two-dimensional distributions are photons emitted under similar angles. The



(a) $z_0 = 30\,\text{cm}$.

(b) $z_0 = 55\,\text{cm}$.

Fig. 6.5.: Two-dimensional histogram of the light distribution in polar coordinates for $y_0 = 4.0\,\text{cm}$.

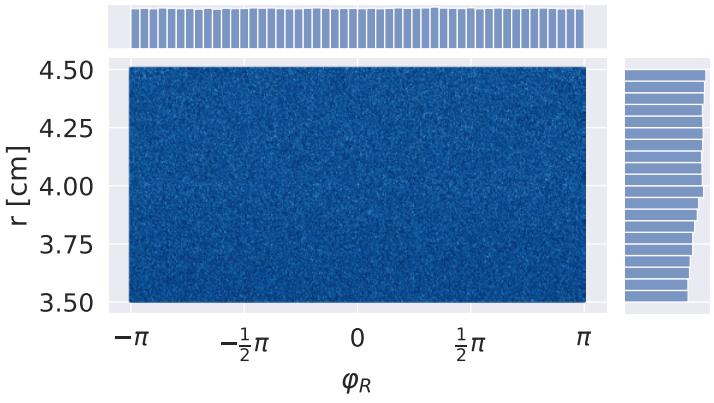


Fig. 6.6.: Two-dimensional histogram of the light distribution in polar coordinates for $z_0 = 5\,\text{cm}$ and $\lambda_s = 10\,\text{cm}$.

corresponding photon paths form a bundle of small spatial broadening. They are only reflected at the outer surface and after one or more reflections they have a common intersection point again exactly at $r = y_0$. Considering the case where this intersection appears after only one reflection, it is obvious that this has to be at $r = y_0$, due to symmetry (see figure 3.7 in chapter 3.5). For emission under similar polar angles, the number of required reflections depends on the variation of the azimuth angle.
The parabola shapes are likely caused by such bundles of rays that do not reach the detection plane exactly at their common intersection point. In that case the bundles still stand out due to their small spatial extension.
Since the isotropy of the light emission has been confirmed in chapter 4.3.1, we can conclude that the emerging patterns are features of the geometry. Besides, similar patterns have been observed in previous simulations of the WOM using FRED [3]. Additional plots for the distributions and screenshots from the renderer can be found in appendix C.3.

The knowledge of the peaks of the distributions of φ_R and r might be useful for improving the readout or analysis of experimental data. For instance, an analysis of data from the WOM test stand in Mainz [50] already uses the simulated distribution of φ_R. In that case, the distribution is employed to get an error estimate for the combination of the non-uniform light output of the WOM and the non-uniform sensitivity of the PMT detection area.

6.2 Tube detection efficiency

In this section, we study the simulated detection efficiency of the tube as a function of the distance between light emission point and detection plane. Unlike in the brief treatment of the simulated capture efficiency of the geometry in chapter 4.3.4, scattering and absorption are now included.
We define the efficiency for one side $\epsilon_{one} = \frac{N_{det}}{N}$ as the ratio between the number of photons reaching the specified detection plane N_{det} and the initial number of simulated photons N. We conduct simulations with varying scattering and absorption lengths λ_s and λ_a, respectively, and distance of light emission z_0 based on the setup in table 6.2.

Tab. 6.2.: Parameters for the efficiency simulations.

N	R_o[cm]	R_i[cm]	L[cm]	n_1	n_2	λ_a[cm]	λ_s[cm]	z_0[cm]	y_0[cm]
5×10^6	4.5	4.3	120	1.5	1.0	λ_a	λ_s	z_0	4.4955

Figure 6.7 shows simulation results for a constant attenuation length of $\lambda_{att} = 100\,\mathrm{cm}$ and several combinations of λ_a and λ_s compliant with equation 4.5. As expected,

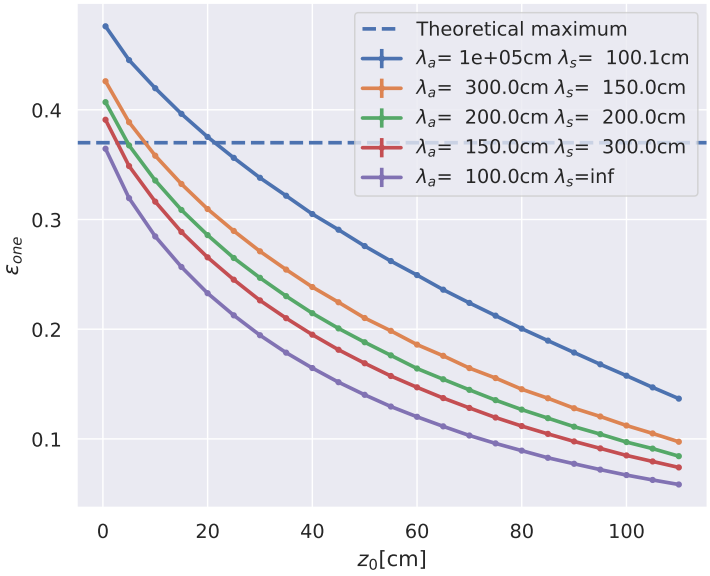


Fig. 6.7.: Simulated efficiencies for combinations of λ_a and λ_s that all yield $\lambda = 100\,\mathrm{cm}$. The dashed blue line is the theoretical maximum efficiency $\epsilon_{one} \approx 0.37$.

including light attenuation impacts the efficiency in a distance dependent way, reducing it for longer photon paths. The simplified model in equation 3.25 yields a theoretical capture efficiency of $\epsilon \approx 0.75$ for the presented simulation setup. Hence, due to the isotropic light emission, we expect a maximum efficiency of $\epsilon_{one} \approx 0.37$ at each detection plane. Using only absorption (purple curve in figure 6.7), this value is almost reached for z_0 approaching zero.

For all possible combinations of absorption and scattering, pure absorption results in the lowest efficiency. This is reasonable, since the implemented scattering distribution is isotropic. Therefore, scattered photons have the same probability of about 74.5% for being captured in the tube again.

Remarkably, with scattering applied, the highest simulated efficiency values exceed the theoretical maximum efficiency stated above. This is primarily caused by backscattering, i.e. photons that reverse n times in z-direction due to scattering (n is an odd integer). For long distances, this effect cancels out, since, on average, the same number of photons turns in both directions. For light emission close to one of the detection planes, however, the travel distance is short for photons starting in the direction of the closer one. Thus, the probability for backscattering prior to detection is significantly smaller for half of the photons.

Figure 6.8 demonstrates how backscattering affects the efficiency. The two curves only differ in the scattering distribution. For the orange line scattered light is only emitted in its initial direction along the z-axis, while for the blue one an isotropic emission is implemented. As a result, the orange curve only reaches the same

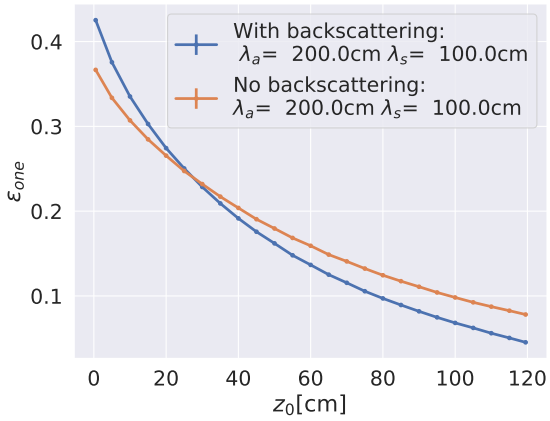


Fig. 6.8.: Simulated efficiencies using the same parameters with and without backscattering.

maximum efficiency as without scattering.
Another very small contribution in the efficiency increase comes from photons whose direction is changed in a scattering event such that they fulfill the TIR criterion afterwards.
Figure 6.9 shows the effect of varying the absorption and scattering length without keeping the attenuation length constant. The blue curve serves as baseline with

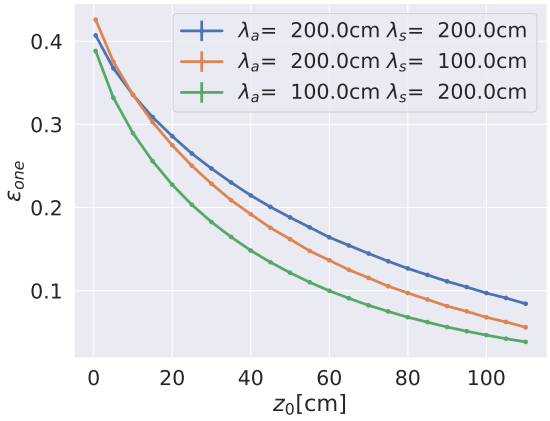


Fig. 6.9.: Simulated efficiencies for different attenuation lengths. The orange curve increases the amount of scattering, while the green one increases the amount of absorption.

equal values for λ_a and λ_s. The green curve is the result of applying stronger absorption, which causes an overall drop of the efficiency. The orange curve is the result of applying more scattering. This leads to an increased efficiency at distances shorter than $10\,\mathrm{cm}$. For longer distances, however, the efficiency drops below the blue curve. This is partly due to the increasing directional imbalance in scattering events. However, it is mainly caused by more photons scattering and subsequently not meeting the TIR criterion.

6.2.1 Comparison to flattened model

If the wall thickness is sufficiently small compared to the outer radius of the tube, the light propagation can be approximated in a flat geometry neglecting the tube's curvature. This can be imagined as "cutting" the cylinder wall along the z-axis and subsequently unrolling it (see figure 6.10). This so-called "flattened model" [5] defines three coordinates for the direction of light emission:

- φ_F is defined as the angle to the straight path in the unrolled plane, with $\varphi_F \in [-\pi, \pi]$.
- θ_F is defined as the angle to the plane, with $\theta_F \in [-\frac{\pi}{2}, \frac{\pi}{2}]$.
- d is defined as the distance between light emission point and tube end.

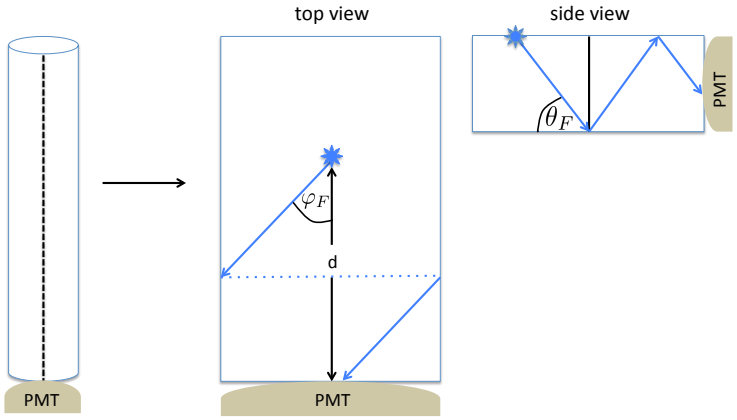


Fig. 6.10.: Schematic of the flattened model.

Applying periodic boundary conditions along the long sides of the plane enables photons reaching one side of the plane to enter it from the opposite side again. A total distance of $2\pi R_o$ in parallel to the PMT plane – crossing the boundary – results in one circumvolution in the tube. Thus, φ_F is associated with the number i of circumvolutions of the light path:

$$\tan(\varphi_F) = \frac{2\pi R_o i}{d} \quad .$$

For $\varphi_F \to \pm\frac{\pi}{2}$ this number goes to infinity. The actual light path length in the plane can be calculated via:

$$s_{plane} = \frac{d}{\cos(\varphi_F)} \quad . \tag{6.1}$$

Since the path lengths between reflections is constant in the side view, trigonometry yields for the total path length:

$$s = \frac{s_{plane}}{\cos(\theta_F)} = \frac{d}{\cos(\varphi_F)\cos(\theta_F)} \quad . \tag{6.2}$$

Note that this result neither depends on the thickness of the wall, nor on the radius of the tube. Using equation 4.3 and equation 6.2 yields the probability for attenuation on a light path in the flattened model:

$$\Pr(att; d, \varphi_F, \theta_F) = \exp\left(-\frac{d}{\cos(\varphi_F)\cos(\theta_F)\lambda_{att}}\right) \quad .$$

In the flattened model, the efficiency can be determined analytically by calculating the solid angle coverd by light paths captured in TIR, i.e. $\theta_F \in [-\theta'_c, \theta'_c]$ with $\theta'_c = \frac{\pi}{2} - \theta_c$. Additionally, attenuation can be taken into account by including the attenuation probability as a weight in the calculation of the solid angle. This yields the one-sided efficiency as function of d:

$$\epsilon(d) = \frac{1}{4\pi} \cdot \int_{-\pi/2}^{\pi/2} \int_{-\theta_C}^{\theta_C} \exp\left(-\frac{d}{\cos(\varphi_F)\cos(\theta_F)\,\lambda_{att}}\right) \cos(\theta_F)\, d\theta_F\, d\varphi_F \quad . \tag{6.3}$$

Unfortunately, an analytical solution does not exist for this integral. However, using an elaborate Taylor approximation scheme [50], the integral can be solved with $1\,\mathrm{ms}$ per evaluation.

