TU WIEN Informatics

Differenzierbares Rendern für Computer-Tomographie Rekonstruktion

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Aleksandar Vucenovic
Matrikelnummer 01635282

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Dr.techn. Dipl.-Ing. Thomas Auzinger, PhD
Dr.techn. Dipl.-Ing. Christoph Heinzl, Universität Passau

Wien, 4. September 2022

Aleksandar Vucenovic	Eduard Gröller

TU WIEN Informatics

Differential Rendering for Computed Tomography Reconstruction

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Aleksandar Vucenovic
Registration Number 01635282

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Dr.techn. Dipl.-Ing. Thomas Auzinger, PhD
Dr.techn. Dipl.-Ing. Christoph Heinzl, Universität Passau

Vienna, 4th September, 2022

Aleksandar Vucenovic	Eduard Gröller

Erklärung zur Verfassung der Arbeit

Aleksandar Vucenovic

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, 4. September 2022

Aleksandar Vucenovic

Danksagung

An dieser Stelle möchte ich mich bei Thomas Auzinger und Christoph Heinzl bedanken, die mich während des Verfassens dieser Arbeit dabei unterstützt haben und mir bei Blockaden die richtige Richtung und wertvolle Hinweise für das Überhürden dieser gegeben haben. Weiters gilt ein großes Danke an meine Familie, die mich stets während meines Studiums emotional und finanziell unterstützt hat.

Acknowledgements

At this point I would like to thank Thomas Auzinger and Christoph Heinzl, who supported me during the writing of this work and gave me the right direction and valuable hints for overcoming blockades. Furthermore, a big thank-you goes to my family, who have always supported me emotionally and financially during my studies.

Kurzfassung

Die kürzlich erreichte Differenzierbarkeit von Path-Tracing-Algorithmen, die der Standard für die Erzeugung fotorealistischer Bilder sind, eröffnet Optimierungsmöglichkeiten für die 3D-Rekonstruktion eines Objekts anhand von Bildern, die durch Röntgenaufnahmen gewonnen wurden. Die Rekonstruktion erfolgt durch „Invertierung der Rendering Pipeline“, was in der Praxis bedeutet, dass 3D-Szenenparameter wie z.B. volumetrische Daten aus 2D-Bildern gewonnen werden. Die Bilder dienen als Referenz für unseren Algorithmus, der die Parameter der Szene so lange optimiert, bis sich das von der gerenderten Szene aufgenommene Bild nur noch minimal vom Referenzbild unterscheidet. In dieser Veröffentlichung stellen wir ein Proof-of-Concept und erste Experimente für das differenzierbare Rendering für die CT-Rekonstruktion vor. Unsere Implementierung ist in der Lage, die Geometrie und das Volumen von Proben erfolgreich zu rekonstruieren, wobei nur Bilder verwendet werden, die von einem software-simulierten Röntgenscan stammen.

Abstract

The recently achieved differentiability of path-tracing algorithms, which are the standard for generating photo-realistic images, opens up optimization possibilities for the 3D-reconstruction of an object using images acquired by X-ray scans. The reconstruction is accomplished by ”inverting the rendering pipeline”, which in practice means obtaining 3D scene parameters, such as volumetric data, from 2D images. The images act as a reference in our algorithm, which is optimizing the scene parameters until the image acquired by our rendered scene minimally differs from the reference image. In this publication, we represent a proof-of-concept and early experiments for differential rendering for CT reconstruction. Our implementation is able to successfully reconstruct the geometry and volume of specimens, using only images acquired from a software-simulated X-ray scan.

Contents

CHAPTER 1

Introduction

X-ray computed tomography (CT) is a widely-used and powerful imaging technique in industrial and medical applications since its emergence [CDL18]. Setting up CT scans physically demands extreme precision and can be very time consuming, especially if working with industrial CT. The parametrization of machines alone can take up to hours, and often needs to be done empirically by professionals who are familiar with the technology and surrounding theory. Considering this and how expensive a CT scan can be, software simulations of CT environments enable a cheap and fast alternative for the parametrization of CT scans in order to produce reliable results. Furthermore, the complex process of computed tomography reconstruction - using X-ray attenuation measurements for the reconstruction of volumetric data - is constantly being improved by new breakthroughs and research in the fields of physics, graphics, and computer science/engineering.

There have been many different developments in the history of CT reconstructions [LK20] [LWA+15] [FB11] [WN19]. While iterative algorithms [Sti18] were experimented with in the early days of the technology, analytical methods such as filtered-backprojection algorithms (commonly referred to as FBP) [SKT+20] [WN19] are computationally much less expensive. Therefore they are used in clinical environments. The use of iterative algorithms for CT-imaging, e.g., Deep Learning frameworks [WN19] [LK20], has been facilitated due to the constant increase of computational power in recent years achieved by parallel algorithms running on dedicated hardware. While a lot of improvements can be credited to hardware related innovations, performance and accuracy optimizations have also been made through new advancements in data-processing algorithms and software architectures [WN19].

One such novel approach is presented in this paper, which is using the recently achieved differentiability of the path-tracing algorithms [NDVZJ19] [WW22] [LADL18] [GCG+20] [ZDDZ21] to achieve precise results. By inverting the rendering pipeline used to generate images from of a set of input parameters (such as surfaces, cameras, lights, ...), we can use

the attenuation data as provided by X-ray scans to reconstruct those input parameters - namely, the heterogeneous volumetric object being scanned. Our research is a preliminary exploration of the new possibilities arising from this technology.

Chapter 2 will briefly describe related work in the fields of computed tomography simulation, reconstruction and differentiable rendering. In particular, the publications mentioned in the previous section will be summarized in more detail. Furthermore, in Chapter 3, we describe the methodology and theory behind our implementation. Chapter 4 deals with the implementation itself, and in Chapter 5 we present the results. Furthermore, we evaluate the results in Chapter 6, and give some ideas for future work in Chapter 7.

CHAPTER 2

Related Work

To get an idea of the underlying methods and technologies being used to achieve our results, we briefly describe related works in the fields of CT simulation, CT reconstruction, and differentiable rendering. This should give the reader a general idea about different approaches to reconstructing CT scans, while also showcasing possible usages of inverse rendering for heterogeneous volume data reconstruction in different and related fields. Furthermore, a short description of differential render engines, which could potentially be used to achieve our results is presented in this chapter.

2.1 Computed Tomography Simulation

CT simulation can be described as a tool or a set of tools for modelling the process of CT imaging purely via dedicated software. As already mentioned in Chapter 1, software-based simulations provide huge benefits such as a less time-consuming and less complex setup, easier way of parametrization of the scan and easier reproducibility compared to an analogous scan, as the latter requires expertise in setting up a hardware-device and approval of use. To make sense of how a CT simulation works, one must first understand the basic principles of a CT scan and the various elements at play.

A CT scanner usually consists of at least three components: The X-ray source, the specimen, and the detector. The X-ray source is responsible for generating the X-ray spectrum and penetrating the specimen with photons, while the detector is used to acquire how much energy was absorbed by the specimen. The obtained data can further be used for CT imaging/reconstruction. A schematic representation can be seen in Figure 2.1. Software simulations of computed tomography scans usually provide some parameters and control over the modelling of these components. Furthermore, simulations have to deal with the modelling of the physics at play when X-rays interact with matter.

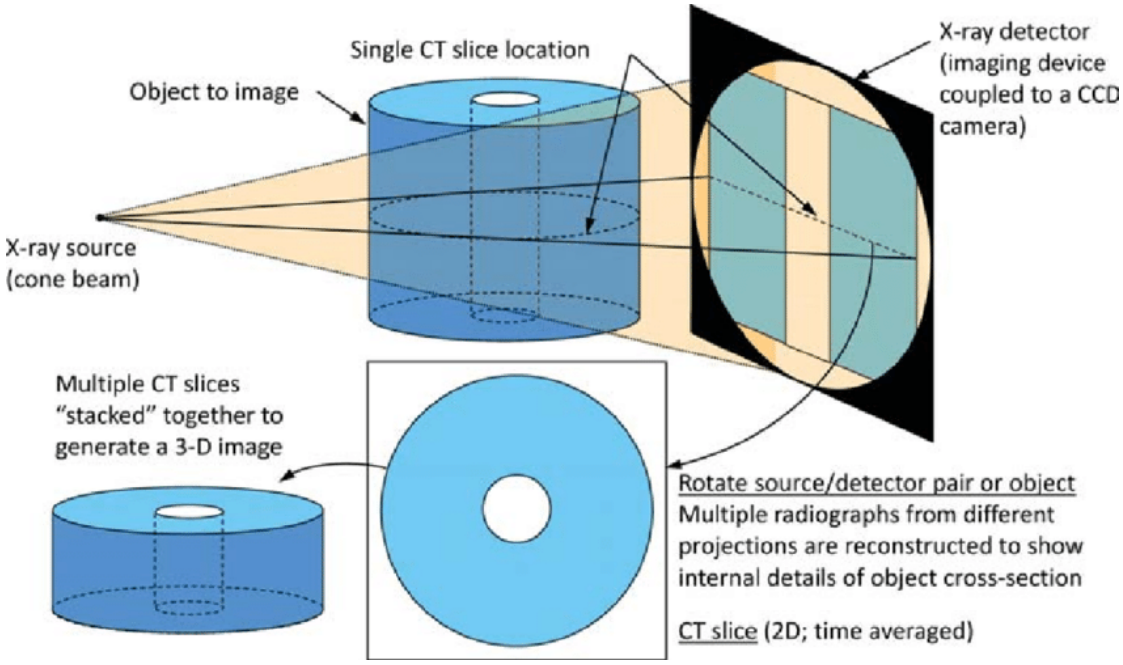


Figure 2.1: A schematic representation of a CT scan, displaying the three components (X-ray source, specimen, detector) and their role in CT imaging. Taken from [Hei11].

Reiter et al. [RHS+10] published the simulation tool SimCT, which features a complete simulation pipeline from analytical X-ray beam generation to the reconstruction of images using an FBP algorithm. The generated X-ray spectra can be parameterized and filtered by plates of arbitrary materials. Furthermore their tool is able to model phenomena such as Rayleigh and Compton scattering, and takes image noise into account. Jaenisch et al. [JBE08] created a similar tool called aRTist.

Agostinelli et al. [AAA+03] propose a more extensive framework for simulating all kinds of particle interactions with matter. Geant4 is a collaboration between an international team of physicists and software engineers, and can even be used to simulate large-scale detectors such as the Large Hadron Collider.

