TU WIEN Informatics

X-ray Path Tracing for CT Imaging

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Philipp Hochhauser
Matrikelnummer 01527619

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. DI Dr.techn. Eduard Gröller
Mitwirkung: DI Dr.techn. Thomas Auzinger
DI Dr.techn. Christoph Heinzl, Universität Passau
DI Sarah Rendl, FH Oberösterreich

Wien, 28. Jänner 2023

Philipp Hochhauser	Eduard Gröller

TU WIEN Informatics

X-ray Path Tracing for CT Imaging

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Philipp Hochhauser
Registration Number 01527619

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. DI Dr.techn. Eduard Gröller
Assistance: DI Dr.techn. Thomas Auzinger
DI Dr.techn. Christoph Heinzl, Universität Passau
DI Sarah Rendl, FH Oberösterreich

Vienna, 28th January, 2023

Philipp Hochhauser	Eduard Gröller

Erklärung zur Verfassung der Arbeit

Philipp Hochhauser

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 28. Jänner 2023

Philipp Hochhauser

Danksagung

An dieser Stelle möchte ich mich bedanken bei all jenen, die mich während meiner Studienzeit unterstützt haben. Ein besonderes Danke vor allem an meine Betreuer DI Dr.techn. Thomas Auzinger und DI Dr.techn. Christoph Heinzl. Auch wenn wir uns nie persönlich treffen konnten, habe ich in den fast wöchentlich stattfindenden Skype Konferenzen viel gelernt und bekam immer die Hilfe, die ich brauchte. Ein großer Dank auch an DI Sarah Rendl, die sich ebenfalls fast wöchentlich meinen Fragen in den Treffen gestellt hat und mich immer tatkräftig unterstützt hat, als ich auf Fehlersuche war.

Außerdem möchte ich mich bei meiner Familie und Freunden bedanken, die mich stets unterstützt haben in den vergangenen Studienjahren.

Acknowledgements

In this section, I want to thank each and everyone that supported me during my studies. Most of all, I would like to thank my advisors DI Dr.techn. Thomas Auzinger and DI Dr.techn. Christoph Heinzl. Even though we could never meet in person, we had almost weekly Skype sessions where I would always get the help I needed. Furthermore, I would like to thank DI Sarah Rendl, who also was a big help in finding and fixing different problems while working on my thesis.

A big thank you also to my family and friends, who have always supported me over the years.

Kurzfassung

Computertomographie (CT) ist ein wichtiges bildgebendes Verfahren in der Materialwissenschaft. Es wird verwendet, um die Eigenschaften und Strukturen verschiedenster Materialien herauszufinden, ohne die Werkstoffe dabei zu zerstören. Außerdem eignet es sich, um Spannungen in Bauteilen zu visualisieren und potenzielle Bruchstellen im Vorhinein zu identifizieren. Auch im Medizinbereich spielt die Computertomographie eine wichtige Rolle, dies wird allerdings in dieser Arbeit nicht genauer behandelt.

Computertomographien sind jedoch zeitaufwendig und kostspielig. Es ist also wichtig, genaue Simulationen durchführen zu können, um die Anzahl tatsächlicher Durchläufe zu minimieren. Deshalb präsentieren wir in dieser Arbeit ein Verfahren, um mittels Monte Carlo Pfadnachverfolgung (Path Tracing) Einzelbilder einer solchen Computertomographiesimulation zu erzeugen. Unsere Methode wird mit einer existierenden Lösung in Bezug auf Genauigkeit und Laufzeit verglichen.

Abstract

Computed tomography (CT) has a wide range of applications. It is an important technique in industrial non-destructive testing, where engineers want to find flaws in materials or reverse engineer the material composition of an object. Another application in material science is to visualize stresses that occur inside of an object. Furthermore, it is also often used in medical applications, however, the focus of this thesis lies on industrial computed tomography.

Actual computed tomography scans can take quite a long time and may become expensive. Therefore, it is essential to create software simulations that allow fast and accurate prototyping of such scans, in order to keep cost and time consumption to a minimum. In this thesis, we present a method to create such scans using Monte Carlo based rendering methods and compare it to an existing solution regarding accuracy and time.

Contents

CHAPTER 1

Introduction

1.1 Overview

In this thesis, we implement a system that can be used to generate virtual computed tomography (CT) images similarly to CT simulations. Simulating X-ray scans and outputting the generated images is not new. However, by using state-of-the-art programming frameworks and tools, we lay the groundwork for future improvements and innovations built upon our system. One of the major driving factors in this work is the future ability to use our proposed system for differentiable rendering, allowing the users to reverse the problem of rendering. This method of inverse rendering would allow the output of real computed tomography scans, to estimate features of the original scene, such as object geometry, material properties, or even properties of the X-ray source. To be able to do this, we first of all need a stable system that can accurately render images of CT scans as fast and precisely as possible. This is the main focus of this thesis. Further topics in X-ray simulations, such as reconstruction of different computed tomography scans into the scanned objects, will not be covered.

1.2 Approach

To create renderings of X-ray simulations, we used the rendering engine called Mitsuba 2, as proposed by Nimier-David et al. in late 2019 [NDVZJ19]. To be able to accurately model the underlying physics effects, we used a database of X-ray matter interaction [SBG+11] as well as data provided by the authors of SimCT [REK+16], an existing CT simulation software. We evaluate the results of our system and compare these to reference images as obtained by SimCT.

1.3 Thesis Structure

This thesis starts off by giving an overview of computed tomography and our motivation for improving existing methods. In Chapter 3 we present the general concepts of X-ray and physically based rendering that were used in the implementation of this thesis. Chapter 4 covers the actual setup we used, explaining how we implemented the physical effects and the used algorithms. One important goal of this thesis is to allow further work on the topic with the Mitsuba 2 renderer. We thus include what we learned while using Mitsuba 2 in Chapter 5. Chapter 6 covers an evaluation of our proposed method in comparison with an existing X-ray simulation method. Furthermore, we present ideas on how our initial work could be improved upon and ideas for applications that can be built using our proposed system. Finally, we summarize the results outlined in this thesis.

CHAPTER 2

State-of-the-art Systems

In this chapter, we briefly go over different ways of generating virtual CT scans. There are usually two different general approaches that state-of-the-art software systems use to generate respective images, which are deterministic simulations and Monte Carlo simulations. In the following subsections, the difference between these methods is described together with some examples. For a more detailed explanation of the theory the reader is referred to the sections about ray tracing and Monte Carlo methods in Chapter 3.

While there is more to computed tomography simulation than just generating CT images, such as reconstructing the 3D scene from the obtained 2D images, discussing all the different aspects of CT imaging is out of scope for this thesis. For an overview of different reconstruction algorithms, the reader is referred to a survey about CT reconstruction algorithms used in medical systems presented by Geyer et al. [GSM+15].

2.1 Deterministic Simulations

Most, if not all, CT Simulations are modeled with algorithms originating from the ray tracing method [Whi80]. The idea is to place an X-ray detector as well as an X-ray source in the scene, and then to analytically calculate the energy absorption due to radiation that passes through objects right between source and detector. Linear rays get shot from the source to the detector. The spectral response at the detector is calculated by multiplying the source X-ray spectrum with the detector spectrum, multiplied with the absorption at discrete energy ranges. See the Chapter about X-ray for a more in-depth explanation.

One major disadvantage of using the simple ray tracing approach, combined with analytically calculating the X-ray absorption, is that the physical processes cannot be modelled as precisely as compared to more sophisticated methods. For example, when a photon

scattering event happens, secondary rays with a different direction than the original ray get introduced. These rays are not covered with the linear ray tracing approach.

There are ways to approximate some of the physical effects, to get a more realistic image. Furthermore, multiple iterations of the simulation with a slightly different radiation origin can be averaged to approximate the existence of a focal spot. Some artifacts can also be added as post-processing effects, like artificially generating noise or filtering the resulting image with a kernel [Sch18].

The simulation tool SimCT [REK+16] uses purely analytical methods to calculate CT scans. The aRTist tool [BJ07] by the Institute for Materials Research and Testing (BAM) is another example for deterministic simulations.

2.2 Monte Carlo Simulations

To get more accurate results, Monte Carlo methods can be used instead of deterministic simulations. In contrast to the analytical method introduced before, a particle is not constrained to travel along a single straight path, it can scatter in different directions and even create other particles, that can also be traced through the scene. This of course requires much more computational effort, since more paths need to be traced through the scene. Another shortcoming is, that opposed to the analytical method, the spectral response is not calculated over all discrete energy ranges for each ray. Instead, only a single photon with a sampled wavelength is traced through the scene in each iteration, thus requiring even more paths, as multiple photons with different wavelengths have to be sampled for each pixel on the detector to get accurate results. Penelope [Age19] is a state-of-the-art simulation system, using Monte Carlo methods.

2.3 Hybrid Solutions

Many state-of-the-art CT systems nowadays use a hybrid approach to generate realistic CT scans. While originally being a tool to analytically simulate radiography, the aRTist system also switched to a hybrid solution in version 2. The authors implemented an (optional) integration of the Monte Carlo method, to more accurately describe scattering events [BDGJ12].

The CIVA system [FCT+12], developed by the French institute CEA-LIST, is a simulation tool for multiple methods of non destructive testing. In their component to simulate computed tomography, they also use deterministic methods to simulate the effects of direct radiation, while using Monte Carlo methods to describe scattering.

CHAPTER 3

Methodology

In this chapter, we go through some theory on computed tomography, especially on X-ray imaging and related topics, to give the readers some background on the physical processes that are modeled with our proposed system. Also, some theory on ray tracing and Monte Carlo methods is given, to help readers understand the implementation of the system. Figure 3.1 gives an overview of an industrial CT system, with an X-ray source on the one side, an X-ray detector on the other side and a specimen that is analyzed in the middle. The following subsections give some information on each part of such a system.

