Synthetic X-Ray Image Generation Using FPGA-Based Hardware Acceleration

by

P.J. Aanhane

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday March 7, 2025 at 14:00

Student number:	4644581	
Project duration:	May, 2024 - March, 2025	
Thesis committee:	Dr. ir. C. Strydis	TU Delft, Chair
	Dr. ir. R. Remis	TU Delft, Core
	R. de Jong, BSc.	Philips Medical Systems, supervisor
	Dr. ir. Z. Al-Ars	TU Delft, supervisor

An electronic version of this thesis is available at `https://repository.tudelft.nl/`

TU Delft

Abstract

Synthetic image generation involves the creation of artificially generated images that are indistinguishable from real ones. This field is an answer to challenges in the world of data acquisition, where the need for data is outpacing the availability. In cooperation with Philips Medical Systems, the generation of synthetic X-ray images is studied. Using datasets derived from such images, equipment testing and physician training can be improved. Additionally, training data can be generated for machine learning purposes.

The generation of synthetic X-ray images has been an area of research since at least 1994. The images have traditionally been generated using ray-tracing techniques on CPUs or GPUs. While effective, these methods are computationally expensive and demand high memory bandwidths. More recently, machine learning techniques have been explored for X-ray image generation. These approaches are promising. However, they require large labelled datasets which are often unavailable and the quality of the results is difficult to predict.

The aim of this thesis is to investigate whether hardware acceleration using a field programmable gate array (FPGA) can solve the challenges other methods face. Specifically, it discusses an architecture that can handle the large amount of computations in parallel. The memory architecture required to handle the high bandwidth demands is also explained. The performance of the proposed architecture is studied to see whether it is a viable solution.

By simulating the traversal of rays through a voxelized model, an attenuation map was computed which can be used to determine X-ray intensities on a detector. The design separates computational tasks between a host machine and an FPGA, with an optimized High Bandwidth Memory architecture to maximize data throughput. Results demonstrated that the simulation produced realistic images with minimal error (2.26% - 3.00% deviation from CPU results), and performance is dependant on the detector resolution, achieving frame rates between 123 and 378 frames per second which are well above the goal of 60 frames per second. If more performance is required, upsampling can be used to speed up image generation by 33% at an increased error of 0.6% for an upsampling factor of two. These findings highlight the advantages of FPGA acceleration for deterministic, high-speed synthetic image generation without the need for large labelled datasets as required by machine learning algorithms.

Preface

During my master studies, my interest in computer hardware continued to grow. My study programme was filled with as many performance- and architecture related courses as possible. Outside of my studies, I have worked as a software developer because of my enjoyment of programming. These interests came together perfectly in the research topic for this thesis. Being able to apply them in the field of medical technologies proved to be very satisfactory. Effectively using hardware to accelerate software is something I will definitely be spending more time on in the future.

I consider myself lucky to have been able to work on a topic which remained interesting for the entire nine month period. After the usual start-up problems everybody experiences, as soon as the project gained direction I have been nothing but enthusiastic. This made the months fly by and I am proud of what I was able to achieve.

I am grateful to everyone who supported me throughout my studies and this project. This includes my family, friends, and roommates, who patiently listened to my many discussions on the topic. I also want to thank my fellow TASTI team members for their valuable insights. A special thanks to Melis Umay Tekbas for her support in developing the hardware and to Klaus Juergen Engels for his deep knowledge and helpful guidance on the physical characteristics of X-rays and their simulation. I am also thankful to Per Knops for his assistance in transferring the project and for his openness to all my questions.

I would also like to thank my supervisor Zaid Al-Ars for his supporting role in the project and his helpful feedback. Finally, a special word of gratitude to my supervisor from Philips, Rob de Jong. You have taught me a lot about everything ranging from medical imaging to FPGAs, and I really enjoyed our many off-topic discussions on things like computers and cameras. I could not have done it without you and I look back on this internship with a lot of enjoyment, regardless of the four-hour-long travel days.

P.J. Aanhane
Rotterdam, February 2025

Contents

Nomenclature

List of abbreviations

ASIC Application Specific Integrated Circuit

AXI Advanced eXtensible Interface

BRAM Block Random Access Memory

CLB Configurable Logic Block

CT Computed Tomography

DDR Double Data Rate

DRR Digitally Reconstructed Radiographs

DSP Digital Signal Processing

FIFO First In First Out

FPD Flat Panel Detector

FPGA Field Programmable Gate Array

FPU Floating Point Unit

GAN Generative Adversarial Network

GPU Graphics Processing Units

HBM High Bandwidth Memory

ISO Isocenter

LUTRAM Lookup Table Random Access Memory

PCIe Peripheral Component Interconnect Express

RISC Reduced Instruction Set

SDRAM Synchronous Dynamic Random-Access Memory

SID Source-Image-Distance

TASTI Application-Tailored Synthetic Image Generation

URAM Ultra Random Access Memory

Engine variables

x_{step}/y_{step} Horizontal or vertical distance between projection pixels

x_{inc}/y_{inc} Increment in x or y position between layers

$x_{inc\_step}/y_{inc\_step}$ Change in x or y increment between projection pixels

$x_{inc\_start}/y_{inc\_start}$ Increment in x or y position for the first projection pixel

$x_{inc\_start}/y_{inc\_start}$ Increment in x or y position for the first projection pixel

x_{min}/y_{min} Lowest x or y coordinate of the projected boundary

x_{max}/y_{max} Highest x or y coordinate of the projected boundary

1

Introduction

Synthetic image generation involves the creation of artificially generated images that are indistinguishable from real ones. This field is an answer to challenges in the world of data acquisition for machine learning and artificial intelligence applications. As the demand for high-quality datasets continues to grow, reliance on real data alone proves increasingly impractical. In the healthcare industry especially, gathering real-world data can be unsafe for the patient, unethical, or restrictive due to privacy legislation protecting against sharing sensitive data. Synthetic data offers a good alternative, enabling the usage of safe, high-quality, and privacy compliant datasets that can be used in the development of the smart systems of tomorrow.

This thesis is part of an internship project at the Dutch company Philips Medical Systems. The project is part of TASTI (Application-TAilored SynThetic Image generation) [32], which is a European funded initiative with the goal of developing a modular framework of transferable technology to innovate synthetic image generation tailored towards applications in several industries. Philips has been an industry leading expert in medical imaging systems and has done a lot to innovate the healthcare industry. The interventional X-ray department aims to improve patient healthcare and build systems that can save lives. These systems have strict real-time processing requirements which are often solved using streaming-based processing. At the TU Delft EEMCS Computer Engineering group, a lot of effort has gone into techniques that can meet these demands, ranging from the development of stream-based processing for image processing applications [16] to composable streaming interfaces defined using software [10].

For Philips specifically, the TASTI project is about creating a virtual testing platform using synthetic image generation, for which three different applications have already been identified:

1. **Equipment testing:** Synthetic images can simplify the process of system integration tests. Real tests require the X-ray source, which can be inconvenient. The restriction because of safety, though justified, can also be time consuming. Through elimination of the X-ray source while maintaining the ability to produce an image, system development can be accelerated and quality is maintained.
2. **Physician training:** A virtual environment that simulates the operation of medical equipment accurately allows a physician to gain hands-on experience in a safe setting without relying on harmful radiation sources or involving actual patients. Such a platform ensures risk-free training, making the process both efficient and accessible.
3. **AI data generation:** Computer vision algorithms are employed for the purpose of system validation. They can be used to judge image quality and perform simple medical analysis. Such algorithms require large datasets for training purposes. These can be generated using the virtual testing platform.

1.1. Challenge

The virtual platform currently under development consists of an exam room containing the controls and a technical room containing a system cabinet for simulations. This simulation is CPU-based and operates on a full-body model, also known as a phantom. Currently, it produces about 5 frames per second, which is insufficient for real-time performance.

The bottleneck of the simulation lies in the high memory throughput necessary to produce each frame. Without optimizations, generating a frame using a whole-body phantom requires a memory transfer of approximately 2 GB. To achieve a speed of 60 frames per second, a transfer rate of 120 GB per second would be required. Such speeds are infeasible for regular PC hardware. This estimate does not even consider the huge amount of calculations required for processing this data.

1.2. Research questions

The aim of this Master's thesis is to determine whether computational- and memory throughput challenges existing solutions face can be addressed using hardware acceleration kernels. Specifically, it seeks to investigate whether a custom FPGA architecture can be used for real-time synthetic image generation of X-ray images at the rate of existing detectors. To answer this question, the following sub-questions are posed:

1. What system architecture is needed to support real-time generation of synthetic X-ray images?
2. What memory architecture is required to handle the high data throughput?
3. What is the maximum performance achievable using FPGAs and does it satisfy the necessary application performance?

1.3. Thesis outline

This thesis will start with some background information in Chapter 2. It will give a short introduction on X-ray radiation and imaging systems. It will also explain the FPGA. Next, Chapter 3 will show the steps and components required to simulate an accurate X-ray image. The following chapters talk about the architecture. Chapter 4 explains the design of the system and introduces the architecture. In Chapter 5, the implementation of the components is discussed. This includes both the software and hardware components. How the memory communication is implemented is mentioned in Chapter 6. Finally, Chapter 7 provides an analysis of the simulation results and Chapter 8 will conclude the thesis.

2

Background

In this chapter, the required background knowledge is discussed to provide context into X-ray based imaging systems. In Section 2.1, the physics behind X-rays is summarized and the concept of X-ray imaging systems is introduced. Section 2.2 will provide an explanation on FPGAs, heavily utilised in this thesis. Section 2.3 will cover related work.

2.1. X-rays and imaging systems

The history of medical and industrial X-rays and the evolution of the technology of its sources has been ongoing for about 125 years. The first indirect discovery has been attributed to Julius Pluecker, who reported a greenish fluorescence on a glass wall while studying Geissler discharge tubes in a partial vacuum [8]. This glow is characteristic evidence of electrons hitting a wall and producing X-rays, but the underlying mechanism was not yet known at the time . In 1895, while experimenting with electric current in cathode-ray tubes, Wilhelm Röntgen observed similar effects and theorized that electrons striking the glass tube produced unknown radiation. He called the phenomenon X-radiation because of its uncertain nature and connection to light [13].

X-rays are electromagnetic radiation. Electromagnetic radiation transports energy, also called radiant energy, through space using waves and photons just like radio waves, visible light or microwaves. As with all forms of light, X-rays are characterized by their frequency or wavelength. In literature, most sources define the wavelength range of X-rays to be between 0.01 nanometres and 10 nanometres [31] [21]. This corresponds to an energy range of 100 eV up to 100 keV.

2.1.1. X-ray generation

There are two main ways X-rays are generated. The first is related to their initial discovery. This method, known as Bremsstrahlung derived from the German word for slowing down, involves the interaction of high-speed electrons with the anode of an X-ray tube. Electrons are accelerated by the tube's acceleration voltage, moving them from the negative cathode to the positive anode. When these high-energy electrons collide with the anode material, they are decelerated and deflected by the electric fields of the atoms in the anode. The deceleration produces X-rays with a continuous energy spectrum [21].

Characteristic X-rays are produced when high-energy electrons eject inner-shell electrons from atoms in the anode material, creating vacancies. To stabilize, electrons from outer shells transition into these vacancies, releasing X-ray photons with energies equal to the difference between the two shell levels. These energies are unique to the atomic structure of the target material, resulting in a discrete spectrum with sharp peaks corresponding to specific transitions [21]. When the continuous and discrete spectra are combined, the spectrum shown in Figure 2.1 is produced. An X-ray beam consisting of radiation with multiple energy levels is referred to as polychromatic. When only a single energy level is present, it is called monochromatic.

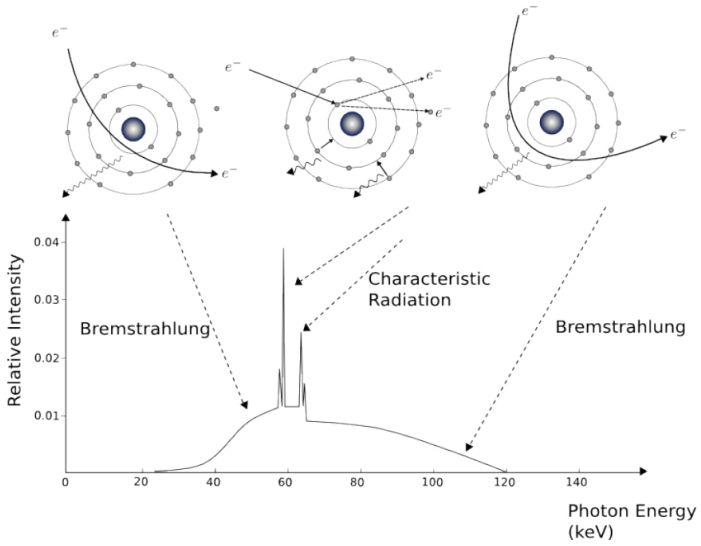


Figure 2.1: Energy spectrum produced by a tungsten tube [21].

2.1.2. Matter interaction

X-rays have the ability to penetrate matter, but the amount is dependent on both the material as well as the X-ray energy. In the energy range that is used for medical imaging, there are three kinds of relevant interactions, described in [21], that can occur when X-rays pass through matter. The first two are physical and occur when X-rays collide with either nucleons or electrons. The third happens when X-rays are influenced by an electric field.

These interactions can lead to radiation absorption, but are also responsible for scattering. Both are responsible for X-ray attenuation. The linear attenuation coefficient μ is a material property and a measure for how much the intensity of an X-ray beam is reduced the further it penetrates matter. The linear attenuation takes the density of a material into account, expressed at $\frac{\mu}{\rho}$. Figure 2.2 shows how the mass attenuation can change depending on the material and the X-ray photon energy. Using the linear attenuation of a material, the Lambert-Beer's law [21] describes the radiation intensity as it passes through a material:

$$I = I_0 e^{-\int_0^l \mu(x)dx} \tag{2.1}$$

where I_0 describes the incident X-ray intensity, l the thickness of the material, and $\mu(x)$ the linear attenuation at x.

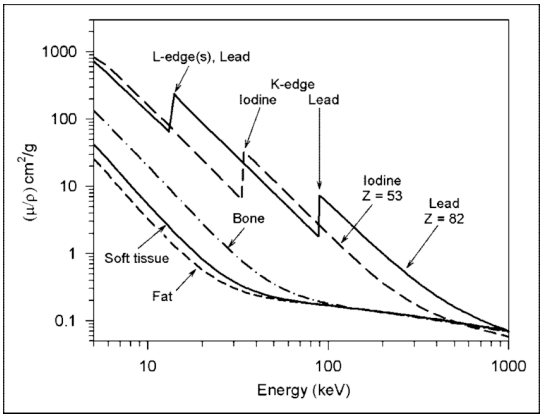


Figure 2.2: Plot which shows the linear attenuation for different materials and different energies [28].

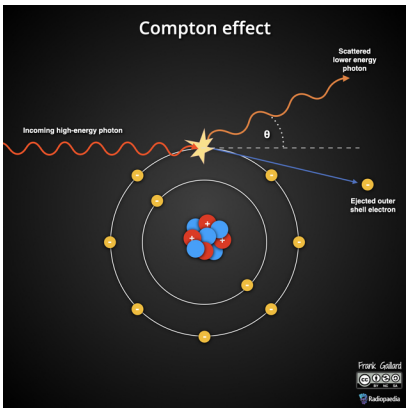


Figure 2.3: Illustration describing the Compton effect, where an incident photon strikes an outer shell electron, scattering the photon [15].

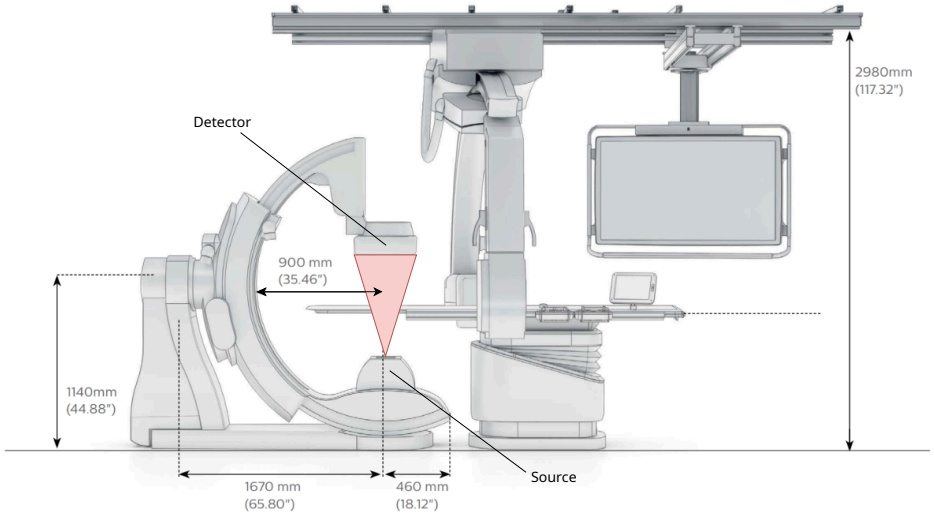


Figure 2.4: Philips Azurion 7 B20/15 bi-plane imaging system showing the location of the source and the detector, adapted from [1].

In the case of scatter, the most prevalent for high-energy radiation is Compton scattering [15]. The cause of this type of scattering is an incident X-ray photon hitting an electron which ejects it from its orbit. The photon is also scattered. The ejected electron is referred to as the recoil electron. The scattered photon will have both a different direction as well as a different energy. Figure 2.3 shows an illustration of the phenomenon.

2.1.3. Imaging systems

Figure 2.4 shows an interventional X-Ray system by Philips with its most important components. The process begins at the X-ray tube, also known as the source. The source produces the radiation which moves through the subject. In medical imaging, low energy X-rays are removed using a thin metal plate as a filter. This is because the low energy rays are largely absorbed by a patient, resulting in a higher patient dose without improving the image quality. The type of beam is referred to as a cone beam, due to its outward portraying direction [21].

In modern systems, the flat panel detector (FPD) has become the standard. FPDs can either directly or indirectly capture the radiation. Direct conversion FPDs contain an X-ray sensitive photoconductor which detects the presence of radiation. Indirect conversion FPDs contain a scintillating layer which first converts X-ray photons into visible light. The detectors used by Philips have indirect conversion FPDs. The detector is covered by an anti-scatter grid, which limit the degrading effects of this phenomenon. A diagram of the detector used in Philips systems can be seen in Figure 2.5. Note the scintillating layer, shown in blue, that covers the detector matrix.

2.2. FPGA

An FPGA is a type of integrated circuit which is designed to be reconfigurable. Unlike traditional logic devices such as application-specific integrated circuits (ASICs), the connections between logic gates and components on an FPGA can be modified after the chips have been produced. This ability makes FPGAs suitable devices for prototyping high-performance applications.

Reconfiguration is achieved through an array of fixed configurable logic blocks (CLBs) and flexible interconnects that can be configured to perform either complex operations or to serve as simple logic gates. The structure is shown in Figure 2.6. FPGAs can also include static hardware ranging from dense memory arrays to dedicated hardware multipliers which can be incorporated in a design through the interconnect [17]. FPGAs are useful for prototyping high-performance applications and can support most designs, provided it has the required resources available. Highly parallel parts of an application can efficiently be implemented with custom logic. This is different from a conventional CPU where one

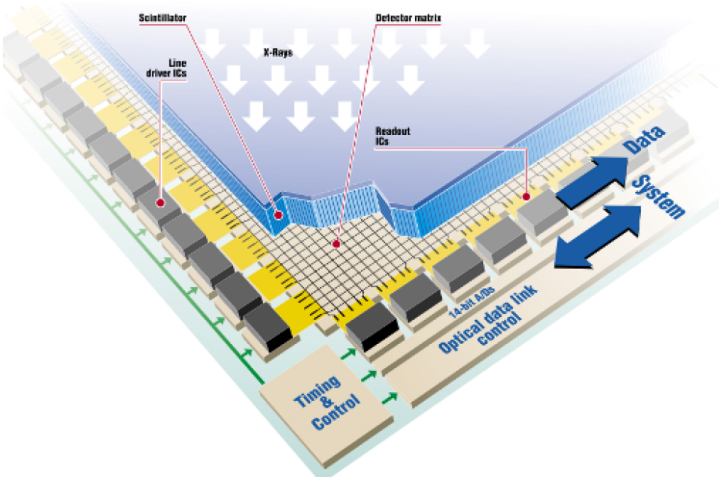


Figure 2.5: Diagram showing the components that make up a detector [34].

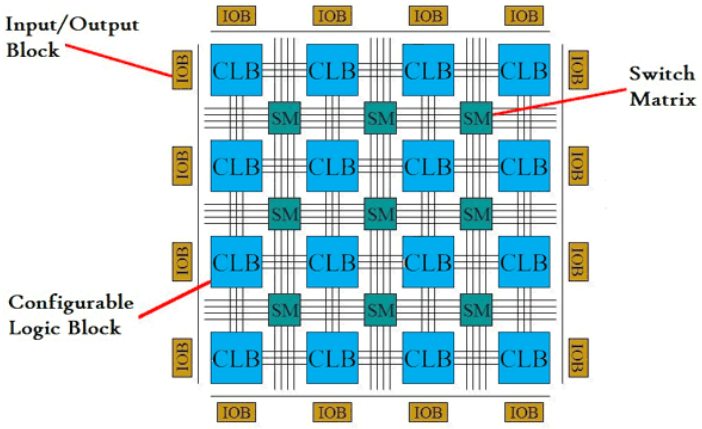


Figure 2.6: Structure within an FPGA [25].

is restricted by the hardware implementation which cannot be altered after production.

2.3. Related work

The generation of synthetic X-ray imagery is not a novel field, with research on rendering techniques and computational algorithms dating back to at least 1994. Over the past two decades, there has been a strong focus on digitally reconstructed radiographs (DRRs) as the primary method for simulating X-ray images. DRRs are computational approximations of X-ray images. They are based on 3D imaging datasets derived from computed tomography (CT) or reconstructed rotational X-ray images. There are several approaches to the usage of these dataset for the image generation process.

It was discovered early on that approaches involving the simulation of rays moving through the model, also known as ray tracing, are computationally expensive and require a high memory bandwidth. [27] describes the usage of attenuation fields that extend ray tracers, allowing most computations to be performed in a preprocessing step. This method significantly accelerates DRR generation. The attenuation field are more memory efficient than storing precomputed DRR tables used in conventional methods. Using this approach, a speed of roughly twenty images per second can be achieved.