2.2 CT Reconstruction

As already mentioned in the introduction, CT reconstruction algorithms can in general be divided into two typical categories: analytical and iterative. Due to the recent advancements of computational power, iterative approaches are in the center of research, especially with technologies such as Deep Learning emerging. It is also possible to combine steps of both categories in various permutations, and create hybrid algorithms for CT reconstructions. Geyer et al. compiled a report on state-of-the-art iterative techniques in 2015 [GSM+15], which are being used in healthcare facilities around the

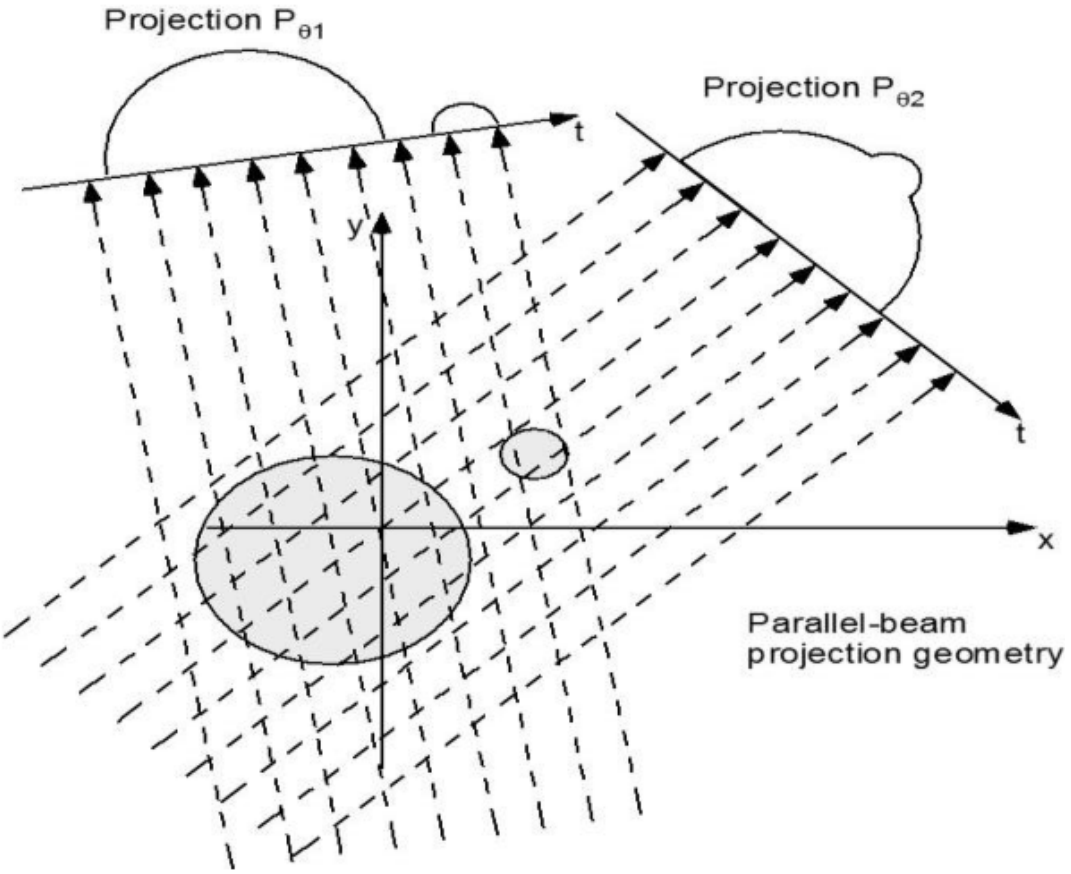


Figure 2.2: Filtered Backprojection: Acquisition of projection data from different angles. Taken from [KH16].

globe. While the report gives a good overview on how integrating iterative reconstruction algorithms has changed and advanced existing systems, we want to focus on more recent research in this chapter.

We will provide some fundamental knowledge about the origins of the algorithms used in recent publications. The most fundamental algorithm, which is still widely used for high-precision CT reconstructions, is called the *Filtered Backprojection Algorithm* or FBP [SKT+20] in short. It refers to the process of transforming acquired data from the projection space, i.e., the detector being used, back to the image space. The filter is used to de-blur the resulting images. A very simplified version of the FBP algorithm could look like this:

1. Shoot parallel beams through an object into a detector. Each point on the detector corresponds to the summation of the absorbed energy along its corresponding beam

(Forward Projection).

2. Repeat Step 1 while rotating the detector and beams at an angle $< 360°$.
3. Transform the resulting detector function into the Fourier Domain (FFT).
4. Multiply the resulting fourier-transformed function with a high-pass filter.
5. Transform the resulting function back into the projection domain (IFFT).
6. Use backprojection to transform the resulting function into the image domain.

A schematic representation can be seen in Figure 2.2.The more angles are used to acquire data, the more precise the backprojected image will be. As homogeneous areas are sampled more densely than sharp edges and details, the resulting image would be very blurry without the filtering steps (Steps 3 to 5). The most simplified example would be to shoot beams through a homogeneous material, meaning that the backprojection algorithm divides the amount of absorbed energy equally along a beam per angle. FBP algorithms can also be used for heterogeneous materials, taking physical phenomena such as beam hardening as well as various scattering effects into account [RHS+10]. Iterative algorithms are still often used in combination with FBP [GSM+15].

Compared to FBP, iterative methods do not perform the transformation of the projection space into the image space in a single step. Analytical algorithms such as FBP are known to introduce artifacts and noise in the resulting image. By approaching the reconstructed image in multiple steps iteratively, these artifacts can be reduced. One of the first uses of an iterative algorithm for CT imaging was in fact one of the first uses of a reconstruction technique for computed tomography ever. Gordon et al. [GBH70] proposed a so-called algebraic reconstruction algorithm back in 1970. It is based on an algorithm used to solve a system of linear equations (Kaczmarz method).

Most publications in the recent years are experimenting with Deep Learning algorithms for CT reconstruction. Lell et al. [LK20] provide a good general overview of how such algorithms can be used to improve medical imaging. In 2022, Khodajou et al. [KCHA22] proposed a framework which can be used to decrease the amount of radiation patients are exposed to, while also speeding up the process of data acquisition using parallel residual conventional neural networks. *Deep Learning Reconstruction* (DLR) has proven to be one the leading techniques for medical imaging because of these two properties of DLR [MCG+21].

Differentiable rendering [KBM+20] allows the use of gradient-descent based algorithms for CT reconstruction purposes, such as the ones used for convolutional neural networks and other Deep Learning architectures. While our research does not make use of such networks, there is definitely a lot of room for future research and advancements in this area. Figure 2.3 shows an overview of the mentioned approaches to CT reconstruction.

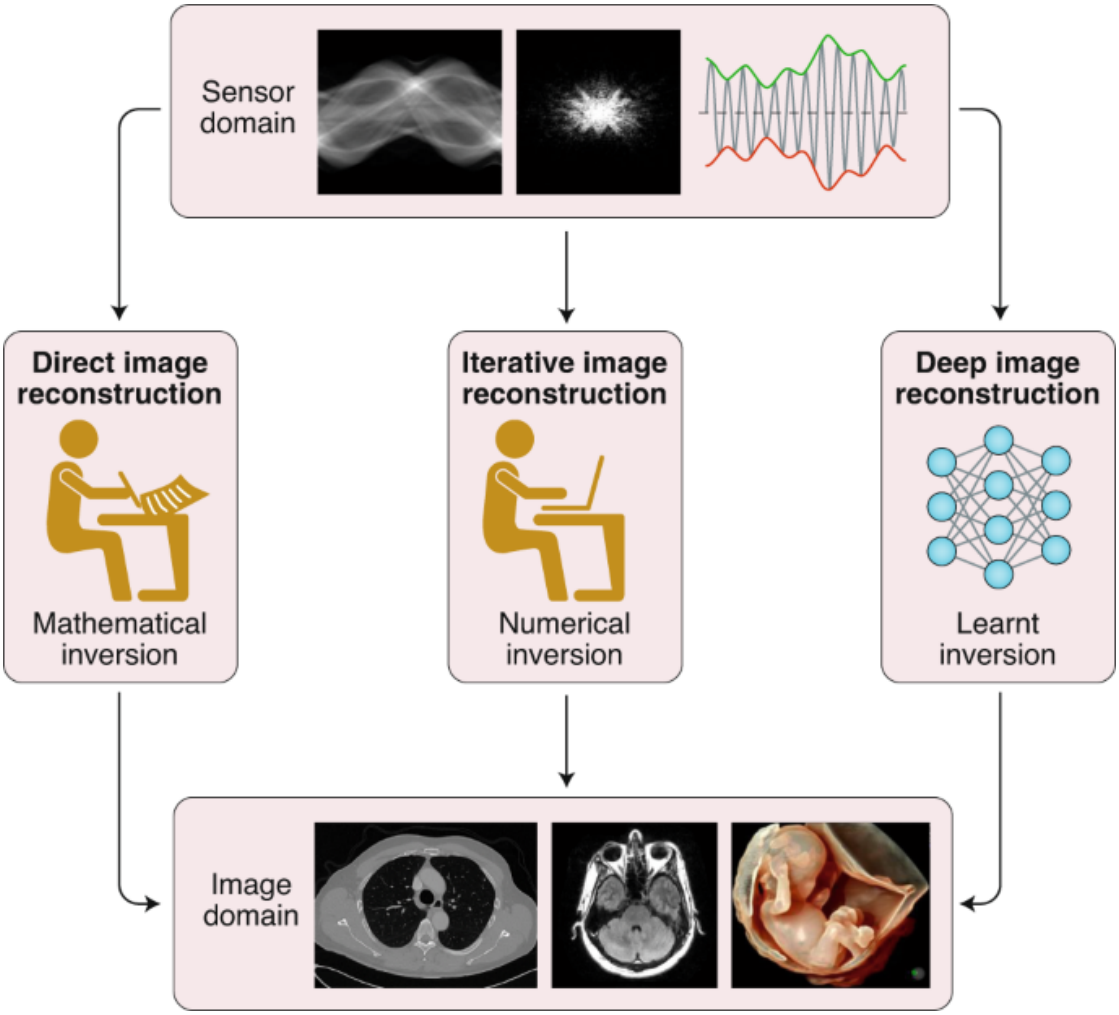


Figure 2.3: Different approaches to reconstruction for CT imaging. Taken from [WYDM20]

2.3 Differentiable Rendering

Physically-based render engines have to deal with a plethora of phenomena regarding the interaction of light and matter. While many algorithms used to model these phenomena are well-suited for *Forward Rendering*, creating a 2D image of a 3D scene, they cannot be differentiated. Inverting this process to create an *Inverse Rendering*, i.e. reconstructing the 3D scene from a 2D image, can be achieved by computing the derivatives of the forward rendering function in respect to the scene parameters. This process is known as *Differentiable Rendering*. The derivatives acquired by differentiable rendering enable the use of optimization algorithms, such as *Gradient Descent Based Algorithms*. Figure 2.4 illustrates an example of a differentiable rendering pipeline. Some algorithms used in a forward rendering pipeline would introduce discontinuities when being differentiated.