In figure 6.11, we compare the distance dependent efficiency of the flattened model to several simulations, using the parameters in table 6.3. The scattering is turned

Tab. 6.3.: Parameters of the efficiency simulations for the comparison to the flattened model.

N	R_o[cm]	R_i[cm]	L[cm]	n_1	n_2	λ_a[cm]	λ_s[cm]	z_0[cm]	y_0[cm]
5×10^6	R_o	R_i	120	1.5	1.0	100	∞	z_0	4.4955

off, since the attenuation in equation 6.3 cannot model changes of light paths.
The top panel shows the absolute values of the simulated efficiency for different tube sizes and the efficiency according to equation 6.3 (blue line). The different parameter sets of WOMs were chosen such that a regularly sized one (orange), an exorbitantly oversized one (green), a small one (red), one with a very thick wall (purple) and one with a thin wall (brown) were simulated. First, the plot illustrates that the size of the WOM does not matter for the simulated efficiency from ray tracing. This has already been demonstrated for the attenuation-free case in chapter 4.3.4. Consequently, the result including attenuation implies that changing the size of the WOM does not change the overall length of the light paths. The actual paths taken, however, still differ.

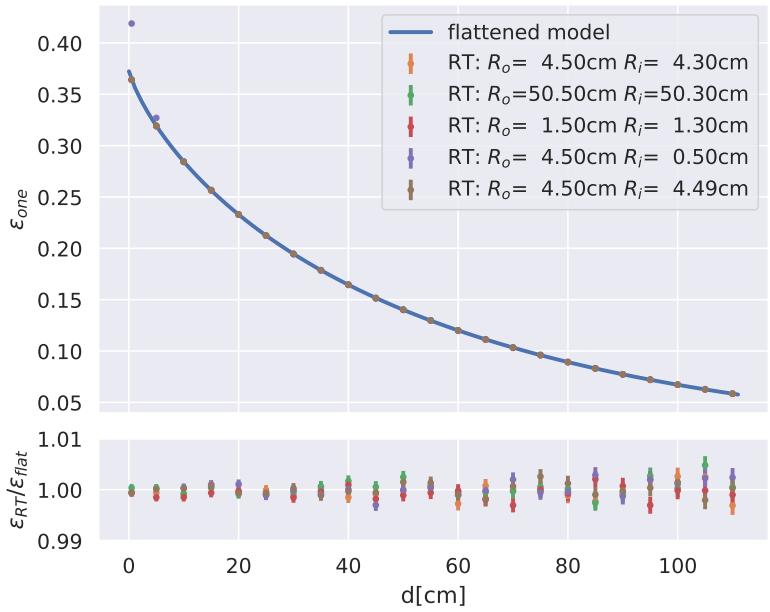


Fig. 6.11.: Comparison of the efficiency calculated in the flattened model to ray tracing simulations. The upper panel shows plots for the absolute efficiencies from ray tracing for several WOM sizes and the efficiency from the flattened model. The lower panel shows the corresponding ratios between ray tracing and flattened model results.

In order to visualize deviations of both models, the bottom panel shows the ratio between the simulated efficiencies and the result from equation 6.3. In almost all cases the deviation is less than 1%. For longer distances, the deviations and the statistical errors are larger due to the lower event counts and the larger accumulated error from floating-point arithmetic[1].

The high level of agreement is very impressive, considering that the flattened model does not know the size of the tube. Moreover, the photon paths in the flattened model and the ray tracing differ, especially because the ray tracing allows reflections at only one surface. Apparently, the path lengths are still approximately the same in both models, allowing for similar attenuation. Especially the case of a thick wall stands out since the approximation of a flat plane should be hardly valid. Still, only for $d \leq 10\,\mathrm{cm}$ the efficiency differs more than 1% (the respective data points fall out of the plotting range of the lower panel). This is caused by a significant amount of light reaching the detection plane without reflections in the ray tracing, including rays that would otherwise fail the TIR criterion. The efficiency calculated in the flattened model does, of course, not include this effect.

In conclusion, the flattened model yields a very good approximation of the efficiency in the absorption-only case.

[1] Errors from floating-point arithmetic are not displayed here.

6.2.2 Fit to experimental data

In this section, the possibilities and issues of comparing simulated efficiencies to measurements of a tube via a fit are explored. A fit with the ray tracing simulation would allow for the determination of both λ_a and λ_s, while fits with the flattened model can only determine λ_a.

For a number of n measurement points (d_i, ϵ_i) with associated uncertainties σ_{ϵ_i} and a function $S(d; \mathbf{\Gamma})$ with free parameters $\mathbf{\Gamma} = (A, \lambda_a, \lambda_s)$ and associated uncertainties $\sigma_{S(d_i;\mathbf{\Gamma})}$ the following measure of similarity can be defined [51]:

$$\chi^2(\mathbf{\Gamma}) = \sum_{i=0}^{n-1} \frac{(\epsilon_i - S(d_i; \mathbf{\Gamma}))^2}{\sigma_{\epsilon_i}^2 + \sigma_{S(d_i;\mathbf{\Gamma})}^2} \quad . \tag{6.4}$$

The non-linear function $S(d; A, \lambda_a, \lambda_s) = A \cdot \epsilon_{RT}(d; \lambda_a, \lambda_s)$ is the result of an efficiency simulation ϵ_{RT} with the parameters λ_a and λ_s for absorption and scattering, respectively, scaled with a factor A. In principle, more parameters of the simulation could be incorporated in $\mathbf{\Gamma}$, but for the start the dimensionality $D = \dim(\mathbf{\Gamma})$ should be kept low in order to make converging easier for the fit. All other parameters of the tube are known from direct measurements anyway. However, extending $\mathbf{\Gamma}$ to also cover the ellipse parameters might be worthwhile in order to find the deviation of a tube from a circular cross section.
Solving the optimization problem

$$\widetilde{\mathbf{\Gamma}} = \arg\min_{\mathbf{\Gamma}} \chi^2(\mathbf{\Gamma})$$

yields the set of parameters for a simulation that has the best fit to the data points. The goodness of the fit can be determined by calculating

$$\chi^2_{red} = \frac{\chi^2(\widetilde{\mathbf{\Gamma}})}{n - D} \quad . \tag{6.5}$$

As a simplified rule of thumb, equation 6.5 should yield approximately 1 for a good fit. Then the squared residual of fit function and measurement is on average equal to their associated squared uncertainty. Values much greater than 1 can indicate a poor fit. Values smaller than 1 on the other hand can indicate overfitting, since the fit tries to describe the data more precisely than the actual measurement allows given the uncertainty.

Most algorithms for non-linear minimization require that the objective function is differentiable because its gradients are employed in order to find the minimum. However, this is not the case for the objective function in equation 6.4, since it involves evaluations of the ray tracing simulation which obviously does not provide

a gradient.
An algorithm that is suitable for objective functions that are not differentiable is the Nelder-Mead downhill simplex algorithm [52]. The algorithm uses $D+1$ test points $\mathbf{\Gamma}^i$ that define a D-dimensional simplex in the space of parameters. By evaluating the objective function at the test points and comparing the results it determines a direction in parameter space to find the minimum and whether to expand or shrink the simplex in that direction. In the best case the simplex shrinks to a point which is the optimum.
The algorithm is present in the module scipy.optimize of the Python library SciPy [26] as the function `minimize(method='Nelder-Mead')`. The fit is performed using this function with the objective function from equation 6.4. Implementing the objective function requires to make the simulation program callable from a Python script, which can be done using Python's subprocess module. In that way, the evaluation of $\epsilon_{RT}(d;\lambda_a,\lambda_s)$ within Python takes place by generating an input text file from the given simulation parameters, subsequently calling the simulation program as a subprocess, reading the generated HDF5 output file and finally evaluating the efficiency from that.
The fit converges if $|\Gamma_k^0 - \Gamma_k^j| \leq xtol$ and if $|\chi^2(\mathbf{\Gamma^0}) - \chi^2(\mathbf{\Gamma^j})| \leq ftol$ for all $j \in [1, D]$ and $k \in [0, D-1]$. This means that the parameters of all simplex vertices are at most $xtol$ away from the best vertex $\mathbf{\Gamma^0}$ and the values of the objective function differ at most by $ftol$ from the best one. Here, $xtol$ and $ftol$ are tuneable hyperparameters of the minimization routine.

Testing the fitting routine for data of a tube measured at the Mainz test stand [50] yields the result in figure 6.12. For comparison, the plot also includes a fit with the

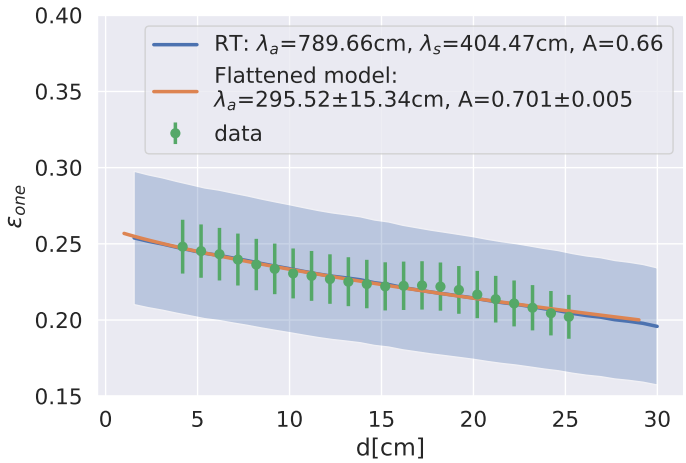


Fig. 6.12.: Fit of the ray tracing simulation (blue) and the flattened model (orange) to an efficiency measurement (green) of the tube "Q3" with the "South PMT". The ray tracing fit converged with $\chi^2_{red} = 0.0124$. The flattened model fit converged with $\chi^2_{red} = 0.0222$. The shaded blue area is the the simulation error magnified by a factor of 100.

flattened model using the numeric solution of equation 6.3 for ϵ in the objective function and the standard routine `curve_fit` in the scipy.optimize module. Fitting results for another tube can be found in figure C.13 in the appendix.
The uncertainties of the measurement σ_{ϵ_i} include both the statistic and the systematic error. The uncertainties of the simulation $\sigma_{S(d_i;\mathbf{\Gamma})} = A \cdot \frac{\sqrt{N_{det}(d_i)}}{N}$ are the result of the statistic error of the count of detected photons. Those are tiny compared to the measurement uncertainties. In order to visualize the difference in magnitude of both, the simulation error is magnified by a factor of 100 in the plot (shaded blue area).
For the simulation evaluations the parameters (see table 6.4) were set to match the measured tube (code name "Q3"). For the chosen value of N values of $xtol = 10^{-4}$

Tab. 6.4.: Simulation parameters for the fit.

N	R_o[cm]	R_i[cm]	L[cm]	n_1	n_2	λ_a[cm]	λ_s[cm]	z_0[cm]	y_0[cm]
2×10^6	2.2	1.67	30	1.49	1.0	λ_a	λ_s	d_i	2.1978

and $ftol = 0.1$ were used to achieve convergence. Higher values of N for now turned out infeasible, since the time for writing this amount of data to HDF5 files starts dominating when this is done at a high frequency. It appears that in consecutive iterations at high frequency the simulation is able to produce data at a rate that exceeds the writing capabilities of the hard drive.
The fit converged after about 20 minutes using 185 evaluations of the ray tracing simulation at 22 distance points for 2×10^6 photons each. The final value of χ^2_{red} is too small with $\chi^2_{red} = 0.0124$. This is a result of the high, correlated uncertainties of the measurement. The same applies for the χ^2_{red} of the flattened model fit with $\chi^2_{red} = 0.0222$. Apart from that, the curves of both fits are very similar and also close to the actual measurement points. Using equation 4.5 yields an attenuation length of $\lambda_{att} = 267.47\,\mathrm{cm}$ for the ray tracing which can be compared to the absorption length of the flattened model. Considering the fact that the attenuation in both models is still different[2], those results are quite close to each other.