In order to speed up the computations, graphics processing units (GPUs) are a good candidate due to the large number of processing cores available. [29] achieves a performance of approximately 100 images per second through algorithmic simplifications and specialized ray casting techniques. [33] optimizes DRR rendering on GPUs and evaluates performance across four commercially available devices. They were able to achieve a performance ranging from 190 to 370 images per second. Both

techniques used small models however of approximately 60 MB, which means the models used were either not very large or not very detailed.

More modern approaches advocate the usage of machine learning algorithms to aid the X-ray image generation process. The problem with these techniques is that they rely on large amounts of labelled images which are rarely available. To address this, [9] introduces a multi-stage Generative Adversarial Network (GAN) to generate synthetic images with semantic labels for data augmentation. This approach performs well on small datasets. [24] attempts to generate synthetic CT images from Magnetic Resonance Imaging (MRI) data using GANs with cycle consistency. Cycle consistency refers to the similarity between images translated from one domain to another, and back to the initial domain. The implementation included additional improvements such as perceptual loss, coordinated convolutional layers, and super-resolution techniques. Both methods report accurate images compared to other methods, though no note is made of relevant performance metrics.

3

Simulation outline and algorithm explanation

In this chapter an outline of the simulation setup will be provided. This outline includes a description of the simulation model, including some definitions and the components modelled by the system. Furthermore, the algorithm itself is explained which consists of a projection step, a computation step, and a detector step.

3.1. Simulation model

In order to simulate an X-ray image, three important components need to be simulated. These three components, visible in Figure 3.1, are the source, physical model, and detector. The source should accurately represent how X-rays are produced and how they move through physical space. The model serves as a stand-in for the subject and describes how X-rays are altered as they penetrate material. The detector is responsible for registering the X-ray intensities which are used to produce the final image.

3.1.1. Simulation arrangement

To set up the simulation of X-ray images, the spatial arrangement of the components is critical. Figure 3.2 shows the arrangement and coordinate system used in the simulation. The source is positioned to emit X-rays directed towards the model. The model is placed at the centre, with the frontal bottom-left corner placed at the origin. Behind the model, the detector is placed to capture the X-rays measuring their intensity to construct the final image.

The isocentre (ISO) is defined as the point around which the source and detector rotate. it is not a fixed point but it can move as well. The movement is dictated by the rotation and translation of the source and detector with respect to the model. For the purpose of consistency, all simulations used throughout this thesis have placed the ISO at the centre of the model. Any realistic position is supported however. The ISO-to-source distance is also fixed in all simulations. The source-to-image distance (SID) can vary however, affecting the shape of the beam. This is summarised in table 3.1.

Table 3.1: Value range of SID and ISO to source.

Parameter	Value
SID	89.5 - 119.5 cm
ISO to Source	81 cm

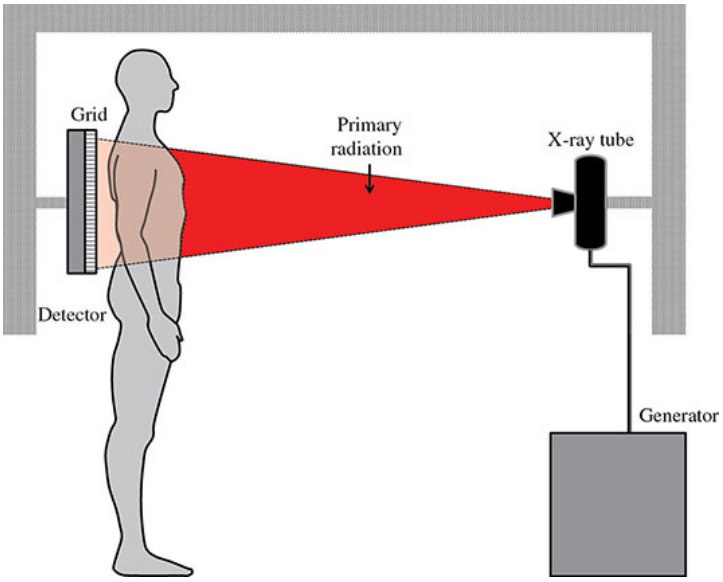


Figure 3.1: Diagram showing the source, model, and detector which need to be modelled in the simulation [18].

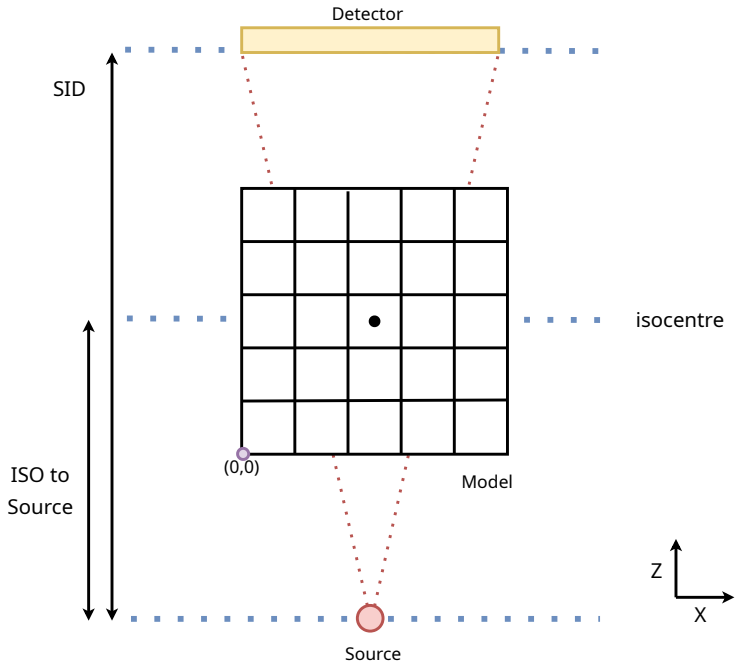


Figure 3.2: Diagram showing the arrangement of the components in the coordinate system.

Figure 3.3: Example voxel model of a skull. This model is used throughout this thesis.

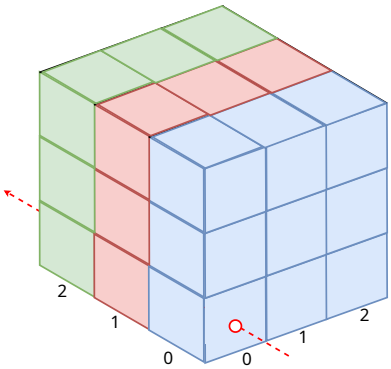


Figure 3.4: Example of how the voxel model is constructed out of individual layers or planes.

3.1.2. Source

The source is defined by a three-dimensional vector containing an X, Y, and Z component. Throughout the simulation this vector can be transformed, as a real imaging system can also move to different positions. This transformation, which can be a combination of rotation and translation, also affects the trajectory of the rays that the source produces.

The source is modelled as a single point in space and produces a pyramid-shaped beam consisting of individual rays. The simulated source is a simplification compared to its physical counterpart because of two reasons. First, the beam produced by the source is modelled as radiation with an energy of 75 keV. As explained in Chapter 2, this means the beam is monochromatic rather than polychromatic. This assumption simplifies the simulation by eliminating the need to account for the varying absorption of X-rays at different energy levels. This approximation is considered valid because, as shown in Figure 2.2, the linear attenuation coefficients for soft tissue, fat, and bone show a similar dependency on the X-ray energy for higher levels. However, this simplification is not able to represent physical effects like beam hardening, which is an increase in the mean energy of the X-ray spectrum as radiation moves through an object. Artifacts caused by such effects like cupping and streak artifacts [3] are therefore not visible.

Second, the simulated source produces radiation which does not scatter. This is not representative of a physical system, which predominantly suffers from Compton scatter in the medical imaging energy range [15]. Compton scatter is not only a source of noise, but can also cause artifacts in dense materials appearing as blurs in the image [14]. Because such effects do not dominate the final image, is was not considered for this thesis and its inclusion is recommended as future work.

3.1.3. Voxel model

The model depicts the subject of the imaging process. In the simulation, the model consists of a large amount of voxels, also known as three-dimensional pixels, containing linear attenuation coefficients which describe how the X-ray intensity is reduced as rays move through the voxel. Throughout this thesis, a voxel model of a skull is used. This model is shown in Figure 3.3. The model is divided into layers which are traversed from the front to the back. This is shown in Figure 3.4. There are little restrictions on the size of the model. It can both be a cube or a rectangular cuboid. The only requirement is that all layers run parallel to each other. The skull model has a size of 384 x 297 x 384 voxels. The largest model which must be supported has a size of 1024 x 1024 x 1024 voxels. In case the model is larger than this, a slice can be created which fits within these dimensions. As long as the rays do not pass voxels outside of this slice, these voxels can safely be ignored. In case the source and detector move, the slice can be reconstructed.

The voxel values of the model represent linear attenuation coefficients. Initially, these coefficients

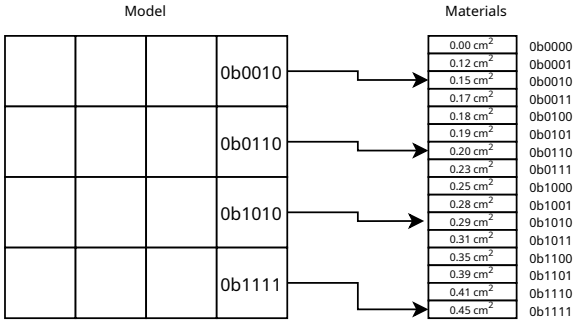


Figure 3.5: Example of the voxel model compression concept, adapted from [19]. Each voxel contains an index into a materials table, greatly reducing the memory footprint.

are represented using 32-bit floating point numbers making the range of possible values practically continuous. This is inconvenient for two reasons. First, floating point operation require a large area of dedicated hardware. Second, many voxels which are close to each other will have values in a similar range because they represent the same physical material like tissue or bone. If these sets of roughly equivalent voxels can be grouped, the memory footprint can be decreased if the groups can be referenced using a single number instead of 32 bits.

To implement this, the model is limited to 16 different material values. This limit means that a material can be referenced by a 4-bit integer acting as the index for a lookup table instead. This lookup table contains the actual linear attenuation coefficient represented using fixed point notation for efficient processing. An uncompressed model of 1024 x 1024 x 1024 voxels would require roughly 4.3 GB of storage. A model utilizing the compression trick only requires 536 MB of storage. The compression concept is illustrated in Figure 3.5.

The compression of the model is a trade-off between processing speed and representation accuracy. The main advantage is a large reduction in memory bandwidth, enabling faster processing. However, averaging attenuation coefficients introduces some error compared to the original model. In Chapter 6, various strategies will be explored to minimize this error while maintaining performance. If the resulting error is deemed too large, future work could explore the impact of increasing the material count on image quality and system performance.

3.1.4. Detector

The detector is the final component in the simulation. Its functionality is not modelled, but it is still an important part of our representation. This is because it plays a large part in the generation of the X-ray beam. The detector is treated as a rectangle for simplicity, but it does not need to be. Different shapes would work as well. The detector is based on the Allura Xper FD20, which has a size of 293.2 x 398.2 millimetres.

The detector is defined by a centre point and horizontal- and vertical spans. Based on this information, the detector corners can be accurately represented. The most significant parameter of the detector is its resolution, which directly influences the quality and computational complexity of the simulation. Each pixel in the detector corresponds to a single ray traced from the X-ray source, resulting in a computational complexity which is linear to the number of detector pixels.

3.1.5. Computational model

At its core, the simulation is about calculating the intensity of a ray using the Lambert-Beer law stated in Equation 2.1. A line integral needs to be evaluated to determine the intensity. The voxel model is discrete, so the integral can be replaced by a simple summation:

$$I = I_0 e^{-\sum \mu_{x,y,z} \cdot l_{x,y,z}} \tag{3.1}$$

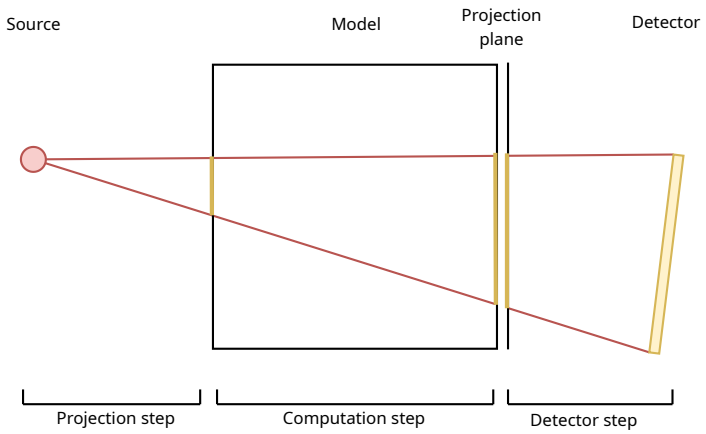


Figure 3.6: Overview of the 3 steps making up the algorithm, adapted from [19].

where $\mu_{x,y,z}$ represents the linear attenuation coefficient of a specific voxel and $l_{x,y,z}$ the length of the path through this voxel.

For the purpose of computational efficiency, only a single voxel per layer is considered. This voxel contains the longest path segment of the ray. The contribution of neighbouring voxels are ignored and the length of the ray through those voxels are added to the voxel under consideration. Thus, since the angle of a ray does not change as it moves through the model, for each particular ray its length through every intersected voxel is constant. Therefore, Equation 3.1 can be further simplified to include a constant length L, shown in Equation (3.2)

$$I = I_0 e^{-L \cdot \sum \mu_{x,y,z}} \tag{3.2}$$

Equation 3.2 shows that the intensity of a ray can be calculated if the scaled sum of linear attenuation coefficients is known. This is not a costly operation. The summation of the attenuation coefficients is the expensive part. The goal of the algorithm is therefore to compute this sum for every ray efficiently. The end result is a two-dimensional matrix containing the total linear attenuation for every ray, referred to as the attenuation map.

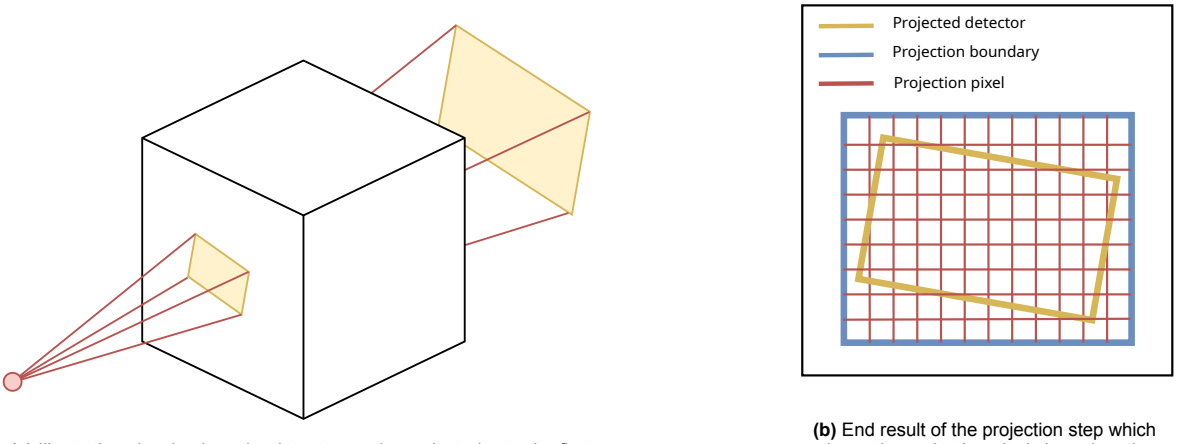
3.2. Algorithm

This section explains the algorithm used to compute the attenuation map for a given voxel model and detector, outlining the steps involved in the process. The algorithm is structured into three steps: the projection step, the computation step, and the detector step. These steps are shown in Figure 3.6. The projection step focuses on projecting the detector onto the voxel model in order to determine the boundary of the beam. This boundary is used to calculate a grid of projection pixels which mark the entry points of the rays in the model. The computation step handles the propagation of the rays through the model, determining the cumulative linear attenuation on the projection plane. Finally, the detector step is responsible for transforming the attenuation map from the projection plane onto the detector.

3.2.1. Projection step

In the first step of the algorithm, the detector is projected onto the first model layer. This step is important because the projection confines the X-ray beam. Figure 3.7a shows a visualization of how the detector can be projected onto the first model layer.

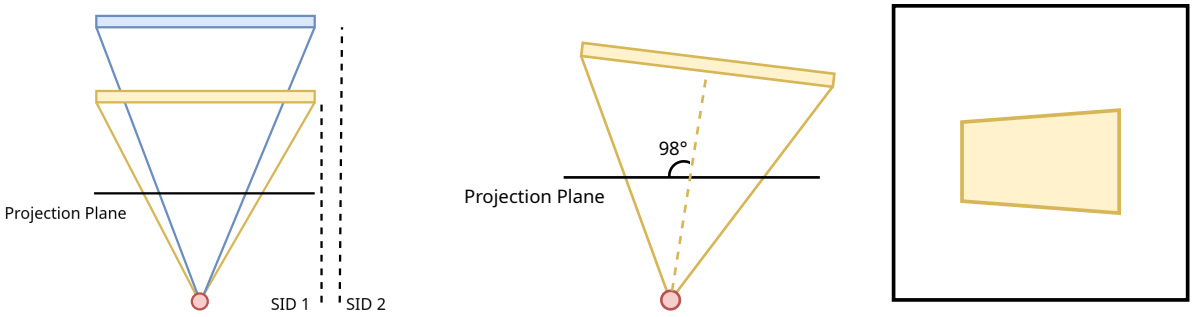
There are a couple of parameters that influence the size and shape of the projection. The first is the SID, which determines the size of the projection which is shown in figure 3.8a. The closer the detector is to the voxel model, the larger the projection will be for a given detector shape. The second parameter is the orientation of the source and detector. They are able to rotate around the ISO centre inside of the model. Rotation along either the X- or Y-axis influences the shape of the projection, as shown in figure 3.8b and figure 3.8c.



(a) Illustration showing how the detector can be projected onto the first model layer.

(b) End result of the projection step which shows the projection pixels based on the projected detector.

Figure 3.7: Illustration of projected detector and resulting projection grid.



(a) SID determines projection size

(b) Orientation determines projection shape

(c) Resulting trapezium shape caused by rotation

Figure 3.8: Effects of distance and orientation of the detector on the projected projection.

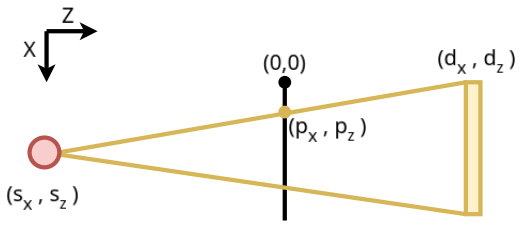


Figure 3.9: Using simple trigonometry, a point can be projected on the first model layer based on the source and the detector.

Before the detector can be projected, the position of the source and the detector in three-dimensional space must be known. For the first image, this position is equal to the initial configuration. For the other images, the positions need to be adjusted according to the provided trajectory if present. In the case of translation, all relevant positions can simply be moved since they all move an equal amount. In the case of rotation, a few calculations are required to determine the new position.

Based on Euler's rotation theorem [20], an arbitrary three-dimensional rotation can be split up into three two-dimensional rotations. To achieve this, a standard rotation matrix described by [36] is used. This results in the following equations:

$$\begin{aligned} R_x(\theta) &= \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} z \\ y \end{bmatrix} \\ R_y(\phi) &= \begin{bmatrix} cos(\phi) & -sin(\phi) \\ sin(\phi) & cos(\phi) \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \\ R_z(\psi) &= \begin{bmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned} \tag{3.3}$$

Once the position is known, the detector can be projected onto the first layer. The equations shown in Equation 3.4 can be derived from the setup shown in Figure 3.9.

$$\begin{aligned} p_x &= \frac{-s_z}{d_z - s_z} * (d_x - s_x) + s_x \\ p_y &= \frac{-s_z}{d_z - s_z} * (d_y - s_y) + s_y \end{aligned} \tag{3.4}$$

Once the detector has been projected a boundary can be constructed. This boundary covers the projection with some additional padding, which benefits the detector step. Within this boundary a grid can be constructed. This grid determines the entry points of the ray in the model. An example of this grid is shown in Figure 3.7b.

3.2.2. Computation step

In the computation step the traversal of rays through the voxel model is simulated. The path of the rays is based on the result from the projection step. Equally spaced rays are generated with their entry points chosen at the centre of the projection pixels shown in Figure 3.7b. For every voxel intersected by a ray, the linear attenuation coefficient corresponding to this voxel is added to the sum of the previous layers. Figure 3.10 shows how a set of rays move through the centred model.

The attenuation coefficients provided by the model are based on a length of one unit, which would describe a path perpendicular to the xy voxel plane. Because the ray can move angulated against the voxel plane normal, the end result needs to be scaled according to the length of the ray in a voxel in accordance with Equation 3.2. Let $\overrightarrow{R}(\alpha, \beta)$ be the path of a ray through a voxel in the z-direction, where α and β describe the slope in the x- and y-direction respectively. The length of a segment is then equal to the magnitude of this vector, shown in Equation 3.5. $\widehat{x}$, $\widehat{y}$, and $\widehat{z}$ are the unit vectors. An example of an attenuation map produced by the computation step can be seen in Figure 3.12.

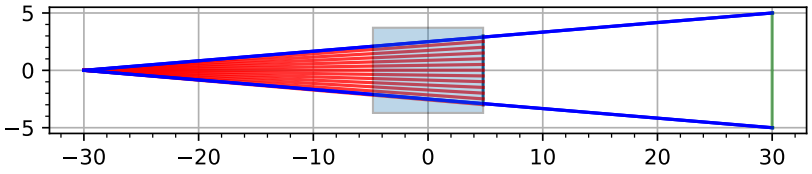


Figure 3.10: Side view of a set of rays (red) as they move through a model (light blue), generated with code from [19].