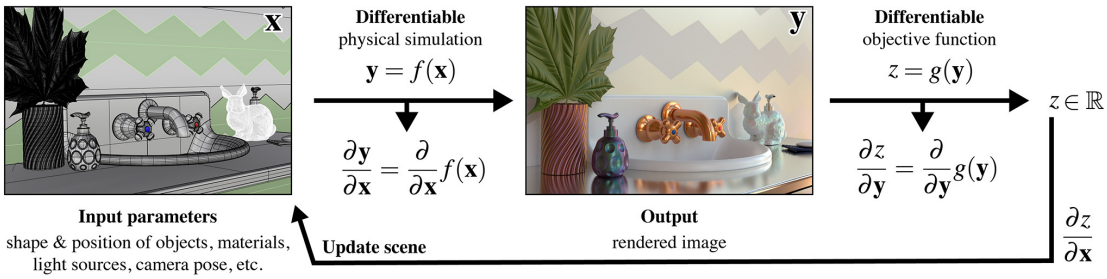


Figure 2.4: Differential rendering pipeline. Taken from [ZJL20].

Therefore, building an inverse rendering engine with the same capabilities as conventional forward rendering engines is a complex task and requires redesigning and -implementing existing systems. In this section we present the differentiable render engine we used for our research, as well as possible alternatives and other use cases.

Nimier-David et al. [NDVZJ19] proposed a render engine that was built for differentiable rendering purposes from the ground up. By using modern programming language features such as template-metaprogramming and support for modularity via a plugin-based design, Mitsuba 2 is extremely versatile when it comes to its use cases and can be easily adjusted and expanded to fit an application's needs. The original implementation offers different modes, which must be defined at compile-time. These properties make it well suited for any research regarding differentiable rendering. Their publication includes information about some of the possible use cases such as differentiable volumetric path tracing, under which category our research is falling and was inspired by.

Also Li et al. [LADL18] introduced a general purpose differentiable renderer a year earlier, which makes use of edge sampling to achieve its capabilities. Their implementation includes a physically-based mode, which can also be used for modelling phenomena caused by the interaction of light and matter. However, their algorithm assumes static scenes with no participating media, so it was not suited for us. Furthermore, the architecture of their proposal is not as easily configurable and adjustable as Mitsuba 2 [NDVZJ19], so it might take more effort to start working with it for more specific applications (such as working in the X-ray spectrum instead of with visibile light).

Weiss et al. [WW22] proposed an algorithm specifically tailored for differentiable volume rendering. It can be applied for the same use cases as the Nimier-David et al. [NDVZJ19] Monte-Carlo path tracing based solution for inverse volume rendering.

CHAPTER 3

Methodology

3.1 X-Ray Spectrum

To be able to penetrate matter without a full loss of energy, the wavelength of electromagnetic rays has to be very short, as the energy carried by photons is inversely proportional to it. The relationship is defined by the formula presented in Equation 3.1, where h is Planck's constant, c is the speed of light, λ is the wavelength and f is the resulting electromagnetic frequency:

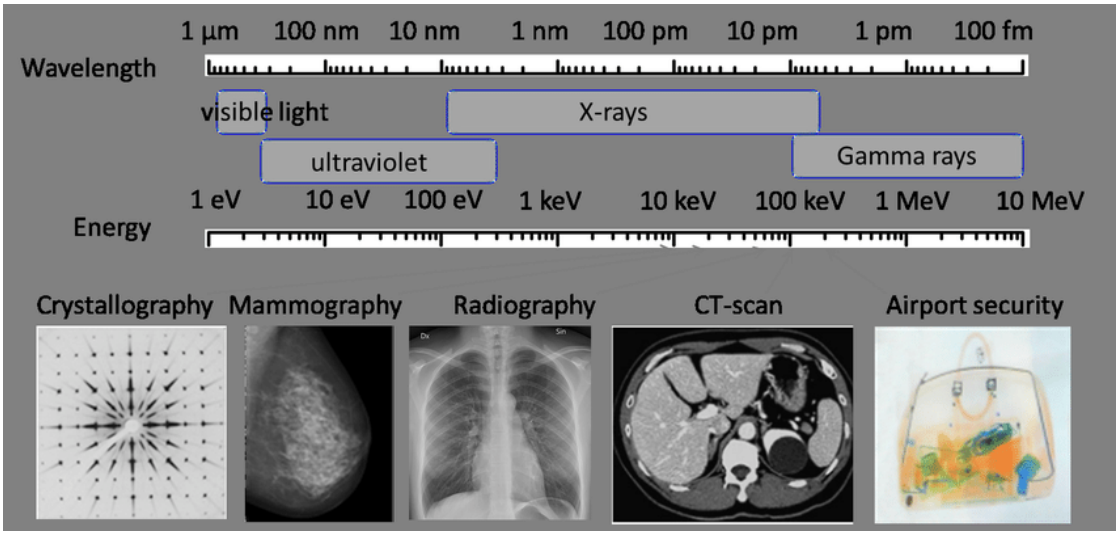


Figure 3.1: Part of the electromagnetic spectrum with wavelength and energy of visible light, ultraviolet light, X-rays and gamma rays. Taken from [Shr18].

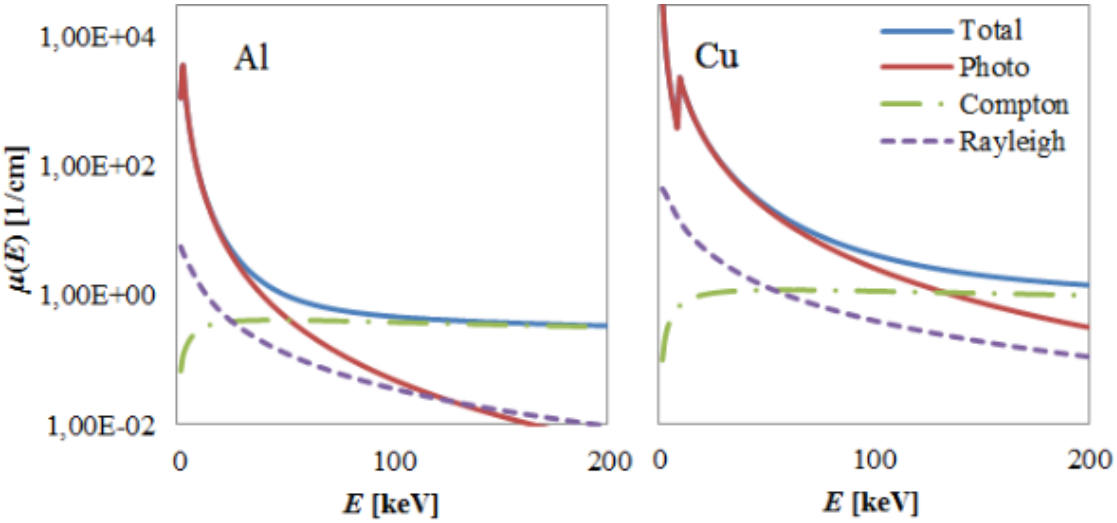


Figure 3.2: Linear attenuation coefficients of aluminium ($Z = 13, \rho = 2.694 g/cm^3$) and copper ($Z = 29, \rho = 8.94 g/cm^3$). Z denotes the atomic number of the element, and ρ denotes its respective density. Taken from [RHS+10].

$$E = hf = h(c/\lambda) \tag{3.1}$$

X-ray spectra usually range between 0 and 500 keV (but can also be higher, e.g. for XXL-CT), and are categorized as *ionizing radiation*, meaning that photons in this spectrum carry enough energy to detach electrons from the matter they are interacting with. The X-ray spectrum can further be divided into *soft X-rays* and *hard X-rays* depending on their wavelength, whereas the latter is used for X-ray and CT imaging purposes. An overview of the spectrum compared to other spectra can be seen in Figure 3.1. Further information about the X-ray Spectrum can be found in the work of Carmignato et al. [CDL18].

There are three different physical phenomena that describe the interaction of X-rays with matter. Depending on the energy carried by X-rays and the material composition of the object they are being shot at, these phenomena occur in different probabilities. Photoelectric absorption is the dominating effect in X-rays carrying lower energies, while Compton scattering dominates for X-rays with high energies. Furthermore, a material with a high atomic number is more likely to influence the effect of photoelectric absorption, and a material with low atomic number strengthens the influence of Compton scattering. The third phenomena is Rayleigh scattering, and it is most probable at low atomic numbers and lower energies. More about this can be found in the SimCT paper [RHS+10].

3.2 Path Tracing

Path Tracing is the prevalent Monte-Carlo based method for obtaining high-precision, physically-based renderings [PJH16]. In 1986, Kayija et al. [Kaj86] introduced the *rendering equation* (as shown in Equation 3.2, taken from [PJH16]), which describes how much light is being radiated from a particular point p on the surface of an object towards a direction ω_0. The outgoing radiation is the sum of the emitted radiance at that point, denoted as $L_e(p, \omega_0)$, and the incident radiance $L_i(p, \omega_1)$ from all directions on the hemisphere S^2 around p scaled by the BSDF $f(p, \omega_0, \omega_1)$ and a cosine term.

$$L_0(p, \omega_0) = L_e(p, \omega_0) + \int_{S^2} f(p, \omega_0, \omega_1) L_i(p, \omega_1) |\cos \theta_i| d\omega_1 \tag{3.2}$$

Solving this equation turned out to be very computationally expensive as the amount of light a point in space receives is not only influenced by direct illumination, but also from all the reflections that bounce light into its direction. Therefore different approximating algorithms have been introduced to tackle the problem efficiently. Path Tracing algorithms are able to solve the rendering equation along individual rays, and the higher the sampling count of individual rays the more realistic the resulting rendering turns out to be, as it approaches the solution of the rendering equation. Similar techniques include Ray Tracing, Photon Mapping and Metropolis light transport. Path Tracing is also referred to as *Monte-Carlo Ray Tracing*, as it is based on Ray Tracing.