However, this fitting routine has a number of issues that require improvement. First, it is apparent that the parameter estimates from the fit are lacking statistic uncertainties. Unfortunately, the used SciPy routine for the Nelder-Mead algorithm `minimize(method='Nelder-Mead')` does not provide uncertainties on its own. The original paper [52] contains an approach for obtaining uncertainties from the final simplex, but due to time constraints it was not implemented anymore for this fitting routine.
Moreover, the Nelder-Mead algorithm does not always converge to the global optimum. The solution is quite sensitive to the initial simplex choice and it can easily get

[2] Bear in mind that the comparison in the previous section did not include scattering.

stuck in local optima. Also, the two parameters λ_a and λ_s of course are correlated via the more general attenuation length λ_{att}, hence the minimization is quite flexible in the choice of these two. In order to get more reliable results, a more sophisticated analysis has to be applied. For instance, evaluating equation 6.4 on a grid for a subset of the parameters, while keeping the rest constant should help narrowing down the search of the global optimum. Creating a two dimensional map of χ^2 depending on λ_a and λ_s in that way should visualize the correlation of the two.
Finally, speed is still a concern that can and should be improved. As stated above, the interface between SciPy and the simulation program via HDF5 files is a severe bottleneck at the rate of data generation during the optimization. It should be considered to implement the simulation, such that it is callable directly with a Python wrapper function that returns the results in RAM rather than on the hard drive. Removing that bottleneck makes the minimization process at least by a factor of 10 faster. Subsequently, utilizing even higher photon counts in the simulation enables smaller values for $ftol$ and thus the optimum can be determined with better precision.
In summary, this section demonstrated that generally speaking the simulation is now fast enough to be employed in a fitting routine. The actual results of this section should only be treated as proof of concept that has to be extended in the future.

6.3 Light exit angles

The simulation program returns two angles for the final ray of each photon (see figure 6.13). φ is the angle between the final ray projected to the x-y-plane and

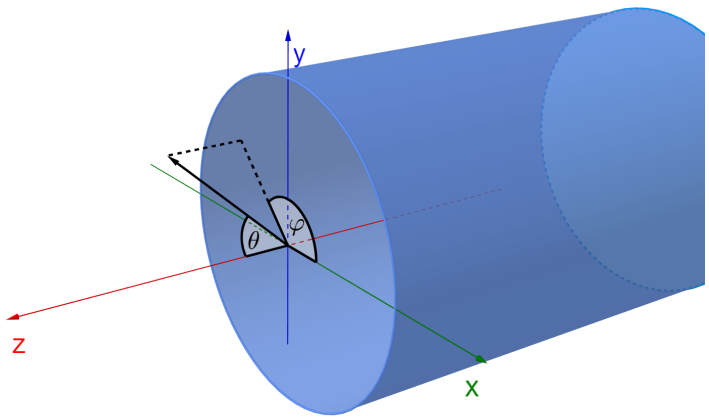


Fig. 6.13.: Definition of the angles of the final ray.

the x-axis with $\varphi \in [0, 2\pi]$. θ is the angle between the final ray and the z-axis with $\theta \in [0, \frac{\pi}{2}]$. We analyze the angles for simulations with the parameters in table 6.5. Figure 6.14 shows distributions of the angle φ, the first for light emission close to the detection plane and the second for light emission close to the opposite end of

Tab. 6.5.: Parameters for the simulation of the angle distributions.

N	R_o[cm]	R_i[cm]	L[cm]	n_1	n_2	λ_a[cm]	λ_s[cm]	z_0[cm]	y_0[cm]
2×10^7	4.5	4.3	60	1.5	1.0	100	100	z_0	4.4955

the tube.

The distribution for the short distance features a peak at $\varphi = \frac{3}{2}\pi$. That means, most

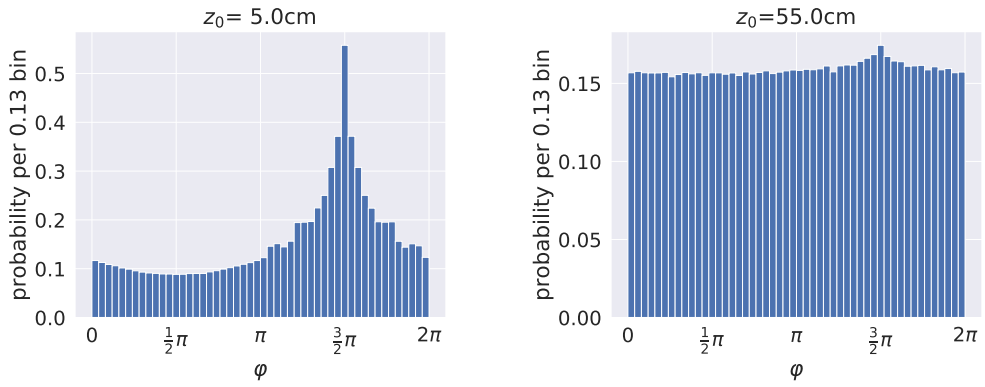


Fig. 6.14.: Distributions of the angle φ for simulations with different light emission distances.

photons exit favoring the negative y-direction. This has to be a consequence of the light entry point position, which sits at $(0, y_0, z_0)$. Therefore, at this distance, photons still favor a direction that heads away from the light entry point. At greater distances the distribution eventually becomes uniform due to the light spreading out. However, compared to the distribution of φ_R in section 6.1 it takes much greater distances to even out. Even for $z_0 = 55\,\mathrm{cm}$ the peak position is still slightly noticeable.

More meaningful, however, are the distributions of θ. In figure 6.15 the distribution is depicted for a simulation of the attenuation-free case. The distribution is asymmetric with a discontinuity defined by the critical angle at $\theta_c' = \frac{\pi}{2} - \theta_c$, which is marked by the green line. We have learned in chapter 4.1.2 that for an isotropic emission the angle θ follows the PDF $f_\theta(\theta) = \frac{1}{2}\sin(\theta)$. The left part of the distribution still follows that initial distribution.

The right part, however, is the result of TIR. In two dimensions, the angle θ_c' results in an angle of incidence of θ_c at the surface. Therefore, angles greater than that do not fulfill the TIR criterion at the surface in two dimensions, which causes the decrease of the probability density in the right part. Nevertheless, the distribution is not plainly cut at that angle, since the TIR criterion can still be accomplished in three dimensions. It appears that the full distribution is a convolution of the initial distribution and some kind of decay for angles above θ_c'.

Figure 6.16 visualizes the distribution of θ for simulations including attenuation for two different light emission distances. Since increasing the angle θ increases the

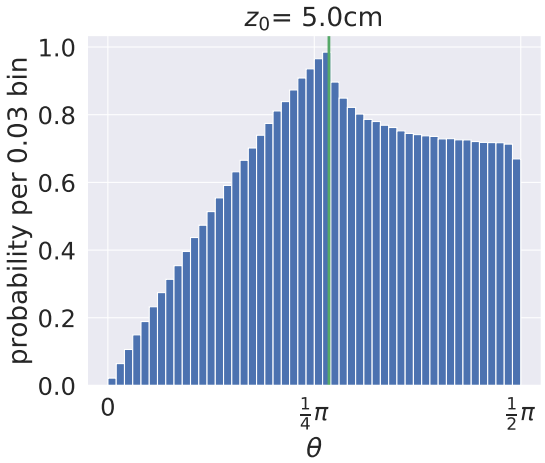


Fig. 6.15.: Distribution of the angle θ for light emission at $z_0 = 5\,\mathrm{cm}$ and $\lambda_a = 10^6\,\mathrm{cm}$, $\lambda_s = \infty$. The green line marks $\theta = \frac{\pi}{2} - \theta_c$, which is defined by the critical angle θ_c.

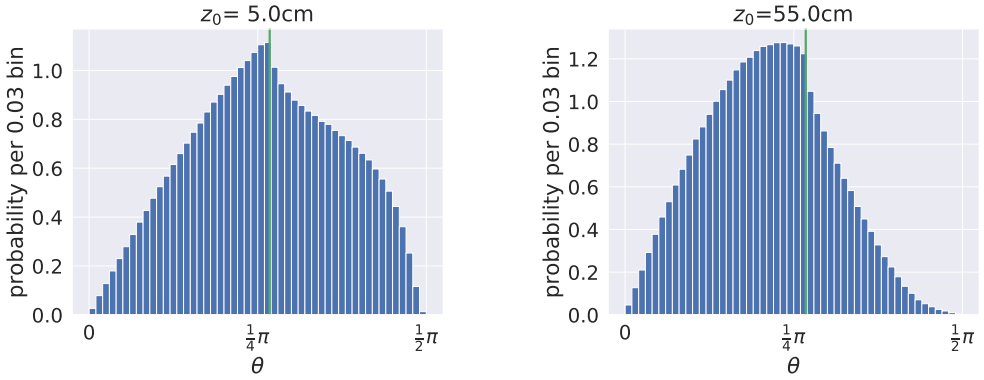


Fig. 6.16.: Distributions of the angle θ for simulations with attenuation using $\lambda_a = 100\,\mathrm{cm}$ and $\lambda_s = 100\,\mathrm{cm}$.

distance a photon has to cover before reaching the detection plane, the attenuation leads to a decline of the distribution there. For the greater light emission distance, the effect on great angles increases but it also starts expanding to smaller angles due to the overall longer distances.
In summary, the distributions demonstrate that the WOM does not give a directed output that is aligned with the normal of the exit surface. Since θ defines the angle of incidence at a detector or a lightguide attached to the tube the knowledge of the distributions are useful to estimate the efficiency of such attachments (see section 6.4.1).

So far we have only examined angles in the global coordinate frame. However, those do not expose the orientation of a final ray in relation to the tube surfaces since this also depends on the final position on the detection plane. Hence, we have to switch to a local coordinate frame and calculate angles from there.
We define three basis vectors $\hat{\boldsymbol{n}}$, $\hat{\boldsymbol{n}}_\perp$ and $\hat{\boldsymbol{z}}$ for a suitable coordinate system in the following way: $\hat{\boldsymbol{n}}$ is a normal vector to the cylinder surface and it depends on the final photon position $\boldsymbol{p} = (x, y, z)$; $\hat{\boldsymbol{z}}$ is the basis vector of the z-axis in global coordinates which is already orthogonal to $\hat{\boldsymbol{n}}$; last, $\hat{\boldsymbol{n}}_\perp$ is defined such that it is orthogonal to both $\hat{\boldsymbol{n}}$ and $\hat{\boldsymbol{z}}$ and such that $(\hat{\boldsymbol{n}}_\perp, \hat{\boldsymbol{z}}, \hat{\boldsymbol{n}})$ yields a right-handed orthonormal basis. From equation 3.8 we find for the cylinder:

$$\hat{\boldsymbol{n}} = \frac{1}{\sqrt{x^2 + y^2}} \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} \quad .$$

Subsequently, we obtain the right-handed orthonormal basis with:

$$\hat{\boldsymbol{n}}_\perp = \hat{\boldsymbol{z}} \times \hat{\boldsymbol{n}} = \frac{1}{\sqrt{x^2 + y^2}} \begin{pmatrix} -y \\ x \\ 0 \end{pmatrix} \quad .$$

The basis vectors define the transformation matrix $\mathbf{M} = (\hat{\boldsymbol{n}}_\perp, \hat{\boldsymbol{z}}, \hat{\boldsymbol{n}})^{-1}$ for switching the coordinate system via $\boldsymbol{p}' = \mathbf{M} \cdot \boldsymbol{p}$. After the transformation, we again examine angles in common convention of spherical coordinates [3]:

$$\varphi_F = \operatorname{atan2}(p'_x, p'_y) \quad ,$$

$$\theta_F = \arccos(p'_z) - \frac{\pi}{2} \quad .$$

The two angles defined in this way, are comparable to the two angles in the flattened model, if the wall of the tube is sufficiently small.

[3] Note that the old z-axis takes on the role of the y-axis in this coordinate system, thus we get a different angle θ.