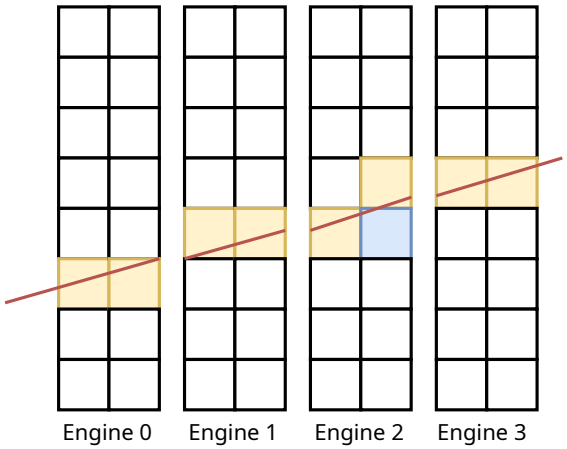


Figure 3.11: Path which a ray traces through the model. In this illustration, one engine is responsible for two model layers. The yellow voxels represent intersected voxels. The blue voxel is not considered.

$$\begin{aligned} \overrightarrow{R}(\alpha,\beta) &= \alpha\hat{x} + \beta\hat{y} + \hat{z} \\ l = |\overrightarrow{R}(\alpha,\beta)| &= \sqrt{\alpha^2+\beta^2+1} \end{aligned} \tag{3.5}$$

Many rays can be simulated in parallel through the use of compute engines. These engines, described in detail in Chapter 5, are responsible for contiguous model layers and compute the path and accumulated attenuation for a given ray. Figure 3.11 shows an example of the path of a ray through a voxel model. It is important to note that in this algorithm only a single voxel per layer is considered. This is a simplification however, because in reality up to three voxels can be crossed at a time in three-dimensional space. As a suggestion for future work, quantifying and possibly reducing the error caused by this simplification can be investigated.

3.2.3. Detector step

After the computation step, an attenuation map of the X-rays has been produced. This map is based on the projection plane, which is the exit plane of the final model layer. The rays are equally spaced in this plane. This is a result of the computation step. However, the detector pixels which detect the X-rays are not equally spaced. Their location can significantly differ from the projection pixels based on the amount of padding used to construct the boundary and rotation of the system. Figure 3.13a shows an example of an attenuation map for approximately 15 degrees of rotation around the y-axis. This map is in the projection plane. The orange line shows the border of the detector projected onto this plane. All detector pixels are within this boundary. To determine the attenuation registered by these pixels, they have to be interpolated based on the projection pixels. Section 5.5 will explain this process in more detail.

Using interpolation, intensities from one pixel grid are resized to another by estimating the intensities at the interpolated positions. Several algorithms are available to do this. In order to make a decision, three different algorithms will be explored. The three different algorithms are discussed in detail in Section 5.5.

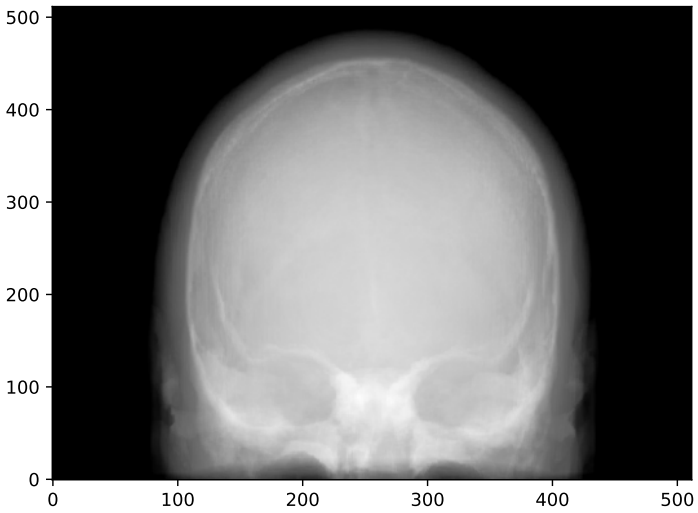
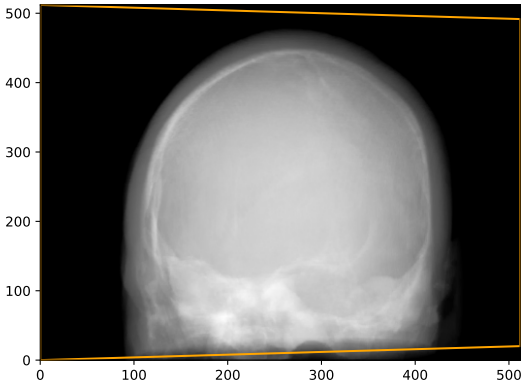
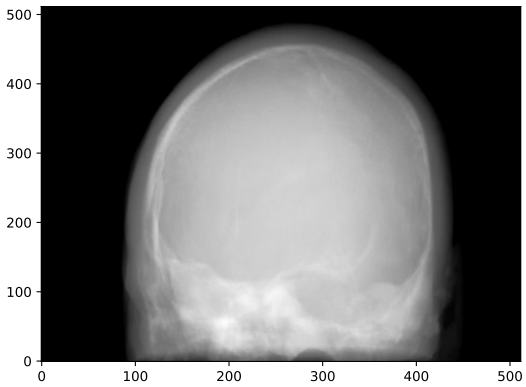


Figure 3.12: Example of an attenuation map produced by the computation step.



(a) Attenuation map on the projection plane. The orange line indicates the projected detector.



(b) Attenuation map interpolated to the detector plane.

Figure 3.13: Attenuation before and after the detector step, generated with approximately 15° of vertical rotation.

The first interpolation method is the nearest neighbour interpolation. Several variations of this algorithm exist. For this thesis the easiest variant will be considered, which simply finds the nearest pixel and assigns its value to the pixel to be interpolated. The second interpolation which will be studied is the bilinear interpolation. It is a well known extension of a linear approximation to two-dimensional space. Using this approach, an interpolated value is based on four points surrounding it. The third interpolation method which will be studied is the Lanczos interpolation. This technique is proven to work well in three-dimensional reconstruction in the context of medical volumes [22] and is based around the Lanczos kernel.

4

System architecture

In this chapter, an overview of the system architecture will be provided, starting with a list of requirements that serve as the foundation for the design. Based on these requirements, a design is presented which is also motivated by the need for real-time behaviour.

4.1. Requirements

As outlined in Chapter 1, the purpose of the simulator is to replicate the experience of a real imaging system, delivering realistic X-ray images. Ultimately, the goal is to generate image passable as real images at speeds on par with real imaging systems, providing a safe and effective alternative for machine testing and operator training. Based on this goal the following list of requirements can be established.

1. The hardware must target the Alveo U50 Data Accelerator card;
2. The simulator must support models up until a depth of 1024 voxels;
3. The simulator must support a resolution of at least 512 projection pixels;
4. The simulator must produce realistic results in the operating range of a real X-ray system;
5. The simulator must produce images at a speed of 60 images per second;
6. The accuracy of the implementation must be precise up to 1/16th of a projection pixel;

4.2. System design

An overview of the main system components needed to meet the requirements and achieve real-time behaviour can be seen in Figure 4.1. The diagram specifies two independently operating regions. The host machine is connected to the X-ray system and accepts simulation parameters. The FPGA is connected to the host through the Peripheral Component Interconnect Express (PCIe) and computes the X-ray attenuation maps.

4.2.1. Host machine

The host machine is a Linux PC which houses the FPGA and handles all forms of communication. Its main responsibility is supplying the FPGA with the correct simulation data. This data includes the following information:

- The voxel model;
- The material values of the voxel model;
- The source position;
- The detector position and resolution;
- The trajectory describing the change in source-detector orientation and position.

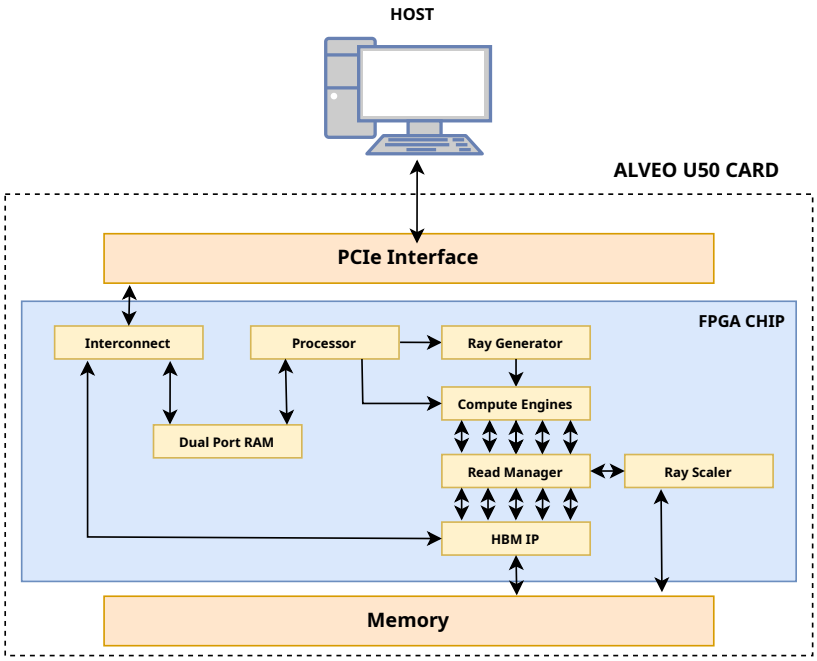


Figure 4.1: Diagram showing the main components of the system. It shows how the host interfaces with the FPGA and how the components required to run the algorithm are interconnected.

The voxel model and material values need to be send ahead of time to the FPGA. The largest model has a compressed size of 536.9 MB. One model covers approximately 22.5 degrees of rotation in each direction. If this range is exceeded, the model needs to be recalculated in order to adjust the orientation such that the total rotation does not exceed the 22.5 degree limit. This can be done ahead of time. For every base model, the material values only need to be send once. The host must however make sure that each model is present on the FPGA ahead of time. The model data is stored in the memory of the Alveo U50 card. The material values are stored in Lookup Table Random Access Memory (LUTRAM) blocks within each engine, later referred to as the parameter RAM.

The other parameters are inputs which define the simulation configuration. The positions of the source and the detector determine the path the rays take through the model. The resolution of the detector defines the amount of rays which are simulated, which has a large impact on the image quality. The trajectory describes how the source-detector orientation changes between frames. These parameters only need to be send once to the FPGA, as long as the trajectory does not change and the source-detector orientation is within the specified range. The inclusion of the trajectory parameter means the simulation configuration can be updated on the FPGA. This decreases the amount of communication between the systems, which is necessary to achieve real-time behaviour. The input parameters are stored in Block RAM (BRAM) connected to a processor on the FPGA. Communication with the host is also supported through the BRAM.

Detector

The final stage of the X-ray image simulation algorithm involves transforming the produced attenuation map onto the detector. While the majority of the simulation is implemented on the FPGA, the detector step is performed on a PC. This stage begins by taking the output from the FPGA, an attenuation map, and interpolating it to account for distortions introduced by the projection onto the model.

Although this process could be implemented on an FPGA, it is not timing-critical and involves relatively few calculations, making it more practical to handle on a PC. Additionally, performing this step on a PC enables post-processing operations, such as filtering and upsampling, which can enhance image quality. This approach also provides flexibility for future extensions, such as integrating GPU-based processing to further accelerate or enhance the simulation pipeline.

4.2.2. FPGA components

The FPGA handles the execution of the simulation. It should independently operate as a pipeline, requiring as little directions from the host machine as possible. To realize the algorithm outlined in Section 3.2 in hardware, the system shown in Figure 4.1 was created. The diagram shows the most important components and how they are connected.

As described in the previous section, the initial parameters and model data are send to the FPGA through a PCIe interface and the interconnect of the chip. The parameters are stored in memory blocks exposed to a micro controller. These blocks allow for bidirectional communication between the FPGA and the host, though this has not yet been realized for this project. The model data is stored in memory chips on the Alveo U50 card.

Processor

The first task of the FPGA is to project the detector onto the model. Because this step requires complex operations which are expensive to implement accurately in hardware like trigonometric functions, it is implemented using a MicroBlaze processor. The MicroBlaze is a 32-bit soft-core processor and has many different hardware configurations. It can be optimized for area or performance and can be configured to include additional hardware like fast multipliers. It can be programmed using C allowing a flexible implementation of the non-critical parts of our system. Once the micro controller has finished processing the initial parameters, its output is communicated to the rest of the system. For the ray generator component this includes the projection boundary which confines the rays. For the compute engines, this includes the material values stored in the parameter RAM and a line offset.

Ray generator

The ray generator is responsible for providing the input to the compute engines. It does this by calculating where a ray enters the model as well as how the position of a ray changes between model layers. This information is all the engines need to determine how the ray traverses the model.

Compute engines

The compute engines play a crucial role in the system. They calculate the sum of attenuation coefficients for specified ray directions and positions and update the position of rays as they move through the model. Their functionality and implementation was first introduced in [19].

Each engine handles eight model layers. The engines form a chain. This means that the number of rays which can be processed concurrently is equal to the number of engines in the system. The engines requests voxel lines from memory in advance to accelerate processing. They operate in groups of sixteen. Within a group, the engines are connected to the same read manager. They must wait for their memory requests to be processed if other requests are handled first. This can potentially delay individual computations.

Ray scaler

The ray scaler handles the final step of the computational model explained in Section 3.1.5. This step includes properly scaling the computed attenuation according to the length of a ray segment. This is because the attenuation calculated by the engines assume a ray length of one unit. It is important that the ray scaler does not introduce any latency in the system, because every single ray must be processed by this component. Therefore, a fast implementation of this component is important.

4.2.3. Memory components

Earlier work focused on the Versal FPGA device, where memory bandwidth proved to be a significant bottleneck. Based on request and access times gathered through simulations, engines had to be run at half their possible speed in order to prevent delays. To address this issue, the Alveo U50 Data Accelerator card was considered due to its High Bandwidth Memory (HBM) architecture. This architecture is well-suited for the application, as it can handle large volumes of data in parallel, benefiting from the highly predictable access patterns of the processing engines.

As shown in Figure 4.2, the Alveo U50 card features two 32 Gb HBM stacks, composed of DDR4 (Double Data Rate) memory chips. Each stack consists of sixteen pseudo channels, each with a dedicated memory interface and a capacity of 2 Gb. These channels can be accessed almost entirely in parallel,

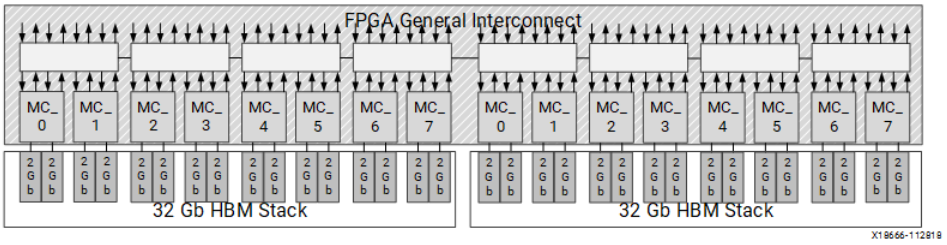


Figure 4.2: Alveo U50 HBM two stack configuration [5]

Table 4.1: Summary of the amount of bits returned for a given model width.

Model Width	Bits	32-byte Bursts
256	1024	4
384	1536	6
512	2048	8
1024	4096	16

significantly improving data throughput. However, engines are not directly connected to the memory interface. Instead, a read manager is responsible for the retrieval of voxel lines requested by the engines, providing access to the corresponding pseudo channel.

Read manager

In software, the operating system is responsible for handling the memory management of an application. In an FPGA, this is the responsibility of the application itself. The read manager is therefore one of the most important components in the system. One read manager is responsible for serving voxel data to sixteen engines at most. It sits between the engines and the Alveo U50 memory.

Compute engines can request a model line with a single request. Memory is returned in bursts of 256 bits. The read managers must therefore be able to handle these bursts and return them in the correct order to the compute engines. Table 4.1 shows how many bits are returned per line for a given model width and the number of bursts during which this happens.

5

System implementation

This chapter will discuss the implementation details of the system. Specifically, it will explain the hardware and software components required for the computation of an attenuation map. An overview of the hardware components and their connections are shown in Figure 5.1. The details of the read manager and memory are discussed in Chapter 6.

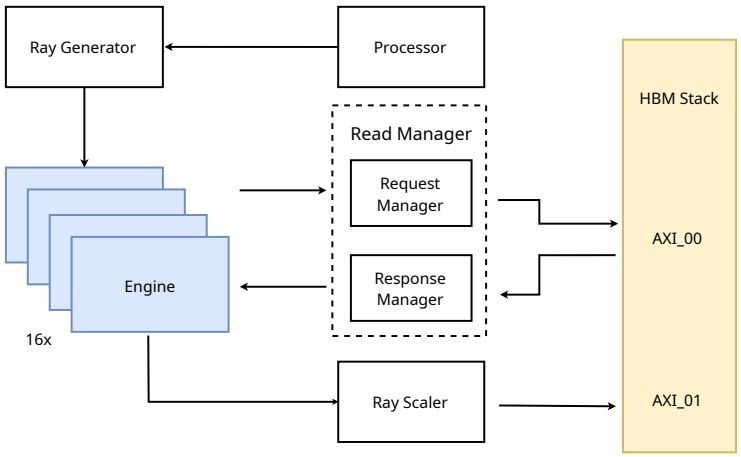


Figure 5.1: Functional diagram showing how the components discussed in this chapter are related. The number of components will vary depending on the model size.

5.1. Processor

The purpose of the processor is to take the parameters supplied by the host machine and compute the input parameters for the simulation. It is implemented using the MicroBlaze Micro Controller System IP supplied by Xilinx. The MicroBlaze is a 32-bit Reduced Instruction Set (RISC) soft-core processor with a three-stage pipeline. It has an integrated IO module which can be used to communicate with the rest of the system.

The MicroBlaze calculates a set of parameters used by the ray generator component to generate the rays. It calculates an initial vertical offset which describes the entry point of the first ray for the engines. Additionally, it sends the material coefficients used for the attenuation computations to the engines. The processor operates at 100 MHz. The other hardware components operate at 400 MHz. To prevent data hazards from interfering with the communication, clock domain crossing logic is required. This logic consists of double buffering the MicroBlaze IO bus and generating a simple ready-valid handshake in the corresponding clock domain using a pulse generator provided by Xilinx [6]. Figure 5.2 shows how the processor is connected to the other components.

Table 5.1: Execution time of different parts of the algorithm running on the MicroBlaze.

Task	Time
Precompute trigonometric values	123 μs
Update source and detector according to the trajectory	239 μs
Project detector points onto the model	178 μs
Construct boundary used for ray generation parameters	3 μs
Compute ray generation parameters	59 μs
Send ray generation parameters over IO bus	2 μs
Calculate and send vertical offsets to engines	255 μs
Total:	**859** μs

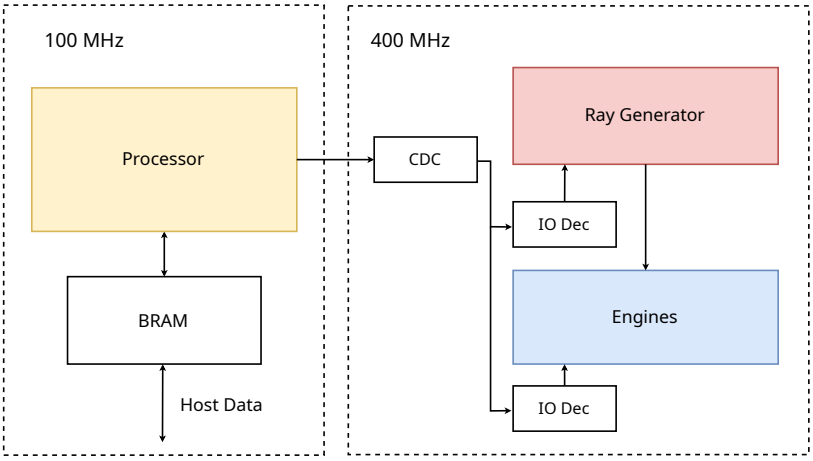


Figure 5.2: Diagram showing how the processor is connected to the rest of the system.

Software

The software, implemented in C, must be fast due to its placement in the critical path. This is because the projection step of the algorithm described in Section 3.2 is executed for every image. The projection step can be executed in advance while the engines are handling the computation step. If the host parameters change however, the result will have to be recomputed. To determine the available time to run the code, consider the worst-case scenario where new parameters arrive just before the new frame is required. This means the processor and the engines have to complete their work within 16.5 ms in order to achieve the desired performance of 60 images per second. To provide the engines with a wide margin to complete their work, the processor should finish its part of the algorithm within 2 ms.

Execution begins with fetching parameters from the host. However, as host/FPGA communication has not yet been realized for this thesis, it is not discussed further. The execution times of the remaining operations for the projection step have been measured using simulation and are summarized in Table 5.1. Based on the total duration, completion within 2 ms is easily achieved. To ensure a fast execution time, the algorithm uses fixed point numbers for all computations. This speeds up the execution dramatically, since the processor does not have a Floating Point Unit (FPU) for the arithmetic. To determine the number of fractional bits required, the fixed point implementation was compared with a floating point implementation. The number of fractional bits was adjusted until the requirement of accuracy up to 1/16th the size of a projection pixel was met. In addition to the fixed point implementation, an approximation of the sine- and cosine function is also implemented in the software. The values are computed once and reused for each vector that needs to be rotated. The approximation was confirmed to meet the level of accuracy required.

5.2. Ray generator

The ray generator component supplies the engines with data about the X-rays, including the x- and y-position within a model layer as well as their respective changes, known as the x- and y-increments.

To determine this for every ray, knowledge of the projected boundary as well as positional information from the processor is used.

It operates based on a state machine. When the reset signal drops, the ray generation component waits for the necessary parameters as mentioned in Section 5.1. It receives these parameters through an IO decoder attached to the IO bus of the processor. Once all parameters have been received, the IO status signal is asserted and rays can be generated.

X- and y-position

The rays are generated from the bottom-left to the top-right corner of the boundary. To compute the initial position of the ray, the x_{step} and y_{step} parameters are introduced. These parameters are equal to the width and height of a projection pixel respectively. They are computed by the processor and equal to the width or height of the boundary divided by the resolution in the same direction. The initial x- and y-position are calculated according to Equation (5.1), where x_{min} and y_{min} are the bottom-left corner of the boundary.

$$\begin{aligned} x &= x_{min} + 0.5 \cdot x_{step} \\ y &= y_{min} + 0.5 \cdot y_{step} \end{aligned} \tag{5.1}$$

For each additional ray the x-position is incremented by the x_{step} parameter as long as it is within the boundary. If the horizontal boundary is passed, the x-position is restored to the initial x-position and the y-position is incremented. If the vertical boundary is passed, all rays have been generated.