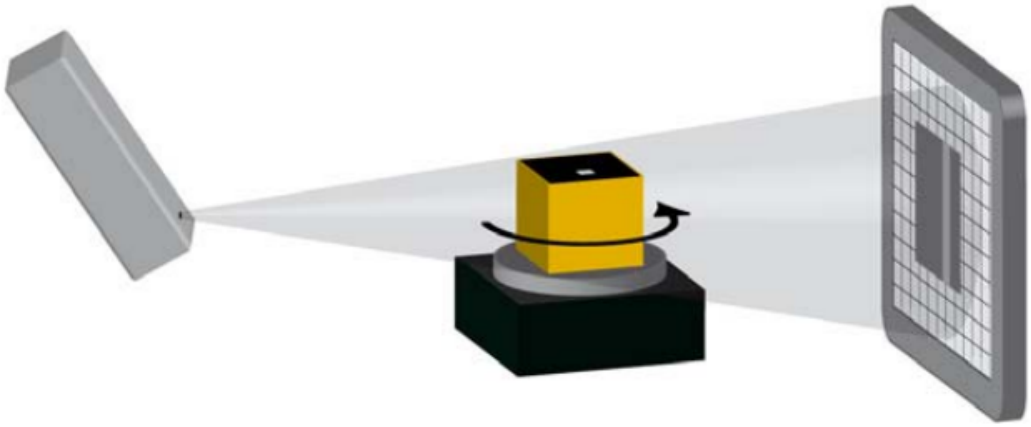


Figure 3.1: Illustration of a CT scene, with a cone beam X-ray source on the left, a flat panel detector on the right, and a specimen on a rotating table in the middle. Image obtained from [RHS+10].

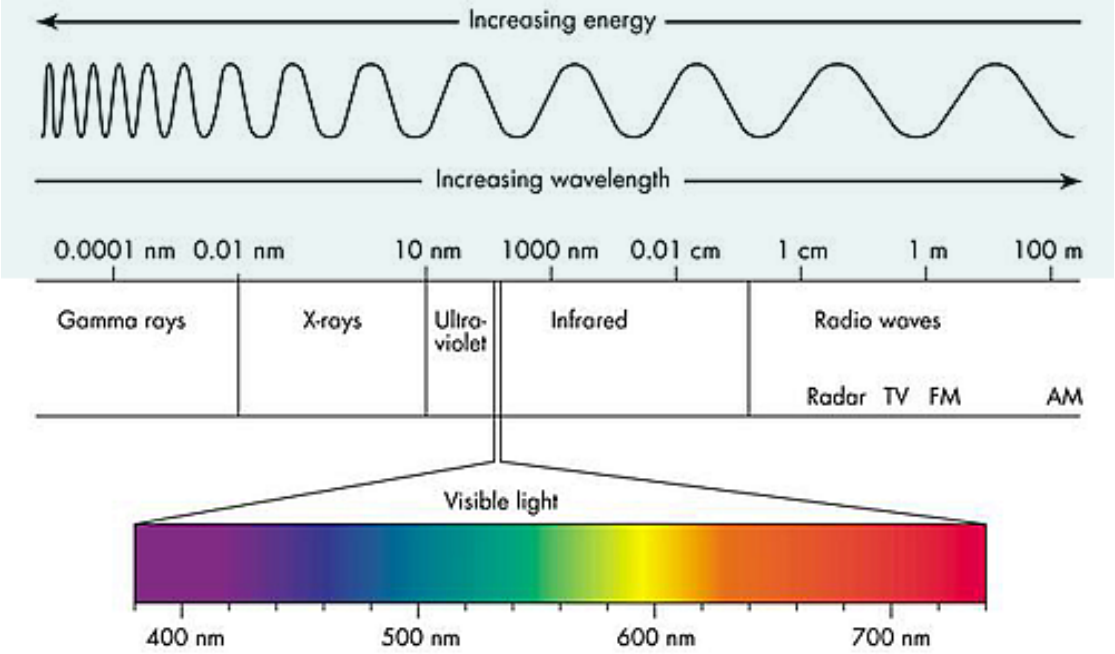


Figure 3.2: Electromagnetic Spectrum. Image obtained from [Cyb21].

3.1 X-rays

X-ray radiation is a type of electromagnetic radiation. As shown in Figure 3.2, it features much smaller wavelengths compared to visible light, but the energy of these rays is much greater. Electromagnetic radiation can also be described by small particles called photons. Photons have a fixed energy and frequency, where the frequency is directly proportional to the energy. The energy of a photon is inversly proportional to the wavelength of the electromagnetic radiation. The usual unit for X-ray energy is kilo electron volt (keV), with typical ranges of 0,12 to 120 keV according to the work by Wenjuan et al. [SBL12]. However, in modern high-energy CT systems, photons with an energy up to 15 MeV are used according to the work by Cantatore and Müller [CM11].

Figure 3.3 shows an example of an X-ray spectrum. The total energy of a spectrum can be described by the area under the curve.

3.1.1 X-ray Matter Interaction

Interaction of matter with electromagnetic radiation is usually explained by the Beer-Lambert law.

$$I = I_0 e^{-\mu d} \tag{3.1}$$

Where I is the energy after absorption, I_0 is the initial energy, d is the length of the absorbing material, and μ is the linear attenuation coefficient of the material. This

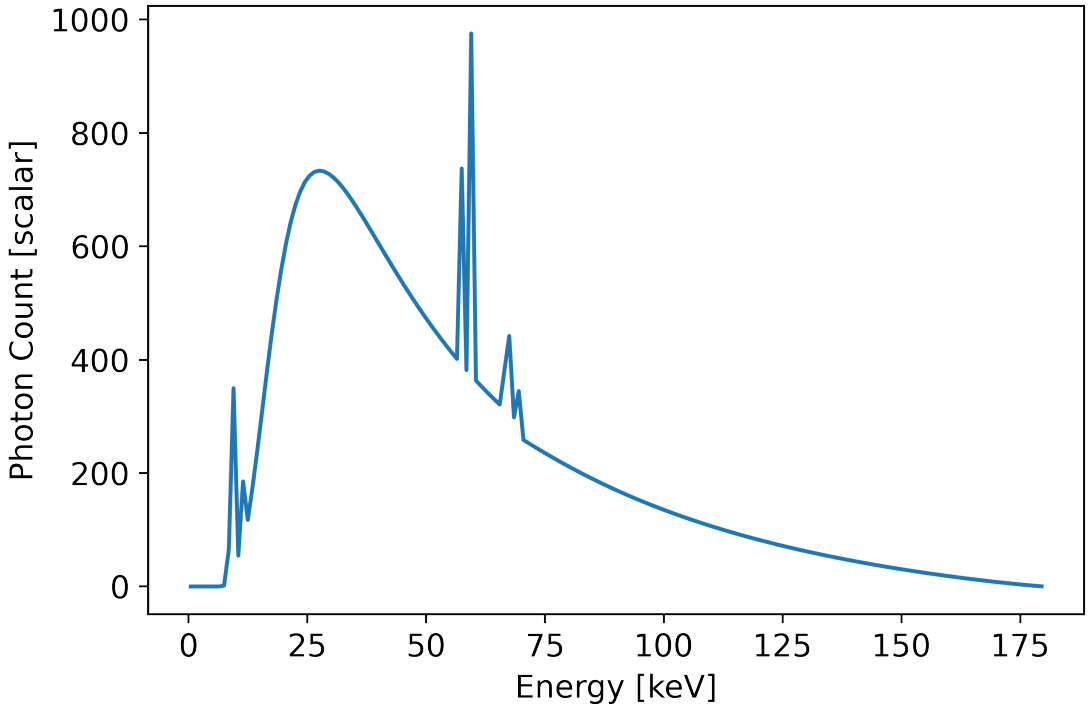


Figure 3.3: Typical spectrum of an X-ray source with a voltage of 180 kV, and current of 90 µA. The peaks in the plot represent the characteristic lines. Data obtained from the authors of SimCT [REK+16]

equation shows how energy is decreased if photons with a specific energy are passing through matter with a certain density and for some length.

3.1.2 Linear Attenuation Coefficients

The attenuation coefficients mentioned in Equation 3.1 are modeled after different physical processes of photons interacting with matter. The most important ones are the photoelectric effect (1), Compton scattering (2) and Rayleigh scattering (3) [BM09].

(1) The photoelectric effect occurs if a photon hits an electron and passes all its energy onto the electron. As this effect absorbs all the photons' energy at once, this effect is also called the photoelectric absorption.

(2) Compton scattering occurs if a photon interacts with an electron, but the electron does not absorb all its energy. Instead, part of the energy of the photon is absorbed by the electron. The photon is scattered in a different direction and because part of its energy was absorbed, the photon now has a lower energy.

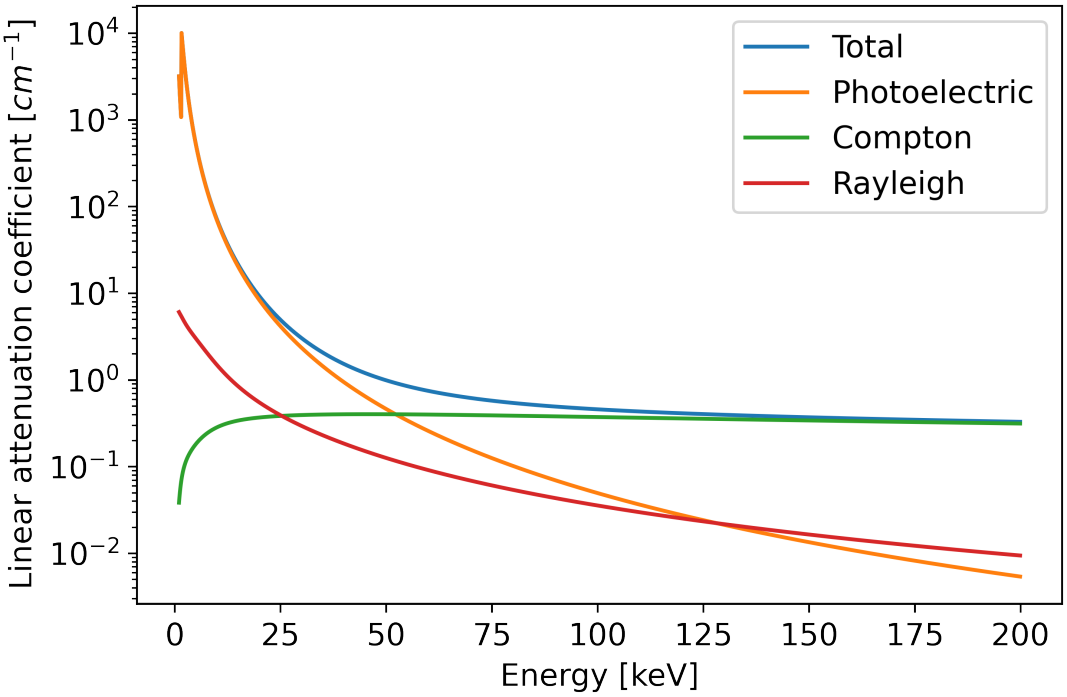


Figure 3.4: Spectrum of linear attenuation coefficients for Aluminium with a density of 2.6989 g/cm^3. Figure created with data obtained from xraylib [SBG+11].