Figure 6.17 shows the distribution of φ_F, again without attenuation for the start. The distribution appears to be very uniform in this case. This makes sense, since in

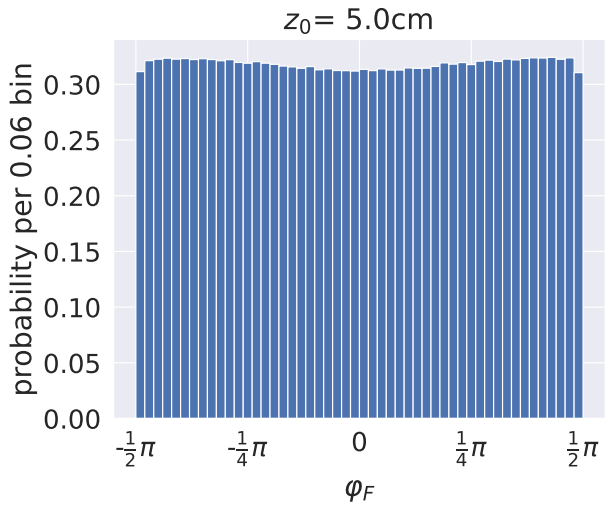


Fig. 6.17.: Distribution of the angle φ_F for light emission at $z_0 = 5\,\mathrm{cm}$ and $\lambda_a = 10^6\,\mathrm{cm}$, $\lambda_s = \infty$.

the flattened model all paths in the range of φ_F are equally likely. Figure 6.18 shows the distributions for the addition of attenuation, again one close to the detection plane and another one far away. The distributions are still symmetrical and centered

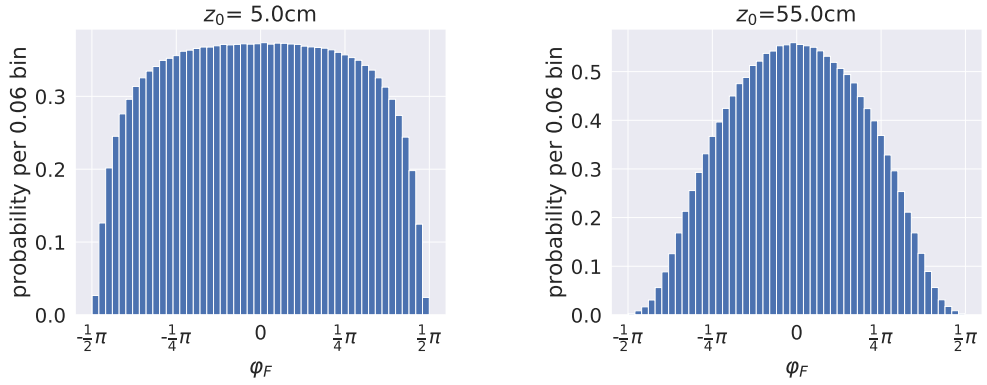


Fig. 6.18.: Distributions of the angle φ_F for simulations with attenuation using $\lambda_a = 100\,\mathrm{cm}$ and $\lambda_s = 100\,\mathrm{cm}$.

at $\varphi_F = 0$, which corresponds to a straight path in the plane of the flattened model. Towards the edges, however, both are decreasing. For the light emission at $z_0 = 5\,\mathrm{cm}$, this mainly concerns only angles close to the edges while the center still features a plateau. For the light emission at the opposite end of the tube, a wider range of angles is impacted by this and the distribution appears with a shape similar to a Gaussian.

This is expected, since increasing the angle in either direction, increases the photon path length in the flattened model plane (see figure 6.10 and equation 6.2). Hence,

photons at the edges with many circumvolutions in the tube have the highest attenuation probabilities. Increasing z_0 then magnifies the effect.

Finally, we examine the last angle θ_F. If the wall is sufficiently thin and the flattened model approximation holds, this angle relates to the angle of incidence α at the surface via $\alpha = \frac{\pi}{2} - \theta_F$. Figure 6.19 depicts distributions of θ_F. It is again symmetric

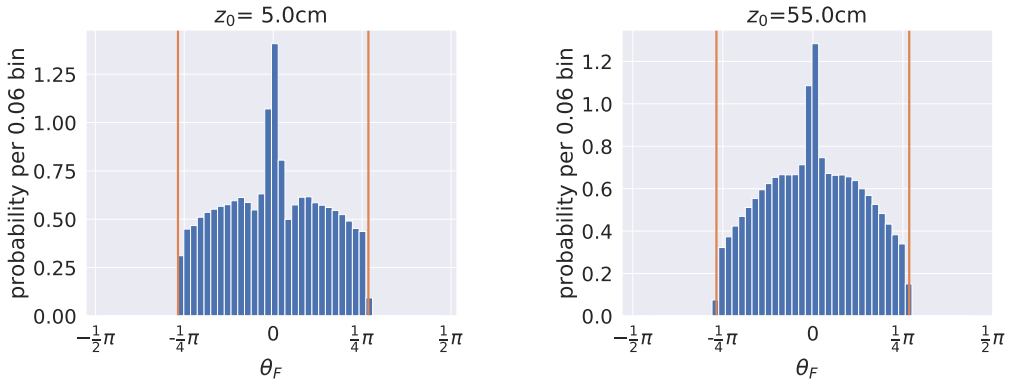


Fig. 6.19.: Distributions of the angle θ_F. The orange lines mark the angle $\theta_F = \frac{\pi}{2} - \theta_c$, which is defined by the critical angle θ_c.

and centered at $\theta_F = 0$. The distribution is pruned by the critical angle at both sides exactly at $\theta_F = \frac{\pi}{2} - \theta_c$. The peak at $\theta_F = 0$ corresponds to light that arrives at very flat angles at the surface ($\alpha \approx \frac{\pi}{2}$). It is likely caused by the high number of photons that circulate close to the outer wall and also cause the high density ring at the outer surface in figure 6.1. Increasing the light emission distance, again yields a decline of the distribution at the edges due to the longer path lengths and hence the higher attenuation probability.

6.4 ALG detection efficiency

Previous measurements with the manufactured ALG attached to the tube have shown a drop in the WOM's detection efficiency. The optical coupling could be the reason for this, but other sources for a decreased efficiency must be considered as well. We are thus primarily concerned with verifying the lossless concentration of light in the geometry first, in order to rule out flaws in the design concept.

We simulate the light propagation in assembly of tube and ALG with the spline approach discussed in chapter 3.4. We perform the simulation of the ALG itself in a separate simulation run, which enables comparisons of tube and ALG light output in a simple way. Instead of a light source undergoing a wavelength-shift, we thus use the output data from a simulation run of the tube alone as input for the main

simulation loop for the ALG. Photons, which did not reach the detection plane in the tube simulation are filtered out of the input. We assume that ALG and tube are made of the same material and are attached with perfect optical coupling, i.e. photons are transitioning lossless and without refraction between the two. In this simulation, both the entrance and exit plane of the ALG serve as detection planes causing a photon to terminate. This of course means, that by splitting the simulation in two runs we lose the ability of simulating photons, which cross the boundary between tube and ALG several times due to multiple scattering. However, this is only a minor effect.

We want to simulate the same geometry that was calculated by Falke [31]. For the hyperbola in equation 3.18, we use the parameters stated in the thesis, i.e. $r_0 = p = 4.5\,\mathrm{cm}$ and $\epsilon = 1.4502$ and $r_{inner} = 4.3\,\mathrm{cm}$. We obtain spline coefficients with 10 sample points of $r^2(z)$ from SciPy and use this as input for the geometry of the ALG simulation.

For the tube simulation we use a perfectly circular tube that fits to the ALG, using the set of parameters in table 6.6. We analyze the ALG efficiency for different settings

Tab. 6.6.: Parameters for the simulation of the ALG.

N	$R_o[\mathrm{cm}]$	$R_i[\mathrm{cm}]$	$L[\mathrm{cm}]$	n_1	n_2	$\lambda_a[\mathrm{cm}]$	$\lambda_s[\mathrm{cm}]$	$z_0[\mathrm{cm}]$	$y_0[\mathrm{cm}]$
1×10^7	4.5	4.3	60	1.49	1.0	100	λ_s	z_0	4.4955

of λ_s and z_0 in the tube simulation. For the simulations of the ALG itself, scattering is disabled by setting $\lambda_{sc} = inf$ and only little absorption is used with $\lambda_{abs} = 1\,\mathrm{km}$, since the first goal is the investigation of the efficiency of the geometry alone.

Figure 6.20 shows efficiency plots from repeating the ALG simulation for several lengths of the ALG. Here, length means cutting away the end of the body of revolution in order to get different levels of light concentration [4].

Unfortunately, for all simulated tube parameters the efficiency decreases significantly with increased length. Changes of the two parameters mainly result in changes of the slope and curvature of the plots. Shorter distances of light emission on the tube (blue and green curve) or longer scattering distances (green and red curve) result in lower efficiencies at all lengths. Attaching the full-length manufactured ALG results in a simulated drop of the signal by up to 80%. In contrast to that, the calculation of Falke expects a constant efficiency of 100%.

We find the reason for this by examining photon paths in the interactive renderer. Figure 6.21 shows that, unlike predicted by Falke, photons starting with an angle λ to a cross-sectional plane through the symmetry axis do not circulate around the symmetry axis of the ALG. Instead, they travel more or less directly towards the

[4] The body of revolution always has to be cut before the root of the inner curve, since otherwise the cross sectional area is not conserved.

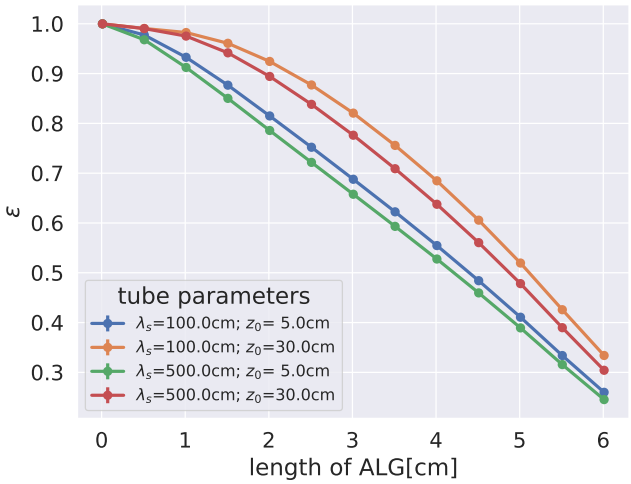


Fig. 6.20.: Efficiency of the ALG depending on the cutting length. The efficiency $\epsilon = \frac{N_{det}}{N}$ is defined as the ratio between the number of photons reaching the end N_{det} and number of incoming photons N. The statistical error $\Delta N_{det} = \sqrt{N_{det}}$ is not displayed here, since it is not visible.

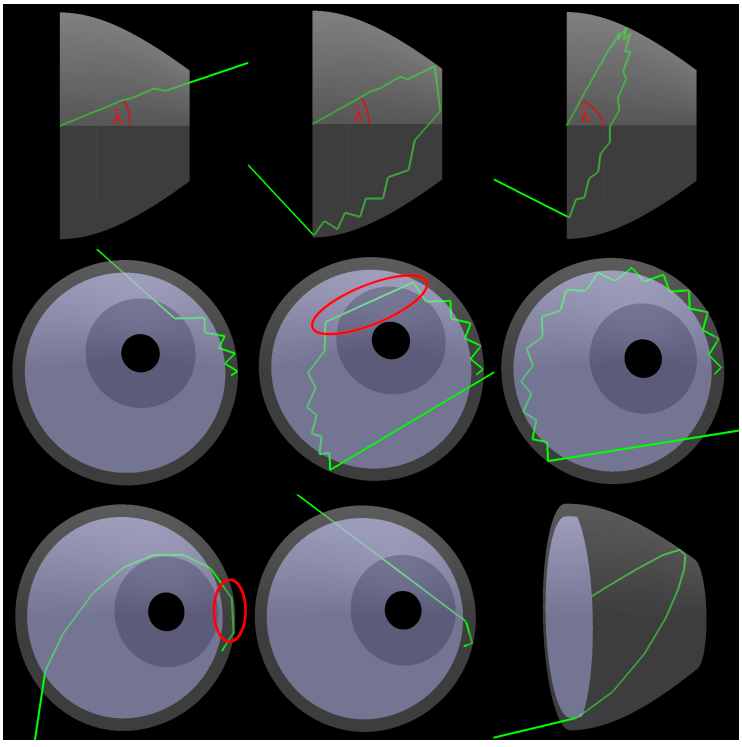


Fig. 6.21.: Nine different photon paths inside the ALG. Depending on the angle λ and the cutting length, photons can turn back before they reach the detection plane. Multiple subsequent reflections at the outer surface are possible, as highlighted by the red circles.

detection plane but turn around at some point. Whether or not they reach the detection plane depends on whether the turning point is located in front of the cutting length. Longer cutting lengths then lead to less photons making it to the detection plane, hence the efficiency decreases.
This could be an artifact of the normal vectors that are not smooth with the quadratic spline approach. However, switching to exact normals, derived as the gradient of equation 3.19, only results in a minor change of the photon paths and does not lead to an increased efficiency.
The angle λ here is the same angle as φ_F in section 6.3. There we have seen that increasing the distance of light emission or the attenuation length mostly causes a loss of light at higher angles φ_F. Hence, in both cases a higher fraction of light enters the ALG on paths that are – under the observations above – more favorable for detection. Therefore, the distributions of φ_F are able to explain the variation between the results for the different parameters in figure 6.20 and give further evidence that this angle is indeed the problem.

Reviewing the thesis of Falke reveals that a wrong assumption in appendix A2 is likely to be the reason for the failure of the design: Starting from the requirement that the angle λ must not increase between reflections to prevent photons from eventually turning their direction, he derives a condition for angle updates in his simulation. This calculation uses several Taylor approximations based on the assumption that photons are always reflected between inner and outer surface of the ALG. However, figure 6.21 also indicates that this is not necessarily true.