X- and y-increment

The x- and y-increment values, which describe how the x- and y-positions change as the rays move between layers, are calculated using a similar process. The initial increment values are however provided by the processor, since they require a division operation to calculate. All additional values are determined by adding an $increment\_step$ value to the previous increment value. To proof that this is valid, consider two adjacent projection points on the xz-plane. The first x-increment value can trivially be calculated using Equation (5.2), where P_n and P_{n-1} are neighbouring projection points, S is the source, and D_{smd} is the source-model-distance.

$$P_{n-1} : x_{increment} = \frac{P_{n-1,x} - S_x}{D_{smd}} \tag{5.2}$$

If P_n is rewritten in terms of P_{n-1} it is shown that the increment value changes by a constant factor, which has previously been referred to as the $increment\_step$. This is shown in Equation (5.3). The same relation holds for the y-increment of a ray.

$$\begin{aligned} P_n : x_{increment} &= \frac{P_{n,x} - S_x}{D_{smd}} \\ &= \frac{P_{n-1,x} + x_{step} - S_x}{D_{smd}} \\ &= \frac{P_{n-1,x} - S_x}{D_{smd}} + \frac{x_{step}}{D_{smd}} \end{aligned} \tag{5.3}$$

5.3. Engines

The compute engines, introduced in a different Master's thesis [19], are responsible for computing the path of X-rays through a model. An overview of the components within the engine can be seen in Figure 5.3. Most of the functional details of the compute engines will not be covered in this thesis. However, in the context of the entire system an understanding of its basic functionality is required. Figure 5.4 shows a diagram describing the simplified interface of a single engine.

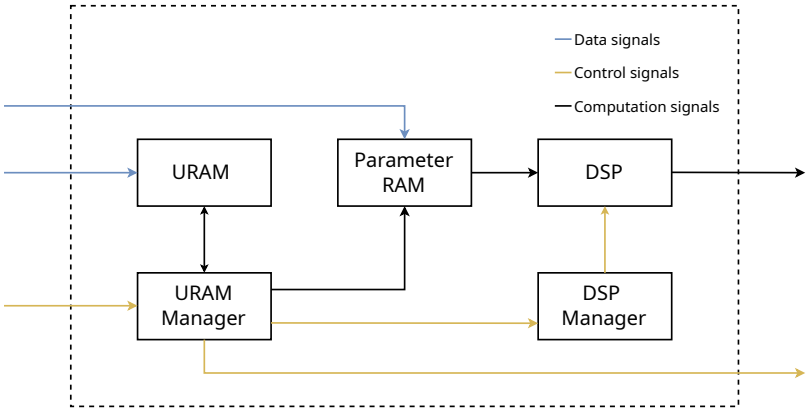


Figure 5.3: Diagram showing the components which make up a compute engine [19]. The most important components are the URAM manager, which handles all control signals within the engine, and the DSP manager, which handles the control signals of the addition component.

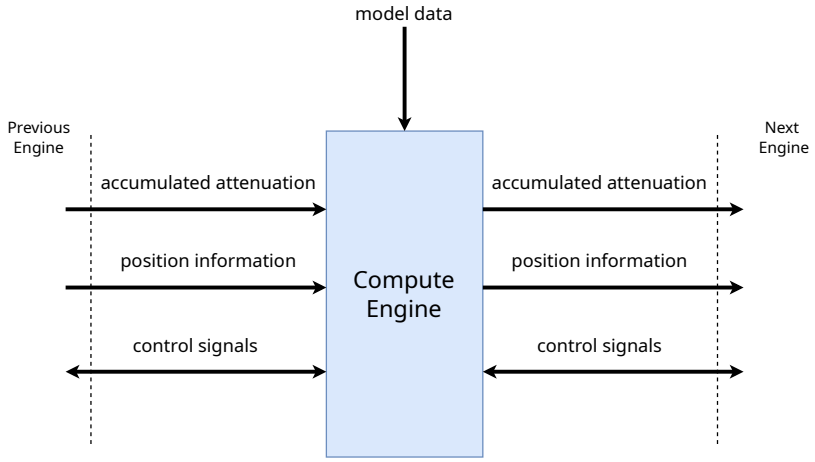


Figure 5.4: Simplified diagram describing the interface of a compute engine.

5.3.1. Computations

The compute engines are responsible for executing the computation step of the algorithm, as described in Section 3.2.2. The engines are connected in a chain, each responsible for calculating the path of a ray through eight model layers. The number of cascaded engines is therefore dependent on the depth of a model. A model with a depth of 256 layers requires 32 engines. The largest model supported has a depth of 1024 layers, which means 128 engines would be required. The three main pieces of information exchanged between engines are as follows:

- **Partial results:** Since the engines operate in a cascaded fashion, each engine takes the partial result of a previous engine and computes a new partial result which includes the eight layers it is responsible for.
- **Position information:** In order to determine the ray position within a layer, four values describing the position of a ray are shared between engines. These values include the x- and y-coordinate of a ray within a layer, as well as an x- and y-increment describing how the coordinate changes between layers. This logically means that the output position of a ray is equal to the sum of the input position and eight layer increments.
- **Control signals:** To facilitate the communication between engines some control signals are required. These signals describe when an engine is ready to receive data from the previous engine or to send it to the next one.

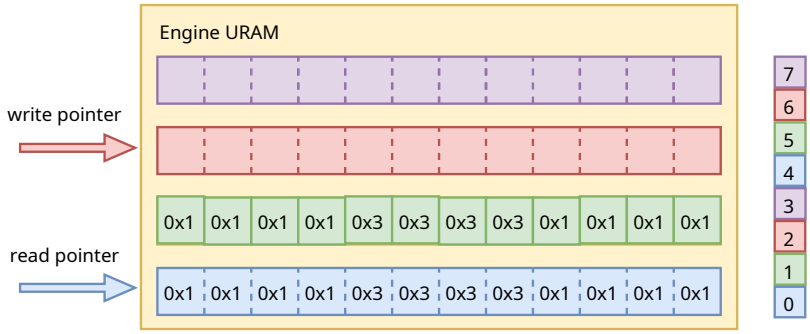


Figure 5.5: Implementation of the engine line buffers. The number in the boxes indicate the line numbers. The colour indicates the buffer a line is stored in.

Table 5.2: URAM addressing structure. P[n] denotes the nth plane bit, L[n] denotes the nth line bit, and I[n] denotes the nth index bit.

22 - 11	10	9	8	7	6	5	4	3	2	1	0
0	P[2]	P[1]	P[0]	L[1]	L[0]	I[5]	I[4]	I[3]	I[2]	I[1]	I[0]

5.3.2. Engine memory

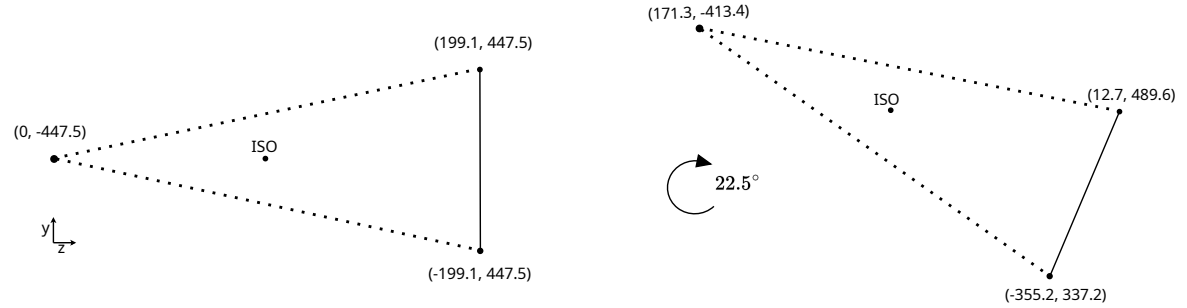
In order to compute results, each engine is responsible for fetching the voxels it requires. It does this by utilizing a set of buffers which can each store an entire line of voxels which are requested ahead of time. Each engine has four line buffers per model layer, meaning 32 lines in total can be buffered. The aim of the buffers is to store lines before they are required. Using this approach, the amount of time an engine spends waiting on voxels to be retrieved is limited. The buffers are implemented as circular buffers within an UltraRAM (URAM) block. URAM blocks are the largest dedicated on-chip storage elements on an FPGA and have plenty of memory to hold all the lines. The circular buffer implementation means they have a read and write pointer as shown in Figure 5.5. When a line has been read, meaning no more rays will pass through it, the read pointer is incremented. When a line has been read from memory the write pointer is incremented, as long as it will not overwrite the read pointer. Every line only needs to be read once, since the rays traverse the model from the bottom to the top for any given plane.

A URAM block is a specialized memory resource available in FPGAs. URAM is designed to provide high-capacity and high-performance on-chip memory. A single URAM block offers 288 Kb (36 KB) of memory, which is significantly larger than a Block RAM block which has 36 Kb (4 KB). Each engine has its own URAM block to implement the buffers. The URAM is accessed using a 23-bit address. There are three bits required to select the relevant plane. Two are required to select a line buffer. The data line of a URAM block is 64 bits. This means a block of sixteen voxels can be accessed at once. Since the largest line contains 1024 voxels, there are $log_2(1024/16) = 6$ bits required to index a block in the buffer. Based on these parameters the URAM addressing structure shown in Table 5.2 can be defined. This structure needs to be followed by the read manager, since it is responsible for writing the correct voxels to these addresses.

Before the engines can process rays, they must first be initialised with parameters provided by the processor. These parameters include the material coefficients used to decompressed the voxel values. These parameters are supplied by the host application and stored in the parameter RAM within the engines. Additionally, each engine requires a vertical offset to determine the entry point of the first ray for each of the layers an engine is responsible for. This offset is necessary in order to request the correct voxel lines in advance.

5.4. Ray scaler

The final operation within the FPGA is the scaling of the computed attenuation by the engines. This is done by the ray scaling component, which takes the output of the final engine and scales it based on the length of the ray. Based on the Pythagorean theorem, x_inc (Δx) and y_inc (Δy) , this length can



(a) Configuration of the source and detector drawn at scale for the smallest SID.

(b) Configuration of the source and detector after 22.5° of rotation.

Figure 5.6: Setup resulting in the largest angle of the rays for an ISO placed at the centre. This angle occurs for the smallest SID with maximum rotation.

be computed using Equation (5.4).

$$L(\Delta x, \Delta y) = \sqrt{1 + \Delta x^2 + \Delta y^2} \tag{5.4}$$

This computation requires both a square root operation as well as two square operations. This is not feasible for the hardware implementation due to the latency and circuit area such mathematical operations introduce. In order to still be able to scale the ray a length approximation is required. The approximation has to fulfil a few criteria in order to be useful:

1. **Accuracy:** The approximation must not introduce significant error in the computed attenuation. Therefore, the average error may not exceed 2%;
2. **Speed:** The approximation may not be expensive to compute and add any latency. This means that the approximation must be able to process rays immediately. Therefore, only operations which can be pipelined and take a single cycle can be used;
3. **Resources:** The approximation may not use an excessive amount of resources.

In order to adhere to the speed requirement, the length of the ray will be approximated using a linear function. The additions required for such a function can be implemented in a single cycle using little resources. There will be a delay of one cycle for every addition, but this will not be noticeable at the output. The approximation needs to be valid for every $\Delta x < 1.0$ and $\Delta y < 1.0$, since the engines do not support a higher increment. To verify that this increment is never exceeded, the maximum increment can be computed using some trigonometry and knowledge about the physical system. The maximum increment of the rays occur for the smallest SID with maximum rotation applied to the source and detector. This situation is drawn in Figure 5.6. The smallest SID is 895 millimetres. The largest span of the detector, resulting in the maximum ray increment, is 199.1 millimetres. This information is based on the system specifications provided in Section 3.1.4. The maximum rotation for which a model is still valid is 22.5°. Based on these parameters, if the ISO is positioned at the centre of the system the maximum increment is equal to $|\frac{-369.6-189.1}{321.4--405.0}| = 0.70$. This means the range for which the approximation is valid will never be exceeded.

5.4.1. First order approximation

The first approximation is reliant on the fact that $1 + \Delta x^2 + \Delta y^2 \approx 1 + \Delta x + \Delta y$. This is true because the constant is the dominating term, especially for small values. Assuming this approximation is valid, the square root operation is not yet considered. To approximate this with an addition operation as well, a correction coefficient α can be included.

Before the coefficient is explained, a mathematical definition to select the appropriate coefficient based on Δx and Δy is required. This will be referred to as the coefficient index. First, Δx and Δy are represented in fixed point notation. Let us assume they are positive, so the sign can be ignored. If the

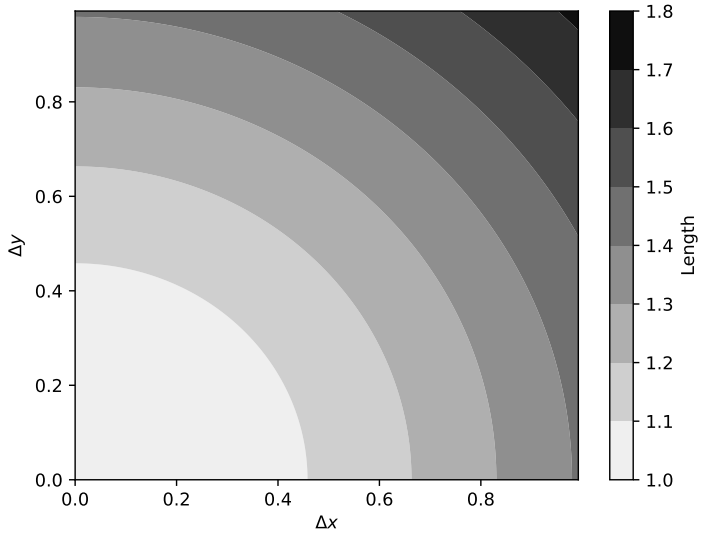


Figure 5.7: True ray length for a given Δx and Δy.

number of coefficients is equal to 2^n, the coefficient index can be computed using Equation (5.5). As an example, consider the index if $\Delta x = 0.453125_{10} = 0.011101_2$. If 16 coefficients are stored, which is 2^4, the index is equal to $MSB_4(0.011101_2) = \lfloor 2^4 \cdot 0.011101_2 \rfloor = 0111_2 = 7$

$$\begin{aligned} MSB_n(\Delta x_2) &= \lfloor 2^n \cdot \Delta x_2 \rfloor \\ MSB_n(\Delta y_2) &= \lfloor 2^n \cdot \Delta y_2 \rfloor \end{aligned} \tag{5.5}$$

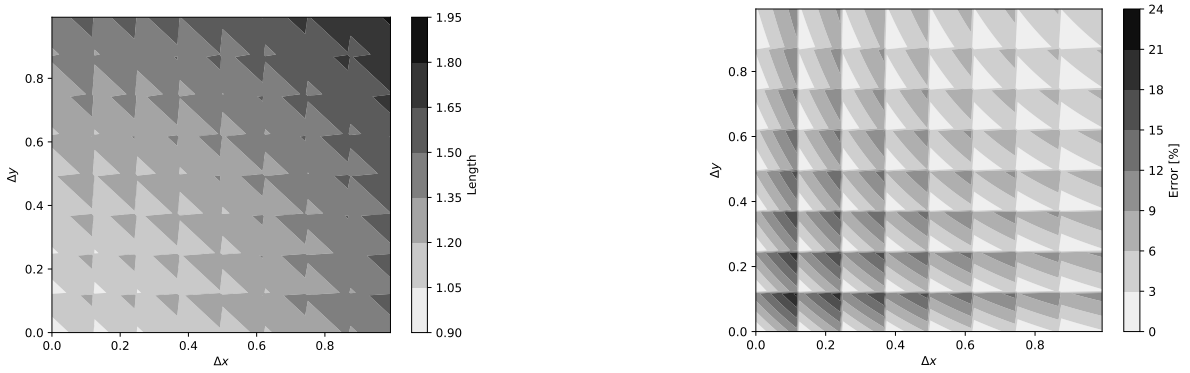
Alpha

The coefficient α, which is based on Δx and Δy, is used to account for the fact that $\sqrt{1 + \Delta x^2 + \Delta y^2} \leq 1 + \Delta x + \Delta y$. To calculate the correction coefficient α, the difference between the actual length and the approximation without correction is calculated as shown in Equation (5.6). Since α will be stored in memory on the FPGA, only a discrete number of coefficients can be stored. This is indicated by the indices i and j, which are computed based on the prior coefficient index definition. Figure 5.7 shows the true length of a ray for a given Δx and Δy. The low-resource memory implementations within an FPGA store multiples of 64 32-bit words. This means 64 coefficients can be stored at a low cost. The error should be the same in both directions. A grid of eight by eight equally spaced Δx_i and Δy_i values is therefore used to calculate the corresponding correction coefficients. Equation (5.7) shows the resulting linear approximation.

$$\alpha_{ij} = (\sqrt{1 + \Delta x_i^2 + \Delta y_j^2}) - (1 + \Delta x_i + \Delta y_j) \tag{5.6}$$

$$L(\Delta x, \Delta y) \approx 1 + \Delta x + \Delta y - \alpha_{ij} \tag{5.7}$$

Figure 5.8 shows a comparison between the true length computed using Equation (5.4) and its estimate using Equation (5.7). There is a clear grid visible. At the 64 sample points within the grid, the error is 0.0. This is because those sample points can perfectly be compensated using the precomputed α_{ij} term. However, everywhere else there is still a significant error present. The average error of this approximation is 6.02%. The maximum error is 21.78%. This error is too significant and the estimate therefore needs to be refined. The straight contour lines in Figure 5.8a do not match the curves in Figure 5.7. This indicates that the estimate can be improved by adding a correction for the square operation as well.



(a) First order approximation of ray length for given Δx and Δy.

(b) Approximation error as a percentage compared to the true length.

Figure 5.8: Comparison between the true length and the first order approximation of the length for a given Δx and Δy.

5.4.2. Second order approximation

For the second order approximation, we further refine the estimate by adding two additional terms. These terms, β_k and β_l, compensate the error introduced by the estimate of the square operation.

Beta

The β correction term can be computed in advance using the relation shown in Equation (5.8). Similar to the α_{ij} term, β is not continuous since it is stored in physical memory. The indices k and l are computed based on the prior definition. β is stored using the same memory block consisting of 64 x 32-bit words. Because β is based on only a single variable-both Δx and Δy are individually compensated-the number of coefficients that can be stored can be doubled to 128 if the precision of a single term is dropped from 32 bits to 16 bits. A coefficient can then be addressed using an odd number of bits, which was not possible for the α_{ij} term.

$$\begin{aligned} \beta_k &= \Delta x_k^2 - x_k \\ \beta_l &= \Delta y_l^2 - y_l \end{aligned} \tag{5.8}$$

Using both the α and β correction terms, the new approximation is shown in Equation (5.9). A comparison between the true length and the estimate using the new approximation can be seen in Figure 5.9. The grid in the error plot is still apparent, but the overall error has decreased significantly. For this approximation, the average error is 4.72%. The maximum error is 17.73%. Again, the error is still too significant. To achieve the desired accuracy, more correction coefficients are needed. This requires an increase in memory.

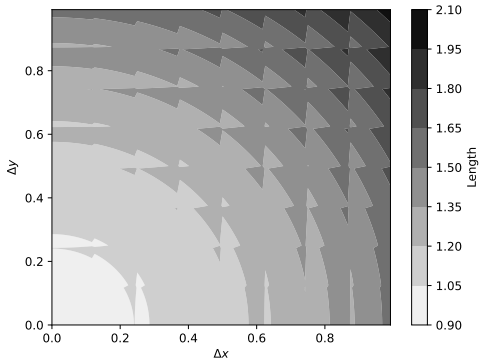
$$L(\Delta x, \Delta y) \approx 1 + \Delta x - \beta_k + \Delta y - \beta_l - \alpha_{ij} \tag{5.9}$$

5.4.3. Final approximation

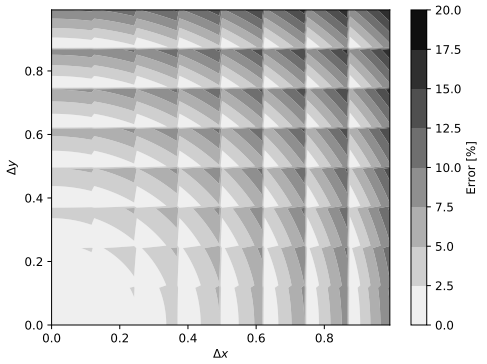
To ensure the approximation is valid for the chosen range, the number of stored coefficients can be increased. This means more points can be represented accurately, lowering the overall error. To achieve the desired accuracy, the number of α coefficients had to be increased from 64 to 1024. The number of β_k and β_l coefficients were increased from 128 to 256. Figure 5.10 shows the comparison between the true length and the estimate with these additional coefficients. The average error of this estimate is 1.25%. The maximum error is 4.40%. This means the error of the approximation is now in the acceptable range.

5.4.4. Hardware implementation

The ray scaler must be able to process a new ray every cycle. The additions in Equation (5.9) are therefore split into several stages. The individual stages can be computed in a single cycle. The

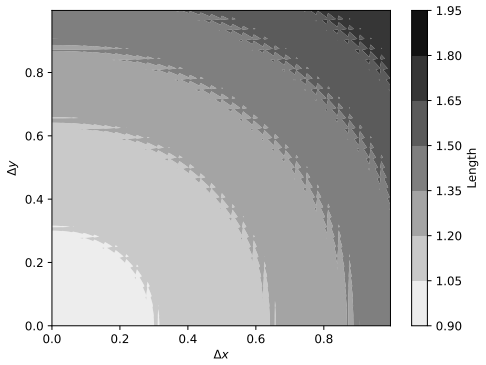


(a) Second order approximation of ray length for given Δx and Δy.

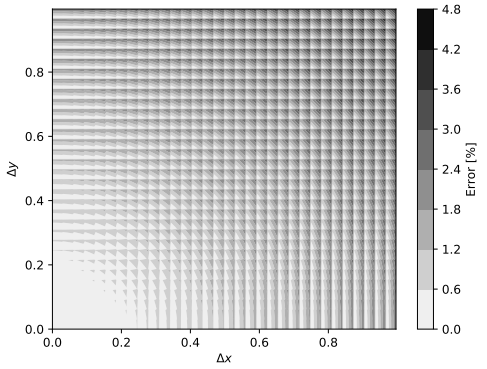


(b) Approximation error as a percentage compared to the true length.