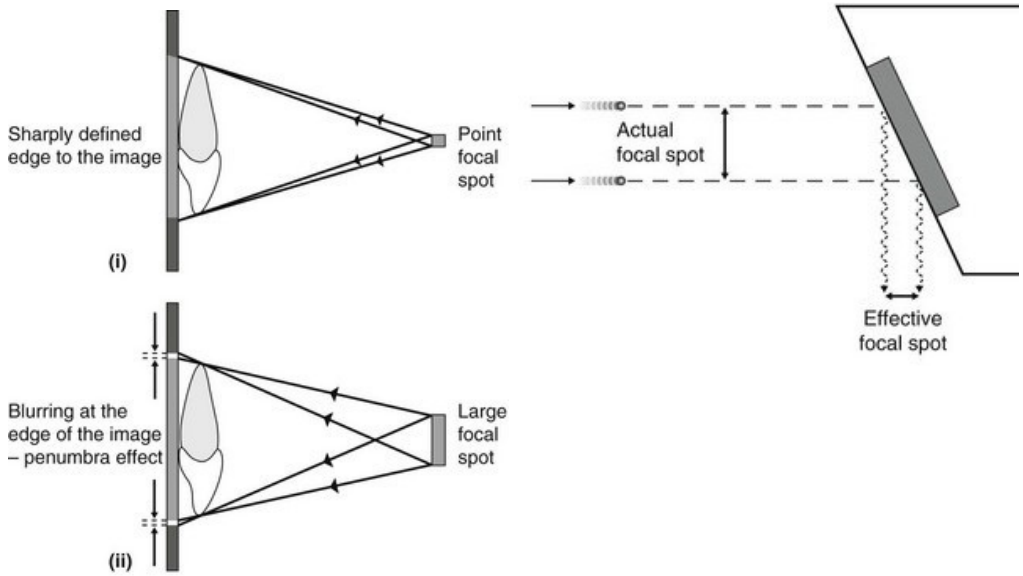
(3) In Rayleigh scattering, the photon is also scattered in some direction because of the interaction with an electron. The photons has the same wavelength before and after the interaction, and therefore also the same energy.

Comparing the three effects, Figure 3.4 shows that the photoelectric effect is the main contribution to the attenuation of X-ray through matter interaction in the lower energy ranges.

3.1.3 X-ray Generation

Section 3.1.1 shows how X-ray matter interaction can be modeled using the Beer-Lambert law, while Section 3.1.2 provides background on the physical phenomena that happen at the particle level. This subsection explains the practical process of of generating X-rays.

One method to create the X-ray radiation is to accelerate ionized particles and shooting them onto matter inside an X-ray tube. The idea is to heat up a cathode so that it radiates ionized particles. Those particles are then focused onto an anode. There it generates X-rays by hitting the target material and thus slowing down in the process, creating the *Bremsstrahlung*. This radiation generates a smooth distribution of energy.



(a) Blurring effect occurring due to a large focal spot. Edges of the object will become unsharp as rays will project the same point onto different pixels of the detector

(b) Effect of lowering effective focal spot size due to focusing onto target at a 90-degree angle

Figure 3.5: Visualization of focal spot effects. Original images obtained from Pocket Dentistry [Den15]. Images were slightly modified.

Sometimes the incoming electron interacts with an electron in the innermost shell of the target material and kicks it out of its orbit. The atom needs to fill this orbit again, using an electron from a higher shell. Depending on the energy difference of the shells, a photon with a specific energy is emitted, leading to the characteristic lines, depending on the material used in the anode target. The characteristic lines can be seen in Figure 3.3 at the peaks in the plot. It takes high voltages to generate X-rays, but most of the energy is converted into heat. To withstand those conditions of high heat, the tube is often made of tungsten, as it is highly heat-resistant. Ideally the generated ray of ionized particles are focused onto a single point on the anode. This is practically impossible, so the ray will hit an area of the target instead of a single point. This is called the focal spot and for example leads to multiple artifacts when generating X-ray images. To minimize these effects, the target ray is taken from a 90-degree angle of the incoming ray, to reduces the size of the focal spot. The effects of the focal spot on generated X-ray images are visualized in Figure 3.5.

3.1.4 X-ray Detectors

As briefly mentioned in the last subsection, the focal spot drift can cause artifacts in generated X-ray images. This subsection explains how X-ray detectors can be used to

create such images. In general, an X-ray detector is a device that measures incoming X-ray radiation over an area. A detector typically consists of several smaller units covering that area to measure the amount of radiation hitting each unit. The units are often arranged in a matrix, such that the output of the detector is an image with the same resolution. We use the term detector pixel to refer to a single measurement unit of an X-ray detector throughout this thesis.

There are different methods that detectors use to count the photons. These approaches vary based on the different energy ranges they can detect, as well as on the actual hardware cost. Some X-ray detectors work by counting the number of photons hitting the detector in each detector pixel and outputting a grayscale image depending on the energy of the counted photons. Single photon counting detectors are not the standard yet. Most X-ray detectors in use integrate the light of incoming photons over a specified integration time. The reader is referred to the work by Baba et al. [BKUI02] for a comparison of two different detector architectures, the flat-panel detector and the image-intensifier detector.

3.2 Ray Tracing

In Section 3.1, an overview of X-ray radiation is given, as well as a practical example how to generated images of X-rays by using an X-ray source and an X-ray detector. In order to simulate those processes using software, algorithms to model the generation of images are needed.

There are two distinct approaches for rendering images on a computer. On the one hand there is rasterization, which essentially places objects in a 2D or 3D world and by using a rasterized camera, it determines which objects can actually be seen. Unless objects are (semi-)transparent, only the objects in the front are rendered. This results in quickly generated images. However, the generated images do not look very realistic, as only local information about object materials is used and no global illumination effects can be included.

Therefore, almost all systems that try to model some kind of physically-based rendering simulation, rely on the same fundamental technique, the ray tracing algorithm, proposed by Turner Whitted in 1980 [Whi80].

In this algorithm, first a ray is cast from the camera into the scene and is tested for intersection with any object. At the intersection, information about color and lighting are calculated. If this information is immediately returned, this method is often called ray casting. Rays can also be traced further with a different ray direction, if the object is reflective or refractive, to get more accurate information.

As simple as this algorithm may be, it is computationally very expensive. To generate a single image, millions of rays have to be traced through the scene. For this reason conventional graphics cards did not support this algorithm for a long time and were instead built for the rasterization pipeline. Only with the introduction of the NVIDIA

Turing Architecture in 2018 [Bur19], hardware support for ray tracing was introduced into consumer grade computer graphics cards.

3.3 Monte Carlo Path Tracing

Images generated using ray tracing, are physically more accurate than images rendered using rasterization. In reality, the light rays are not simply refracted or reflected at an intersection point, but light is scattered or absorbed into multiple directions, depending on the physical properties of the object.

In contrast to simple ray tracing, path tracing is trying to calculate whole paths of light rays traversing the scene. In every interaction-point of the scene, a ray is either scattered into a new direction or absorbed by the material. The probability of a path getting reflected or absorbed at a surface is given by the bidirectional scattering distribution function (BSDF), which is a combination of the bidirectional reflectance distribution function (BRDF) and bidirectional transmittance distribution function (BSDF) [BDW81]. This results in paths of infinite length, and therefore in an infinitely dimensional integral. So there is no method to solve this problem analytically. Monte Carlo path tracing numerically approaches this integral by using a statistical method called Monte Carlo integration [Kaj86].

In theory, this method converges to the correct solution eventually. In practice, the actual number of times the method is run is chosen empirically, until a good enough result is achieved.

There are many techniques to improve image quality or rendering time for path-tracing, such as multiple important sampling and the usage of spatial acceleration structures respectively. Furthermore, there are several different variants of the Monte Carlo path tracing algorithm, such as bidirectional path tracing [LW96, Vea97] and Metropolis light transport [Vea97], which are usually more efficient.

Refer to the book about physically-based rendering [PJH16] for an overview of all the different variants and techniques. Going in to depth on all the different variants and techniques would be out of scope for this thesis, and many alternatives are already implemented in the Mitsuba 2 rendering system.

3.4 Reciprocity

This chapter so far presents the theoretical background on a system to generate X-ray images, by using the Monte Carlo path tracing algorithm, and modeling the interactions of X-ray and matter using the Beer-Lambert law. The system that is presented in this thesis in the following chapters uses a scene that traces rays generated from an X-Ray Detector back to an X-ray source, which is reciprocal to the theoretical setup described in this chapter.

The Helmholtz reciprocity describes how the optical properties of a light ray, such as reflection and absorption, are reversible in an optical system. According to Veach [Vea97], the Helmholtz reciprocity does not imply the direct reversibility of light rays, if BSDFs are used to model physical materials, as the original reciprocity principle only applies to rays reflected by mirrors. However, by using the Kirchhoff's equilibrium radiance law in combination with the principle of detailed balance, he presents a reciprocal principle that holds for arbitrary BSDFs [Vea97].

We use this reciprocity, to verify that results obtained from our system can be compared with a reference simulation, that uses a setup that is reciprocal to ours. More details on this reciprocal setup in the following Chapter 4.

CHAPTER 4

Implementation

In Chapter 3, we present the theoretical background of how hardware computed tomography systems work. In this chapter, we show how the Mitsuba 2 framework can be modified to simulate such a system to create X-ray images.

4.1 Scene Setup

As is shown in Section 3.1, a typical hardware system to generate X-ray images consists of an X-ray source emitting X-ray radiation, which then passes through one or more objects in the scene. The radiation that is not absorbed or scattered while traversing the scene will finally hit an X-ray detector screen, to generate an image. The area where the

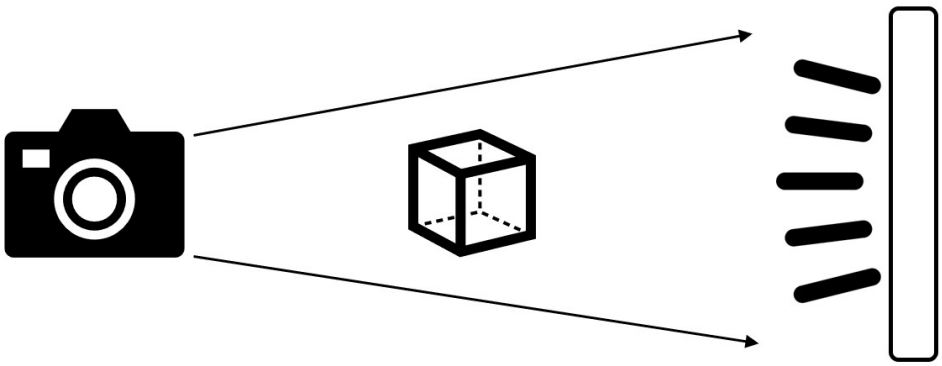


Figure 4.1: Schematic visualization of the system proposed in this thesis. There is a camera on the left, an area emitter on the right, and a specimen in the middle.

radiation of the X-ray source is emitted corresponds to the size of the focal spot. This area is relatively small compared to the size of the X-ray detector.