6.4.1 A possible alternative for the Falke ALG

An apparent alternative could be the Compound Parabolic Concentrator (CPC, also called "Winston Cone"), which is a standard solution from non-imaging optics [32]. This concentrator sends light entering through an opening with radius R to another opening with a smaller radius r with one reflection at a single surface of revolution. For a given R the whole geometry is defined by the acceptance angle θ_a with $\sin(\theta_a) = \frac{r}{R}$. Light entering at an angle θ to the symmetry axis with $\theta > \theta_a$ is sent back to the entrance with a second reflection.
For the light output of the WOM, this angle θ is the same angle as discussed in section 6.3. Unfortunately, the results show that the θ distribution of the WOM covers the whole range $[0, \frac{\pi}{2}]$. Although angles below the critical angle are favored, this always results in some light loss. Quantifying the loss requires the CDF of the θ distributions. The value $\mathrm{CDF}_\theta(\theta_a)$ corresponds to the relative amount of light from the WOM output that is accepted by a CPC with acceptance angle θ_a.
Figure 6.22 visualizes the result of this approach for the above tube simulation with $\lambda_s = 100\,\mathrm{cm}$ and $z_0 = 5\,\mathrm{cm}$ and compares it to the results of the ALG simulation.

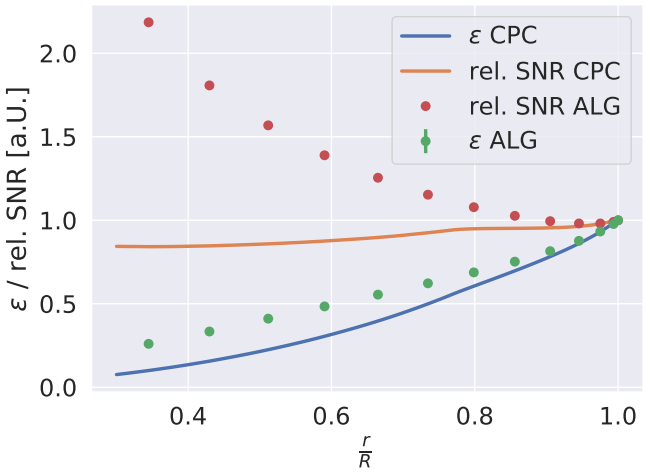


Fig. 6.22.: Comparison of efficiency and SNR of CPC and ALG. See text for details.

The x-axis features the level of concentration as $\frac{r}{R}$, such that a value of 1 means the light concentrator does not reduce the radius at all. The y-axis features both the efficiency ϵ and the theoretical SNR of the concentrator relative to their respective values for $\frac{r}{R} = 1$. For the SNR this assumes that the noise is approximately proportional to the surface area of the detecting PMT and thus SNR $\sim \frac{\epsilon}{r^2}$.
First, the plot demonstrates that the flawed ALG design still theoretically increases the SNR up to a factor of 2 in that setting. For the CPC, however, we find that it performs worse than the existing ALG design, both in SNR and efficiency. Thus, it does not qualify as an alternative.

In conclusion, the ALG design by Falke does not come up to expectations for an application in the WOM. Although it theoretically increases SNR and still performs better than a standard solution from the field of non-imaging optics, it is flawed if the goal is the reduction of the noise rate without losing in efficiency.

6.5 Detection time resolution

Last, we want to analyze the simulated time distribution of the WOM tube. The simulation tracks the absolute traversed distance d of each photon. From this we can calculate the elapsed time inside the tube for each photon using the speed of light in vacuum c:

$$t = \frac{d \cdot n_1}{c} \quad .$$

We analyze the simulation setup in table 6.7 varying the distance of the light emission point and for $\lambda_{att} = 300\,\mathrm{cm}$ and $\lambda_{att} = 100\,\mathrm{cm}$ both with equal scattering and absorption. Figure 6.23 compares the results of the time distributions for several

Tab. 6.7.: Parameters for the simulation of the time distributions.

N	R_o[cm]	R_i[cm]	L[cm]	n_1	n_2	λ_a[cm]	λ_s[cm]	z_0[cm]	y_0[cm]
2×10^7	4.5	4.3	50	1.5	1.0	λ_a	λ_s	z_0	4.4955

values of z_0 and for both attenuations. The distributions are plotted on a log scale

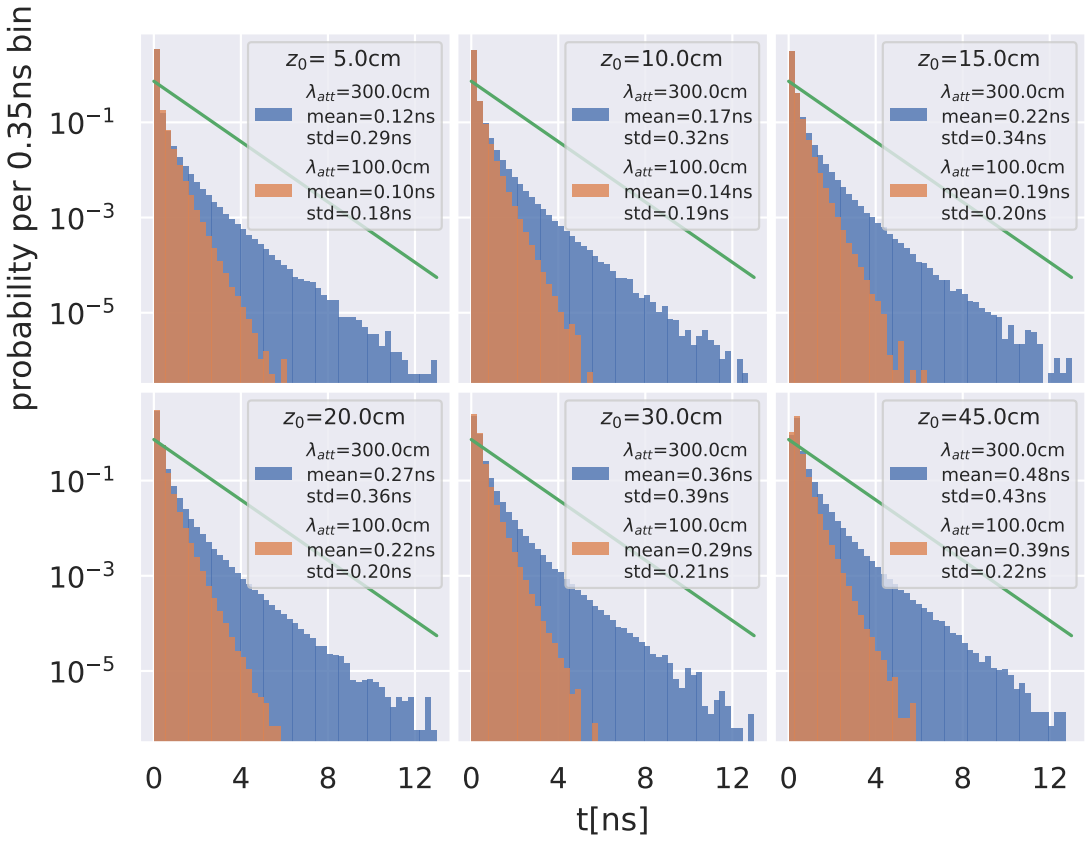


Fig. 6.23.: Simulated time distributions on a log scale for two levels of attenuation and at different distances of light emission. The green line represents the probability distribution for the decay of the paint, which is not yet included in the output distributions.

in order to visualize the extensive tails. Each distribution has a rapidly decreasing peak close to $t = 0$. The long tails have probabilities several orders of magnitude smaller than the peak area. They are caused by the relatively few photons that circle many times in the tube. Increasing z_0 moves the distribution slightly along the t-axis, which can be seen from the increasing values of the mean. But also the standard deviation increases and the height of the peak slightly drops. Thus, it also broadens the distribution and gives more weight to the tail.
The depicted distributions are only the distributions for the propagation inside the

tube wall and do not include the time delay of the WLS paint. The light emission time of the paint complies to a law of exponential decay with probability density

$$p_t(t) = \frac{1}{\tau} \exp\left(-\frac{t}{\tau}\right) \quad ,$$

which is the green line in each panel of figure 6.23. The time constant $\tau = (1.37 \pm 0.06)\,\mathrm{ns}$ has been measured at DESY Zeuthen [53]. Using this we can obtain the simulated time resolution of the tube with the paint combined. To do so, we actually need a single time distribution with light emission at random positions z_0 instead of a fixed position. This would be closer to the use case with light hitting the WOM at random positions. Unfortunately, the simulation has been developed with comparisons to the rather artificial efficiency test measurements at fixed positions in mind. Thus, at the time of writing this, the program is lacking this feature.
However, we can try to emulate the use case by simulating the tube in steps of $5\,\mathrm{cm}$ for z_0 over the whole length of the tube. The results of the simulations are then merged into a single time distribution and binned. Subsequently calculating the discrete convolution of the binned data and the probability density for the decay time of the paint yields the time distribution of the combination.
Figure 6.24 shows the result of this procedure for the attenuation length $\lambda_{att} = 300\,\mathrm{cm}$. Compared to figure 6.23 the distribution becomes broader, shifts towards

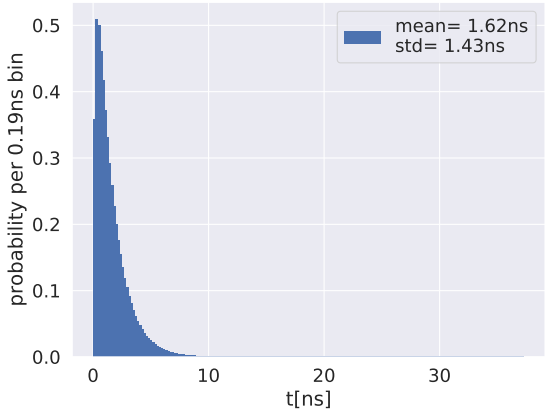


Fig. 6.24.: Time resolution of the WOM tube, including decay time of the paint for $\lambda_{att} = 300\,\mathrm{cm}$.

larger t and gets an even longer tail.
Concluding, for this simulation setup we find $1.62\,\mathrm{ns}$ for the mean time elapsed between photons hitting the tube and their arrival at the detector with a standard deviation of $1.43\,\mathrm{ns}$. This is, however, only for the tube alone. For a fully assembled WOM module, the time resolution of the detector has to be considered as well.

7 Conclusion and outlook

This work presents a dedicated simulation tool for the WOM, which allows for an overall better understanding of the sensor by providing unprecedented statistics at high simulation speed. It is a redevelopment of a preceding prototype [9] and has improved the simulation in numerous aspects.
The simulation is based on the ray tracing algorithm, which provides a model for the simulation of light propagation. It enhances the ray tracing algorithm with total internal reflection, absorption and scattering of light and is prepared for further extensions.

It has been demonstrated, that the algorithm reproduces all theoretical predictions (see chapter 4.3). It can thus be concluded that the light propagation is correctly simulated within the implemented physical framework.

In order to perform the ray tracing itself, a framework for intersections of rays and general quadrics has been developed (see chapter 3). This framework has been employed to implement the WOM geometry using two elliptic cylinders, as well as the geometry of the adiabatic light guide (ALG) via an approximation by quadratic splines.

The simulation has been optimized for high throughput of photons on a GPU. Optimizations were applied for the generation of random numbers and to reduce performance losses caused by branch divergence and load imbalance. Using a simplified replication of the parallel algorithm, it has been demonstrated that the applied optimizations yield a performance that is close to a scenario without branch divergence and load imbalance (see chapter 5).
After all optimizations have been applied, the simulation reaches a throughput of 7 million photons per second on an outdated mid-range GPU for typical simulation settings. This is a significant improvement compared to the 3000 photons per second achieved with Fred [5], but also compared to the prototype, which reaches 50 000 photons per second with similar settings.

Finally, the developed tool has been used to simulate and analyze properties of the WOM (see chapter 6). Here, the simulation tool has enabled the observation of scattering effects on the WOM efficiency for the first time. Additionally, the simulated

efficiency without scattering has been compared to the simplified "flattened model" and a consistency of more than 99% between both models was found. This result further supports the usage of the simpler and faster flattened model.
Furthermore, simulations of the ALG geometry have revealed that the current ALG design does not yield a lossless concentration. As a result, alternatives for the ALG are currently under investigation. One approach uses the simulation tool in an attempt of optimizing the design via the spline knots.

Due to the achieved speed, it turned out feasible to deploy the simulation in a χ^2-based fitting routine for experimental data. However, the fitting results and the routine still require additional work. On the one hand, uncertainties have to be implemented and a more thorough examination of the objective function in the parameter space is required in order to cope with the flexibility of the minimization algorithm. On the other hand, writing results to HDF5 files is not reasonable with the achieved simulation throughput, since this is the major bottleneck during the optimization. Therefore, the simulation tool should receive a Python interface with NumPy arrays in order to speed up the fit at least by a factor of 10.