A simplified algorithm would start by shooting a ray at a pixel, for each pixel in the output image. The value of the color of the pixel is first being calculated by checking if the ray is intersecting with an object. If yes, the variable gets populated with the color of the object's material, otherwise it gets populated with a value representing black. Then, a new ray is being shot from the origin of the previous intersection, into a random direction of the intersection's normal hemisphere. This new ray is going through the same steps and shooting new rays recursively. The number of times new rays are being generated should be capped by a maximum depth parameter supplied to the algorithm. The final color of the pixel then depends on the summation of the colors calculated through recursion and their respective BSDFs. The implementation of shooting a new ray from the original ray's intersection with the object depends on the use case. While shooting rays into a random direction might be sufficient for diffuse materials, specular material might need a more sophisticated calculation. There are also various ways of improving the number of rays needed to bounce off from incoming rays' origins, such as *Russian Roulette Algorithms* [PJH16]. As we can not send an infinite number of new rays from new origins to calculate an unbiased result, Path Tracing algorithms need some mechanism to determine the termination of this procedure. Instead of using a maximum depth value, or a certain threshold, Russian Roullette algorithms randomly cancel out paths that are contributing less to the final result with a higher probability than those paths that contribute a lot to the result. The implementation of the BRDF calculation is beyond the scope of this section, it is used to calculate light added by reflectance.

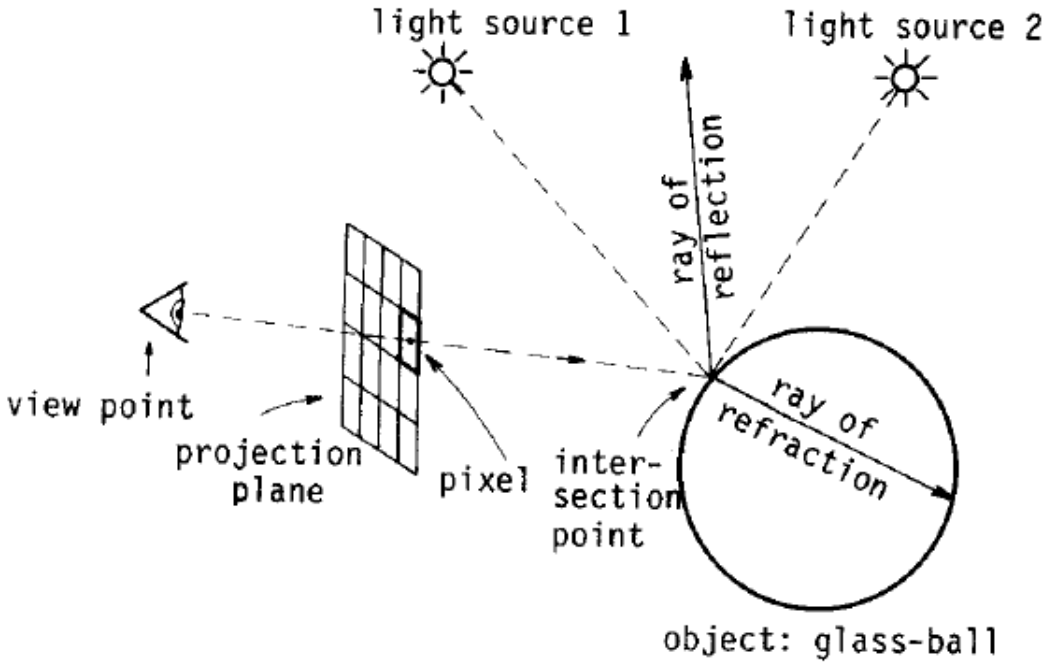


Figure 3.3: A schematic representation of a Ray Tracing Algorithm. For each pixel in the output image, a ray is being shot into the scene, which either intersects with an object or does not. The color of the pixel then depends on the intersecting object's material, the light illuminating it, and rays of reflection/refraction reaching the intersection. Taken from [SML88].

Paths can generally be traced in both directions, from the light source to the camera and vice versa, with both methods having their pros and cons, e.g. backwards-tracing (from camera to light source) has problems with properly calculating bright pixels caused by caustics. This is why a combination of both is used in most modern algorithms, and it is therefore called *Bidirectional Path Tracing.*

More about Path Tracing and Physically Based Rendering Techniques can be found in Pharr et al.'s work [PJH16]. A simple schematic overview of the algorithm is presented in Figure 3.3.

3.3 Participating Media

In the real world, photons are travelling through media such as air or water before they land on surfaces. These media impact the travelling paths of light by scattering photons into different directions. To achieve photo-realistic renderings, the effect of participating media has to be taken into account using path tracing algorithms. It would be computationally very expensive to model the effect of each individual particle (that could interact with light) of a medium through equations. Instead, calculations are made

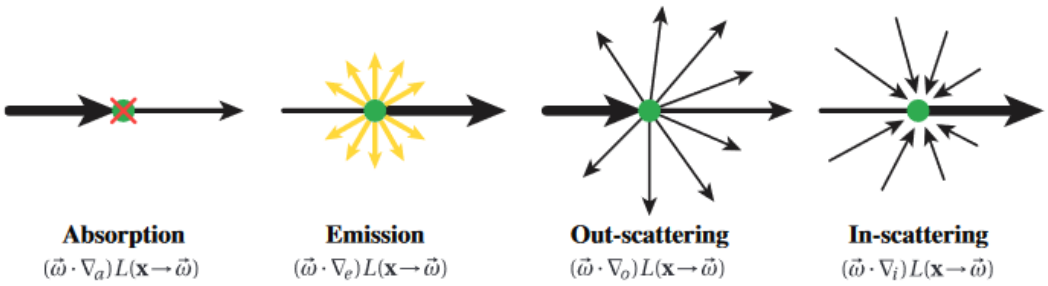


Figure 3.4: An illustration and the underlying equations for the different types of interactions a photon can have with the medium. Taken from [Jar08].

considering the probabilities of light interacting with such a medium. As illustrated in Figure 3.4, interacting with participating media can result in different phenomena. The probability of the energy of a photon being absorbed by the medium is modelled by the *absorption coefficient*, denoted as σ_α, while the probability of a scattering event is denoted as σ_s and called *scattering coefficient*. Summing up these coefficients results in the *extinction coefficient*, $\sigma_\epsilon = \sigma_\alpha + \sigma_s$, which describes how much the incident radiance $L(x \rightarrow \overrightarrow{\omega})$ is being weakened through travelling through the medium.

A more detailed description can be found in the work of Jarosz et al. [Jar08].

3.4 Volume Rendering

Rendering scenes that are partly composed of participating media requires calculations that take the four interaction types (absorption, emission, out-scattering, and in-scattering) into account. Therefore, a volume rendering equation, also called *Radiative Transfer Function* has been proposed by Chandrasekhar et al. [Cha13]. In simple terms, the equation states that the radiance on a path is the summation of the surface, in-scatter and emitted radiance minus the radiance that has been absorbed and out-scattered. Solving this equation amounts to an equally sized body of work as solving the traditional rendering equation. The radiative transfer function is computationally harder to solve, as it incorporates a lot more calculations introduced by the various interactions.

3.5 Differentiable Rendering

Differentiable Rendering is a fairly recent invention used to solve *inverse problems*. Instead of generating a 2D image from a 3D scene, differentiable rendering is used to invert this pipeline and optimize 3D scene parameters from reference images. This is done by computing the gradients of the output image over some scalar loss function in respect to the scene parameters, as shown in Equation 3.3. In the formula, π represents the parameter vector of our scene, L a scalar loss function (e.g., mean squared error of the image and a reference, as used in our research), and $I_i(\pi)$ describes the pixel value

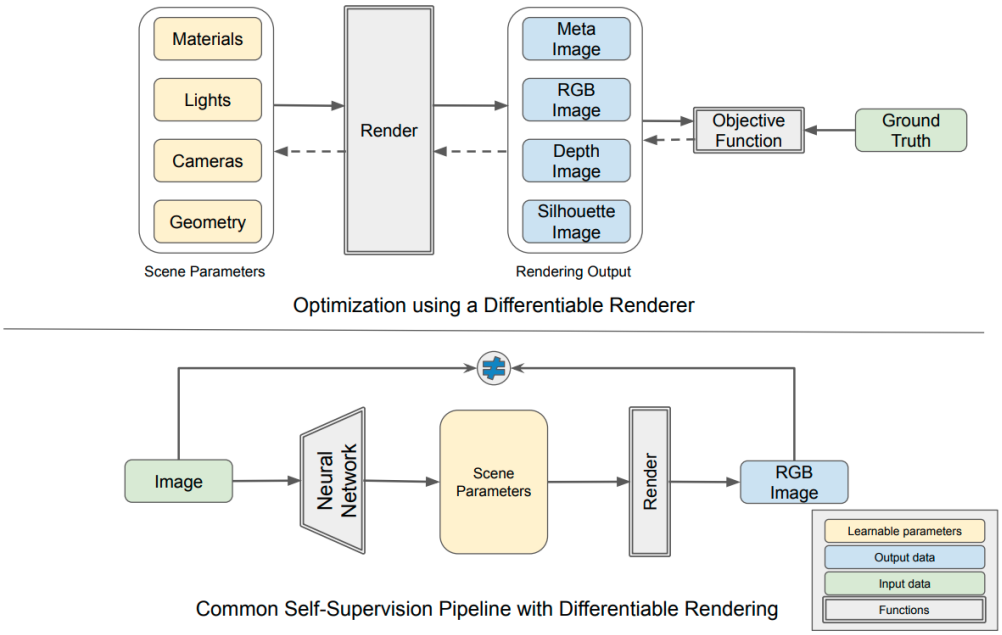


Figure 3.5: Differentiable Rendering pipelines. Taken from [KBM+20].

at position i in respect to our scene parameters. The gradients for the MSE can be calculated as shown in Equation 3.4, whereas $\hat{I}$ denotes the reference image.

$$\frac{\partial}{\partial \pi} L(I(\pi)) = \sum_i \frac{\partial L}{\partial I_i(\pi)} \frac{\partial I_i(\pi)}{\partial \pi} \tag{3.3}$$

$$\sum_i 2(I_i(\pi) - \hat{I}_i(\pi)) \frac{\partial I_i(\pi)}{\partial \pi} \tag{3.4}$$

3D scenes can have different types of representations, e.g., participating media is often described with voxels, while 3D meshes are made of triangulated surfaces. Therefore, different algorithms for inverse rendering are required. Designs of differentiable rendering algorithms have to take each step of the underlying rendering functions into account, as computing gradients in respect to scene parameters means integrating over the whole rendering function. If a step of the complex rendering function is discontinuous when calculating its derivatives, the step has to be overcome by some sort of approximation that produces a continuous function. Approximations can be used in the forward- or backward pass of the rendering pipeline. Furthermore, an image can consist of a high number of pixels and the underlying scene that generated the image can consist of a high number of scene parameters. The resulting gradient matrix easily explodes in regard to how many entries it is holding, producing memory issues in trying to compute the gradients of the render function. Improvements on this issue have been made by Nimier

et al. [NDSRJ20] by using radiative back-propagation. A reasonable initialization of the to-be-optimized parameter can further help to accelerate the performance and precision of the algorithm. Possible architectures of a differentiable rendering pipeline are illustrated in Figure 3.5. A more extensive description and mathematical insights can be found in the work of Zhao et al. [ZJL20].