Figure 5.9: Comparison between the true length and the second order approximation of the length for a given Δx and Δy.



(a) Second order approximation with additional coefficients for given Δx and Δy.



(b) Approximation error as a percentage compared to the true length.

Figure 5.10: Comparison between the true length and the second order approximation of the length with additional coefficients.

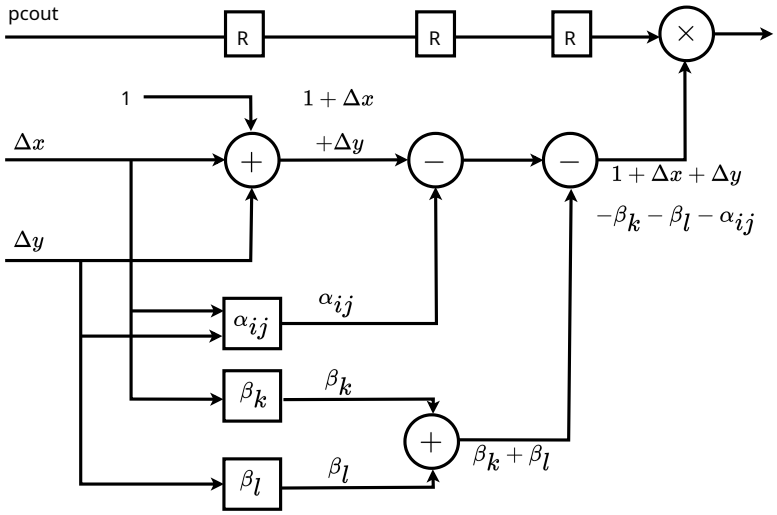


Figure 5.11: Hardware implementation of the ray scaler which computes the length of a ray for a given x- and y-increment.

correction terms are precomputed and stored in read-only memory elements. These can be accessed in a single cycle as well. Using this approach, the rays are scaled without introducing any latency, apart from a delay which is not noticeable at the output.

Since α_{ij} is based on Δx and Δy, both are used to compute the address which contains the corresponding correction coefficient. There are 256 coefficients, so the four most significant bits of both Δx and Δy are required to form the address. To determine β_k and β_l, the eight most significant bits from Δx and Δy are used, since there are 256 coefficients for both as well.

The implementation of the ray scaler is shown in Figure 5.11. There are three cycles required to compute the approximation. The linear attenuation, computed by the engines and supplied using the pcout line of the final engine, therefore needs to be delayed three cycles using registers. Afterwards, it can be multiplied by the length using a Digital Signal Processing (DSP) multiplier element.

5.5. Detector

The goal of the detector step is to produce the final attenuation map used to create an X-ray image. The total attenuation computed for each ray by the engines is based on the projection plane which is always parallel to the model plane. However, in most configurations this is not the case for the detector plane. Therefore, the attenuation needs to be interpolated based on the detector pixels. That is why the detector step is introduced in the algorithm.

As mentioned in Section 4.2.1, this step is implemented in software instead of in hardware on the FPGA. This is because the step is not timing-critical due to the relatively small amount of calculations required. To support this decision, an additional post-processing step will be conducted to evaluate whether an attenuation map can be upsampled while maintaining accuracy. Given that the computational cost of the algorithm scales linearly with the number of projection pixels, this technique offers significant potential for time savings.

Both techniques require interpolation algorithms. Three interpolation methods will be introduced and compared in this section. The methods investigated are the bilinear interpolation, the nearest neighbour interpolation, and the Lanczos interpolation. Using a computer simulation, an attenuation map can be generated directly on the plane of the detector. This map can then be used to evaluate the interpolation techniques using the error definition explained in Section 5.5.1.

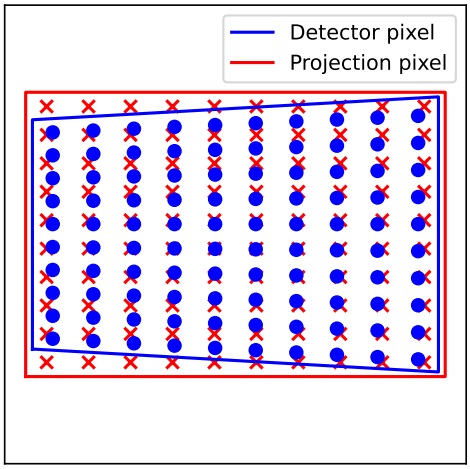


Figure 5.12: Illustration of how the true location of the detector pixels vary compared to the projection pixels.

5.5.1. Error definition

In order to evaluate a simulation result, the error can be computed based on a reference map. Consider a reference attenuation map and a candidate attenuation map, both consisting of N rows and M columns containing detector pixels. Let $X_{i,j}$ be the attenuation value of the reference pixel in row i and column j. Let $x_{i,j}$ be the attenuation value for the same pixel of the map being compared. P_{99} is the 99th percentile value of the maximum attenuation of the reference map and is used to normalize the data. The usage of percentile normalisation removes outliers. The error $\epsilon_{i,j}$ of a simulated pixel compared to a reference pixel is then defined according to Equation (5.10). This error will be referred to as the normalized difference throughout this thesis.

$$\epsilon_{i,j} = \frac{X_{i,j} - x_{i,j}}{P_{99}} \tag{5.10}$$

The study of perceived image quality, which is about the level of accuracy captured by imaging systems, is an entire field on its own. The human eye is complex and it is difficult to define what criteria a realistic result would meet. As shown in [12], there is no single objective metric which judges the quality of an image well in every situation. For this thesis the simple Mean Absolute Error (MAE), shown in Equation (5.11), was chosen. The MAE provides a single figure to summarize the error of a map based on the 99th percentile attenuation value.

$$MAE = \frac{1}{N \cdot M} \sum_{i=1}^{N} \sum_{j=1}^{M} |\epsilon_{i,j}| \tag{5.11}$$

5.5.2. Interpolation

Figure 5.12 illustrates the variation between detector pixels and the corresponding points targeted by the rays. The part of the detector closest to the model exhibits the least amount of distortion. Detector pixels in this region are relatively well-aligned with the ray targets. The part of the detector further away from the model experiences the most amount of distortion. Here, the difference between the detector pixel and the projection pixel is larger.

The process begins with generating a detector pixel grid by determining the exact positions of the detector pixels on the projection plane. The attenuation at these grid points are then interpolated using the output from the compute engines.

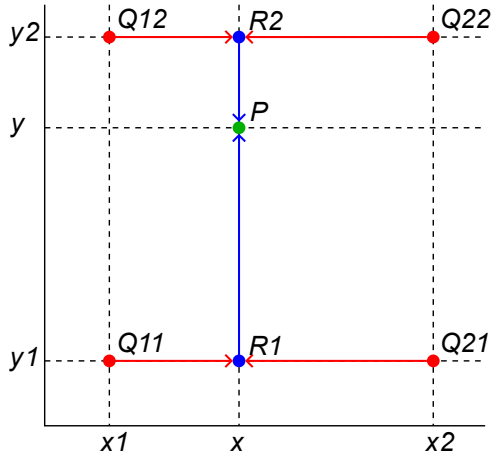


Figure 5.13: Application of the bilinear interpolation algorithm. The green dot is the value to be interpolated. The red dots show the known values [35].

Nearest neighbour interpolation

The nearest neighbour interpolation method finds, as the name implies, the pixel in the projection grid closest to the detector pixel which needs to be interpolated. The detector pixel is assigned the value of this neighbour. The method is computationally efficient and easy to implement, though it is possible that blocky artifacts start to appear [26].

Bilinear interpolation

The bilinear interpolation method can be formulated as a weighted mean of a set of points surrounding the point to be interpolated. Consider a point P as shown in Figure 5.13. The set of points surrounding it are marked as Q_{11}, Q_{12}, Q_{21}, and Q_{22}. If $f(x, y)$ denotes the the interpolated value at (x, y), the bilinear interpolation can be written as Equation (5.12) [35]. The coefficients w_{ij} are defined according to Equation (5.13). Using $f(x, y)$, every point on the detector grid can be interpolated.

$$f(x,y) \approx w_{11}f(Q_{11}) + w_{12}f(Q_{12}) + w_{21}f(Q_{21}) + w_{22}f(Q_{22}) \tag{5.12}$$

$$\begin{aligned} w_{11} &= \frac{(x_2 - x)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} \\ w_{12} &= \frac{(x_2 - x)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} \\ w_{21} &= \frac{(x - x_1)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} \\ w_{22} &= \frac{(x - x_1)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} \end{aligned} \tag{5.13}$$

Lanczos interpolation

Lanczos interpolation is a popular image resampling and resizing technique often used in image processing. The technique is based on the Lanczos kernel, which is a dynamic kernel based around the sinc function which needs to be recomputed for every value to be interpolated. This method is known for its ability to produce high-quality results with minimal artifacts when scaling and transforming images.

The interpolation process involves a convolution filter which is applied to an image. The filter is defined as a sum of $sinc$ functions. It is windowed to limit the influence of far-off pixels and improve computational efficiency. The one-dimensional kernel used for the filter is given by Equation (5.14) [22], where x is the real value to be interpolated and s_i is a real value from the source image. The parameter a dictates the number of lobes of the sinc function which are used. It also defines the size of the kernel.

Table 5.3: Mean and maximum MAE of different interpolation methods for a 0.0° to 22.5° range of horizontal- and vertical rotation.

Method	Mean	Max.
Nearest neighbour	0.25%	0.27%
Bilinear	0.17%	0.19%
Lanczos (a = 4)	0.89%	0.92%
Lanczos (a = 7)	0.46%	0.49%

[22] shows that a kernel size of 2 can lead to ringing artifacts. These did not occur for kernel sizes of 3 and 4. For $a = 4$, most interpolation methods should be outperformed.

$$\begin{aligned} S(x) &= \sum_{\lfloor x \rfloor - a + 1}^{\lfloor x \rfloor + a} s_i L(x - i) \\ L(x) &= \begin{cases} sinc(x)sinc(x/a) & if -a < x < a \\ 0 & otherwise \end{cases} \end{aligned} \tag{5.14}$$

Since Lanczos resampling uses a separable filter, the result in the horizontal direction can be multiplied by the result in the vertical direction in order to achieve a two-dimensional result. This means that the final interpolated value at $S(x, y)$ can be calculated using Equation (5.15).

$$S(x, y) = \sum_{j=\lfloor y \rfloor - a + 1}^{\lfloor y \rfloor + a} \sum_{i=\lfloor x \rfloor - a + 1}^{\lfloor x \rfloor + a} s_i L(x - i) L(y - j) \tag{5.15}$$

Interpolation results

The interpolation methods are compared for the total 22.5 degree range of horizontal- and vertical rotation. For this experiment, the axis of rotation was placed at the centre of the voxel model. For a given rotation, a reference attenuation map is generated first. For this reference, the projected detector pixels are used as targets for the simulated rays. Because the same points are used in the interpolation, their actual value can be calculated this way. This is different from the regular simulation, where the targets are based on the projection points. Based on the reference, the MAE of the interpolated attenuation maps can be determined.

For each interpolation method, both the maximum and average MAE were evaluated, with the results summarized in Table 5.3. Among the tested methods, bilinear interpolation demonstrated the best performance, achieving the lowest error rates. Surprisingly, the Lanczos method performed the worst, with noticeable artifacts at a kernel size of four. While increasing the kernel size slightly reduced the error, it remained higher than that of other methods. It did remove the artifact. Simulation images suggest that the Lanczos method struggles to accurately interpolate high-contrast regions, particularly at the boundary between the skull and air, leading to significant errors in these areas. In contrast, other interpolation techniques handle these transitions more effectively. Given that bilinear interpolation produces the smallest error, averaging just 0.17% of the maximum reference attenuation, it has been selected for the final implementation.

5.5.3. Upsampling

In an effort to improve the computational efficiency of the algorithm, the potential use of upsampling was investigated. The goal was to determine whether simulating a lower-resolution detector and subsequently upsampling it could maintain an acceptable image quality while reducing computational costs. This approach leverages the fact that the computational complexity of the algorithm scales linearly with the number of detector pixels. By simulating fewer pixels, significant time savings could be achieved, provided that the quality of the upsampled image remains comparable to that of a higher-resolution simulation.

Table 5.4: Comparison of average MAE over a range of 0.0 to 22.5 degrees of rotation in the horizontal- and vertical direction for various upsampling methods (upsampled to 512 x 512).

Resolution	r	Nearest Neighbour	Bilinear	Lanczos (a = 7)
512 x 512	1.00	0.25%	0.17%	0.46%
486 x 486	1.10	0.26%	0.17%	0.48%
458 x 458	1.25	0.27%	0.18%	0.49%
428 x 428	1.43	0.28%	0.18%	0.51%
396 x 396	1.67	0.30%	0.19%	0.54%
362 x 362	2.00	0.32%	0.20%	0.57%
323 x 323	2.51	0.35%	0.22%	0.62%
280 x 280	3.34	0.39%	0.24%	0.69%
256 x 256	4.00	0.41%	0.25%	0.74%
228 x 228	5.04	0.45%	0.27%	0.83%

The same interpolation methods from Section 5.5.2 were used to investigate the feasibility of upsampling. First, a reference attenuation map was generated at a resolution of 512 by 512 pixels. Similar to the approach taken in Section 5.5.2, this reference was generated on the detector plane and does not require an interpolation step. Next, attenuation maps were generated at lower resolutions. These resolutions were determined by the upsampling factor, which is defined as $r = \frac{N_{ref}}{N_{up}}$, where r is the upsampling factor, N_{ref} is the number of simulated pixels at the reference resolution, and N_{up} is the number of simulated pixels at the lower resolution which is upsampled. Several factors were tested ranging from one, meaning no upsampling is performed, to five, meaning the original resolution is more than halved.

The quality of the upsampled attenuation maps were quantified using the MAE metric. The maps were generated for a range of 0.0 to 22.5 degrees in both the horizontal- and vertical direction and the MAE compared to the reference was calculated. These errors were averaged in order determine the feasibility of the different upsampling techniques. The average error for the different upsampling factors and methods can be seen in Table 5.4.

All upsampling methods introduce a small error. The error seems to increase as the upsampling factor grows. This result can be expected, because more pixels need to be interpolated. Again, the bilinear method outperforms the other upsampling methods. Because the error introduced using upsampling in the detector step is of the same magnitude compared to no upsampling, it will be considered a viable approach to increasing the performance of the simulation.

6

Memory architecture

This chapter covers the memory architecture of the application. Section 6.1 will discuss the memory present on the Alveo U50 card. Section 6.2 will cover how the requests for voxel lines from the engines are handled. Section 6.3 explains how these requests are answered. Section 6.4 will talk about the layout of the voxel model in memory and Section 6.5 discusses how the voxel model is compressed into the different material values.

6.1. HBM architecture

In order to understand the memory architecture of the system, it is important to understand the physical memory architecture of the Alveo U50 card. This card contains an AXI HBM Memory Controller which provides access to a 1024-bit wide HBM stack. There are two stacks present on the card. The FPGA targeted is referred to as a 4H device, meaning it has two HBM stacks which both have a capacity of 8 GB. The reported bandwidth of a single stack is 201 GB/s [4]. The internal structure of such a stack is visible in Figure 6.1. Each HBM stack is divided into eight independent memory channels, with each channel further subdivided into two 64-bit pseudo channels.

Pseudo channels

The pseudo channels are the lowest subdivision of memory. They divide each memory channel into two independent 64-bit wide buses which allow two separate operations to occur simultaneously within a single channel. This increases the effective utilisation of the bus by reducing the latency and improving parallelism. One pseudo channel contains 1/16th of the stack capacity and is responsible for 1/16th of the total throughput. The pseudo channels are exposed through sixteen AXI ports. Each port operates at a 4:1 clock ratio, meaning a port width of 256 bits is required.

Each port can be configured to provide access to the entire HBM addressing space, referred to as global addressing, but this comes with a latency penalty. To prevent this, the system is setup in such a way that this is not necessary. Each pseudo channel will be managed by a read manager which is responsible for managing the memory access of up to sixteen engines. Since it is known in advance which layers of the model are relevant to these engines, access to different pseudo channels is never required.

6.2. Request manager

A request manager handles requests from up to sixteen engines and translates them into AXI addresses. As shown in Figure 6.2, the read manager receives the requested line and layer from each engine. Since a model can have up to 1024 lines, ten bits are needed to address them. One read manager is in charge of $16 \cdot 8 = 128$ layers, so seven bits are required. This means that a request consists of seventeen bits in total. These requests are stored in first in first out buffers, also known as FIFOs, with each engine having its own FIFO. An engine can have at most one outstanding request, but using FIFOs to store the requests provides flexibility to change this in the future if desired.

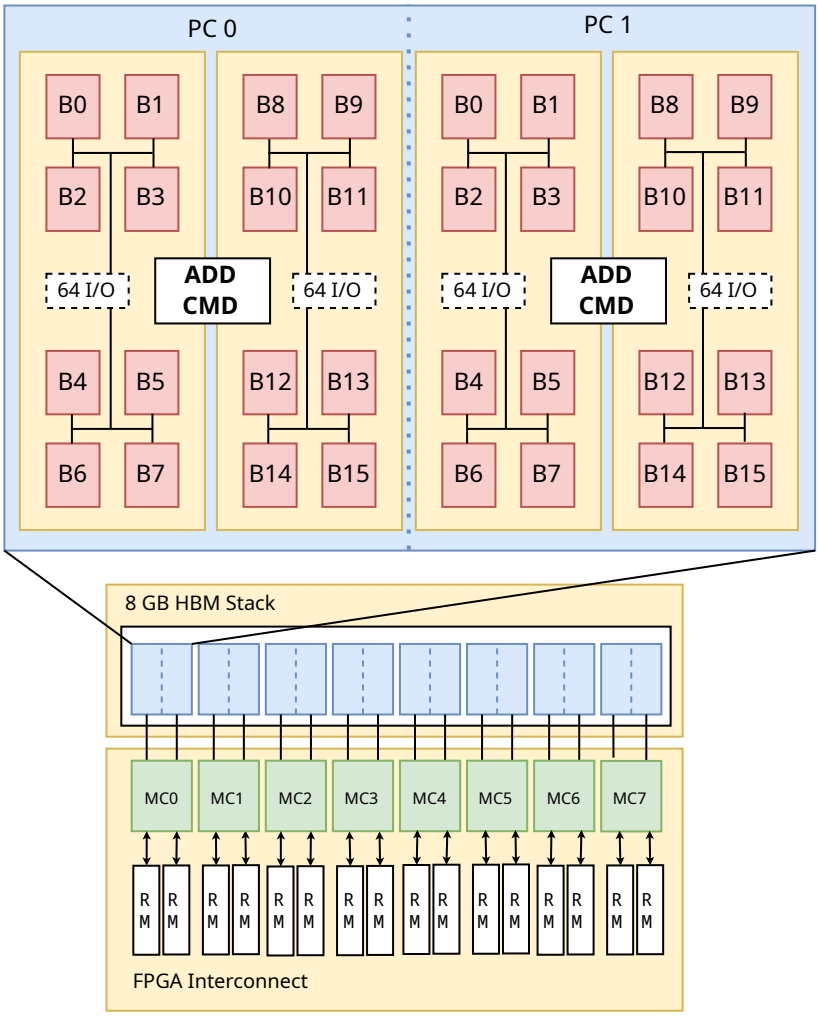


Figure 6.1: Overview of the pseudo channels within the HBM architecture, inspired by [7].

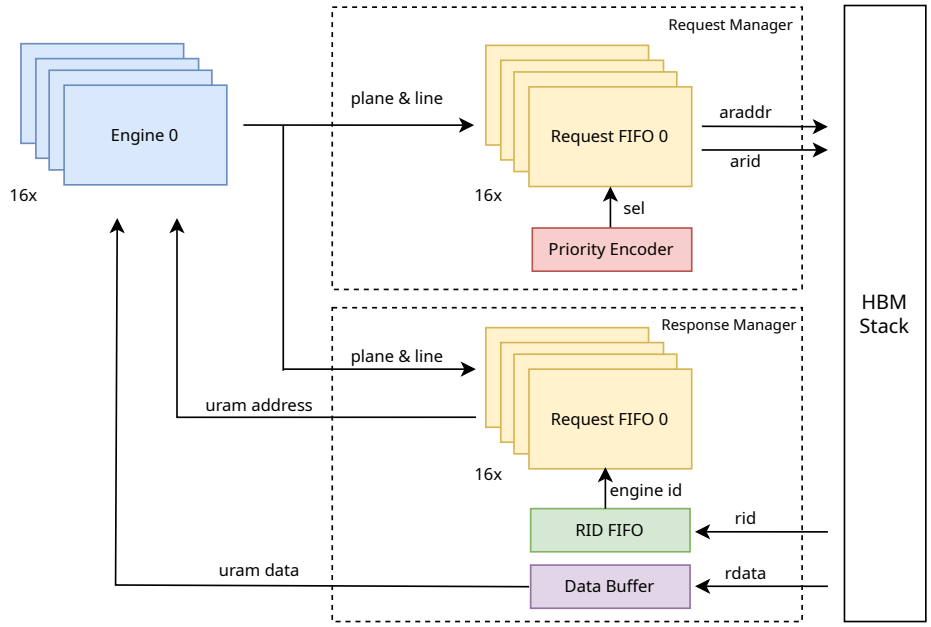


Figure 6.2: Diagram showing how read- and response manager return the voxel data to the engines from the HBM.

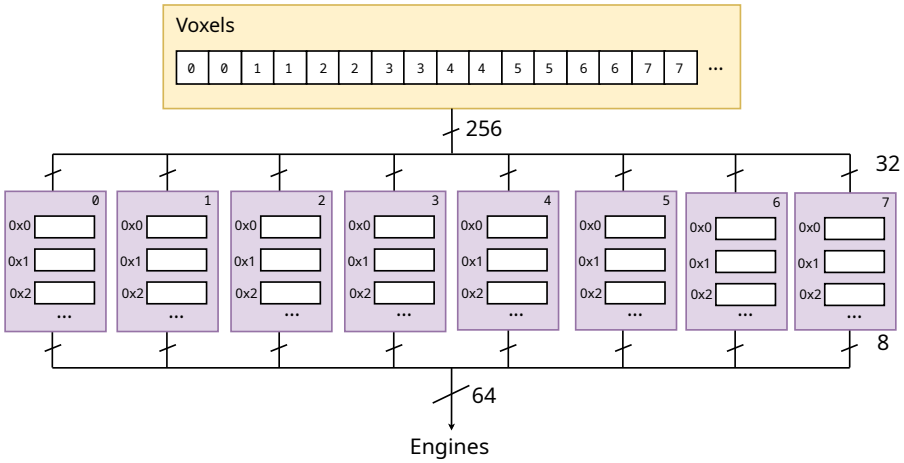


Figure 6.3: Implementation of the memory buffers which convert the 256-bit responses from the HBM to 64-bit blocks for the engines. The numbers in the voxels indicate the buffer number.