The Mitsuba 2 framework provides plugins that model different kinds of emitters and sensors. Emitters are entities that emit radiation, whereas sensors are entities that detect radiation. We could create a small emitter that sends radiation towards a large sensor, which would then model an X-ray source and an X-ray detector respectively. However, it turned out that implementing such a sensor was not trivially possible without changing large parts of the Mitsuba 2 code base.

Therefore, we use a reciprocal system, where there is an emitter that sends out radiation from a large area, corresponding to the area of an X-ray detector. To detect the radiation after traversing the scene, we use a small camera with an aperture that corresponds to the size of the focal spot of an X-ray source. The terms camera and sensor are used interchangeably throughout the following chapters of this thesis. A schematic visualization of this setup is showin in Figure 4.1.

This reciprocal setup should generate the same output compared to a non reciprocal setup. In Section 3.4 we provide additional information on the theory of reciprocal optical systems.

4.2 Material Generation

To model the physical properties of different specimen, we generate different spectra using xraylib [SBG+11], a Python API with an included database for X-ray matter interaction. These spectra contain the linear attenuation coefficients for different materials. These spectra are referred to as materials throughout this thesis.

A material for an element with atomic number Z is generated by taking the scattering cross-section (i.e., the probability of a scattering event) and multiplying it with the density of the material. This results in a distribution showing linear attenuation coefficients for a range of energies. Figure 3.4 shows an example of such a spectrum. The higher the energy of a photon, the less it gets absorbed by the material. The data is converted into a Mitsuba 2 compatible file format (.psd) and then passed into the scene file as materials for the desired objects. This spectral distribution determines how much energy is absorbed when rays pass through an object. As surface scattering can be disregarded in the case of X-rays, we do not use any BSDF.

4.3 X-ray Spectra

To obtain the spectra for our emitters, we relied on data obtained from the authors of SimCT [REK+16]. The spectra contain the number of photons per second per defined solid angle, for an energy range between 0 and 180 keV.

The spectra are only accurate for a specific solid angle. We cannot simply change the distance of the emitter to the sensor without also using a new spectrum. Furthermore, the

values in the calculated spectra describe the number of photons hitting the target solid angle of the detector plane for an exposure time of one second. To change exposure time, the spectra can simply be multiplied by the desired factor. The spectra were calculated for a solid angle with pixel area of 100 nm^2. Therefore, we additionally multiply the spectra by a factor of four, to get the desired irradiance for our modeled detector with a pixel size of 400 nm^2. To pass them to our simulation, we converted the spectra into the Mitsuba 2 compatible .psd format. This is used to create a rectangular area emitter with the spectral distribution as the radiance.

4.4 Focal Spot

To measure the amount or radiance traversing our scene, a sensor plugin has to be implemented. However, a pinhole camera should not be used to model a sensor that corresponds to an X-ray Source of a traditional CT system. This is due to the focal spot effect that was explained in Section 3.1.3. An idealized perspective pinhole camera represents a sensor with an infinitesimally small aperture, which would correspond to an X-ray source generating radiation from an infinitesimally small point on the anode, which is physically impossible. So instead we implemented the camera for our system with a small lens in front of it, where the aperture of the camera is exactly the same as the size of the focal spot.

To achieve this effect of modeling a camera where the origin is not a single point, we first sample a discrete 2D distribution. This distribution shows the probability of a ray not starting perfectly from the camera's origin, but rather starting from an offset position in the 2D plane perpendicular to the camera's viewing direction. Figure 4.2 shows such a distribution. The x and y coordinates describe the offset of the ray's starting point from the ideal infinitesimally small point the rays should start from, and the z axis describes the probability of a ray starting at exactly this x and y coordinate.

Our implementation consists of two different distributions obtained by the authors of SimCT [REK+16]. This data was generated by measuring the focal spot drift of different X-ray tubes. We created a sensor plugin for Mitsuba 2 to implement the camera. This sensor plugin is used to instantiate the rays passing through our scene. We refer to the code snippet in Section 4.9 to convey how rays are sampled from this camera with a focal spot offset.

4.5 Image Generation

So far, this chapter gives an overview of our software system. There is a camera with a small aperture where rays get generated, an emitter that radiates with a specific spectrum, and a specimen in between that absorbs radiation traversing the scene. A more detailed overview of the image generation using this system is presented in this section.

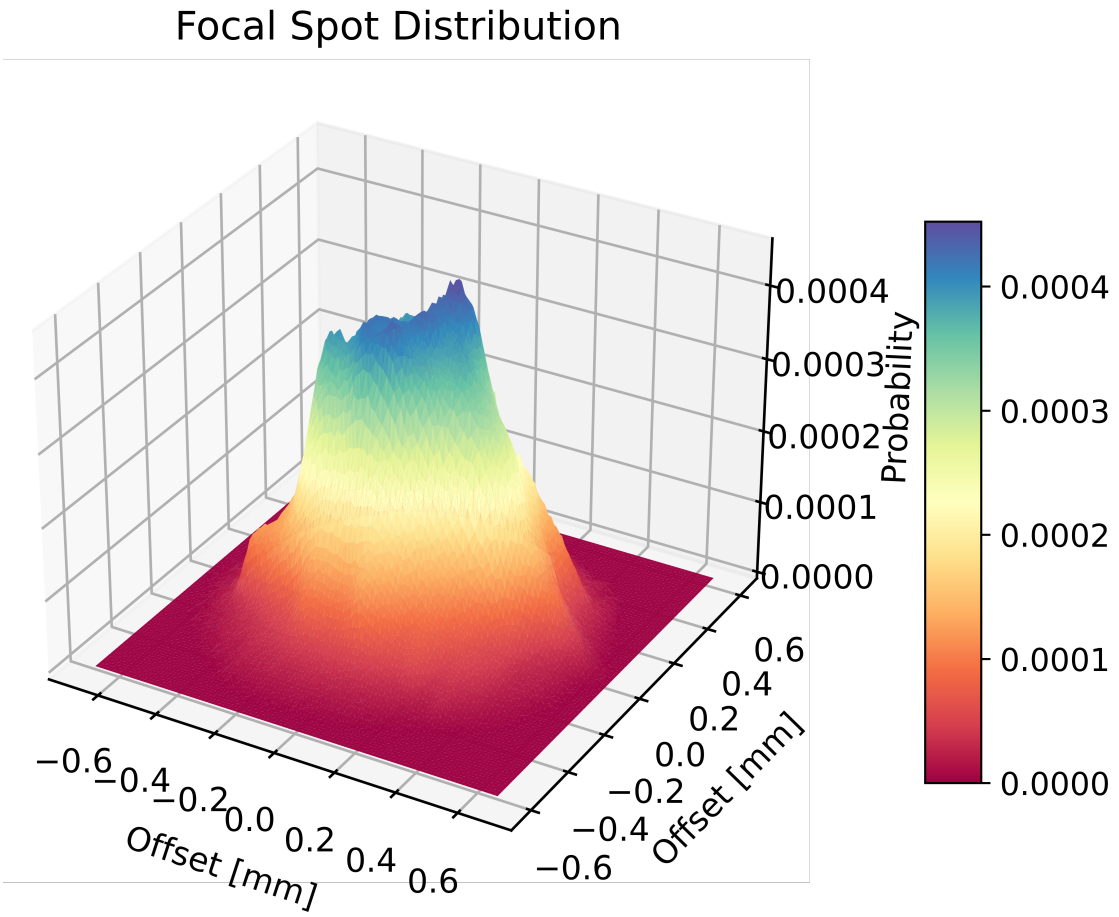


Figure 4.2: Distribution for a focal spot

First, we generate a ray inside our camera. Each ray has a specific starting position in the x-y plane of the camera, obtained by sampling the focal spot distribution. To take samples from the focal spot distribution, we use a method called (binary search) inverse transform sampling. This method works by taking the discrete distribution and calculating an array where each element contains the summed probabilities of prior elements. The newly generated array is now filled with values ranging from zero for the first index, to one for the last index. With a random variable in the range of zero to one, we can now select the index of the array, where the corresponding element is closest to the random variable. Finding this index is done in a computationally efficient way using binary search. For a more detailed explanation on this method and other sampling strategies, the reader is referred to the work by Little [Lit19].

Each generated ray has a specific wavelength, obtained by uniformly sampling the energy range defined for our scene. Mitsuba 2 includes advanced sampling strategies for spectral distributions in the visual spectrum, no such methods are present for arbitrary spectra. The detector efficiency that is used to calculate the final output image could in theory

be used as a distribution to optimize the wavelengths of the sampled rays. This means that the probability of choosing a ray with a specific wavelength should be proportional to the contribution a ray with this specific wavelength would have on the final output image. We tried implementing this using the built-in Mitsuba 2 method for sampling regular discrete distributions that is also based on (binary search) inverse transform sampling, but additionally uses interpolation to sample the value. However, we disabled this sampling method because it caused our system to produce incorrect renderings.

The generated rays then intersect objects in our scene and finally hit the emitter. The spectral absorption is calculated using the Beer-Lambert law and the provided materials of the objects the ray passed through.

At this point, we have a spectral response for a pixel on the camera. Mitsuba 2 usually generates a color image from spectral responses by looking up the corresponding tristimulus values in the CIE distribution. Our implementation instead generates outputs by using the spectral responses and calculating a corresponding greyvalue, using a detector efficiency table. We see an example of a detector efficiency table in Figure 4.3. The vertical values of this graph show the contribution of a single photon with a specific energy onto a pixel of the detector. As the spectral response in our program models the number of photons, we calculate the respective greyvalue by simply multiplying the response with the detector efficiency. The graph as given in Figure 4.3 assumes the program internally calculates with 16 bit integers, so we divide the vertical axis by 65536, to fit the input data to the Mitsuba 2 framework.