3.6 Differentiable Rendering for CT Reconstruction

The task to reconstruct volumetric data of a given object via CT imaging can be formulated as an inverse rendering problem. Given a scene with a detector, volumetric object, and camera we can differentiate the resulting render function in respect to the absorption coefficient matrix of the volumetric object (denoted as π_0), as described in Equation 3.5 (as further detailed in the work of Zhao et al. [ZJL20]).

$$\frac{\partial}{\partial \pi_0} \iint f(x, y; \pi)\, dxdy = \frac{\partial}{\partial \pi_0} I_i \tag{3.5}$$

To achieve this, we first generate a reference image (Ground Truth), which is needed to compute a loss-function that can be properly back-propagated. This can be done by either synthesizing the reference image externally with the same camera settings as used in the scene setup, or by forward rendering the scene initially. In practice though, the goal would be to use reference images acquired through measurements obtained from a CT scan. In the next step, we initialize the absorption coefficient parameter of our volumetric object with a matrix holding constant entries (such as 0). We then iteratively render the new scene, and compare the resulting image with our reference by computing the *Mean Squared Error* using the squared Euclidean distances of the images. The loss function is then being back-propagated to our input parameters, and finally *ADAM* [KB15], a method for stochastic optimization, is used for optimizing the absorption coefficients using the acquired gradients from our image. These steps are being repeated with the newly optimized scene parameters until convergence is reached, meaning that our optimization algorithm does not produce any further changes in regards to the scene parameters. As a single-view reconstruction of a 3-dimensional object is insufficient for capturing the volumetric properties correctly, we proceed to rotate the X-ray source and detector around our specimen, and initialize our absorption coefficient matrix with the resulting values from the previous optimization loop, and finally repeat the optimization steps from a different angle.

CHAPTER 4

Implementation

We implemented our application using the Mitsuba 2 [NDVZJ19] framework. However, our results should be obtainable with any differential rendering engine, provided that the exact same parameters are used. The underlying rendering and optimization implementation with another rendering engine could differ in terms of precision and performance. Furthermore, our scene setup and X-ray rendering is based on the work of Hochhauser et al. [Hoc22]. As our scene setup slightly differs as it was expanded for rendering participating media, we will still still discuss it in more detail.

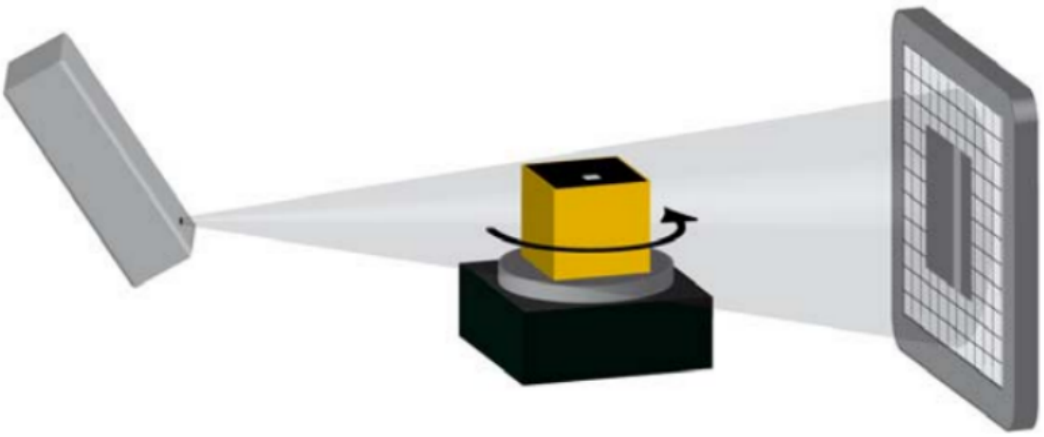


Figure 4.1: Usual CT-simulation scene setup. The X-rays are emitted through the use of a cone beam, the object is being rotated on a table and the detector is capturing the resulting energy on its flat surface. Taken from [RHS+10].

4.1 Scene Setup

A CT simulation scene usually consists of at least these three elements: X-ray source (emitter), object, and detector (projection plane). Our implementation follows this model by making use of a reciprocal setup. While the camera in most scenes is typically used similar to a detector, it is acting as our X-ray source. The detector we use is implemented as a rectangular area light. The rendered image is then generated from the data on the detector. As our rays are following the rules of *Helmholtz reciprocity* [VH67], it does not matter if we follow the rays from the camera or the detector.

4.1.1 Scene parameters

Table 4.1 describes the parameters we used in our setup for rendering a scene. All additional values that are required when computing transform matrices (e.g., for rotating the camera around the object origin) can be derived from this table, e.g., the object distance from the origin is simply SDD - SOD.

Param	Value[m]	Description
SOD	0.02	Object distance from origin
SDD	0.25	Camera distance from origin
FD	0.25	Focus distance of camera
FC	$0.25 + 10^{-5}$	Far clip of camera
DS	0.4096×0.5	Detector size/scale
OS	0.1	Object size/scale

Table 4.1: Scene Parameters used for rendering the scene.

4.1.2 Medium parameters

The material of our medium is described by its absorption coefficients per voxel. As Mitsuba 2 only accepts values between 0 and 1 for this parameter, we had to pass a scale to our material for setting up the scene, which Mitsuba 2 then reapplies before rendering.

4.2 X-ray Spectrum

For the generation of the X-ray spectrum we used Mitsuba 2's internal implementation and data acquired by Hochhauser et al. [Hoc22], which relied on data from the authors of SimCT [RHS+10].

4.3 Detector

The detector implementation is also taken from Hochhauser et al. [Hoc22]. The central idea is to use Path Tracing for generating a spectral response for each pixel on the camera,

and furthermore converting this data to a grey-scale image. As generating images in the X-ray spectrum is a central part of the work of Hochhauser et al. [Hoc22], more can be read about the implementation and algorithm in the publication.

4.4 Reference Creation

To create our participating media, we created a file describing our object as a 16x16x16 voxel grid, with a fixed value of 10^{-4} assigned to each voxel containing object data, which is corresponding to its absorption coefficient. A value of one would mean fully absorbing the energy at this voxel, zero means that no energy is absorbed at this voxel. The steps we took to generate a voxel-based representation from an *.obj* file are as follows:

1. Generate a bounding box fully encapsulating the object around its outer edges.
2. Subdivide this bounding box into 16 same-sized boxes, creating a 16x16x16 grid.
3. For each box in the grid (corresponding to a voxel), check if surface data is present.
4. If surface data is present, assign a non-zero value to this voxel, otherwise assign zero.

We automated this process using the *binVox* 3D mesh voxelizer [NT03] [Min19]. The resulting binvox file was then converted to Mitsuba 2's *.vol* file format using a conversion script. This volume can then be forward rendered and the rendered image can furthermore be used as a reference image from each captured angle before starting the optimization process.

4.5 Rendering Parameters

Mitsuba 2 has a plethora of options to choose from regarding integrators and samplers. Integrators refer to rendering-techniques, such as Path Tracing. For our implementation only two of the available integrators are relevant: the *volpath* and *volpathmis* integrator. Both are volumetric path tracers, but the *volpathmis* integrator additionally uses a method called *Multiple Importance Sampling* [VG95]. Both of those integrators are well-suited for our application, but the *volpathmis* integrator should in theory produce less noise in the outcoming image, which is why we went with it. The sampler is being queried by the integrator to evaluate on which paths the rays are travelling through the scene. Mitsuba 2 has five different samplers available: *independent*, *stratified*, *multi-jitter*, *orthogonal* and *ldsampler*. For our application we chose the independent Sampler and went with its default sample count of four. As going into more detail about samplers and integrators would go beyond the scope of this thesis, more about them can be read in the official Mitsuba 2 documentation [Mit].

4.6 ADAM Optimization

The ADAM Optimization algorithm we use in our implementation to minimize the loss function was proposed by Kingma et al. [KB15] and is since then being widely used instead of the classical stochastic gradient descent algorithm. The algorithm is especially very useful in computer graphics, as it deals well with noisy and sparse gradients compared to alternative methods. Four hyperparameters can be set when initializing an ADAM optimizer. First, there is the learning rate, which describes how big the amount of change in the updated parameter per epoch is. Choosing this parameter can be a tricky task, as a parameter that is set too low will slow down the process made per optimization step, and a learning rate that is too high might lead to divergence. We experimentally initialized this value with a value of 0.002, which lead to a steep decrease in our loss function. Another set of parameters, $\beta 1$ and $\beta 2$, control the exponential averaging of first and second order gradient moments. The fourth hyperparameter ϵ controls the exponential averaging of second order gradient moments. We initialized these parameters as recommended by the authors. A full overview of our chosen parameters can be seen in Table 4.2.

LR	0.002
$\beta 1$	0.9
$\beta 2$	0.99
ϵ	$1e^{-8}$

Table 4.2: ADAM hyperparameters used for optimizing our scene parameters.