Possible future enhancements of the algorithm are transmission and subsequent refraction of light, which is demonstrated in appendix B.1, the correct simulation of reflections described by the Fresnel equations (see chapter 2.1), the wavelength dependence of the refractive index and the simulation of rough surfaces.
Furthermore, the geometry in the ray tracing is entirely replaceable. Thus, the simulation software facilitates the simulation of arbitrary geometries without changes to the core algorithm provided that the required intersection routines are implemented. E.g., based on the quadric intersection framework an implementation of a geometry consisting of more layers of the module – most importantly separate layers for the paint and the tube wall – is immediately possible when transmission is implemented.

Overall, the dedicated simulation tool enables fast simulations of the light position distribution, the exit angle distributions, the detection efficiency and the time resolution as well as parameter estimation via fits to experimental data and much more. All this is possible for the WOM tube, the ALG and for the exploration of many other potential geometries in the future.

Appendices

A Equations

A.1 Scalar results of the coefficients from the quadric-ray-intersection

$$\begin{aligned}\alpha = {} & d_x^2\bar{a}_{00} + d_y^2\bar{a}_{11} + d_z^2\bar{a}_{22} \\ & + 2(d_x d_y\bar{a}_{01} + d_x d_z\bar{a}_{02} + d_y d_z\bar{a}_{12})\end{aligned} \tag{A.1}$$

$$\begin{aligned}\beta = {} & d_x(\bar{a}_{03} + \bar{a}_{00}v_{0x} + \bar{a}_{01}v_{0y} + \bar{a}_{02}v_{0z}) \\ & + d_y(\bar{a}_{13} + \bar{a}_{01}v_{0x} + \bar{a}_{11}v_{0y} + \bar{a}_{12}v_{0z}) \\ & + d_z(\bar{a}_{23} + \bar{a}_{02}v_{0x} + \bar{a}_{21}v_{0y} + \bar{a}_{22}v_{0z})\end{aligned} \tag{A.2}$$

$$\begin{aligned}\gamma = {} & \bar{a}_{00}v_{0x}^2 + \bar{a}_{11}v_{0y}^2 + \bar{a}_{22}v_{0z}^2 + \bar{a}_{33} \\ & + 2(\bar{a}_{01}v_{0x}v_{0y} + \bar{a}_{02}v_{0x}v_{0y} + \bar{a}_{12}v_{0y}v_{0z}) \\ & + 2(\bar{a}_{03}v_{0x} + \bar{a}_{13}v_{0y} + \bar{a}_{23}v_{0z}).\end{aligned} \tag{A.3}$$

A.2 Full quadric matrix of the elliptic cylinder with arbitrary position and rotation

After applying a rotation around the z-axis by α followed by a translation of the quadric, the transformed matrix of the elliptic cylinder calculates from equation 3.7 with the matrix

$$\bar{\mathbf{A}} = \begin{bmatrix} \frac{1}{a^2} & 0 & 0 & 0 \\ 0 & \frac{1}{b^2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

for the untransformed quadric and the matrix

$$\mathbf{M} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & x_0 \\ \sin(\alpha) & \cos(\alpha) & 0 & y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

for the transformations. Using the shorthand notation $s = \sin(\alpha)$ and $c = \cos(\alpha)$ this yields the result:

$$\begin{aligned}\bar{\mathbf{A}}' &= (\mathbf{M}^{-1})^T\, \bar{\mathbf{A}}\, \mathbf{M}^{-1} \\ &= \begin{bmatrix} \frac{c^2}{a^2} + \frac{s^2}{b^2} & c\,s(\frac{1}{a^2} - \frac{1}{b^2}) & 0 & \frac{-c(x_0c+y_0s)}{a^2} - \frac{s(x_0s-y_0c)}{b^2} \\ c\,s(\frac{1}{a^2} - \frac{1}{b^2}) & \frac{c^2}{b^2} + \frac{s^2}{a^2} & 0 & \frac{c(x_0s-y_0c)}{b^2} + \frac{s(-x_0c-y_0s)}{a^2} \\ 0 & 0 & 0 & 0 \\ \frac{-c(x_0c+y_0s)}{a^2} - \frac{s(x_0s-y_0c)}{b^2} & \frac{c(x_0s-y_0c}{b^2} - \frac{s(x_0c+y_0s)}{a^2} & 0 & \frac{(x_0s-y_0c)^2}{b^2} + \frac{(x_0c+y_0s)^2}{a^2} - 1 \end{bmatrix}.\end{aligned} \tag{A.4}$$

A.3 Full quadric matrix of the spline surface of revolution with arbitrary position

After applying a translation to the quadric, the transformed matrix of the spline surface of revolution calculates from equation 3.7 with the matrix

$$\bar{\mathbf{A}}_i = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & \alpha_i & \frac{\beta_i}{2} \\ 0 & 0 & \frac{\beta_i}{2} & \gamma_i \end{bmatrix}.$$

for the untransformed quadric and the matrix

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

for the transformation. This yields the result:

$$\bar{\mathbf{A}}_i{}' = \begin{bmatrix} -1 & 0 & 0 & x_0 \\ 0 & -1 & 0 & y_0 \\ 0 & 0 & \alpha_i & \frac{1}{2}(\beta_i - 2z_0\alpha_i) \\ x0 & y_0 & \frac{1}{2}(\beta_i - 2z_0\alpha_i) & -x_0^2 - y_0^2 + z_0^2\alpha_i - z_0\beta_i + \gamma_i \end{bmatrix}. \tag{A.5}$$

B Details for extensions

B.1 Transmission and refraction

Figure B.1 demonstrates the construction of the refracted ray's direction $\boldsymbol{dir'}$ from the incident ray's direction $\boldsymbol{dir}$, with $\|\boldsymbol{dir}\| = 1$ and the surface normal $\boldsymbol{n}$, with $\|\boldsymbol{n}\| = 1$. Using equation 2.5, we calculate the refraction angle $\beta = \arcsin\left(\frac{n_1}{n_2}\sin(\alpha)\right)$ and can construct $\boldsymbol{dir'}$:

$$\begin{aligned}\boldsymbol{dir'} &= d_\perp \cdot \boldsymbol{n}_\perp + d \cdot (-\boldsymbol{n}) \\ &= d_\perp \cdot (\boldsymbol{n} \times (\boldsymbol{dir} \times \boldsymbol{n})) + d \cdot (-\boldsymbol{n}) \\ &= \sin(\beta) \cdot (\boldsymbol{n} \times (\boldsymbol{dir} \times \boldsymbol{n})) - \cos(\beta) \cdot \boldsymbol{n}\end{aligned} \tag{B.1}$$

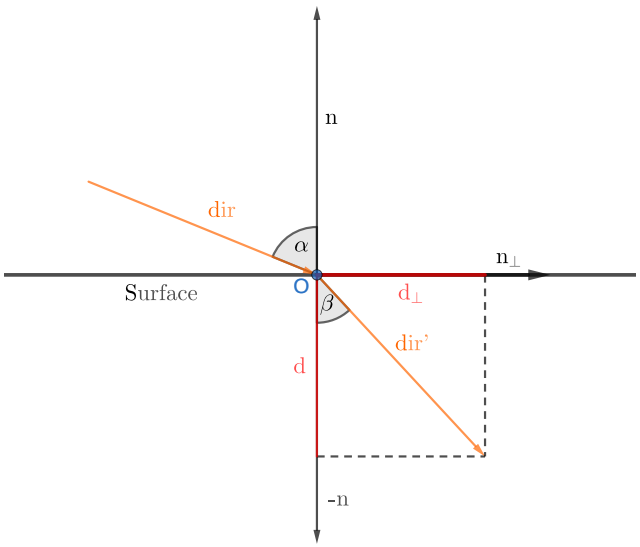


Fig. B.1.: Refraction of a light ray at the point O of a surface with normal n and construction of the refracted ray.

C Additional plots and pictures

C.1 Number of reflections depending on the initial direction

Additional plots for the number of reflections depending on the starting angles in different geometries. The parameters used here are listed in table C.1.

Tab. C.1.: Parameters in the simulations for the analysis of the number of reflections depending on the starting angles.

N	R_o[cm]	R_i[cm]	L[cm]	n_1	n_2	λ_a[cm]	λ_s[cm]	z_0[cm]	y_0[cm]
5×10^6	4.5	R_i	40	1.5	1.0	5×10^5	∞	20	y_0

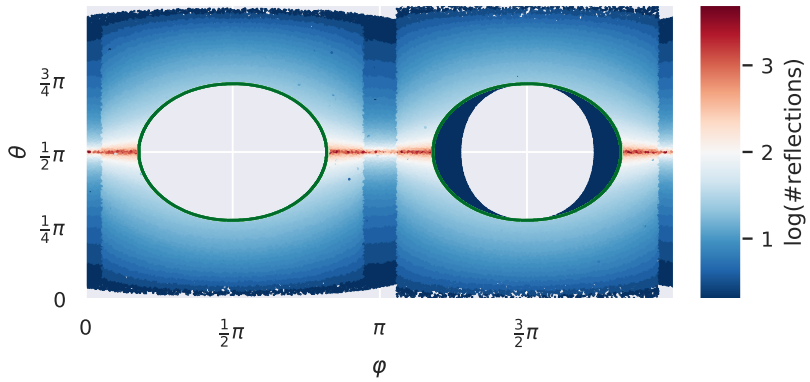


Fig. C.1.: $R_i = 3.5\,\text{cm}$; $y_0 = 3.555\,\text{cm}$

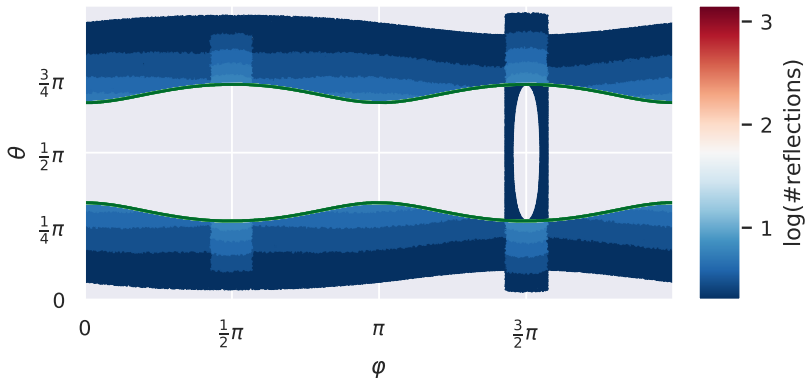


Fig. C.2.: $R_i = 0.5\,\mathrm{cm}$; $y_0 = 2.25\,\mathrm{cm}$

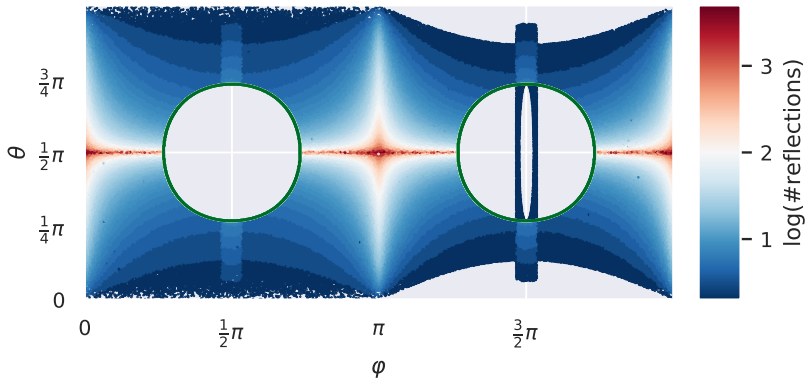


Fig. C.3.: $R_i = 0.5\,\mathrm{cm}$; $y_0 = 4.4955\,\mathrm{cm}$

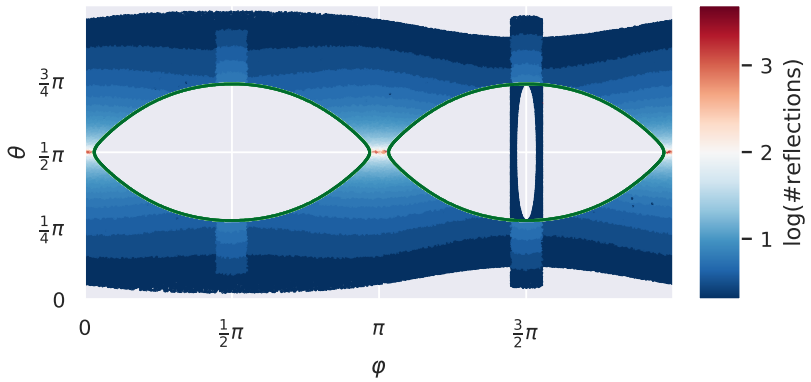


Fig. C.4.: $R_i = 0.5\,\mathrm{cm}$; $y_0 = 3.015\,\mathrm{cm}$

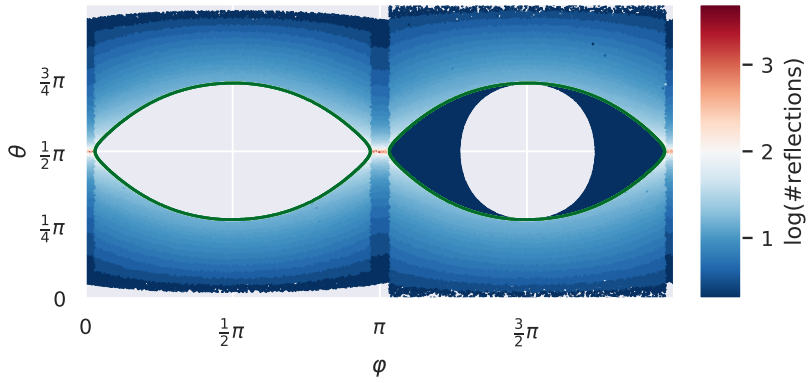


Fig. C.5.: $R_i = 3.0\,\mathrm{cm}$; $y_0 = 3.015\,\mathrm{cm}$

C.2 Light attenuation

Additional plot for the verification of the Beer-Lambert law. The parameters used here are listed in table C.2.