After all rays have been traversed through the scene, we have floating point values for every pixel on the camera plane. If multiple samples per pixel are used, the final color is the average of all calculated values for this pixel. A value of 0 means no photon was detected, and 1 means the greyvalue of the detected pixel is fully saturated. Values higher than 1 will not add any difference in the output image. The output is converted into a 16 bit uncompressed PNG format, to be able to compare it with different implementations.

4.6 Rendering Parameters

In Section 4.5, a detailed overview of the image generation using our proposed system is given. This section provides information on parameters that influence the outcome of our system.

Samplers The Mitsuba 2 framework allows different samplers for sensor objects. We simply forward the chosen sampler to the Mitsuba 2 framework. An overview of the available samplers is given in the documentation [NDVZ+20].

Focal Spot As our system currently supports two different focal spot distributions, we implemented a Boolean flag that can be passed to the Mitsuba 2 framework, determining whether the small focal spot should be chosen over the big focal spot.

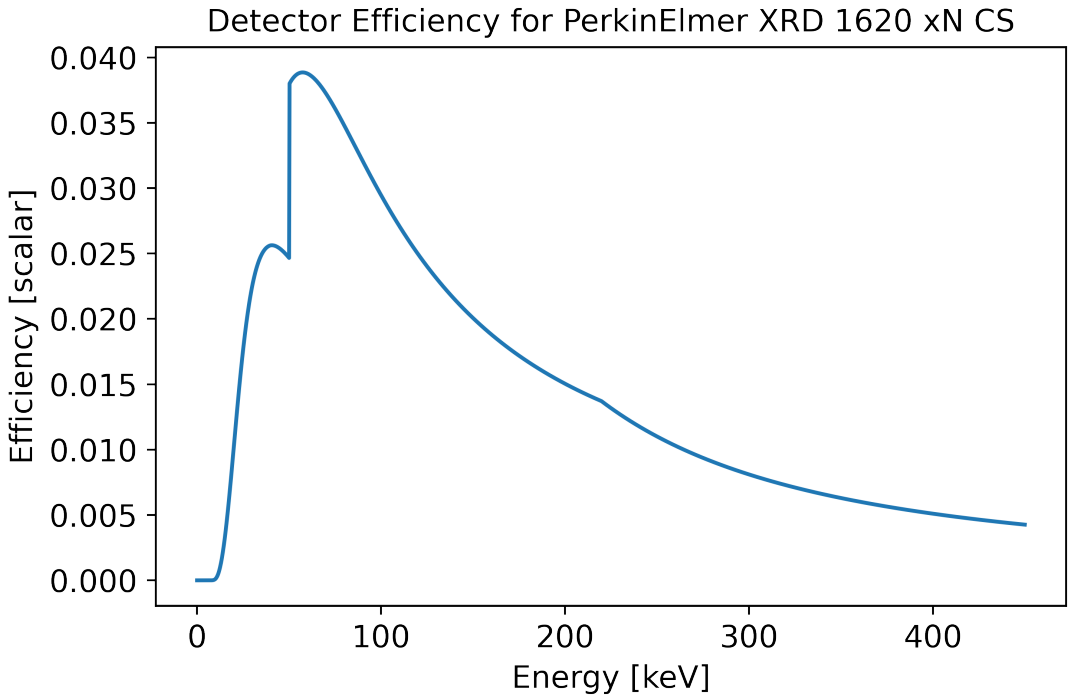


Figure 4.3: Detector efficiency used in our program. The graph shows the detector efficiency of a PerkinElmer XRD 1620 xN CS flat panel detector. Data obtained from the authors of SimCT [REK+16].

Detector Efficiency As all of our reference images were obtained using the same X-ray detector, we currently hard-coded the chosen detector efficiency.

Scene Geometry When changing the distance between the camera and the area light source, the spectrum has to be chosen accordingly. This is because the spectrum is calculated for a specific solid angle, as increasing the distance will lower the total amount of energy hitting the camera. We want to render images as if they were taken from a rectangular X-ray detector. We have to fit our area light emitter exactly onto the rectangular output image. Therefore, the following formula is used to calculate the appropriate camera field of view in degrees:

$$Fov = \frac{360}{\pi} * \tan \frac{detector\_sidelength}{distance\_between\_camera\_and\_arealight} \tag{4.1}$$

The far clipping plane of the camera is set to be just behind the area light source, with a small offset to prevent artifacts. The focus distance is set to be exactly the distance between camera and area light.

X-ray Source Spectrum Different emitting spectra can be assigned to the area light source, by passing a file in the Mitsuba 2 .psd file format. Some calculated spectra are included in our repository.

Object Material Different absorbing spectra can be assigned to objects, by passing in a file in the Mitsuba 2 .psd file format. Some calculated materials are included in our repository. Materials can also be calculated using the provided *data_preparation.ipynb* notebook.

Resolution and Samples Per Pixel The samples per pixel value can be any positive integer. Increasing the number will improve convergence, but will increase render time. Too high values can cause memory overflows. The same applies to the resolution. We only support resolutions with the same dimension in width and height. Note that Mitsuba 2 might change the chosen sample per pixel, if a different sampler than the independent sampler is used.

4.7 Usage and Included Files

To use our simulation software, two python notebooks and a simple python API were implemented. In this subsection, a quick overview of the different files containing our system is given. Refer to the repository [HA20a] for further information.

ct-simulation.ipynb The *ct-simulation.ipynb* Jupyter Notebook shows an example of how our system can be used to render images. We first of all define the different rendering parameters, and then simply call the *renderer.render_scene* method to generate the images. Using the *renderer.plot_rendered_scenes* function, we can visualize the output images and generate plots, showing the distribution of values in the image as well as a slice through the image.

data_preparation.ipynb The *data_preparation.ipynb* Jupyter Notebook is used to visualize the data we obtained from the authors of SimCT [REK+16]. This data consists of reference images, spectral distributions of X-ray sources, as well as distributions showing detector efficiencies. The notebook can be used to save the data in a format that is compatible with the Mitsuba 2 framework.

Furthermore, we used this notebook to simulate the desired outcome of our system on an empty scene. This was used to check the integrity of our system, as we did not achieve the same results as the reference system by the authors of SimCT [REK+16]. See also Section 6.

Originally, this notebook was also used to convert all the input data into different units. All the data on electromagnetic radiation we obtained from the authors of SimCT [REK+16] used (kilo) electron volt as the unit, representing the energy of the radiation. The Mitsuba 2 rendering framework, however, assumes that the unit for the

electromagnetic radiation should be in nanometers, representing the wavelength of the radiation, as this fits well when dealing with radiation in the visible light spectrum.

We discovered that if we converted the X-ray units into nanometers, the distribution of rays became very unbalanced across our energy range. This resulted in rays getting sampled from energies where the ray had almost no impact on the final image, drastically lowering the convergence rate of our rendering system. Figure 4.4 shows this problem, on the graph that was converted to use nanometers as unit for electromagnetic radiation. The graph has a wide range of horizontal values, where the corresponding vertical values are close to zero, and thus have almost no impact on the output of our system.

We also found out, the Mitsuba 2 framework does not need to know internally what units it is using, as long as all distributions like material spectra, X-ray spectra, and detector efficiency are chosen accordingly. We therefore chose not to transform the data into nanometers, but keep the data as is.

renderer.py In the *renderer.py* Python file, we implemented helper functions, to provide an easy-to-use binding for our customised rendering system based on the Mitsuba 2 framework. Furthermore, we included some code to visualize results and to render the scene geometry using OpenSCAD [Ope22].

4.8 Changes to Mitsuba 2

The actual changes to the Mitsuba 2 framework are implemented in a different repository [HA20b]. The most important part is the newly created sensor plugin modelling a camera with a focal spot offset, contained in *focalspot.cpp*. The next section covers this plugin more in-depth.

We had to change multiple constants, as the Mitsuba 2 framework was originally designed to render images in the visible light spectrum. Those constants are used to set the limits of the sampled energy spectra, as well as to ensure no rays are shot through the scene where the wavelength was out of the visible spectrum.

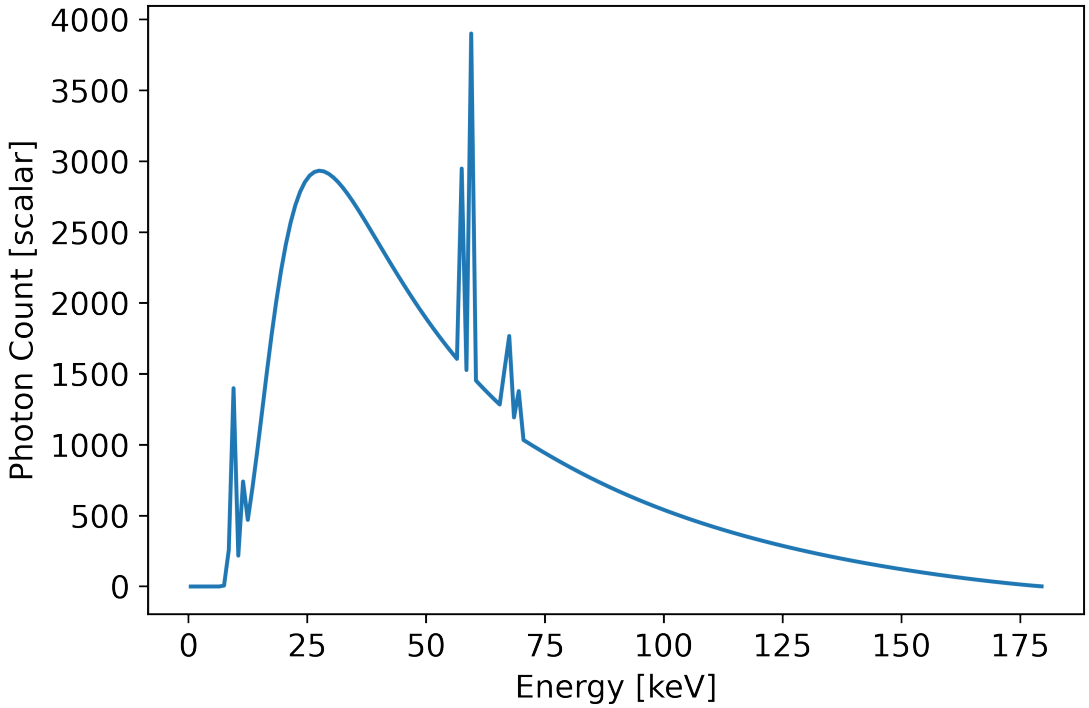
Furthermore, we had to change the way spectral responses are converted into actual images. Originally using CIE lookup tables for tristimulus values, we changed the framework to use spectral responses to generate greyvalue images according to a fixed detector efficiency.