4.7 Reconstruction

The following Algorithm 4.1 outlines our implementation for the reconstruction. We first have to define the number of 45° rotations (N_{rot}), the number of optimization iterations per rotation (N_{iter}) and our scene parameters (π). The scene parameters, including the current rotation, are being used to set up our scene initially. We then create a reference image by forward rendering our scene. Furthermore, we update the scene parameters with a non-absorbing matrix (σ_{update}) in the first iteration. We then start our optimization loop, and update the matrix according to our back-propagated gradients of our loss function. After the loop ends, we use the acquired absorption matrix as starting point for our next optimization/rotation cycle.

Algorithm 4.1: Reconstruction Algorithm

Input: $N_{\text{rot}} \geq 1$, $N_{\text{iter}} \geq 1$, π

for $i \leftarrow 0$ **to** $N_{rot} - 1$ **do**
  $rotation = (i * 45.0) \mod 360$
  $scene \leftarrow loadScene(\pi, rotation)$;
  $image_{\text{ref}} = render(scene)$; // render rerence image
  $\sigma_{\text{ref}} = scene.volume.\sigma.data$
  if $\sigma_{update} == null$ **then**
    $\sigma_{\text{update}} = 0.0 * \sigma_{\text{ref}}.length()$; // initialize non-absorbing volume in first run
  end
  $updateVolumeParameter(scene, \sigma_{\text{update}})$;
  for $j \leftarrow 0$ **to** $N_{iter} - 1$ **do**
    $image = renderDifferential(scene)$;
    $objective = MSE(image, image_{\text{ref}})$ // calculate loss function
    $backPropagate(objective)$;
    $gradientStep()$;
  end
  $\sigma_{\text{update}} = scene.volume.\sigma.data$ // update volume with optimized values
end

CHAPTER 5

Evaluation

5.1 Specifications

For our optimization algorithm, we used the following specifcations:

OS	Windows 10 Pro
GPU	Nvidia RTX 3090
GPU Memory	24 GB
CPU	AMD Ryzen 9 3950X
RAM	32 GB
Differential Renderer	Mitsuba 2 Fork
Mitsuba 2 Base Version	master @f4bc83c68f

Table 5.1: Specifications used for our reconstruction

When starting this thesis, we were using an Nvidia RTX 2060 graphics card instead, but quickly discovered, that Mitsuba 2's autodifferential volume rendering needs a huge amount of GPU memory. This memory issues persisted when we tried to use higher sample counts and resolutions, which is why we had to settle on a resolution of 256x256 pixels and a sample count of ten for all of our renderings.

5.2 Results

In the following section we will showcase our results in the form of diagrams and rendered images. We will compare the reference images with the optimized outcome images after five full camera and sensor rotations around our specimen. Furthermore we will display the efficiency and performance of our algorithm.

5.2.1 Optimized Parameter

Figure 5.1 is showing 3D plots of the optimized absorption coefficient parameter of our volume. After the first iteration, our medium is very cloudy and computing the gradients of a single view was not enough to capture the geometric properties of the volume. Figures 5.2a and 5.2b are showing the front view of the starting position and after five full rotations.

It is observable that our two main goals, approximating the 3D geometry and optimizing the absorption coefficients (which where initialized with a constant value of 0.001 in the reference image), are being reached after a sufficient number of iterations. What can furthermore be observed are the noise clouds around the volume, which are most likely being caused by the resulting noise of our forward rendered reference images. There are also some undesired clouds around the edges, which could be caused by Mitsuba 2's interpolation algorithm, or also by the noise in the reference images.

5.2.2 Renderings

Figure 5.3 is showcasing the reference images of each of the 8 rotation angles (45° steps) in comparison with our optimized output images for each of the angles. Other than very minor differences in the noise profiles of the images, the images look nearly identical. Especially the geometry is captured well by our optimization. Figure 5.4 is showing the differences in the absorption matrix and noise profile with more details.

5.2.3 Loss Function

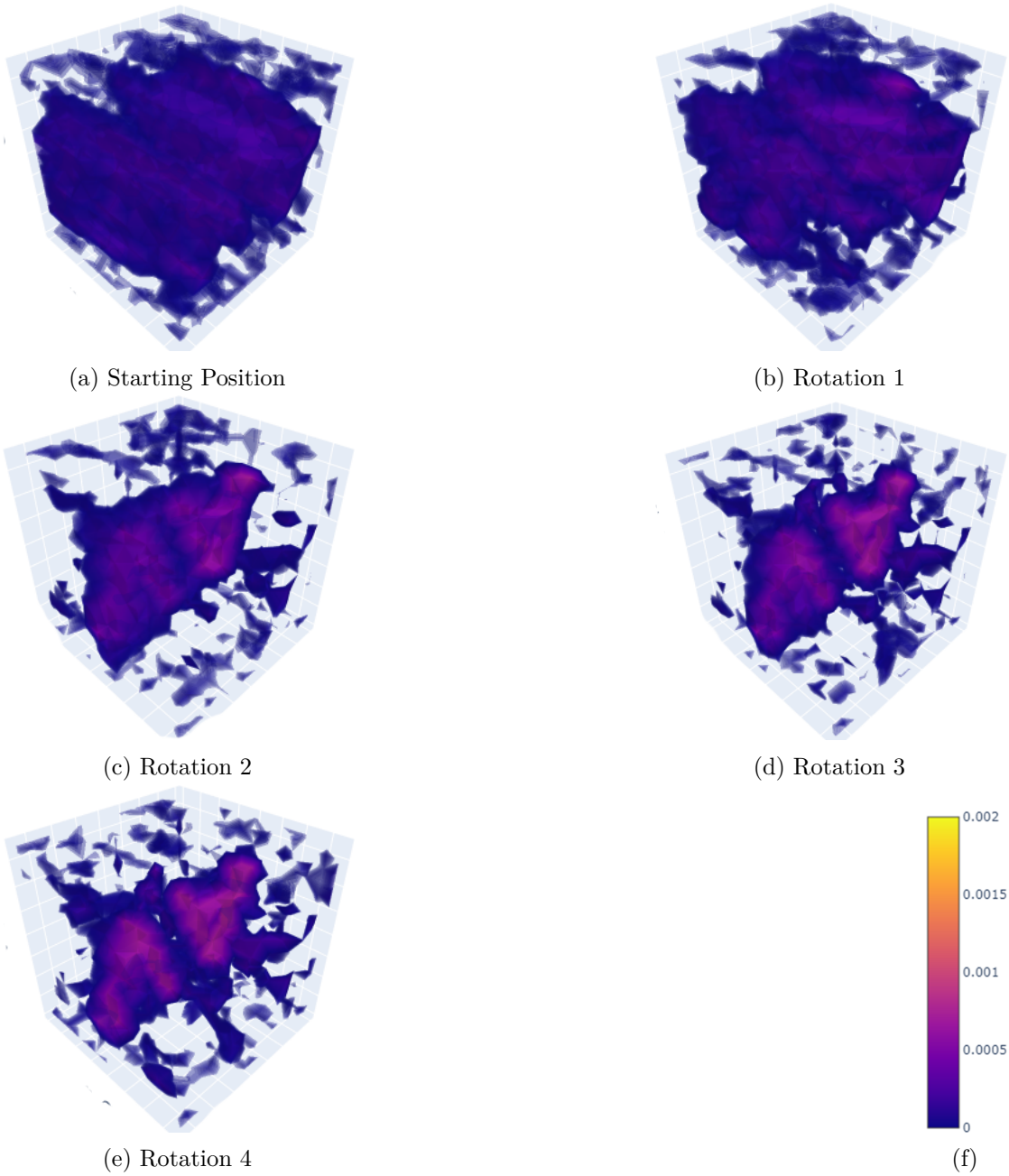
Figure 5.5 is showing the development of the MSE (loss function) during our reconstruction. The first full 360° rotation has descending spikes whenever starting a new 45°-rotation, as the full geometry of our specimen is not known until the completion of a full rotation. After the first full rotation, it can be observed that our loss function is starting to converge around the 0.0010 mark of the MSE. Note that this diagram is omitting runaway values that are extremely small (< 0.0001) and occur only once at the start of each new 45°-rotation after a full 360°-rotation.

5.2.4 Benchmarks

Table 5.2 is showing the maximum, minimum and mean time needed for our various steps in the reconstruction. The total time for reconstruction could be reduced by lowering the number of full 360°-rotations of our camera and sensors around the object, as the data shown in Figure 5.5 is indicating that a lower number could be sufficient for a similar reconstruction outcome.

5.2.5 Memory Usage

We tracked the GPU memory usage of our application with Nvidia NSight Systems. It can be observed that auto-differential renderings take up more memory than usual



(a) Starting Position

(b) Rotation 1

(c) Rotation 2

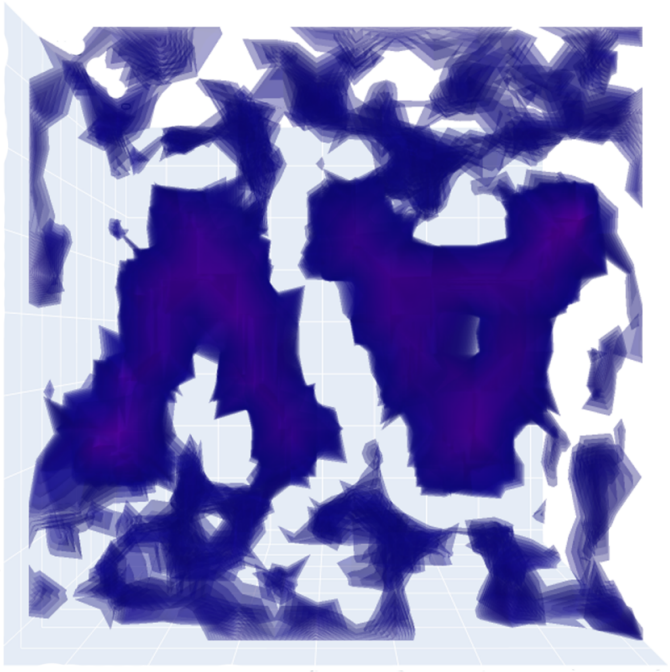
(d) Rotation 3

(e) Rotation 4

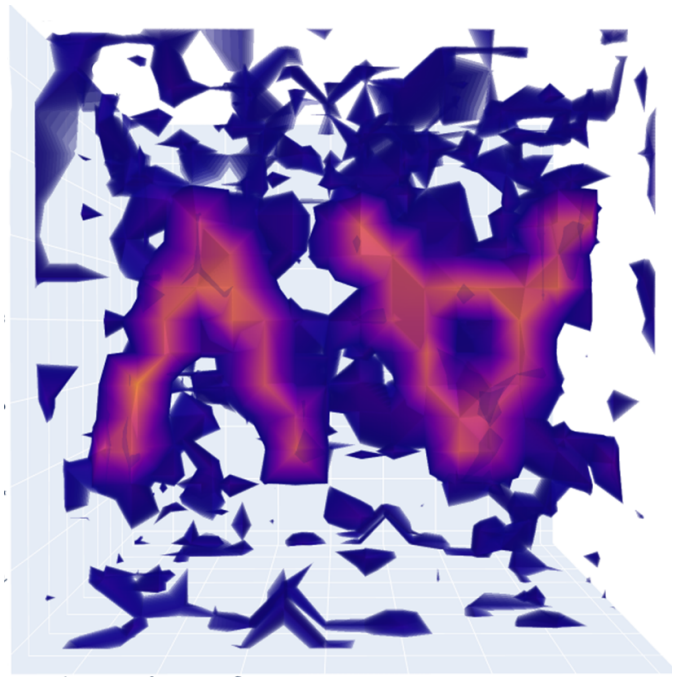
(f)

Figure 5.1: A simple volume rendering visualization of the optimized absorption coefficient matrix. a represents the volume grid after the first optimization iteration and effectively shows how optimizing the volume data from a single view/angle is not sufficient for a correct approximation. The colors of the voxels correspond to the absorption coefficient that was optimized at that voxel, as seen in f

.