Tab. C.2.: Simulation parameters for the verification of the Beer-Lambert law.

N	R_o[cm]	R_i[cm]	L[cm]	n_1	n_2	λ_a[cm]	λ_s[cm]	z_0[cm]	y_0[cm]
10^7	4.5	4.3	10^5	1.58	1.0	λ_a	λ_s	5×10^4	4.499

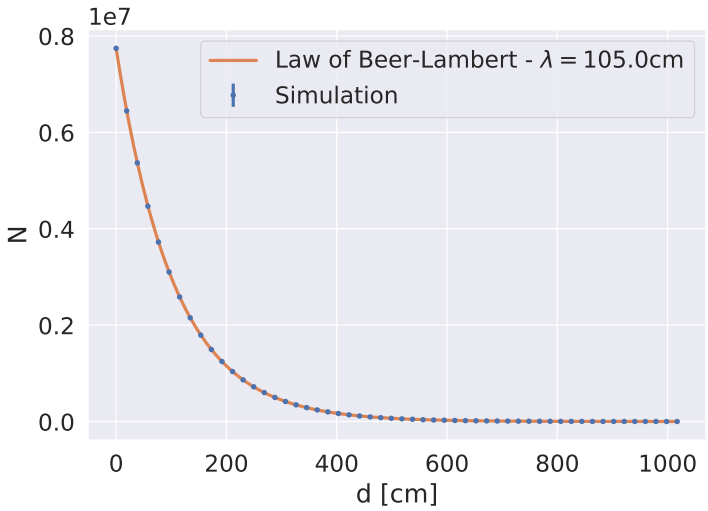


Fig. C.6.: Simulation of absorption and scattering and theoretical curve. The simulation used the parameters $N_0 = 7\,742\,210$, $\lambda_a = 150\,\mathrm{cm}$ and $\lambda_s = 350\,\mathrm{cm}$, which corresponds to $\lambda = 105\,\mathrm{cm}$ for the theoretical curve. The statistic uncertainties of the measurements with $\Delta N = \sqrt{N}$ are not displayed here, as they are too small.

C.3 Light distributions at detection plane

Additional plots for the light distributions at the detection plane. The parameters used here are listed in table C.3.

Tab. C.3.: Parameters for the simulation of light distributions.

N	$R_o[\mathrm{cm}]$	$R_i[\mathrm{cm}]$	$L[\mathrm{cm}]$	n_1	n_2	$\lambda_a[\mathrm{cm}]$	$\lambda_s[\mathrm{cm}]$	$z_0[\mathrm{cm}]$	$y_0[\mathrm{cm}]$
2×10^7	4.5	3.5	60	1.5	1.0	100	100	z_0	4.4955

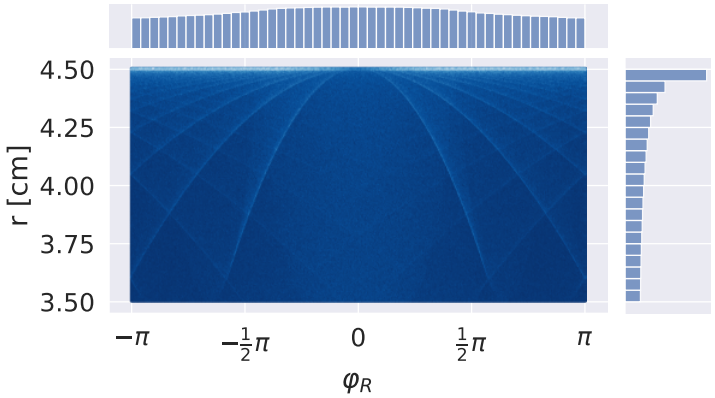


Fig. C.7.: Two-dimensional histogram of the light distribution in polar coordinates for $z_0 = 10\,\mathrm{cm}$.

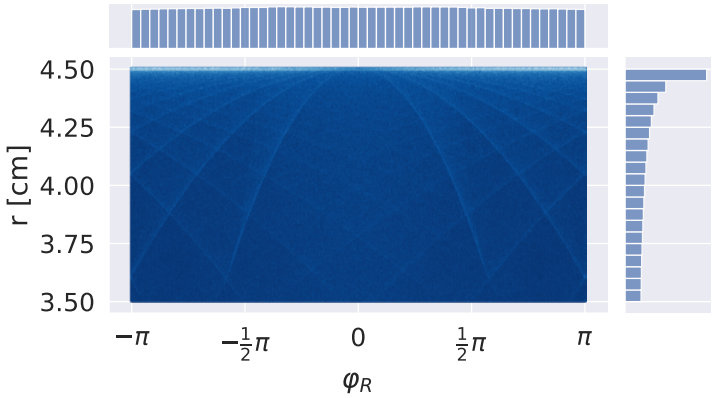


Fig. C.8.: Two-dimensional histogram of the light distribution in polar coordinates for $z_0 = 15\,\mathrm{cm}$.

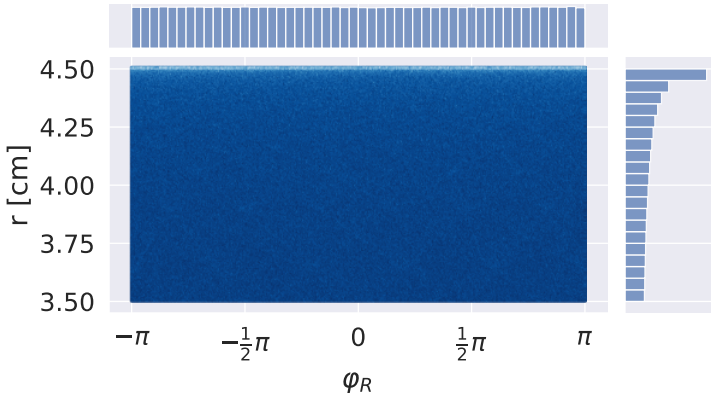


Fig. C.9.: Two-dimensional histogram of the light distribution in polar coordinates for $z_0 = 50\,\mathrm{cm}$.

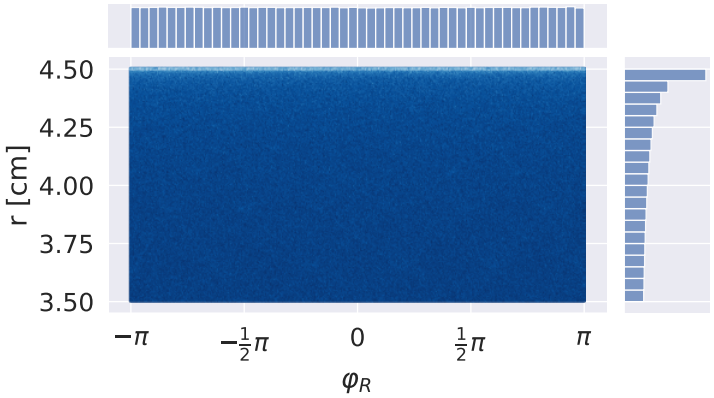


Fig. C.10.: Two-dimensional histogram of the light distribution in polar coordinates for $z_0 = 6\,\mathrm{cm}$. The tube dimensions are similar to a tube measured at DESY Zeuthen with $R_o = 2.24\,\mathrm{cm}$ and $R_i = 1.89\,\mathrm{cm}$. Similar to the measurement the distribution of φ_R is already reasonably uniform at this distance.

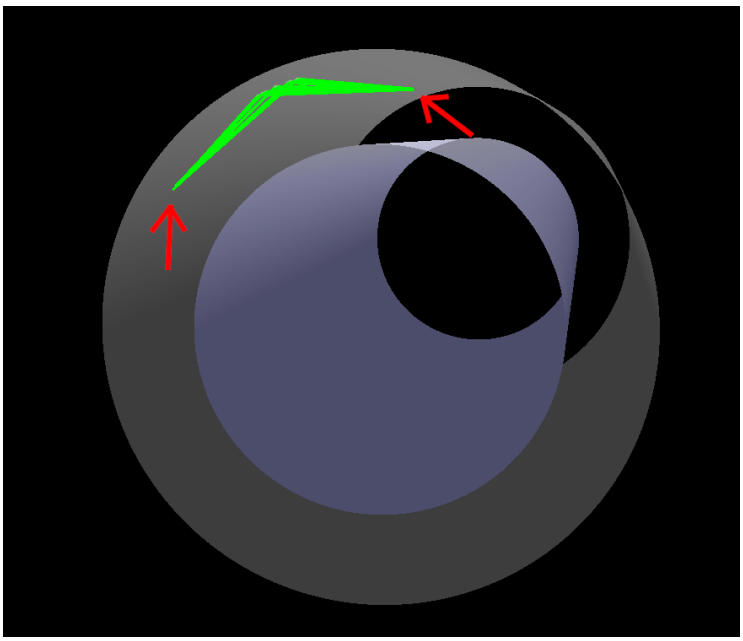


Fig. C.11.: Screenshot from the interactive renderer with rays that cause a peak in the light distribution. The arrows mark the light emission point and the end point.

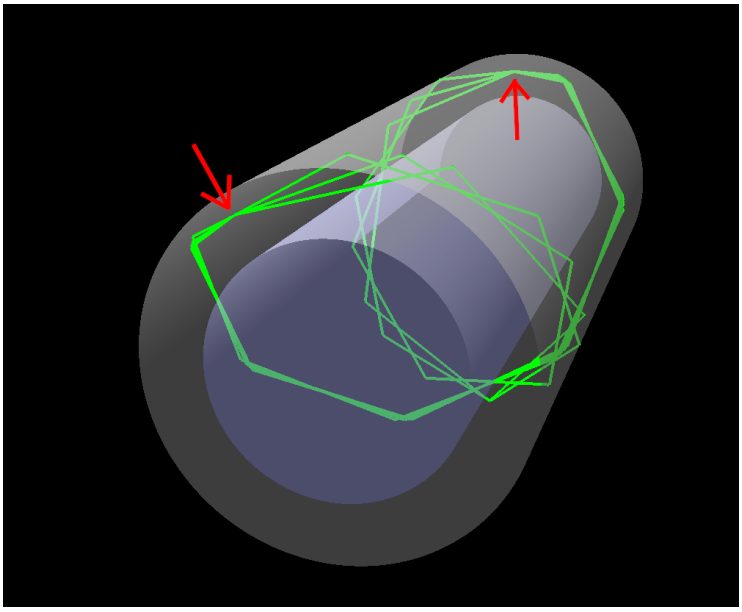


Fig. C.12.: Screenshot from the interactive renderer with rays that cause a peak in the light distribution. The arrows mark the light emission point and the end point. Some of the rays start in the opposite direction but still reach the same point.

C.4 Fit

Additional plot of a fit result for measurements of another tube.

Tab. C.4.: Simulation parameters for the fit.

N	R_o[cm]	R_i[cm]	L[cm]	n_1	n_2	λ_a[cm]	λ_s[cm]	z_0[cm]	y_0[cm]
2×10^6	4.5	4.18	55	1.49	1.0	λ_a	λ_s	d_i	4.4955

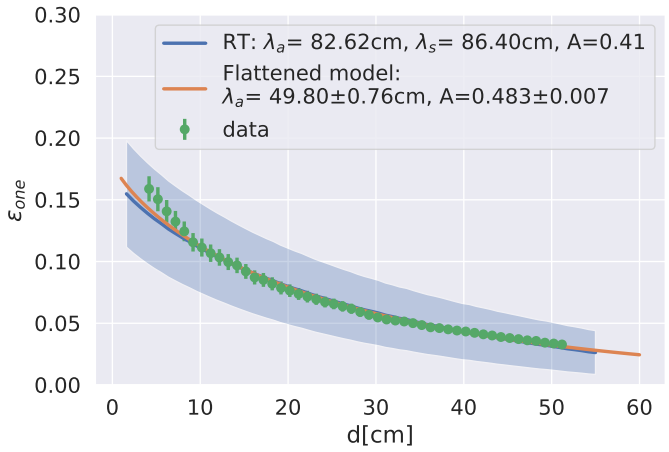


Fig. C.13.: Fit of the ray tracing simulation (blue) and the flattened model (orange) to an efficiency measurement of a tube (green). The measured tube has the code name "Q2" and the measurement was taken with the "South PMT". The fit of the ray tracing converged with $\chi^2_{red} = 0.58$. The fit of the flattened model with $\chi^2_{red} = 0.32$

List of Figures

Acronyms

ALG adiabatic light guide.