Once a request is available, a priority encoder is responsible for deciding which request to handle first. It does this based on a round-robin schedule where each engine has an equal priority. Once a request has been selected it is converted into an AXI address. The read address port, abbreviated as araddr, supplies this address on the HBM interface. Since the memory controller supports multiple outstanding requests, a read address identifier is also provided which can be used to identify the response later. This ”arid” tag is simply the engine number.

6.3. Response manager

The response manager is responsible for handling responses from the HBM memory, transforming bursts of 256 bits into 64-bit responses for the engines. Similar to the request manager, it receives requests from the engines. These are required for computing the URAM destination addresses for the responses. The request- and response manger and their relation to the engines are depicted in Figure 6.2.

Two critical signals managed by the response manager are AXI read identifier and read data signals, abbreviated as ”rid” and ”rdata”. The rid signal can be used to identify the address to which a response belongs, as described in Section 6.2. Through the use of this tag, the initial request can be recovered ensuring the correct data is delivered to the corresponding engine. Since the rid corresponds to the engine number, the request in the corresponding FIFO number is used to compute the URAM address.

Data buffers play a significant role in handling HBM responses, as engines are not capable of managing the high throughput directly. Therefore, the response stream must be managed to prevent bottlenecks or loss of data. As shown in Table 4.1, the response manager must handle up to sixteen 32-byte bursts. Since 32 bytes are returned every cycle, each burst is stored in eight 4-byte wide memory blocks, as illustrated in Figure 6.3.

Voxels need to be returned in the correct order. The data lines are rearranged to facilitate this, since one byte is included from every buffer in the response. Adjacent voxel pairs are therefore placed in adjacent buffers because one byte contains two voxels. This is indicated by the numbers in the voxels in Figure 6.3. Based on the request, a URAM address can be computed which adheres to the engine’s addressing method shown in Table 5.2. The line and layer information is derived from the request, and the buffer index is tracked by the response manager.

6.4. Memory layout

An HBM stack consists of stacks of DDR4 chips. DDR4 memory chips are a type of SDRAM (Synchronous Dynamic Random-Access Memory) that offer higher speed and efficiency compared to previous generations like DDR3.

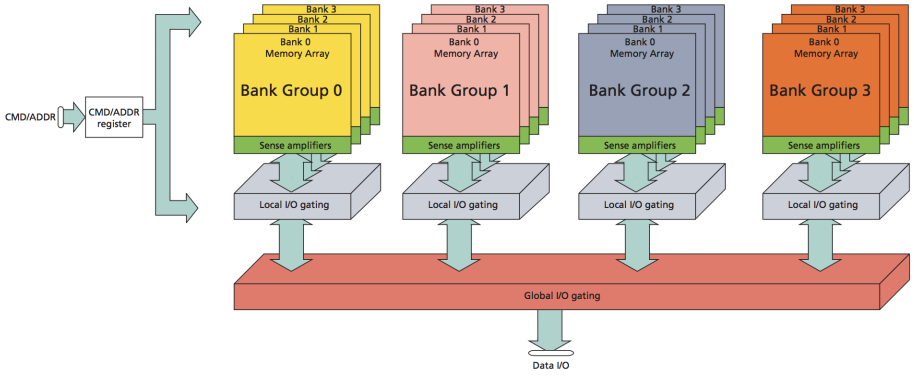


Figure 6.4: Illustration showing the memory bank groups and memory banks used in the DDR4 memory implementation [11].

Memory Banks

Memory banks are subdivisions within a DRAM module that allow for parallel access to different sections of memory. Each bank can be accessed independently, which helps improving memory access efficiency and reducing latency. When a memory request is made, the memory controller reads the row of data into the sense amplifier of the corresponding bank leaving the others untouched. This allows having simultaneous rows of data open in each bank.

Bank Groups

Bank groups are a feature introduced in DDR4 to further enhance memory performance. They were introduced to lower the amount of words that need to be prefetched for each memory access. The term prefetch describes how many words are read for each column command. Memory requested from banks within the same group can be accessed quicker, resulting in memory that can be managed more efficiently, reducing contention and improving overall performance. A pseudo channel consists of four bank groups, each containing four banks. A representation of the internal structure can be seen in Figure 6.4.

AXI address

A custom address map can be created which maps the AXI address bits to the physical ports on the HBM interface. An AXI address is 29 bits, of which the most significant bit is always zero. The lowest five bits should also be zero because of the 32 byte alignment requirement for the AXI port. If this is ignored, multiple memory commands are required for a single read operation, decreasing performance by more than 50%. An AXI address consists of the following physical ports:

- **CA[5:1]:** Five column address bits. Column address bit 0 is unused because it is implicitly included to get an internal bus width of 64;
- **RA[13:0]:**Fourteen row address bits;
- **BA[1:0]:** Two bank address bits;
- **BG[1:0]:** Two bank group address bits.

Reading from memory consists of precharging a row and accessing the column. Once a row is read, each column can be accessed with low latency. However, if a different row needs to be accessed within the same bank, a new precharge command is required. There is a large latency associated with this. The HBM supports one active row per bank, with protocol access times between banks in different bank groups being lower compared to accesses within the same bank group. The currently active row within a bank must be precharged before a different row within that bank can be activated. It is recommended to perform multiple column accesses within an activated row before changing the row, as this results in a high page hit rate and higher efficiency [5].

The memory supports a DDR4 option called bank group interleaving. With this option enabled, the memory controller commands can be executed on two bank groups in parallel, alleviating some latency associated with the protocol. This is because, while one bank is being accessed after a row has been

precharged, a different bank can precharged with the same command [2]. To use this option the least significant bid of the AXI address must be mapped to the least significant bank group bit.

Because the application is memory bound, a memory layout tailored to the access pattern of the application is important to achieve the highest possible bandwidth. This layout will differ slightly between model sizes due to the different sizes in data transfer. First, the memory access pattern of the application will be discussed. This pattern does not differ between model sizes. Next, a layout which supports this pattern will be provided for each model size.

Access pattern

Before an address map can be created, the memory access pattern must be understood. To do this, the worst possible case for the memory access pattern is considered. This is analogous to a detector which projection covers the entire model, and the source being infinitely far way from the detector resulting in the rays travelling in parallel to each other. This is the worst possible case because every voxel of the model will need to be read in this scenario. Based on this situation, three important observations can be made:

- **Observation 1:** Lines at the front of the model are accessed before lines at the back of the model;
- **Observation 2:** In the simulation, lines at the same height are likely to be accessed at the same time. This is especially true for layers which are close to each other;
- **Observation 3:** Each line within a layer only needs to be accessed once.

Based on these observations and knowledge about the memory architecture the following strategy can be constructed:

1. The line number should decide the row within a memory bank, because once a line has been accessed for every layer, it will never have to be precharged again, preventing the associated latency;
2. The layer number should decide the memory bank and column within a row. This way, accessing similar lines leave the row untouched. This prevent additional precharge commands.
3. Bursts should stay within the same page of memory, because this can be processed faster by the memory controller.

6.4.1. Small model

For the smallest model, which is 384 voxels wide, a burst length of six is required to read an entire line of voxels. This means that the three least significant bits of the AXI address cannot be mapped to a line or layer since they are implicitly incremented by the memory controller on every read.

Address map

The goal of the address map is to minimise the response time for engine requests. This is done by minimising the amount of precharge commands from the memory controller. To achieve this, the address map in Table 6.1 is proposed. This map utilises the bank group interleaving option by placing the BG0 bit in the least significant bit position. The three least significant layer bits are assigned to the remaining column bits. This means that a given line is stored in the same row for every layer within an engine.

Using this memory layout, a read bandwidth of 12.069 GB/s is achieved. This means approximately 96% of the maximum throughput is achieved.

6.4.2. Medium model

For the medium model, which is 512 voxels wide, a burst length of eight is required to read an entire line of voxels. This means that the four least significant bits of the AXI address cannot be mapped to a line or layer since they are implicitly incremented by the memory controller on every read.

Address map

The address map for the medium model is similar to the previous address map. However, because an additional bit is required to accommodate the burst length of eight, an extra column bit is required. The

Table 6.1: Table showing the address map for the small model (384 x 297 x 384).

AXI address bit	Physical address bit	Model address bit
5	BG0	0
7 - 6	CA2 - CA1	0
10 - 8	CA5 - CA3	layer[2] - layer[0]
11	BA0	layer[3]
12	BA1	layer[4]
13	BG1	layer[5]
14	RA0	layer[6]
24 - 15	RA10 - RA1	line[9] - line[0]
28 - 25	NA	0

Table 6.2: Table showing the address map for the medium model (512 x 512 x 512).

AXI address bit	Physical address bit	Model address bit
5	BG0	0
8 - 6	CA3 - CA1	0
10 - 9	CA5 - CA4	layer[1] - layer[0]
11	BA0	layer[2]
12	BA1	layer[3]
13	BG1	layer[4]
15 - 14	RA1 - RA0	layer[6] - layer[5]
24 - 15	RA10 - RA2	line[9] - line[0]
28 - 25	NA	0

resulting address map can be seen in Table 6.2.

Using this memory layout, a read bandwidth of 12.114 GB/s is achieved. This means approximately 96% of the maximum throughput is achieved, which is comparable to the smallest model.

6.4.3. Large model

For the largest model, which is 1024 voxels wide, a burst length of sixteen is required to read an entire line of voxels. This means that, similar to the medium model, the four least significant bits of the AXI address cannot be mapped to a line or layer since they are implicitly incremented by the memory controller on every read.

Address map

Initially the same layout for the medium model was attempted, because the number of bits required for the burst is the same. However, tests showed that this layout did not perform as well as for the other model. It is difficult to say what caused this drop in performance. This is because the implementation is encrypted and cannot be inspected. It is possible that the effects of timing calibration performed by modern memory controllers are more noticeable due to the longer transaction lengths. Switching memory bank more frequently resolved the drop in performance. This was done by leaving an additional column bit untouched. The resulting address map can be seen in Table 6.3.

Using this memory layout, a read bandwidth of 12.088 GB/s is achieved. This means approximately 96% of the maximum throughput is achieved. This is the same as the other model sizes.

6.5. Model compression

As explained Section 3.1, it is infeasible to store the model in the original float representation due to its size and the impracticality of the required arithmetic on the FPGA. Therefore, the model is compressed into a four bit value used as an index into a lookup table of sixteen material values. These material values represent attenuation coefficients. It is important that the material values are chosen carefully. Incorrect values can lead to significant errors in the simulated attenuation map. For example, when

Table 6.3: Table showing the address map for the large model (1024 x 1024 x 1024).

AXI address bit	Physical address bit	Model address bit
5	BG0	0
9 - 6	CA4 - CA1	0
10	CA5	layer[0]
11	BA0	layer[1]
12	BA1	layer[2]
13	BG1	layer[3]
16 - 14	RA2 - RA0	layer[6] - layer[4]
24 - 17	RA10 - RA3	line[9] - line[0]
28 - 25	NA	0

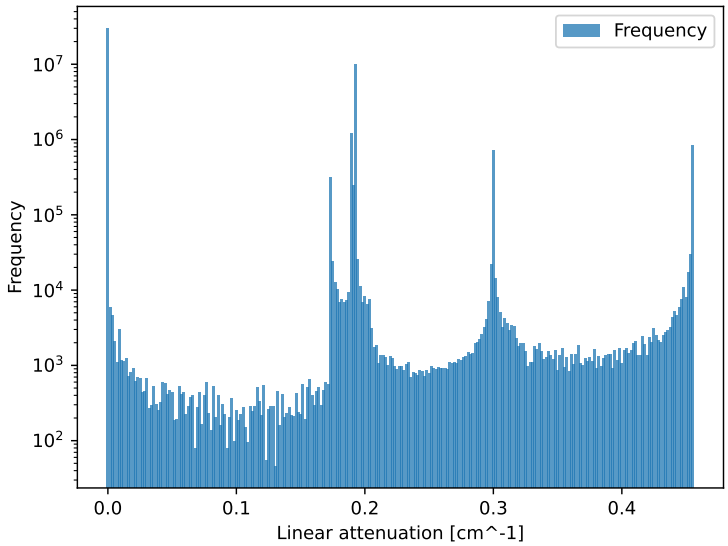


Figure 6.5: Histogram showing the frequency of linear attenuation coefficients in the skull model.

voxels indicating bone are misrepresented the skull can appear to be thicker or thinner than is actually the case. When tissue is misrepresented, subtle features of the brain might no longer be visible.

For the original voxel model, 256 different attenuation coefficients are considered. In order to determine the material values used in the simulation, the model needs to be compressed into sixteen different values. This can be done through binning. Linear attenuation coefficients which are roughly equivalent are grouped and assigned a single value. The material is represented by this single value.

A naive decision, referred to as strategy 0, would be to chose sixteen equally sized bins. This does not work, because the linear attenuation coefficients are not distributed uniformly across all voxels. There is a high variance in the number at which they occur. This is shown by Figure 6.5, which depicts the log of the frequency.

As Figure 6.5 shows, the most occurring voxels are either air, dense bone, or tissue with a linear attenuation coefficient of 0.194 cm^{-1}. Bins which contain these voxels must be narrow, since misrepresenting their value would result in a large error. However, for the other bins it is not very obvious. In order to determine the different bins, two strategies are explored and compared.

6.5.1. Error function

Before the strategies can be compared, the error for any given assignment needs to be defined. In the original voxel model 256 values are considered. These are represented by x_i, where $i \in \{0, 1, ..., 255\}$.

A certain value x_i occurs c_i times, where $i \in \{0, 1, ..., 255\}$. Next, consider some function $f(i)$ which assigns an attenuation value in x_i to X_j with $j \in \{0, 1, ..., 15\}$. X_j represents a bin. The number of voxels in such a bin is simply the sum of voxels assigned to these bins, defined in Equation (6.1).

$$C_j = \sum_{i \in f^{-1}(j)} c_i, j \in \{0, 1, ..., 15\} \tag{6.1}$$

The value corresponding to a bin is defined as the average value of voxels in this bin, which is shown in Equation (6.2)

$$\mu_j = \frac{\sum_{i \in f^{-1}(j)} c_i x_i}{C_j}, j \in \{0, 1, ..., 15\} \tag{6.2}$$

For a single voxel, the error is equal to the absolute difference between the original value and the newly assigned value. The error of a bin is then equal to the sum of errors of each voxel in this bin. The total error is the sum of the bin errors. This is expressed in Equation (6.3).

$$\begin{aligned} e_j &= \sum_{i \in f^{-1}(j)} |x_i c_i - X_j c_i|, j \in \{0, 1, ..., 15\} \\ E &= \sum_{j \in \{0,1,...,15\}} e_j \end{aligned} \tag{6.3}$$

6.5.2. Strategy 1

For the first strategy, bins where created which contain an approximately equal amount of voxels. This is not entirely possible due to some frequency peaks as shown in Figure 6.5. These peaks where assigned their own narrow bin. By definition, the error in these bins is low because the assigned value is close or even equal to the original value.

Once the peaks are assigned, the remaining voxels between the peaks can be assigned to approximately equally sized bins. Note that this was done manually, making the strategy not easily repeatable. Figure 6.6 shows the resulting bins created by this strategy. The red lines indicate the boundary of the range of a certain bin. Table 6.4 shows both the material value as well as the error based on this assignment. The materials are assigned in increasing order of attenuation.

6.5.3. Strategy 2

The second strategy, inspired by [37], seeks to minimize the total bin error, as defined in Equation (6.3), by optimizing the material assignment. This approach is valid because the assignment must be optimal for every subset of voxels. While minimizing the average bin error might yield results similar to the first strategy, where frequency peaks are assigned their own narrow bins, such an approach can lead to an unnecessarily high error because frequently occurring voxels are prioritised. It is not able to compensate the error in one bin using the value of another. In contrast, strategy 2 minimizes the difference between the actual model and the compressed voxel model by treating all voxels equally. Consequently, the sum of linear attenuation coefficients experienced by each X-ray beam is as accurate as possible, hopefully ensuring a highly precise attenuation map.

The algorithm implementing this strategy is a random search-based optimization algorithm, as outlined in Listing 6.1. It begins with an initialization step, where the bins are assigned equal widths, and the error for this configuration is computed. This serves as the initial global best assignment. The optimization proceeds iteratively, aiming to refine the bin assignment to minimize the error.

During each iteration, the algorithm preserves the current best assignment from all previous iterations. Within the iteration, a loop generates new candidate assignments by randomly varying the widths of the bins in the current best assignment. Some bins may remain unchanged, ensuring a diverse set of potential solutions. For each iteration, the 20 best assignments are stored as well. These are used during the current iteration to randomly generate new assignments.

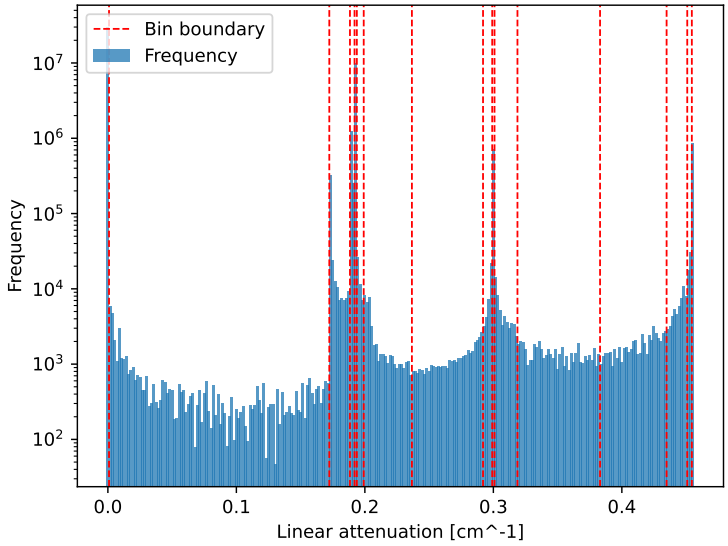


Figure 6.6: Histogram showing the resulting bins based on strategy 1.

Table 6.4: Table showing the resulting assignment of strategy 1.

Material Index	Range [cm^{-1}]	Voxel count	Material value [cm^{-1}]	Bin error [cm^{-1}]
0	0.000 - 0.002	2.9×10^7	0.0	0.000
1	0.002 - 0.173	5.2×10^4	0.055	2532.429
2	0.173 - 0.189	4.0×10^5	0.174	915.633
3	0.189 - 0.193	1.5×10^6	0.189	832.453
4	0.193 - 0.194	9.9×10^6	0.193	0.000
5	0.194 - 0.200	4.5×10^4	0.196	38.080
6	0.200 - 0.237	4.5×10^4	0.210	401.809
7	0.237 - 0.293	3.6×10^4	0.270	515.530
8	0.293 - 0.300	3.6×10^4	0.297	51.302
9	0.300 - 0.302	7.2×10^5	0.300	0.000
10	0.302 - 0.320	5.1×10^4	0.307	225.892
11	0.320 - 0.384	4.9×10^4	0.349	806.583
12	0.384 - 0.436	5.1×10^4	0.413	653.446
13	0.436 - 0.452	5.3×10^4	0.445	194.670
14	0.452 - 0.455	4.8×10^4	0.453	0.000
15	> 0.455	8.5×10^5	0.455	30.418
Total:	-	-	-	**7198.251**

Listing 6.1: Pseudo code for strategy 2

```
#1. Initialize best configuration
global best assignment = equally spaced material bin set; # strategy 0
global best error = error of the equally spaced material bin set;

#2. Iterative optimisation
repeat until the iteration limit has been reached:
  # Preserve current best
  current best assignments = global best assignment; # list initialised with single item

  # Optimisation loop
  repeat 2000 times:
    load previous best assignments from current iteration;
    generate new assignments by randomly changing previous best assignments;
    compute the error for each assignment;
    update current best assignments so it includes 20 best assignments of current iteration;

  # Update global best
  if lowest error of the current 20 best assignments is lower than the best error:
    update global best assignment and error;
```

After completing the optimization loop, the algorithm updates the global best assignment if the lowest error among the current iteration's top 20 assignments is lower than the previous global best error. This process repeats until the iteration limit is reached, continuously refining the bin assignments.

In order to prevent reaching a local minimum, the algorithm was run several times. From these trials the best result was chosen. Figure 6.7 shows the resulting bins created by this strategy. Table 6.4 shows both the material value as well as the error based on this assignment.

6.5.4. Comparison

Before the different material values produced by the two strategies are compared, let us first consider strategy 0 introduced in Section 6.5 as a baseline. This strategy choses bins of equal width. The computed total error produced by such an assignment is 14085.262 cm^{-1}. The total error produced by strategy 1 is 7198.251 cm^{-1}. The total error produced by strategy 2 is 4136.402 cm^{-1}.

Both strategies are an improvement on the baseline. However, they produce vastly different bins. Strategy 1 produces eight bins which are relatively narrow, having a width of less than 0.01 cm^{-1}. For strategy 2, there are only three narrow bins. The error in these bins is low due to the accurate representation of the original value. In the case of strategy 1, the other bins suffer because of this. Table 6.6 shows this. The materials corresponding to narrow bins have a low error per voxel, defined as the total bin error divided by the number of voxels. For the other materials, their error is relatively high compared to strategy 2.

Based on both the total error as well as the average error per voxel, strategy 2 performs better and is therefore chosen for the implementation.