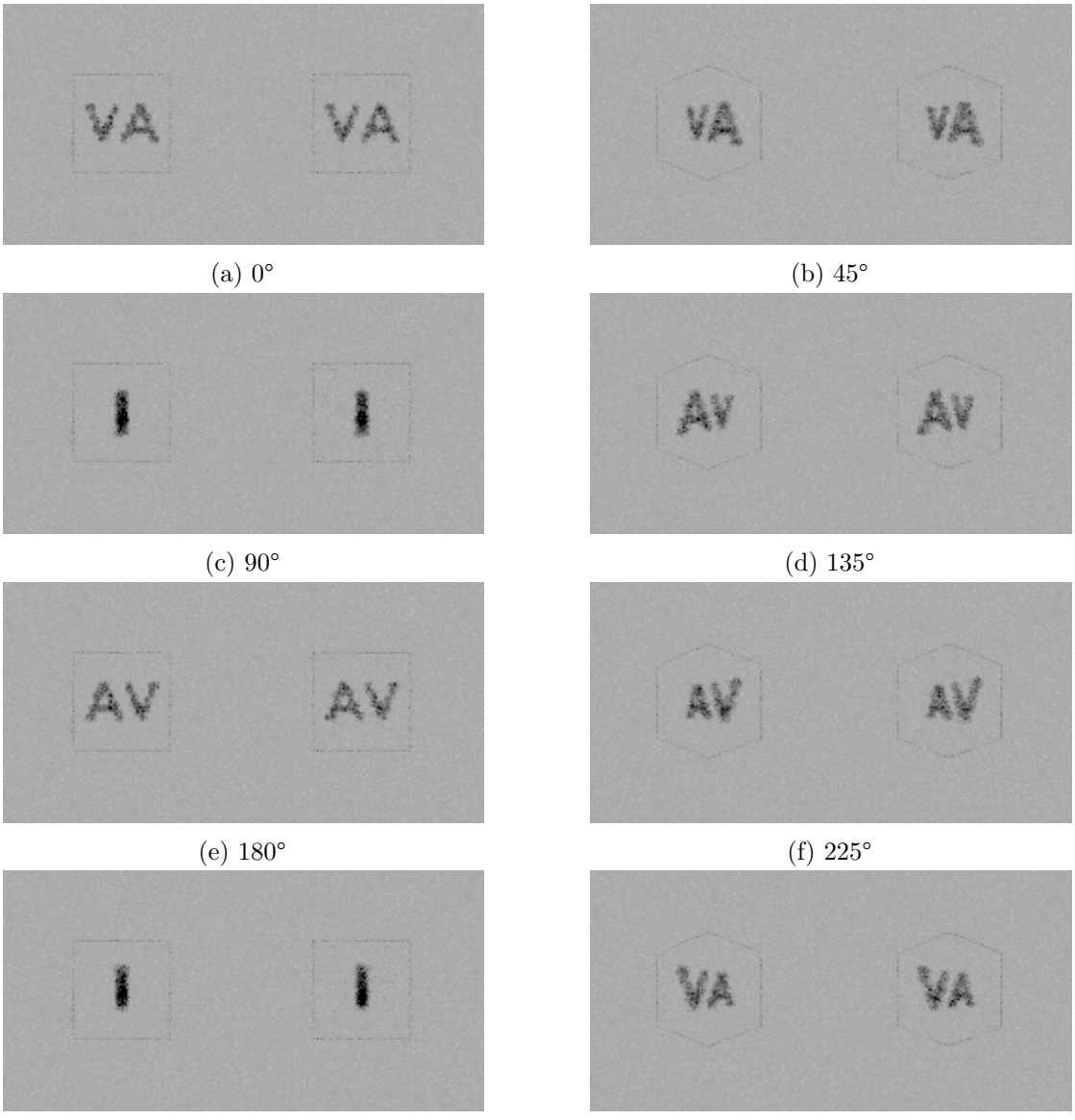
(a) Starting Position



(b) Rotation 40 (45° rotations)

Figure 5.2: Front-view of the first (a) and last (b) optimization iteration.

.



(a) 0° (b) 45°

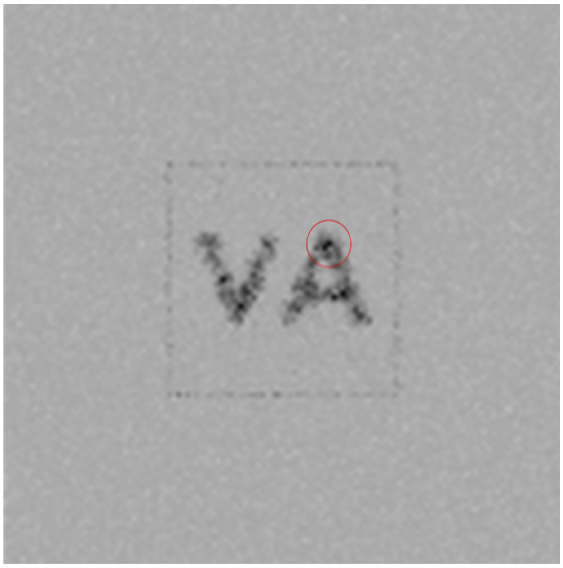
(c) 90° (d) 135°

(e) 180° (f) 225°

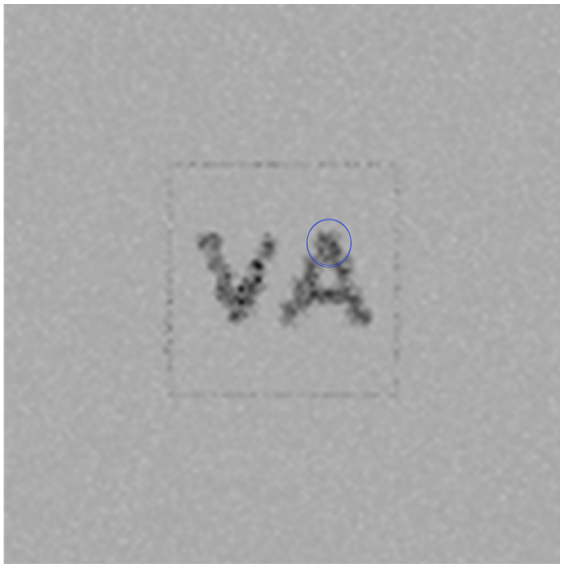
(g) 270° (h) 315°

Figure 5.3: Reference images (left) and optimized output images (right) of all the different angles (45° steps) used in our reconstruction algorithm.

.



(a) Reference



(b) Optimization Output

Figure 5.4: Reference image (a) and optimized output image (b) of the front view (0°) of our specimen. The red and blue circle are highlighting differences in the absorption/noise profile of the two images.

.

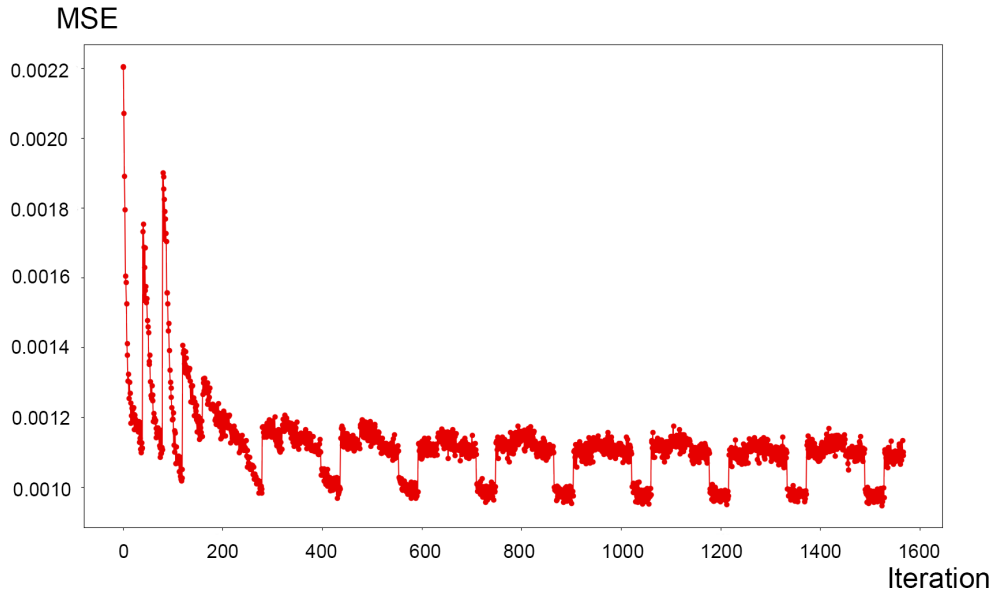


Figure 5.5: Plot of the MSE (loss function) development during our reconstruction. The x-axis is showing each optimization iteration step. As we are optimizing in 40 iterations during one 45°-rotation, and terminate the software after 40 45°-rotations, there are 1600 iterations in total. The spikes during the first full 360°-rotations are a result of wrong geometric assumptions of our optimizer due to not having seen the specimen from each angle yet.

max Optimization	37.693 s
min Optimization	32.706 s
mean Optimization	35.161 s
max 45°-Rotation	25.129 min
min 45°-Rotation	21.804 min
mean 45°-Rotation	23.441 min
Total	**15.627 h**

Table 5.2: Benchmark of our algorithm

forward renderings using the Mitsuba 2-based engine. The memory usage stays constant but almost at a maximum ($> 90\%$) when optimizing. Only four of the 32 CPU kernels where occupied running our reconstruction.