4.9 Ray Sampling

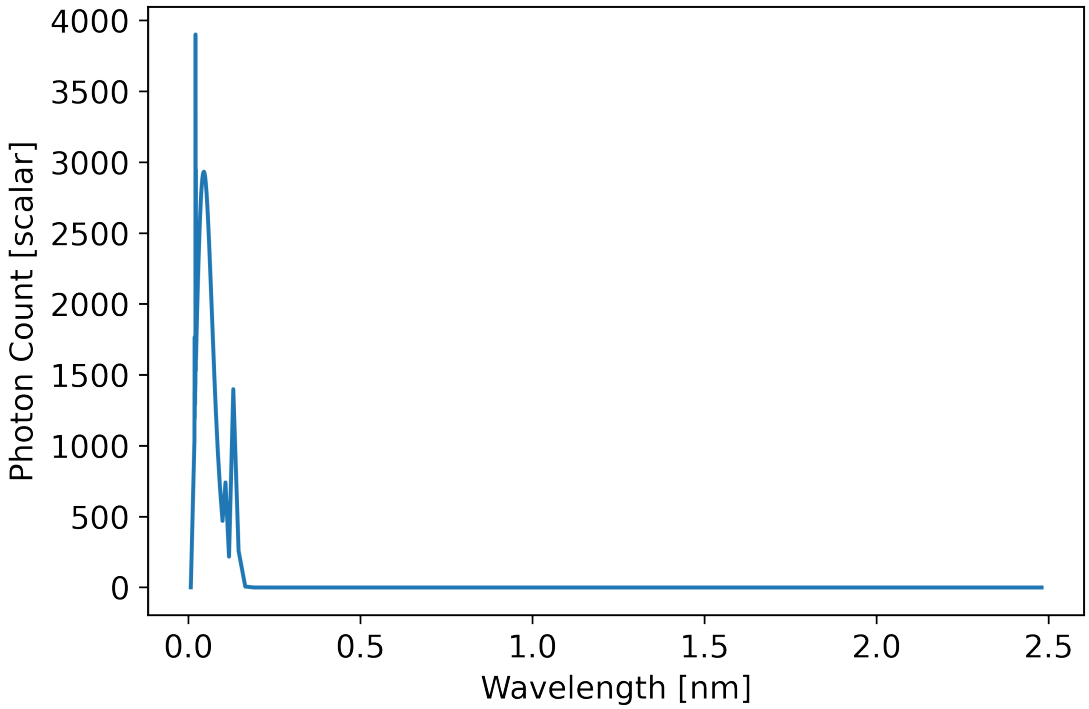
In Section 4.1, we mentioned that our system uses a sensor plugin, with an area corresponding to the size of the focal spot of an X-ray source. We provide the following code snippet to convey how this can be implemented using the Mitsuba 2 framework. The code was heavily inspired by the perspective camera with a thin lens plugin [NDVZ+20].

```
std::pair<Ray3f, Spectrum> sample_ray(
                Float time,
                Float wavelength_sample,
                const Point2f &position_sample,
                const Point2f &aperture_sample,
                Mask active) const override
    {
    MTS_MASKED_FUNCTION(ProfilerPhase::EndpointSampleRay,
    active);


    Wavelength wavelengths;
    Spectrum wav_weight;

    // Determine if uniform sampling or
    // Importance sampling should be used
    if (m_detectorSpectrum == nullptr) {
        std::tie(wavelengths, wav_weight) =
            sample_wavelength<Float, Spectrum>
                (wavelength_sample);
    } else { // Not used as it would cause artifacts
        std::tie(wavelengths, wav_weight) =
            m_detectorSpectrum->sample_spectrum(
                zero<SurfaceInteraction3f>(),
                math::sample_shifted<Wavelength>
                (wavelength_sample)
            );
    }


    Ray3f ray;
    ray.time = time;
    ray.wavelengths = wavelengths;

    // Compute the sample position on the near plane
    // (local camera space).
    Point3f near_p = m_sample_to_camera *
                        Point3f(position_sample.x(),
                        position_sample.y(), 0.f);


    // Aperture position
    Float focalSample = aperture_sample.x();
```

```
        Point3f aperture_p = Point3f(0);

        // Calculate the ray starting point on the aperture
        // plane,  by sampling the focal spot distribution
        UInt32 index =
            enoki::binary_search(0, 12296 - 1, [&](UInt32 index){
                return gather<Float32>(summed_intensities, index)
                    < focalSample;
                }
            );

        Float y = (Float) gather<Float32>(y_values, index);
        Float z = (Float) gather<Float32>(z_values, index);

        aperture_p = Point3f(y * m_scale, z * m_scale, 0.f);


        // Sampled position on the focal plane
        Point3f focus_p = near_p * (m_focus_distance / near_p.z());

        // Convert into a normalized ray direction;
        // adjust the ray interval accordingly.
        Vector3f d  = normalize(Vector3f(focus_p - aperture_p));
        Float inv_z = rcp(d.z());
        ray.mint    = m_near_clip * inv_z;
        ray.maxt    = m_far_clip * inv_z;

        // Update the ray origin and direction,
        // with the world transformation of the camera
        auto trafo = m_world_transform->eval(ray.time, active);
        ray.o      = trafo.transform_affine(aperture_p);
        ray.d      = trafo * d;
        ray.update();

        return std::make_pair(ray, wav_weight);
    }
```



(a) Spectrum of an X-ray source using kilo electron volt as unit for electromagnetic radiation



(b) Spectrum of an X-ray source, converted to use nanometers as unit for electromagnetic radiation

Figure 4.4: Comparison of spectral distribution of an X-ray source, if units of length (nanometers) are used instead of units of energy (kilo electron volts).

CHAPTER 5

Mitsuba 2 Learnings

While using Mitsuba 2 to implement this thesis, we encountered several difficulties. Those problems and some additional tricks for using the Mitsuba 2 framework are presented in this chapter.

5.1 Parameter Names

In Mitsuba 2, scene files are typically given in a Json format or directly as a Python dictionary. Using files in Json format, a parameter can be made accessible to the Python API by naming the parameter inside the scene file with a $ sign followed by the name. The Mitsuba 2 framework will replace every occurrence of *$parameter_name* in the scene file with the value that was used for the parameter. Naming one parameter such that its name contains the name of another parameter as a substring at the beginning of its name will lead to errors or unwanted behaviour. For example, when passing a value for the parameter *object* and another value for a parameter named *object_size*, Mitsuba 2 interprets the string *$object_size* in the scene file as a placeholder for the *object* parameter, and will concatenate the passed value with the remaining string *_size*.

5.2 Double-precision Floats

The Mitsuba 2 framework uses template metaprogramming, as explained in the work by Abrahams and Gurtovoy [AG04], to allow users to select different runtime operation modes. When trying to compile Mitsuba 2 with GPU modes enabled, it will not compile if any double precision floats are present in the code. Usage of the provided float template, which uses either single-precision or double-precision floats, depending on the selected mode, is advised instead.

One has to be especially careful with using define macros to set variables, as these typically do not have an explicit type, and the usage of the provided float template is

not possible for macros. Defining a variable without a trailing f after the value creates double-precision floats.

5.3 Problems with Compiling

The last step in the compile process of Mitsuba 2 is to copy the created files into build/dist. If this step fails, it is most likely not a problem with the code itself, but rather the framework not being able to move the files. This was sometimes the case, when Anaconda was still running in the background, accessing the Mitsuba 2 libraries. Sometimes it was necessary to additionally look at the task manager and close all python processes, to compile successfully.

Another problem was that another program was using some .dlls included inside Mitsuba 2, which also caused the dist copy step to fail. Force closing the programs that used any of those .dlls that are included with Mitsuba 2 fixed this issue. This was most likely caused by adding the Mitsuba 2 path into the environment variables, instead of using the provided scripts to set the environment variables only for the active command prompt.

CHAPTER 6

Evaluation

In this chapter, we evaluate our results and compare the rendered images to reference images.

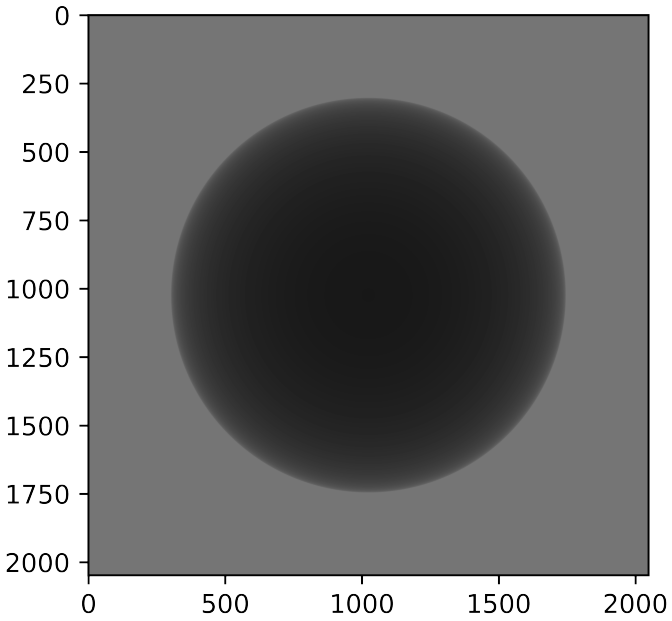


Figure 6.1: Reference image of an Aluminium sphere. Image obtained from the authors of simCT[REK+16]

Reference Images To be able to test our system, we were provided with renderings, spectral distributions and simulation parameters by the authors of SimCT [REK+16]. Figure 6.1 shows a reference image we used to test our system. There were multiple different parameters used in the simulation. The source to object distance (SOD) was 20 mm, while the distance from the source to the detector (SDD) was 250 mm. The sphere between source and detector has a radius of 10 mm, specifying aluminium as material. The used detector distribution was modeled after the PerkinElmer XRD 1620 xN CS flat panel detector, with 2048x2048 pixels at a pixel pitch of 200x200 μm. The X-ray source was specified with an acceleration voltage of 180 kV and current of 90 μA. No prefilter plates were modeled.