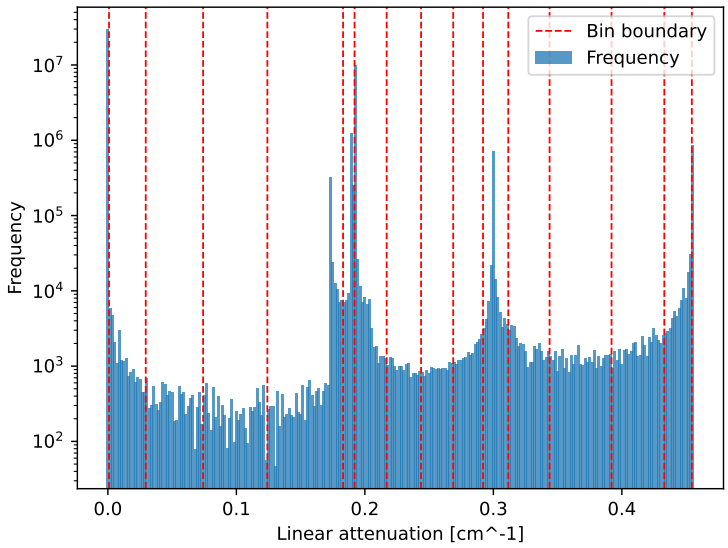


Figure 6.7: Histogram showing the resulting bins based on strategy 2.

Table 6.5: Table showing the resulting assignment of strategy 2.

Material Index	Range [cm^{-1}]	Voxel count	Material value [cm^{-1}]	Total bin error [cm^{-1}]
0	0.000 - 0.002	2.9×10^7	0.000	0.000
1	0.002 - 0.025	2.4×10^4	0.008	123.280
2	0.025 - 0.070	9.9×10^3	0.044	106.798
3	0.070 - 0.123	8.5×10^3	0.095	126.632
4	0.123 - 0.182	3.8×10^5	0.173	418.627
5	0.182 - 0.191	1.3×10^6	0.189	250.295
6	0.191 - 0.216	1.0×10^7	0.193	1001.675
7	0.216 - 0.239	1.3×10^4	0.226	77.437
8	0.239 - 0.270	1.5×10^4	0.254	114.187
9	0.270 - 0.291	1.8×10^4	0.281	94.756
10	0.291 - 0.311	7.9×10^5	0.300	294.401
11	0.311 - 0.348	4.0×10^4	0.325	392.743
12	0.348 - 0.395	3.2×10^4	0.370	376.297
13	0.395 - 0.432	3.9×10^4	0.414	360.225
14	0.432 - 0.454	7.5×10^4	0.445	368.629
15	> 0.454	8.5×10^5	0.455	30.419
Total:	-	-	-	**4136.402**

Table 6.6: Comparison of the error per voxel for the different strategies.

Material Index	Voxel error strategy 1 [cm^{-1}]	Voxel error strategy 2 [cm^{-1}]
0	0.00	0.00
1	4.87×10^{-2}	5.12×10^{-3}
2	2.27×10^{-3}	1.08×10^{-2}
3	5.63×10^{-4}	1.50×10^{-2}
4	0.00	1.10×10^{-3}
5	8.54×10^{-4}	1.99×10^{-4}
6	8.83×10^{-3}	9.80×10^{-5}
7	1.41×10^{-2}	5.84×10^{-3}
8	1.41×10^{-3}	7.50×10^{-3}
9	0.00	5.32×10^{-3}
10	4.46×10^{-3}	3.72×10^{-4}
11	1.64×10^{-2}	9.86×10^{-3}
12	1.27×10^{-2}	1.16×10^{-2}
13	3.70×10^{-3}	9.31×10^{-3}
14	0.00	3.44×10^{-5}
15	3.44×10^{-5}	4.89×10^{-3}
Average:	$\mathbf{7.13 \times 10^{-3}}$	$\mathbf{5.44 \times 10^{-3}}$

7

Results and analysis

In this chapter the simulation results are discussed. Section 7.1 will study the effect of the material choice on the outcome of the simulation. Section 7.2 will compare the results produced by the FPGA simulation with a CPU based simulation. Section 7.3 will analyse the performance of the FPGA simulation for several configurations. Finally, Section 7.4 will compare an upsampled simulation result with a reference in order to evaluate the validity of the method.

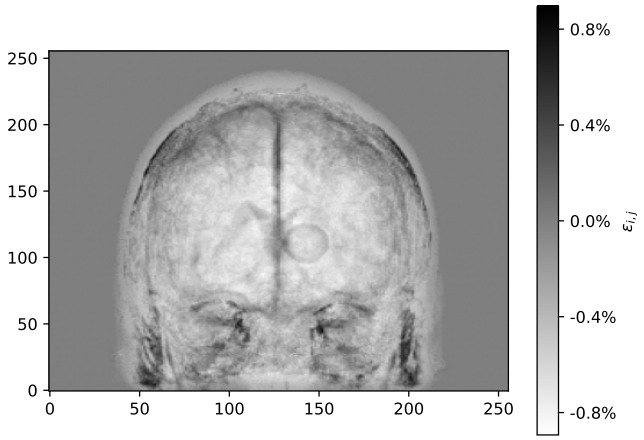
7.1. Material error

To study the effect of the material selection on the attenuation map, three material sets described in Section 6.5 were compared. The material set based on strategy 0 was created by dividing the attenuation coefficient range into 16 equally spaced materials. A second material set was generated using strategy 1, which attempts to include a roughly equal number of voxels in each material. The third material set was based on the results of strategy 2, which tries to optimize the material set through a random search algorithm. Attenuation maps were generated for each material set using a CPU-based simulation under identical configurations. These maps were compared to a reference attenuation map created using an uncompressed model which represents the attenuation coefficients using floating point values. Based on the error defined in Section 5.5.1 the effectiveness of each material selection strategy was evaluated.

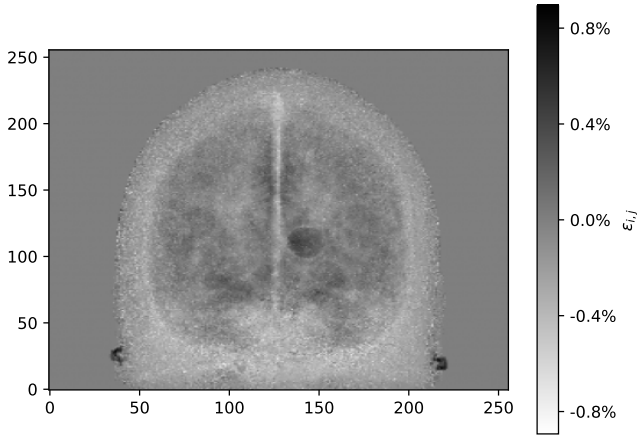
Figure 7.1 shows the normalized difference in attenuation for each material selection strategy. Both the strategy 0 material set and the strategy 1 set produce regions of higher attenuation compared to the reference. For the strategy 0 set, the maximum error is 0.9% of the maximum attenuation, while for strategy 1, this increases slightly to 1.0%. For strategy 2, this maximum error is only 0.1%. All three strategies result in areas with lower attenuation than the reference, with the maximum error in these regions being approximately 0.9% lower for each strategy.

Based on visual inspection, the strategy 0 material set performs the worst among the three strategies. Many regions show a significantly different attenuation compared to the reference, including both high-density materials such as bone and lower-density materials like tissue. For strategy 1 and strategy 2, the differences appear less pronounced. Based on the MAE shown in Table 7.1, strategy 1 performs better.

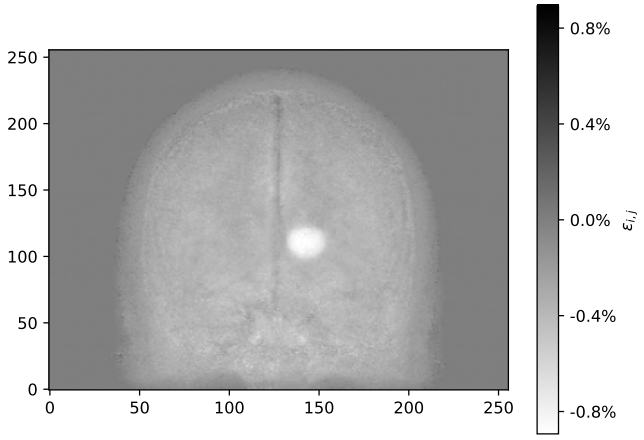
This result aligns with the total bin errors defined in Section 6.5. The attenuation map is largely influenced by the brain due to the coverage on the attenuation map. An inspection of the model shows that most of the tissue within this area falls within the attenuation coefficient range of 0.189 cm^{-1} to 0.194 cm^{-1}, with the majority being 0.194 cm^{-1}. strategy 1 represents this tissue more accurately than strategy 2 as shown in Table 6.6. Furthermore, both attenuation maps show a high-contrast cluster near the centre of the brain, corresponding to voxels with an attenuation coefficient of approximately 0.205 cm^{-1}. This cluster turned out to be a cerebral edema, which is a type of swelling in the brain which causes a slight increase in radiation absorption [23]. Strategy 1 assigns this cluster a coefficient of 0.210 cm^{-1}, which is higher than the true value. However, this is still closer to the true value com-



(a) Strategy 0



(b) Strategy 1



(c) Strategy 2

Figure 7.1: Comparison of the error $\epsilon_{i,j}$ for three different material assignments relative to a reference attenuation map generated using an uncompressed model.

Table 7.1: Mean absolute error for different material choices.

Material choice	MAE
Strategy 0	0.33%
Strategy 1	0.16%
Strategy 2	0.30%

pared to the coefficient assigned by strategy 2, which is 0.193 cm^{-1}. Even though strategy 2 has a lower total bin error compared to strategy 1, the results show that this does not necessarily lead to a more accurate attenuation map.

These results indicate that there is likely no universal solution that benefits all scenarios. Should the user care more about the accurate representation of tissue, this consideration must be made when deciding on the material assignment. If the focus lies with studying bone, more details can be put in that range of voxels. Neither of the current strategies is able to accurately image the edema well. A strategy which is able to do this would be ideal, since this implies that such a strategy would be able to incorporate special features or details from the voxel model. In the end, it is up to the user to decide the level of detail they want to portray in different ranges. Future work can further study the relation between the chosen materials and the resulting attenuation map. In case high tissue attenuation resolution is needed, increasing the number of materials is also still an option, though this comes with a performance penalty.

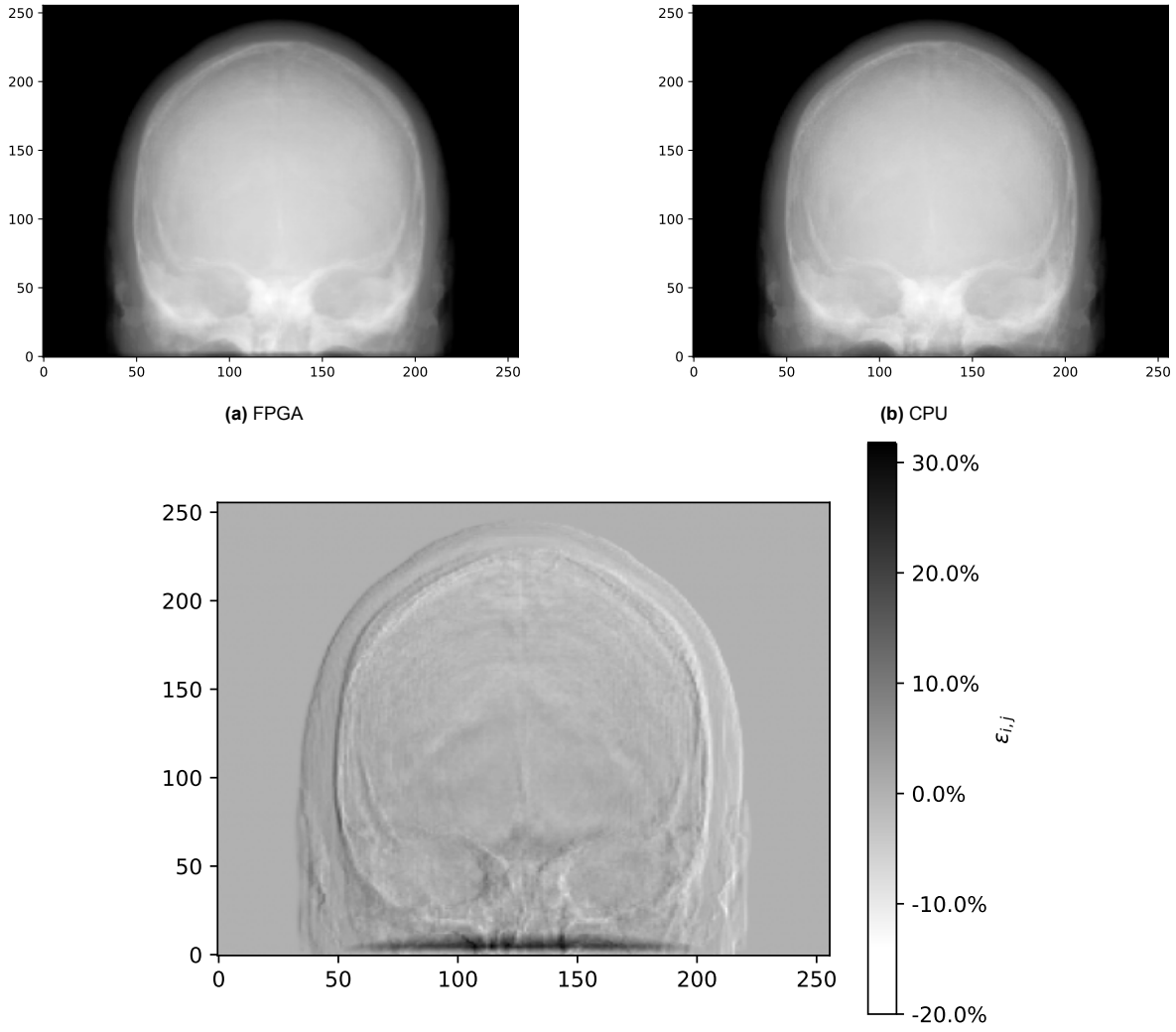
7.2. CPU- and FPGA-based result comparison

To validate the results of FPGA-based simulation, it is important to compare it with a CPU-based simulations. While FPGA-based simulations introduce several sources of error, as discussed throughout this thesis, many of these errors are not present in a CPU-based simulation. An example of this is the type of arithmetic used. The CPU simulation uses single-precision floating-point arithmetic, whereas the FPGA simulation uses fixed-point arithmetic. Additionally, a CPU-based simulations does not require a detector step, since rays can be generated that directly target the detector pixels. Both simulations use the same compressed model as a fair comparison.

The outcomes of a simulation without rotation applied are shown in Figure 7.2. Figure 7.2b provides the reference results from the CPU-based simulation, while Figure 7.2a displays the corresponding FPGA-based simulation. The results from both methods appear to be nearly identical, with the exception of a noticeable area of pixels at the bottom of the FPGA-based simulation. In this area, the map produced by the FPGA contains higher attenuation values compared to the map produced by the CPU. These observations are confirmed by Figure 7.2c, which shows the error according to the definition in Section 5.5.1. A similar comparison was made for a simulation with 15-degrees of rotation applied around the vertical axis. Figure 7.3b shows the reference CPU-based simulation and Figure 7.3a shows the corresponding FPGA-based simulation. The same large error is present at the bottom of the map, which is visible in Figure 7.3c.

The significant error at the bottom of the attenuation maps likely does not have a single cause. Instead, it is probably a culmination of the various error sources, which are most significant near the edges. These sources of error include inaccuracies in ray length estimation and the errors introduced by fixed-point precision. Additionally, it could also be a simulation artifact caused by the fact that the voxel model cuts off this area of the skull, resulting in a perfect two-dimensional plane of high attenuation. This would not be representative of a real head. Ignoring the bottom of the map, the remaining error is the highest near the edge of the skull. The attenuation map produced by the FPGA appears to be slightly shifted towards the left compared to the CPU map. This can be deduced from the error maps by observing that on the left, there is a positive error which means the attenuation computed by the FPGA is higher here compared to the reference. On the right the inverse is true, indicated by the negative error.

The bilinear interpolation method appears to interpolate the high contrast areas well. This is suggested by the top of the skull, which contains a relatively low error. This area would not suffer from the shift towards the left. Despite the areas with a large error, the MAE of the attenuation map without any



(a) FPGA

(b) CPU

(c) Normalized difference between **(a)** and **(b)**.

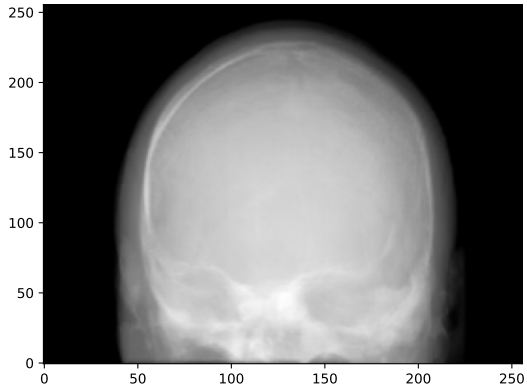
Figure 7.2: Comparison between CPU simulation and FPGA simulation ($\theta = 0°$, $\phi = 0°$).

rotation applied is only 2.26% of the maximum attenuation, excluding areas with no attenuation which are represented by the black pixels. With rotation applied, the MAE is approximately 3.00% of the maximum reference attenuation. This level of error is considered acceptable for the purposes of the simulation.

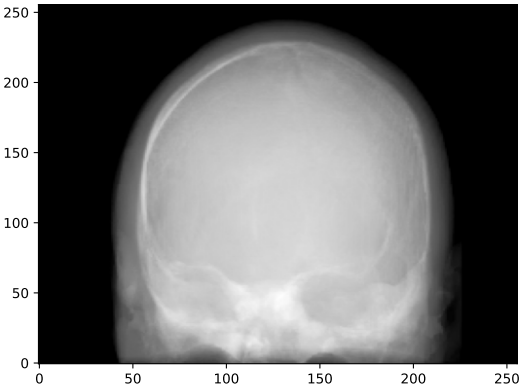
7.3. FPGA performance

To determine whether an FPGA-based simulation is a feasible replacement for a CPU-based simulation, it is important to measure its performance. The duration of a single simulation run is influenced by several factors. As noted earlier, the resolution of the detector is a significant factor, as it directly determines the number of rays that must be simulated. Additionally, the size of the model also has an impact. This is not due to the increased number of layers that need to be traversed, as the addition of more engines compensates for this effect. Instead, the primary factor is the increased memory transfer required for larger models. More voxels must be loaded from memory to process a single line, which is a costly operation.

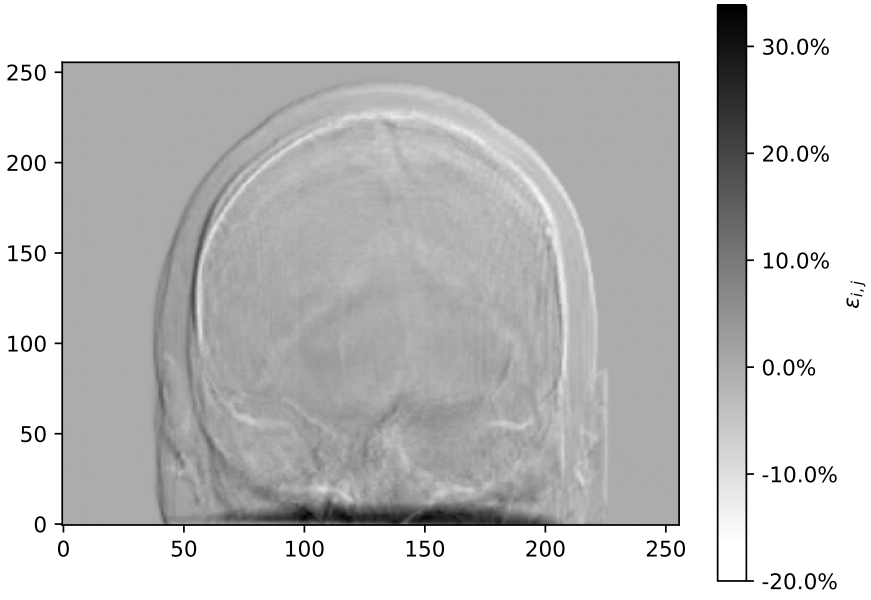
Furthermore, the relative positioning of the source and detector with respect to the model also impacts the simulation time. When the components are closer together, the simulated X-ray beam becomes wider, causing more lines in the model to be intersected. This results in an increase in memory trans-



(a) FPGA



(b) CPU



(c) Normalized difference between **(a)** and **(b)**.

Figure 7.3: Comparison between CPU simulation and FPGA simulation ($\theta = 15°$)

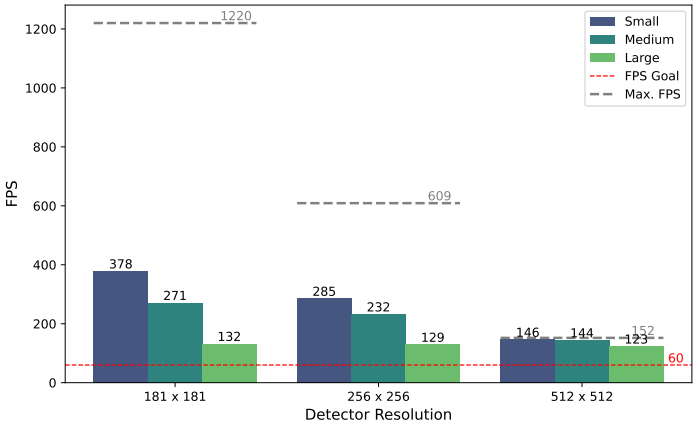


Figure 7.4: Achieved performance for different model- and detector sizes. In the simulation the detector was placed at the maximum distance from the model.

fer affecting the performance. To evaluate the speed of the simulation, first the theoretical maximum number of frames is presented. Next, the runtime of simulations are presented.