CHAPTER 6

Conclusions

In summary, our implementation of a differentiable rendering algorithm for CT reconstruction produces a reliable proof-of-concept for future work in this area. The acquisition of volumetric data using only 2D images opens a lot of new possibilities in various fields, and differentiable rendering could contribute to the efficiency of iterative CT-reconstruction methods in the future. Compared to a lot of recent research that involves deep learning in the iterative reconstruction pipelines, our algorithm works without any pre-trained models and thanks to the emergence of fully auto-differentiable render engines such as Mitsuba 2, setting up the algorithm is pretty straightforward once a scene is configured.

Shortcomings in our application result partly from Mitsuba 2's high memory usage, which limits our renderings to a low sample count, therefore producing noisy images, making the optimization process less efficient and accurate. The reconstruction presented in this publication is also slow, as it is taking several hours to produce reliable outcomes. Ideas for possible improvements are presented in the next chapter.

CHAPTER 7

Future Work

Future work could improve on several sections of our reconstruction algorithm. To improve the performance of our application, optimizing the hyper-parameter fittings for ADAM as well as determining a sensible determination of the algorithm according to the development of the loss-function convergence could be considered.

Furthermore, a new version of Mitsuba was released [JSRV22], which is supposed to solve a lot of bugs and make further improvements in the engine. Especially the memory issues, which let us produce only images with a small resolution and sample count, are supposed to have been improved by this release.

MSE as a loss function could also be exchanged for a more sophisticated approach, like neural networks. As a lot of research is being developed in the recent years in Deep Learning CT reconstruction, combining the developments of differential rendering for path tracing application and DL-reconstruction could produce good results.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AAA+03] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand, F. Behner, L. Bellagamba, J. Boudreau, L. Broglia, A. Brunengo, H. Burkhardt, S. Chauvie, J. Chuma, R. Chytracek, G. Cooperman, G. Cosmo, P. Degtyarenko, A. Dell'Acqua, G. Depaola, D. Dietrich, R. Enami, A. Feliciello, C. Ferguson, H. Fesefeldt, G. Folger, F. Foppiano, A. Forti, S. Garelli, S. Giani, R. Giannitrapani, D. Gibin, J.J. Gómez Cadenas, I. González, G. Gracia Abril, G. Greeniaus, W. Greiner, V. Grichine, A. Grossheim, S. Guatelli, P. Gumplinger, R. Hamatsu, K. Hashimoto, H. Hasui, A. Heikkinen, A. Howard, V. Ivanchenko, A. Johnson, F.W. Jones, J. Kallenbach, N. Kanaya, M. Kawabata, Y. Kawabata, M. Kawaguti, S. Kelner, P. Kent, A. Kimura, T. Kodama, R. Kokoulin, M. Kossov, H. Kurashige, E. Lamanna, T. Lampén, V. Lara, V. Lefebure, F. Lei, M. Liendl, W. Lockman, F. Longo, S. Magni, M. Maire, E. Medernach, K. Minamimoto, P. Mora de Freitas, Y. Morita, K. Murakami, M. Nagamatu, R. Nartallo, P. Nieminen, T. Nishimura, K. Ohtsubo, M. Okamura, S. O'Neale, Y. Oohata, K. Paech, J. Perl, A. Pfeiffer, M.G. Pia, F. Ranjard, A. Rybin, S. Sadilov, E. Di Salvo, G. Santin, T. Sasaki, N. Savvas, Y. Sawada, S. Scherer, S. Sei, V. Sirotenko, D. Smith, N. Starkov, H. Stoecker, J. Sulkimo, M. Takahata, S. Tanaka, E. Tcherniaev, E. Safai Tehrani, M. Tropeano, P. Truscott, H. Uno, L. Urban, P. Urban, M. Verderi, A. Walkden, W. Wander, H. Weber, J.P. Wellisch, T. Wenaus, D.C. Williams, D. Wright, T. Yamada, H. Yoshida, and D. Zschiesche. Geant4—a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 506(3):250–303, 2003.

[CDL18] Simone Carmignato, Wim Dewulf, and Richard Leach, editors. *Industrial X-Ray Computed Tomography*. Springer International Publishing, 1st edition, 2018.

[Cha13] Subrahmanyan Chandrasekhar. *Radiative transfer*. Courier Corporation, 2013.

[FB11] Dominik Fleischmann and F Edward Boas. Computed tomography—old ideas and new technology. *European radiology*, 21(3):510–517, 2011.

[GBH70] Richard Gordon, Robert Bender, and Gabor T Herman. Algebraic reconstruction techniques (art) for three-dimensional electron microscopy and x-ray photography. *Journal of theoretical Biology*, 29(3):471–481, 1970.

[GCG+20] Purvi Goel, Loudon Cohen, James Guesman, Vikas Thamizharasan, James Tompkin, and Daniel Ritchie. Shape from tracing: Towards reconstructing 3d object geometry and svbrdf material from images via differentiable path tracing. In *2020 International Conference on 3D Vision (3DV)*, pages 1186–1195. IEEE, 2020.

[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 DeCecco. State of the art: Iterative CT reconstruction techniques. *Radiology*, 276(2):339–357, aug 2015.

[Hei11] Theodore J Heindel. A review of x-ray flow visualization with applications to multiphase flows. *Journal of Fluids Engineering*, 133(7), 2011.

[Hoc22] Philip Hochhauser. X-ray Path Tracing for CT Imaging, 2022.

[Jar08] Wojciech Jarosz. *Efficient Monte Carlo methods for light transport in scattering media*. University of California, San Diego, 2008.

[JBE08] Gerd-Rüdiger Jaenisch, Carsten Bellon, and Uwe Ewert. artist–analytical rt inspection simulation tool for industrial application. In *Proceedings of the 17th World Conference on Non-Destructive Testing, Shanghai, China, International Committee on NDT, CDrom paper*, volume 64, 2008.

[JSRV22] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. Dr. jit: a just-in-time compiler for differentiable rendering. *ACM Transactions on Graphics (TOG)*, 41(4):1–19, 2022.

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

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[KBM+20] Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Differentiable Rendering: A Survey. *arXiv e-prints*, page arXiv:2006.12057, June 2020.

[KCHA22] H Khodajou-Chokami, SA Hosseini, and MR Ay. Pars-net: a novel deep learning framework using parallel residual conventional neural networks for sparse-view ct reconstruction. *Journal of Instrumentation*, 17(02):P02011, 2022.

[KH16] Ranjith Kumar and Stephane Hans. Filtered-back projection algorithm on computed tomography (ct) scan. 2016.

[LADL18] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018.

[LK20] Michael M Lell and Marc Kachelrieß. Recent and upcoming technological developments in computed tomography: high speed, low dose, deep learning, multienergy. *Investigative radiology*, 55(1):8–19, 2020.

[LWA+15] Michael M Lell, Joachim E Wildberger, Hatem Alkadhi, John Damilakis, and Marc Kachelriess. Evolution in computed tomography: the battle for speed and dose. *Investigative radiology*, 50(9):629–644, 2015.

[MCG+21] CM McLeavy, MH Chunara, RJ Gravell, A Rauf, A Cushnie, C Staley Talbot, and RM Hawkins. The future of ct: deep learning reconstruction. *Clinical radiology*, 76(6):407–415, 2021.

[Min19] Patrick Min. binvox. `http://www.patrickmin.com/binvox` or `https://www.google.com/search?q=binvox`, 2004 - 2019. Accessed: 2022-09-01.

[Mit] Mitsuba 2 documentation. `https://mitsuba2.readthedocs.io/en/latest/generated/plugins.html#integrators`. Accessed: 2022-09-16.

[NDSRJ20] Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. Radiative backpropagation: an adjoint method for lightning-fast differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(4):146–1, 2020.

[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.

[NT03] Fakir S. Nooruddin and Greg Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205, 2003.

[PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation.* Morgan Kaufmann, 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.

[Shr18] Shreetu Shrestha. *Methylammonium Lead Iodide Perovskite for Direct X-ray Detection*. Friedrich-Alexander-Universitaet Erlangen-Nuernberg (Germany), 2018.

[SKT+20] R Schofield, L King, U Tayal, I Castellano, J Stirrup, F Pontana, James Earls, and E Nicol. Image reconstruction: Part 1–understanding filtered back projection, noise and image acquisition. *Journal of cardiovascular computed tomography*, 14(3):219–225, 2020.

[SML88] Alfred A. Schmitt, Heinrich Müller, and Wolfgang Leister. Ray tracing algorithms — theory and practice. 1988.

[Sti18] Wolfram Stiller. Basics of iterative reconstruction methods in computed tomography: a vendor-independent overview. *European journal of radiology*, 109:147–154, 2018.

[VG95] Eric Veach and Leonidas J Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 419–428, 1995.

[VH67] Hermann Von Helmholtz. *Handbuch der physiologischen Optik: mit 213 in den Text eingedruckten Holzschnitten und 11 Tafeln*, volume 9. Voss, 1867.

[WN19] Martin J Willemink and Peter B Noël. The evolution of image reconstruction for ct—from filtered back projection to artificial intelligence. *European radiology*, 29(5):2185–2195, 2019.

[WW22] Sebastian Weiss and Rüdiger Westermann. Differentiable direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):562–572, 2022.

[WYDM20] Ge Wang, Jong Chul Ye, and Bruno De Man. Deep learning for tomographic image reconstruction. *Nature Machine Intelligence*, 2(12):737–748, 2020.

[ZDDZ21] Cheng Zhang, Zhao Dong, Michael Doggett, and Shuang Zhao. Antithetic sampling for monte carlo differentiable rendering. *ACM Transactions on Graphics (TOG)*, 40(4):1–12, 2021.

[ZJL20] Shuang Zhao, Wenzel Jakob, and Tzu-Mao Li. Physics-based differentiable rendering: from theory to implementation. In *ACM siggraph 2020 courses*, pages 1–30. 2020.