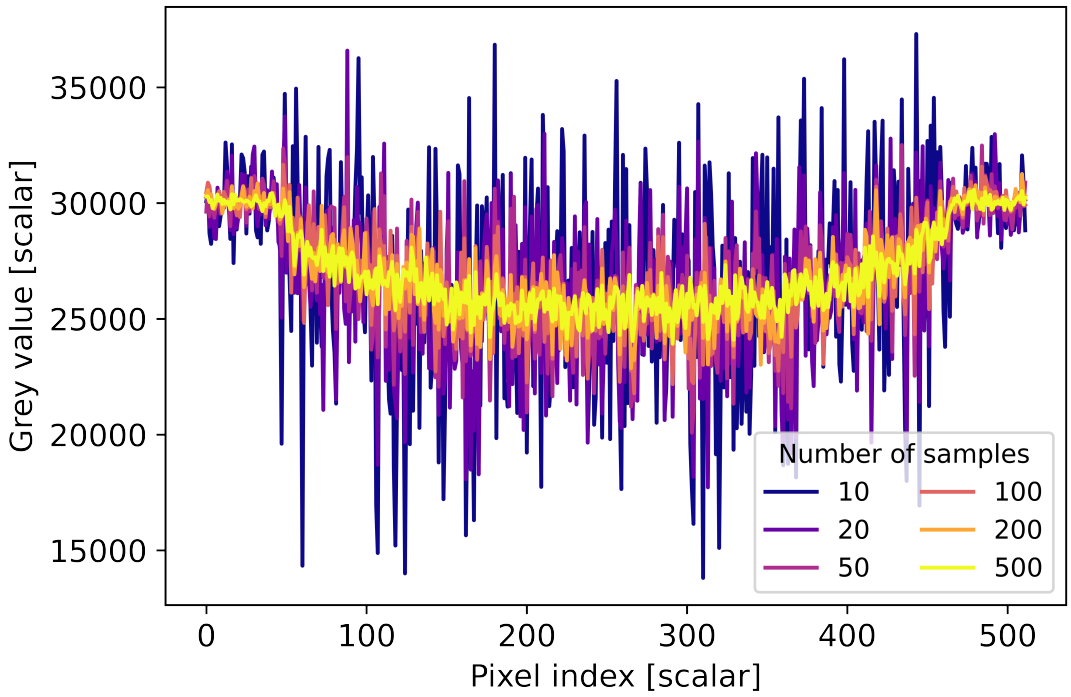


Figure 6.2: Comparison of the convergence of our system if the number of samples per pixel is increased

Biased Reference Comparing the results, we noticed that our obtained values were lower than the values from the reference, around a constant factor of approximately 1,1080. We have tested that our system does generate the right values by analytically solving an empty scene. Both the analytically calculated values, and our empty scene, differ from an empty scene rendered in SimCT [REK+16] by approximately this constant factor of 1,1080. Therefore, we assume that our reference images were obtained by a biased renderer, and assume our generated images are correct, if the values are off by

this factor. We could not find the reason why this bias occurred.

6.0.1 Convergence

To test the convergence of our system, we compared it against the reference images, but also against values we analytically determined using the *data_preparation.ipynb* Notebook. This is described in Section 4.7.

Figure 6.2 shows a plot generated from rendering an aluminium sphere with different values for number of samples per pixel. As is seen, increasing the number of samples per pixel decrease the variance.

Figure 6.3 shows a comparison of a single slice through the middle of an image rendered using our system, to a single slice through the middle of a reference image. Our image was rendered with 100 samples per pixel. We see that a comparison is only meaningful if the values are multiplied by the bias value of 1.1080.

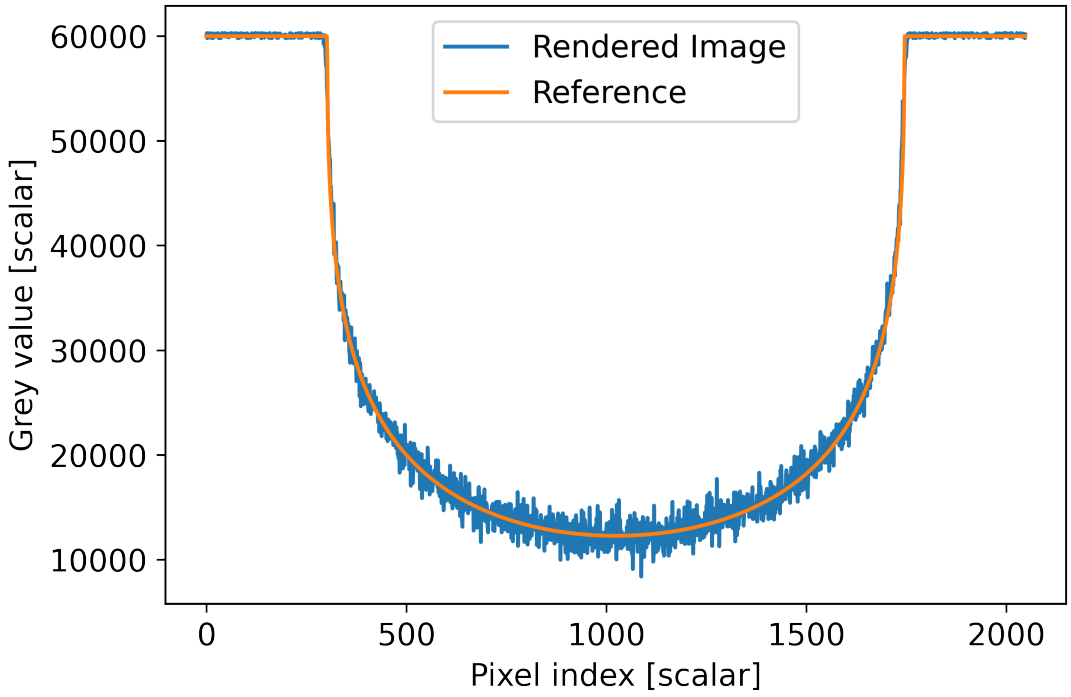
We also calculated the root mean square error, which describes an error in image intensity, comparing the two images mentioned above. In our tested scene, the root mean square error is approximately 33492.7, which is equivalent to 3,67%.

6.0.2 GPU Mode

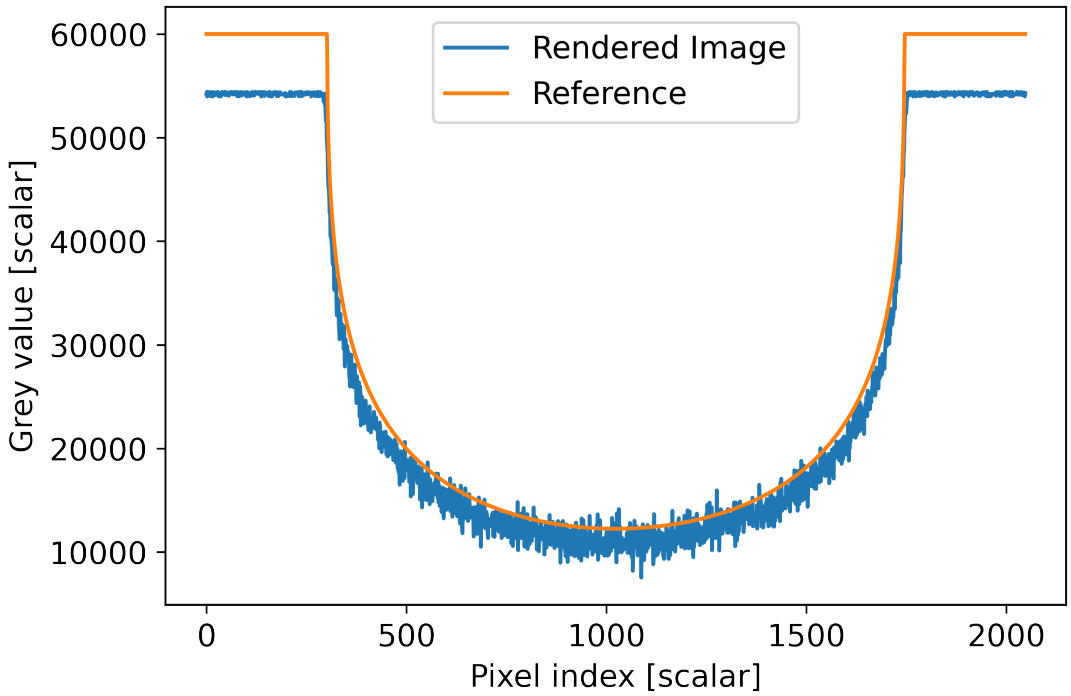
Initially, we wanted to compare the results of rendering images in CPU mode with renderings generated in GPU mode. In theory, the GPU accelerated mode of Mitsuba 2 should allow for much faster rendering speed compared to rendering on the CPU. However, there are still problems with GPU memory consumption in the current Mitsuba 2 version, which only allows us to render at four samples per pixel. Having such a low number of samples per pixel does not allow for a good comparison, as the resulting images would be very noisy. Our system is able to compile in GPU mode and should allow for rendering in GPU mode once the problems with memory consumption get fixed in the original Mitsuba 2 implementation.

6.0.3 Render Time

To test the average rendering time of our system, running Mitsuba 2 in CPU mode, we rendered the same scene 20 times, while varying the number of samples per pixel. The result was an average rendering time of 0.73 seconds per sample used. See Figure 6.4 for the distribution of render time compared to samples per pixel. The approximate linear relation of render time to samples per pixel is not surprising, as increasing the number of samples per pixel by a factor a should also increase the overall computational power required by the same factor.



(a) Rendered Image multiplied with a factor of 1.1080, to allow comparison with the biased reference



(b) Rendered Image without multiplying by a factor of 1.1080, showing the difference to the biased reference

Figure 6.3: Graphs showing the convergence of a rendered image on a slice through the middle of the rendered object.