7.3.1. Theoretical performance

The theoretical maximum number of frames possible, first described in [19], can be calculated using Equation (7.1). The cycles per image, or CPI, is calculated by determining how many clock cycles it will take to process each ray. The first ray will arrive after the engines have completed their first computations. This is equal to nine cycles multiplied by the number of engines. After this initial ray, a new ray will arrive at the output every ten cycles. The number of frames per second, or FPS, is equal to the clock frequency of the FPGA divided by the CPI.

$$\begin{aligned} CPI &= N_{engines} \cdot 9 + (n_{pixels} - 1) \cdot 10 \\ FPS &= \frac{f_{FPGA}}{CPI} \end{aligned} \tag{7.1}$$

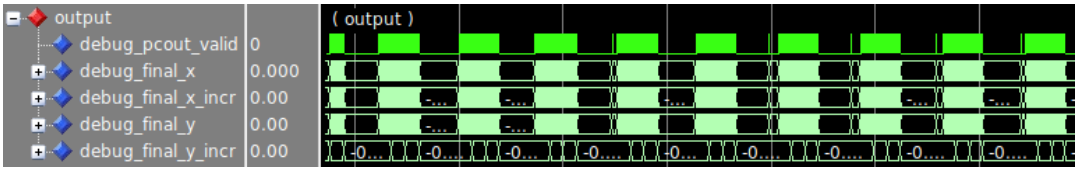
The Alveo U50 card runs at a clock frequency of 400 MHz. Since the number of pixels dominates the CPI, the theoretical maximum is the same independent of the model size. For a detector of 256 x 256 pixel, this is approximately 609 FPS. For a detector of 512 x 512 pixel, this is approximately 152 FPS.

7.3.2. Measured performance

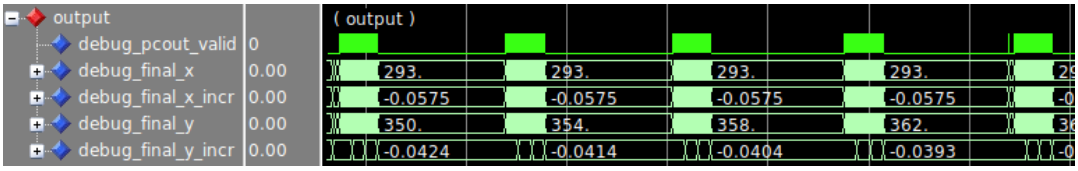
Figure 7.4 shows an overview of the achieved frames per second for different model and detector sizes. The best performance is achieved by the smallest model at the lowest detector resolution, reaching a speed of 378 frames per second. However, this is less than a third of the theoretical maximum at this configuration. For the highest detector resolution, the theoretical maximum is nearly achieved independent of the model size. Contrary to predictions regarding maximum speed, the model size does influence the computation time per frame at the lower detector resolutions, with the time per image increasing for larger model sizes. For the highest detector resolution, the performance remains nearly the same. For every configuration, the goal of 60 frames per second is met.

7.3.3. Analysis

There are two reasons why the theoretical performance is not achieved, both of which are related to the engine buffers used to store the voxel lines. First, Equation (7.1) assumes that the first ray takes nine clock cycles per engine to traverse the model. This is inaccurate, because the time to initialise the buffers is not included in this calculation. To compensate for this, Equation (7.2) can be used to calculate the time it takes for the first ray to arrive at the output. It can also be measured using the simulation waveform, which is more accurate. For the smallest model, the initialisation time is approximately 0.26 ms. For the largest model, it increases to 0.70 ms.



(a) Output of engine chain for the smallest model.



(b) Output of engine chain for the largest model.

Figure 7.5: Waveform showing the output of the engine chain over a span of 0.4 ms.

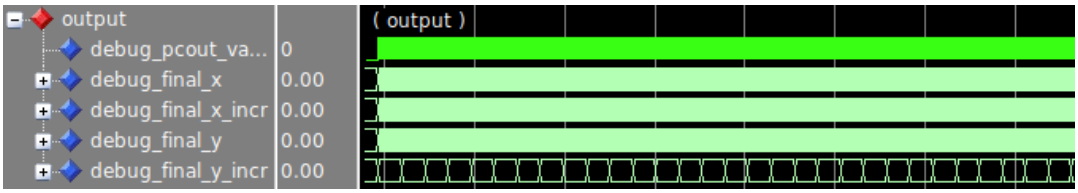


Figure 7.6: Waveform showing the output of the engine chain over a span of 0.4 ms for the smallest model.

$$T_{init} = (T_{request} \cdot N_{buffers} * 8) * N_{engines} \tag{7.2}$$

The second reason the theoretical maximum is not achieved is because Equation (7.1) assumes that only ten clock cycles are required to process a ray. This is only the case if no stalls occur within the engine chain. Stalls happen when rays cannot be send to or received from adjacent engines. The reason for stalls is when voxel lines are required which have not yet been loaded into the engine buffers. During these periods, engines must wait for the requested lines to be fetched, preventing new rays from being processed. These stalls are noticeable at the output of the engine chain. Figure 7.5 shows the output of the chain for both a small and large model. In this waveform, the pcout_valid signal indicates when the result of a ray is ready. As can be seen in both waveforms, there are extended periods where the engines are waiting on new voxel lines, during which no rays are being processed. Because the requests for larger models take longer to complete due to the increase in memory transfer, the stall duration is also increased. This explains the difference in processing times observed for different model sizes.

At the lowest detector resolution, the buffers are processed faster than new ones can be fetched. This is not the case for the larger resolution. The exact number varies depending on the configuration, but quadrupling the resolution means theoretically four times as many rays are processed per line in the buffer, providing the engines with more time to request new lines in advance. Figure 7.6 shows the output of the engine chain in such a scenario for the smallest model. As can be seen in the waveform, stalls no longer occur and rays arrive uninterrupted at the output. For the medium model, similar behaviour can be observed. For the largest model size stalls still occur however, though they are shorter compared to lower resolutions. Still, this lowers the performance. The initial delay is also present at higher resolutions. This explains why the theoretical maximum cannot be achieved.

7.4. Upsampling error

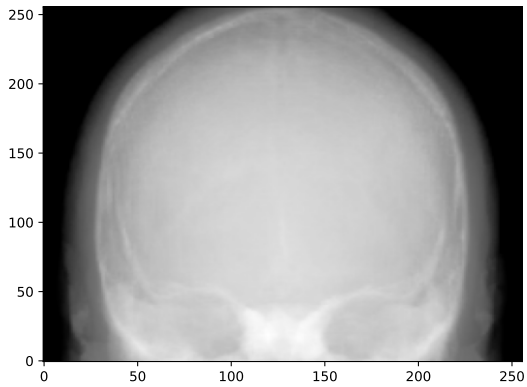
To evaluate the effectiveness of upsampling, two FPGA simulations were compared. First, a reference output was generated using a detector with a resolution of 256 x 256 pixels, positioned relatively close to the model. In the second simulation, a detector with a resolution of 181 x 181 pixels was used, and no rotation was applied. The absence of rotation was not expected to influence the results, as the error introduced by the detection step, when compared to the CPU reference, should be similar for both

simulations. The attenuation map from the lower-resolution detector was then upsampled using the bilinear interpolation method, identified as the most effective method in Section 5.5.3.

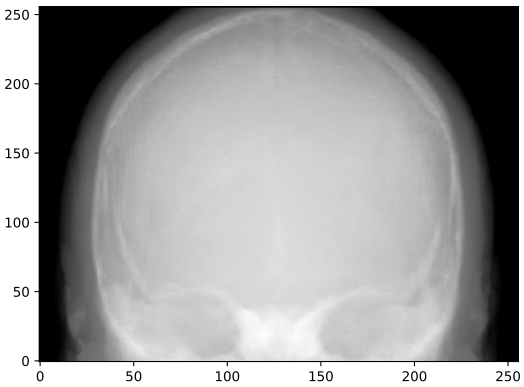
The results are shown in Figure 7.7. The attenuation map at a detector resolution of 181 x 181 was computed in 2.64 ms. At a resolution of 256 x 256, the map was computed in 3.50 ms. This means that a speed up of 33% was achieved for this configuration. The two attenuation maps appear to be nearly identical. The maximum error is 6% of the maximum attenuation. Similar to other results, the largest errors occur in high-contrast areas near the edge of the skull. These differences are attributed to the interpolation method. However, the MAE is only 0.6%, which is a good result. This outcome means that the simulation time can easily be sped up by 33% for only a small increase in the error. At higher resolution detectors, the introduced error will be even smaller. This is true because the differences in distance between the pixels will be smaller, meaning that more of the same voxels will have been interacted with.

While the upsampling approach demonstrates promising results, there are some downsides to consider. The largest errors, up to 6% of the maximum attenuation, are concentrated in high-contrast regions such as the edges of the skull. If the highest accuracy possible must be maintained in these areas, it is better to increase the detector resolution.

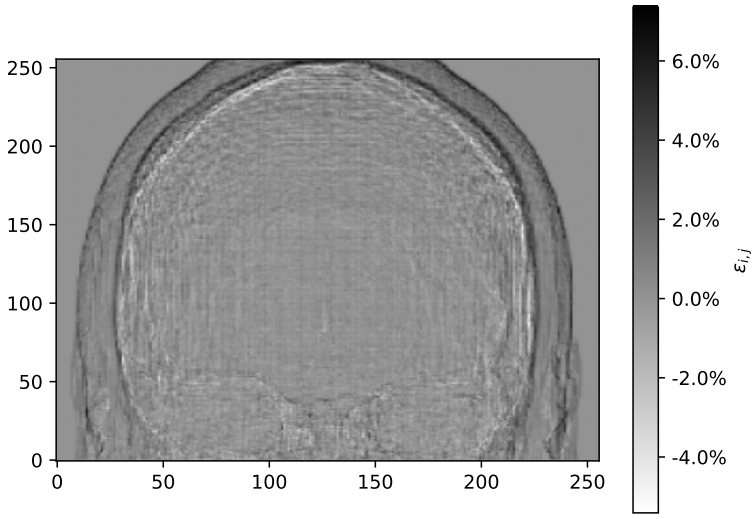
Furthermore, upsampling can introduce an increase in noise in a map, which is an undesirable side effect particularly for machine learning-based applications. Increased noise can reduce the accuracy of classifiers. Additionally, one application of synthetic X-ray images is the testing and improvement of reconstruction methods used in computed tomography (CT). Higher noise levels can degrade image quality, negatively impacting the performance of these reconstruction algorithms [30]. Therefore, for applications that require minimal noise, it is recommended to avoid the use of upsampling techniques.



(a) Upsampled FPGA attenuation map simulated at 181 x 181.



(b) Reference FPGA attenuation map simulated at 256 x 256.



(c) Normalized difference between **(a)** and **(b)**.

Figure 7.7: Comparison between a reference attenuation map and an upsampled attenuation map.

8

Conclusion and recommendations

This thesis addresses the challenges involved in generating synthetic X-ray images for medical purposes. The goal was to produce images which are as close as possible to their real counterpart at the speed of existing imaging systems. This was realised using an FPGA-based hardware accelerator which can simulate the traversal of rays through a voxel model in order to compute an attenuation map which can be used to determine the intensities of the X-rays on a detector. First, an outline of the simulation algorithm was provided. This served as a basis for understanding the computations involved in the process. Next, the architecture required was carefully studied. A separation was made first between the parts of the algorithm that would be implemented on the host machine and what would be implemented on the FPGA. This is possible since not all parts of the algorithm are timing critical. Afterwards, the FPGA implementation was discussed. This included both the hardware components as well as the memory architecture.

8.1. Conclusion

An analysis of the results showed that realistic images can definitely be produced by the simulation. Depending on the amount of rotation involved, attenuation maps can be computed which on average only differ by 2.26% to 3.00% of the maximum attenuation compared to their CPU counterpart. The choice of materials used in the model is also relevant. Based on this selection, the voxel error changes which is visible in the final map. The desired performance is also achieved. The number of frames produced per second is mainly dependant on the detector resolution, since this directly determines the number of rays simulated. If stalling occurs in the compute engine chain however, the size of the model is also a relevant factor due to the increased stall duration caused by larger memory transfers. Performance can be increased through upsampling with a minor 0.6% increase in the error if desired.

This research contributed to the field of synthetic X-ray image generation by studying the system architecture required for such tasks. Established methods are reliant on machine learning techniques which are non-deterministic and unpredictable. A voxel-based simulation is flexible and provides a lot of control over the output. This research has shown that a set of accelerator kernels can be utilised in order to speed up computations otherwise executed on a CPU. In combination with some hardware to support these kernels, real-time generation of images is definitely possible.

The memory architecture required was based on conclusions from earlier work. It quickly became apparent that a specialised memory architecture was required in order to achieve the desired data throughput. To meet this goal, the Xilinx High Bandwidth Memory architecture was carefully studied and optimised. This research has shown that many parallel channels benefit the accelerator kernels and that the maximum throughput can almost be achieved if the access pattern is optimised for the chip access times.

To determine the maximum performance achievable using FPGAs, hardware simulations were studied. FPGAs are a great tool which can effectively be used for parallel work. The use-case discussed throughout this thesis fits the paradigm perfectly, since many accelerator kernels that are configured

as a chain are used for parallel processing. This results in a setup which, dependent on the simulation parameters, can produce images at a speed ranging from 123 frames per second to 378 frames per second. This result satisfies the desired speed of 60 frames per second and is definitely enough for real-time synthetic image generation.

8.2. Recommendations

Future work should focus on validation on real hardware. While the system architecture has been tested extensively in simulation, unknown issues always emerge when implemented on physical hardware. The communication between the host and the FPGA will also have to be implemented. Based on the trajectory of the source and the model, the voxel models which are needed in the future must be predicted. This requires some changes to the algorithms.

Another opportunity for future work lies with assessing the effects of simplifications used in the simulation. The first is related to the source, which is modelled to produce a monochromatic beam. The simplification could be studied by simulating multiple energy levels and combining the results into a composite map. The second simplification involves the path of the ray through a model layer. The current approach assumes that each ray intersects only a single voxel per layer. This is not true and modifications can be studied which can account for contributions from multiple voxels per layer. Lastly, the compression of the model leads to a loss in detail. Not only can the choice of materials further be investigated, but the inclusion of more materials can also be studied. Both may lead to an improved material assignment strategy which better suits the end user of the produced synthetic images.

Finally, incorporating more realistic effects into the simulation is still a possibility. Currently, sources of noise such as radiation noise and scatter are not included. Furthermore, real-world imaging artifacts such as beam hardening and Compton scattering are still absent from the simulated results. Including these effects could aid in the development of methods to mitigate their impact on image quality in real-life.

References

[1] URL: https://www.philips.co.uk/c-dam/b2bhc/gb/resource-catalog/landing/brightontender/philips-azurion-bi-plane-specifications-7b20-15.pdf.

[2] *10.4.3. Bank Interleaving*. URL: https://www.intel.com/content/www/us/en/docs/programmable/683216/23-1-2-7-0/bank-interleaving.html.

[3] Shiras Abdurahman et al. "Beam Hardening Correction Using Cone Beam Consistency Conditions". In: *IEEE Transactions on Medical Imaging* 37.10 (2018), pp. 2266–2277. DOI: 10.1109/TMI.2018.2840343.

[4] *Alveo U50 Data Center Accelerator Card Data Sheet*. DS965. v1.2. Xilinx. Nov. 2019.

[5] AMD. *AXI High Bandwidth Memory Controller LogiCORE IP Product Guide (PG276)*. 2024. URL: https://docs.amd.com/r/en-US/pg276-axi-hbm (visited on 12/09/2024).

[6] AMD. *UltraScale Architecture Libraries Guide (UG974)*. 2025. URL: https://docs.amd.com/r/en-US/ug974-vivado-ultrascale-libraries/XPM_CDC_PULSE (visited on 01/17/2025).

[7] Kazi Asifuzzaman et al. "Demystifying the Characteristics of High Bandwidth Memory for Real-Time Systems". In: *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 2021, pp. 1–9. DOI: 10.1109/ICCAD51958.2021.9643473.

[8] Rolf Behling. "X-ray sources: 125 years of developments of this intriguing technology". In: *Physica Medica* 79 (2020). 125 Years of X-Rays, pp. 162–187. ISSN: 1120-1797. DOI: https://doi.org/10.1016/j.ejmp.2020.07.021. URL: https://www.sciencedirect.com/science/article/pii/S1120179720301812.

[9] Giorgio Ciano et al. "A Multi-Stage GAN for Multi-Organ Chest X-ray Image Generation and Segmentation". In: *Mathematics* 9.22 (2021). ISSN: 2227-7390. DOI: 10.3390/math9222896. URL: https://www.mdpi.com/2227-7390/9/22/2896.

[10] C Cromjongh et al. "Hardware-Accelerator Design by Composition". In: (2024).

[11] *DDR4 SDRAM*. Micron, Sept. 2021. URL: https://www.mouser.com/datasheet/2/671/Micron_05092023_8gb_ddr4_sdram-3175546.pdf.

[12] R. Dosselmann. "An evaluation of existing and emerging digital image and video quality metrics." In: Faculty of Graduate Studies and Research, University of Regina, 2006. ISBN: 978-0-494-20204-3.

[13] The Editors of Encyclopaedia Britannica. *Britannica*. 2024. URL: https://www.britannica.com/biography/Wilhelm-Rontgen (visited on 09/17/2024).

[14] Haruki Hattori et al. "Learning Scatter Artifact Correction in Cone-Beam X-Ray CT Using Incomplete Projections with Beam Ho le Array". In: *Journal of Nondestructive Evaluation* 43.3 (Aug. 2024), p. 99. ISSN: 1573-4862. DOI: 10.1007/s10921-024-01113-5. URL: https://doi.org/10.1007/s10921-024-01113-5.

[15] Trang Hoang and Ayush Goel. *Compton effect*. en. Aug. 2014. DOI: 10.53347/rid-30308. URL: http://dx.doi.org/10.53347/rID-30308.

[16] Joost Hoozemans et al. "Frame-based Programming, Stream-Based Processing for Medical Image Processing Applications". In: *Journal of Signal Processing Systems* 91.1 (Jan. 2019), pp. 47–59. ISSN: 1939-8115. DOI: 10.1007/s11265-018-1422-3. URL: https://doi.org/10.1007/s11265-018-1422-3.

[17] Ian Smalley Josh Schneider. *What is a field programmable gate array (FPGA)?* 2024. URL: https://www.ibm.com/think/topics/field-programmable-gate-arrays (visited on 09/18/2024).

[18] Radiology Key. *Projection X-ray imaging*. Aug. 2020. URL: https://radiologykey.com/projection-x-ray-imaging/.

[19] H.J.M.T. Knops. *Hardware acceleration of artificial X-ray image generation*. 2024. URL: https://resolver.tudelft.nl/uuid:f243012c-1380-45d9-ae98-de6211defac1 (visited on 12/24/2024).

[20] Knowino. *Euler's theorem (rotation) — Knowino, an encyclopedia*. [Online; accessed 15-February-2015]. 2011. URL: http://knowino.org/w/index.php?title=Euler%27s_theorem_(rotation)&oldid=6892.

[21] Andreas Maier et al., eds. *Medical Imaging Systems: An Introductory Guide*. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018. ISBN: 9783319965192 9783319965208. URL: http://link.springer.com/10.1007/978-3-319-96520-8 (visited on 09/17/2024).

[22] Thiago Moraes et al. "Medical image interpolation based on 3D Lanczos filtering". In: *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization* 8.3 (Nov. 2019), pp. 294–300. ISSN: 2168-1171. DOI: 10.1080/21681163.2019.1683469. URL: http://dx.doi.org/10.1080/21681163.2019.1683469.

[23] Sara M Nehring, Prasanna Tadi, and Steven Tenny. "Cerebral Edema". en. In: *StatPearls*. Treasure Island (FL): StatPearls Publishing, Jan. 2025.

[24] Denis Prokopenko et al. "Unpaired Synthetic Image Generation in Radiology Using GANs". In: *Artificial Intelligence in Radiation Therapy*. Ed. by Dan Nguyen, Lei Xing, and Steve Jiang. Cham: Springer International Publishing, 2019, pp. 94–101. ISBN: 978-3-030-32486-5.

[25] Dheeraj Punia. *FPGA Design, Architecture and Applications*. 2023. URL: https://www.logic-fruit.com/blog/fpga/fpga-design-architecture-and-applications/ (visited on 02/03/2025).

[26] Olivier Rukundo and Hanqiang Cao. "Nearest Neighbor Value Interpolation". In: *CoRR* abs/1211.1768 (2012). arXiv: 1211.1768. URL: http://arxiv.org/abs/1211.1768.

[27] Daniel B Russakoff et al. "Fast generation of digitally reconstructed radiographs using attenuation fields with application to 2D-3D image registration". en. In: *IEEE Trans Med Imaging* 24.11 (Nov. 2005), pp. 1441–1454.

[28] James Seibert and John Boone. "X-Ray Imaging Physics for Nuclear Medicine Technologists. Part 2: X-Ray Interactions and Image Formation". In: *Journal of nuclear medicine technology* 33 (Apr. 2005), pp. 3–18.

[29] Jakob Spoerk et al. "High-performance GPU-based rendering for real-time, rigid 2D/3D-image registration and motion prediction in radiation oncology". In: *Zeitschrift für Medizinische Physik* 22.1 (2012), pp. 13–20. ISSN: 0939-3889. DOI: https://doi.org/10.1016/j.zemedi.2011.06.002. URL: https://www.sciencedirect.com/science/article/pii/S0939388911000651.

[30] Wolfram Stiller. "Basics of iterative reconstruction methods in computed tomography: A vendor-independent overview". In: *European Journal of Radiology* 109 (2018), pp. 147–154. ISSN: 0720-048X. DOI: https://doi.org/10.1016/j.ejrad.2018.10.025. URL: https://www.sciencedirect.com/science/article/pii/S0720048X18303747.

[31] Dawood Tafti and Christopher V Maani. "X-ray production". en. In: *StatPearls*. Treasure Island (FL): StatPearls Publishing, Jan. 2025.

[32] *Tasti: Application-Tailored Synthetic Image Generation*. Jan. 2025. URL: https://tasti-project.eu/.

[33] Gábor János Tornai, György Cserey, and Ion Pappas. "Fast DRR generation for 2D to 3D registration on GPUs". en. In: *Med Phys* 39.8 (Aug. 2012), pp. 4795–4799.

[34] Trixell. *Detector Matrix*. [Online; accessed 30-January-2025]. URL: https://www.trixell.com/technology.

[35] Wikipedia. *Bilinear interpolation — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Bilinear%20interpolation&oldid=1250588270. [Online; accessed 09-December-2024]. 2024.

[36] Wikipedia. *Rotation matrix — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Rotation%20matrix&oldid=1250574232. [Online; accessed 08-December-2024]. 2024.

[37] Zelda B. Zabinsky. “Random Search Algorithms”. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Ltd, 2011. ISBN: 9780470400531. DOI: https://doi.org/10.1002/9780470400531.eorms0704. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470400531.eorms0704. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470400531.eorms0704.