ALU arithmetic logic unit.

AoS Array of Structures.

API application programming interface.

CDF cumulative distribution function.

CPU central processing unit.

CU control unit.

CUDA Compute Unified Device Architecture.

DOM Digital Optical Module.

DRAM Dynamic Random Access Memory.

EM electromagnetic.

EMR electromagnetic radiation.

GPGPU general-purpose computing on graphics processing units.

GPU graphics processing unit.

PCIe Peripheral Component Interconnect express.

PDF probability density function.

PMT photomultiplier tube.

PRNG pseudo-random number generator.

SIMD Single Instruction Multiple Data.

SIMT Single Instruction Multiple Thread.

SM Streaming Multiprocessor.

SNR signal-to-noise-ratio.

SoA Structure of Arrays.

TIR Total Internal Reflection.

UV ultraviolet.

WLS wavelength-shifting.

WOM Wavelength-shifting Optical Module.

Bibliography

[1]S. Agostinelli, J. Allison, K. Amako, et al. „Geant4—a simulation toolkit“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506.3 (July 2003), pp. 250–303 (cit. on p. 1).

[2]Timo Karg. *Geant4 and Geometric Tolerances*. WOM Phone Call August 24, 2016. 2016 (cit. on p. 1).

[3]Daniel Popper. „Optical Simulation of the Wavelength-Shifting Optical Module“. Bachelor's Thesis. Johannes Gutenberg-Universität Mainz, Apr. 2017 (cit. on pp. 1, 75).

[5]Sebastian Böser. *Private communication*. 2019 (cit. on pp. 1, 78, 97).

[7]J. Allison, K. Amako, J. Apostolakis, et al. „Recent developments in Geant4“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 835 (Nov. 2016), pp. 186–225 (cit. on p. 2).

[8]HEP Software Foundation, : J Apostolakis, et al. *HEP Software Foundation Community White Paper Working Group - Detector Simulation*. 2018. arXiv: 1803.04165 [physics.comp-ph] (cit. on p. 2).

[9]Frederic Thomas Kirstein. „Partikelbasierte Lichtausbreitung in Rohren mit elliptischem Querschnitt mit Hilfe von CUDA“. Bachelor's Thesis. Johannes Gutenberg-Universität Mainz, Dec. 2017 (cit. on pp. 2, 55, 97).

[10]Ig. Tamm. „Radiation Emitted by Uniformly Moving Electrons“. In: *Selected Papers*. Springer Berlin Heidelberg, 1991, pp. 37–53 (cit. on p. 3).

[11]M.G. Aartsen, M. Ackermann, J. Adams, et al. „The IceCube Neutrino Observatory: instrumentation and online systems“. In: *Journal of Instrumentation* 12.03 (2017), P03012–P03012 (cit. on pp. 3, 4).

[13]Stephen G. Warren. „Optical constants of ice from the ultraviolet to the microwave“. In: *Appl. Opt.* 23.8 (1984), pp. 1206–1225 (cit. on p. 4).

[14]Dustin Hebecker, Markus Archinger, Sebastian Böser, et al. „A Wavelength-shifting Optical Module (WOM) for in-ice neutrino detectors“. In: *EPJ Web of Conferences* 116 (Jan. 2016), p. 01006 (cit. on p. 4).

[15]Dustin Hebecker. „Development of a single photon detector with wavelength shifting and light guiding technology“. MA thesis. Rheinische Friedrich-Wilhelms-Universität Bonn, Sept. 2014 (cit. on p. 5).

[16]Arthur Appel. „Some techniques for shading machine renderings of solids“. In: *Proceedings of the April 30–May 2, 1968, spring joint computer conference on - AFIPS '68 (Spring)*. ACM Press, 1968 (cit. on p. 10).

[17]Wolfgang Demtröder. „Elektromagnetische Wellen in Materie“. In: *Experimentalphysik 2: Elektrizität und Optik*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 209–248 (cit. on pp. 11, 13).

[19]Bertil Schmidt, Jorge González-Domínguez, Christian Hundt, and Moritz Schlarb. „Compute Unified Device Architecture“. In: *Parallel Programming*. Elsevier, 2018, pp. 225–285 (cit. on p. 13).

[20]David Kirk. *Programming massively parallel processors : a hands-on approach*. Cambridge, MA: Morgan Kaufmann, 2017 (cit. on p. 13).

[24]Egbert Brieskorn. *Lineare Algebra und Analytische Geometrie III - Geometrie im euklidischen Raum. Mit historischen Anmerkungen von Erhard Scholz*. 1. Aufl. 2019. Wiesbaden: Springer Fachmedien Wiesbaden, 2019 (cit. on p. 19).

[25]Josef Hoschek and Dieter Lasser. „Allgemeine Splinekurven“. In: *Grundlagen der geometrischen Datenverarbeitung*. Vieweg+Teubner Verlag, 1992, pp. 72–114 (cit. on p. 24).

[26]Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed Sep.03,2019]. 2001– (cit. on pp. 25, 82).

[27]P. Dierckx. „An algorithm for smoothing, differentiation and integration of experimental data using spline functions“. In: *Journal of Computational and Applied Mathematics* 1.3 (1975), 165–184 (cit. on p. 26).

[28]Paul Dierckx. *An improved algorithm for curve fitting with spline functions*. TW Reports TW54. Department of Computer Science, K.U.Leuven, Belgium, July 1981 (cit. on p. 26).

[29]Paul Dierckx. „A Fast Algorithm for Smoothing Data on a Rectangular Grid while Using Spline Functions“. In: *SIAM Journal on Numerical Analysis* 19.6 (Dec. 1982), pp. 1286–1304 (cit. on p. 26).

[30]Paul Dierckx. *Curve and Surface Fitting With Splines* -. New York: Clarendon, 1993 (cit. on p. 26).

[31]Peter Falke. „Entwicklung eines Lichtkonzentrators basierend auf einer Hohlzylinder-Geometrie“. Bachelor's Thesis. Rheinische Friedrich-Wilhelms-Universität Bonn, July 2014 (cit. on pp. 28, 90).

[32]Roland Winston, Lun Jiang, and Melissa Ricketts. „Nonimaging optics: a tutorial“. In: *Adv. Opt. Photon.* 10.2 (2018), pp. 484–511 (cit. on pp. 28, 92).

[33]Jost-Hinrich Eschenburg. *Sternstunden der Mathematik*. pp. 47-54. Springer Fachmedien Wiesbaden, 2017 (cit. on p. 30).

[34]L. Euler, J. Hewlett, F. Horner, J. Bernoulli, and J.L. Lagrange. *Elements of Algebra*. pp. 272-278. Longman, Orme, 1822 (cit. on p. 30).

[36]Roland Waldi. „Monte-Carlo-Rechnung“. In: *Statistische Datenanalyse: Grundlagen und Methoden für Physiker*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 233–260 (cit. on pp. 41, 43).

[37]John P. Snyder. *Map projections: A working manual*. 1987 (cit. on p. 47).

[38]Makoto Matsumoto and Takuji Nishimura. „Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator“. In: *ACM Trans. Model. Comput. Simul.* 8.1 (Jan. 1998), pp. 3–30 (cit. on p. 55).

[40]Mutsuo Saito. „A Variant of Mersenne Twister Suitable for Graphic Processors“. In: *CoRR* abs/1005.4973 (2010). arXiv: 1005.4973 (cit. on p. 56).

[41]George Marsaglia. „Xorshift RNGs“. In: *Journal of Statistical Software* 8.14 (2003) (cit. on p. 56).

[43]Pierre L'Ecuyer and Richard Simard. „TestU01“. In: *ACM Transactions on Mathematical Software* 33.4 (Aug. 2007), 22–es (cit. on p. 56).

[44]Lawrence E. Bassham III, Andrew L. Rukhin, Juan Soto, et al. *SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. Tech. rep. Gaithersburg, MD, United States, 2010 (cit. on p. 56).

[46]John Rack-Helleis. *Private communication*. 2019 (cit. on p. 58).

[48]Benjamin Bastian. „Characterization of cylindrical wavelength shifting optical light guides.“ MA thesis. Humboldt-Universität zu Berlin, Apr. 2019 (cit. on p. 72).

[49]Esther Ana del Pino Rosendo. „Study of the Light Propagation in the Wavelength-shifting Optical Module“. MA thesis. Johannes Gutenberg-Universität Mainz, Feb. 2016 (cit. on p. 73).

[50]John Rack-Helleis. „Efficiency determination of the Wavelength-shifting Optical Module (WOM)“. MA thesis. Johannes Gutenberg-Universität Mainz, Oct. 2019 (cit. on pp. 75, 79, 82).

[51]Roland Waldi. „Statistische Inferenz“. In: *Statistische Datenanalyse: Grundlagen und Methoden für Physiker*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 97–212 (cit. on p. 81).

[52]J. A. Nelder and R. Mead. „A simplex method for function minimization.“ English. In: *Comput. J.* 7 (1965), pp. 308–313 (cit. on pp. 82, 83).

[53]Dustin Hebecker. *Private communication*. 2019 (cit. on p. 95).

Webpages

[4]Photon Engineering. *FRED Optical Engineering Software*. 2019. URL: https://photonengr.com/fred-software/ (visited on Oct. 16, 2019) (cit. on p. 1).

[6]Herb Sutter. *The Free Lunch Is Over*. 2004. URL: http://www.gotw.ca/publications/concurrency-ddj.htm (visited on Oct. 6, 2019) (cit. on p. 1).

[12]Hamamatsu Photonics. *Hamamatsu Data Sheet – Photomultiplier Tube R7081-02 for IceCube Experiment*. 2019. URL: https://icecube.wisc.edu/~kitamura/NK/PMT/031112\%20R7081-02\%20data\%20sheet.pdf (visited on Oct. 17, 2019) (cit. on p. 4).

[18]NVIDIA Corporation. *CUDA Toolkit Programming Guide*. 2019. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (visited on Oct. 11, 2019) (cit. on p. 13).

[21]TechPowerUp. *NVIDIA Tesla V100 datasheet*. 2017. URL: https://www.techpowerup.com/gpu-specs/tesla-v100-sxm2-32-gb.c3185 (visited on Oct. 12, 2019) (cit. on p. 14).

[22]Frank Denneman. *Memory Deep Dive: DDR4 Memory*. 2015. URL: https://frankdenneman.nl/2015/02/25/memory-deep-dive-ddr4/ (visited on Oct. 12, 2019) (cit. on p. 14).

[23]PCI-SIG. *PCI Express-3.0 Frequently Asked Questions*. 2019. URL: https://pcisig.com/faq?field_category_value\%5B\%5D=pci_express_3.0&keys=bandwidth (visited on Oct. 12, 2019) (cit. on p. 14).

[35]Florian Thomas. *WOMRaT Git*. 2019. URL: https://etap-git.physik.uni-mainz.de/WOM/WOMRaT (visited on Sept. 26, 2019) (cit. on p. 39).

[39]NVIDIA Corporation. *cuRAND API reference guide*. 2019. URL: https://docs.nvidia.com/cuda/curand/index.html (visited on Aug. 30, 2019) (cit. on p. 55).

[42]NVIDIA Corporation. *CUDA 6.5 Performance Report*. 2014. URL: http://developer.download.nvidia.com/compute/cuda/6_5/rel/docs/CUDA_6.5_Performance_Report.pdf (visited on Aug. 30, 2019) (cit. on p. 56).

[45]Robert G. Brown. *Dieharder: A Random Number Test Suite*. 2006. URL: http://webhome.phy.duke.edu/~rgb/General/dieharder.php (visited on Aug. 30, 2019) (cit. on p. 56).

[47]The HDF Group. *THE HDF5 LIBRARY & FILE FORMAT*. 2019. URL: https://www.hdfgroup.org/solutions/hdf5/ (visited on Sept. 8, 2019) (cit. on p. 71).

Declaration

I hereby declare that I have written the present thesis independently and without use of other than the indicated means. I also declare that to the best of my knowledge all passages taken from published and unpublished sources have been referenced. The paper has not been submitted for evaluation to any other examining authority nor has it been published in any form whatsoever.

Mainz, October 25, 2019

Florian Thomas