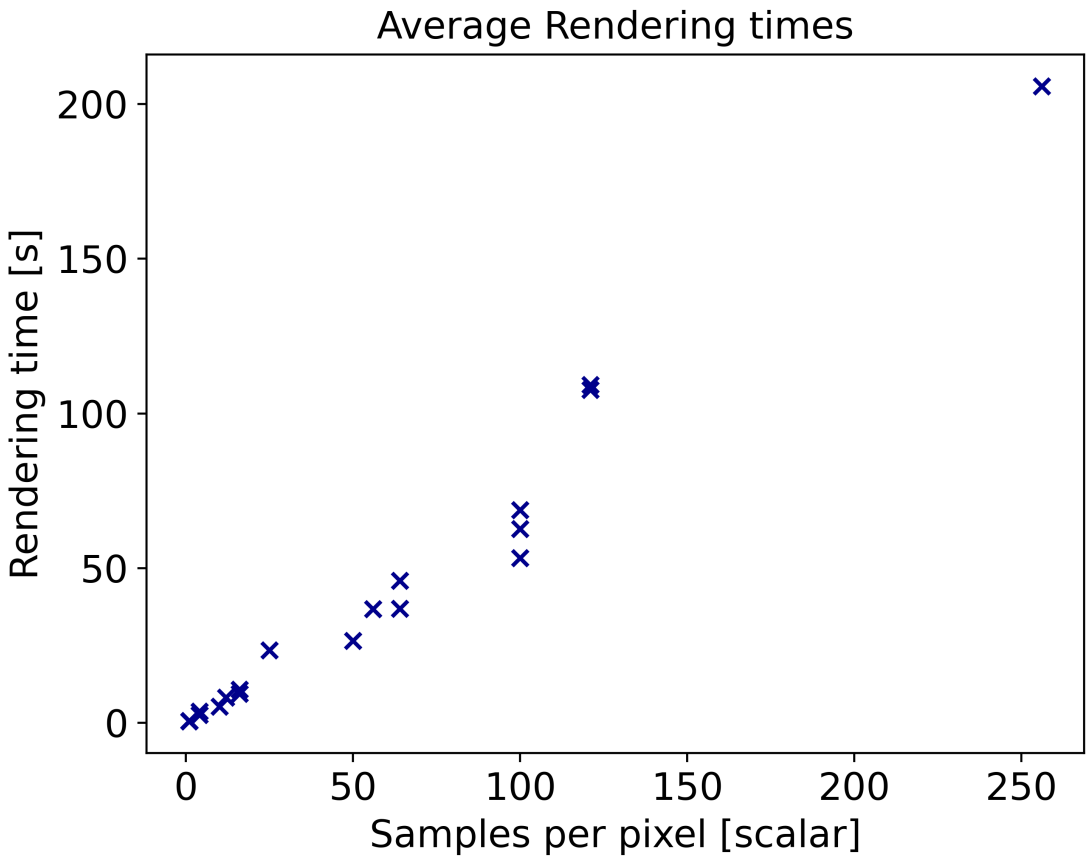


Figure 6.4: Average render time of our system for a resolution of 2048*2048 pixels.

CHAPTER 7

Future Work

One of the big advantages of using Mitsuba 2 is the ability to differentiate the whole rendering process, by using Enokis [Jak19] autodiff framework. Our method relies on this paper, using the accurate description of CT imaging, to reverse the process. Instead of using scenes as an input for the framework and rendering images, we can use images of a real photon detector as an input, and let Mitsuba 2 figure out how the scene was composed.

In this work, we calculated X-ray absorption along a ray by using linear attenuation coefficients. However, we only looked at the absorbing parts of those X-ray matter interactions. If photons are scattered, they also produce other particles that need to be traced. This could be improved in future work.

Mitsuba 2 provides advanced sampling strategies for electromagnetic spectra in the visible range. As we use different kinds of spectra, with much lower wavelengths compared to spectra in the visible range, the Mitsuba 2 framework uses uniform random sampling to obtain wavelength values from the spectra. By implementing sampling strategies optimized for the CT setup, convergence could be improved.

As stated in Section 4.3, we generated our spectra by using data provided to us. This makes it difficult to render a large number of different scenes. To improve our system, it could be better to calculate all necessary data for ourselves as far as possible.

CHAPTER 8

Summary

In this thesis, we presented a system that can simulate a computed tomography scan. A state-of-the-art rendering framework was used, to make the system fast and friendly for future improvements. We compared our results with an existing simulation system, and found out that the used system has some kind of bias. This finding was interesting as the system is already in use, but we could not find a reason why this bias occurred. Furthermore, the problems encountered during the implementation of the systems were summarized, so other people can work with our system and the used framework without hurdles. We mentioned how our system can be improved, especially using automatic differentiation and differentiable rendering. We also identified flaws in our current system, mainly that GPU support is not working at the moment, due to bugs in the used framework.

List of Figures

Bibliography

[AG04] David Abrahams and Aleksey Gurtovoy. *C++ template metaprogramming: concepts, tools, and techniques from Boost and beyond*. Pearson Education, 2004.

[Age19] Nuclear Energy Agency. *PENELOPE 2018: A code system for Monte Carlo simulation of electron and photon transport*. OECD Publishing, 2019.

[BDGJ12] Carsten Bellon, Andreas Deresch, Christian Gollwitzer, and Gerd-Rüdiger Jaenisch. Radiographic simulator artist: version 2. In *Proc. of 18th World Conference on Nondestructive Testing, Durban, South Africa*, 2012.

[BDW81] Frederick O Bartell, Eustace L Dereniak, and William L Wolfe. The theory and measurement of bidirectional reflectance distribution function (brdf) and bidirectional transmittance distribution function (btdf). In *Radiation scattering in optical systems*, volume 257, pages 154–160. SPIE, 1981.

[BJ07] Carsten Bellon and Gerd-Rüdiger Jaenisch. aRTist–analytical RT inspection simulation tool. In *DIR 2007-International Symposium on Digital Industrial Radiology and Computed Tomography (Proceedings)*, pages 1–5, 2007.

[BKUI02] Rika Baba, Yasutaka Konno, Ken Ueda, and Shigeyuki Ikeda. Comparison of flat-panel detector and image-intensifier detector for cone-beam ct. *Computerized medical imaging and graphics*, 26(3):153–158, 2002.

[BM09] Thorsten Buzug and Dimitris Mihailidis. Computed tomography from photon statistics to modern cone-beam CT. *Medical Physics*, 36:3858–, 2009.

[Bur19] John Burgess. Rtx on–the nvidia turing gpu. In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pages 1–27. IEEE, 2019.

[CM11] Angela Cantatore and Pavel Müller. Introduction to computed tomography. *Kgs. Lyngby: DTU Mechanical Engineering*, 2011.

[Cyb21] Cyberphysics. Cyberphysics — electromagnetic spectrum. `https://www.cyberphysics.co.uk/topics/light/emspect.htm`, 2021. (Accessed on 02/15/2021).

[Den15] Pocket Dentistry. Pocket dentistry — 3: Dental x-ray equipment, image receptors and image processing. https://pocketdentistry.com/3-dental-x-ray-equipment-image-receptors-and-image-processing/, 2015. (Accessed on 08/18/2022).

[FCT+12] Roman Fernandez, Marius Costin, David Tisseur, Arnaud Leveque, and Samuel Legoupil. Civa computed tomography modeling. In *proceedings of the World Conference of NDT*, 2012.

[GSM+15] Lucas L Geyer, U Joseph Schoepf, Felix G Meinel, John W Nance Jr, Gorka Bastarrika, Jonathon A Leipsic, Narinder S Paul, Marco Rengo, Andrea Laghi, and Carlo N De Cecco. State of the art: iterative ct reconstruction techniques. *Radiology*, 276(2):339–357, 2015.

[HA20a] Philipp Hochhauser and Thomas Auzinger. ct-path-tracing/ct-simulation: Scripts that simulate ct scans using x-ray-based path tracing. https://github.com/ct-path-tracing/ct-simulation, 2020. (Accessed on 02/23/2022).

[HA20b] Philipp Hochhauser and Thomas Auzinger. ct-path-tracing/mitsuba2 at focalspot_gpu. https://github.com/ct-path-tracing/mitsuba2/tree/focalspot_gpu, 2020. (Accessed on 02/23/2022).

[Jak19] Wenzel Jakob. Enoki: structured vectorization and differentiation on modern processor architectures, 2019. https://github.com/mitsuba-renderer/enoki (Accessed on 03/04/2021).

[Kaj86] James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, 1986.

[Lit19] Max A Little. *Machine learning for signal processing: data science, algorithms, and computational statistics*. Oxford University Press, 2019.

[LW96] Eric P Lafortune and Yves D Willems. Rendering participating media with bidirectional path tracing. In Xavier Pueyo and Peter Schröder, editors, *Rendering Techniques '96*, pages 91–100. Springer Vienna, 1996.

[NDVZ+20] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Wenzel Jakob, Guillaume Loubet, Sébastien Speierer, and Benoît Ruiz. Shapes — mitsuba2 0.1.dev0 documentation. https://mitsuba2.readthedocs.io/en/latest/generated/plugins.html, 2020. (Accessed on 03/04/2021).

[NDVZJ19] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on Graphics (TOG)*, 38(6):1–17, 2019.

[Ope22] OpenSCAD. Openscad — the programmers solid 3d cad modeller. `https://openscad.org/index.html`, 2022. (Accessed on 02/25/2022).

[PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (3rd ed.)*. Morgan Kaufmann Publishers Inc., 3rd edition, 2016.

[REK+16] Michael Reiter, Marco Erler, Christoph Kuhn, Christian Gusenbauer, and Johann Kastner. SimCT: a simulation tool for x-ray imaging. In *Proceedings of the 6th Conference on Industrial Computed Tomography*, 2016.

[RHS+10] Michael Reiter, Christoph Heinzl, Dietmar Salaberger, Daniel Weiss, and Johann Kastner. Study on parameter variation of an industrial computed tomography simulation tool concerning dimensional measurement deviations. In *10th European Conference on Non-Destructive Testing, Moscow, Russia*, 2010.

[SBG+11] Tom Schoonjans, Antonio Brunetti, Bruno Golosio, Manuel Sanchez del Rio, Vicente Armando Solé, Claudio Ferrero, and Laszlo Vincze. The xraylib library for x-ray–matter interactions. recent developments. *Spectrochimica Acta Part B: Atomic Spectroscopy*, 66(11):776–784, 2011.

[SBL12] Wenjuan Sun, Stephen Brown, and Richard Leach. *An Overview of Industrial X-ray Computed Tomography*. NPL report. National Physical Laboratory, 2012.

[Sch18] Richard Schielein. *Analytische Simulation und Aufnahmeplanung für die industrielle Röntgencomputertomographie*. PhD thesis, Julius-Maximilians-Universitat Würzburg, 2018.

[Vea97] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, 1997.

[Whi80] Turner Whitted. An improved illumination model for shaded display. In *Communications of the ACM*, volume 23, pages 343–349. Association for Computing Machinery, 1980.