GPU Monte Carlo scatter calculations for Cone Beam Computed Tomography

JONAS ADLER

Master of Science Thesis
Stockholm, Sweden 2014

GPU Monte Carlo scatter calculations for Cone Beam Computed Tomography

J O N A S A D L E R

Master's Thesis in Scientific Computing (30 ECTS credits)
Master Programme in Mathematics (120 credits)
Royal Institute of Technology year 2014
Supervisor at Elekta was Markus Eriksson
Supervisor at KTH was Michael Hanke
Examiner was Michael Hanke

TRITA-MAT-E 2014:05
ISRN-KTH/MAT/E--14/05--SE

Royal Institute of Technology
School of Engineering Sciences

KTH SCI
SE-100 44 Stockholm, Sweden

URL: www.kth.se/sci

Abstract

A GPU Monte Carlo code for x-ray photon transport has been implemented and extensively tested. The code is intended for scatter compensation of cone beam computed tomography images.

The code was tested to agree with other well known codes within 5% for a set of simple scenarios. The scatter compensation was also tested using an artificial head phantom. The errors in the reconstructed Hounsfield values were reduced by approximately 70%.

Several variance reduction methods have been tested, although most were found infeasible on GPUs. The code is nonetheless fast, and can simulate approximately $3 \cdot 10^9$ photons per minute on a NVIDIA Quadro 4000 graphics card. With the use of appropriate filtering methods, the code can be used to calculate patient specific scatter distributions for a full CBCT scan in approximately one minute, allowing scatter reduction in clinical applications.

Referat

GPU Monte Carlo spridningsberäkningar för volymtomografi

En GPU Monte Carlo kod för transport av röntgenfotoner har implementerats och utförligt testats. Koden är avsed för spridningskorrektion av CBCT-bilder.

Koden har testats mot PENELOPE och resultaten överenstämmer inom 5% för ett antal enklare geometrier. Koden testades också i en verklig uppställning med ett artificiellt huvud. De resulterande felen i de beräknade Hounsfield-värdena minbskade med ca 70%.

Ett antal variansreduktionstekniker har också testats, men de flesta gav ingen förbättring på GPU. Koden är trots detta avsevÃ¤rt snabb och kan simulera ca $3 \cdot 10^9$ photoner per minut med ett Quadro 4000 grafik-kort. Med hjälp av väl valda filtreringsmetoder kan koden användas för att beräkna patientspecifika spridningsfördelningar för ett fullständigt CBCT-scan på under en minut. Detta är tillräkligt för spridningskorrektion i kliniska tillämpningar.

Contents

List of Figures

List of Tables

List of Abbreviations and Nomenclature

CPU	Central Processing Unit
GPU	Graphics Processing Unit
CBCT	Cone Beam Computed Tomography
MC	Monte Carlo
MFP	Mean Free Path
PS	Phase Space
RNG	Random Number Generator
CUDA	Compute Unified Device Architecture
CURAND	CU(DA) Random Numbers
XORWOW	XOR-shift with Weyl sequence
MT	Mersienne Twister
NIST	National Institute of Standards and Technology
PDF	Probability Density Function
CDF	Cumulative Distribution Function
DCS	Differential Cross Section
SIMD	Single Instruction, Multiple Data
GPGPU	General-Purpose computing on Graphics Processing Units
RR	Russian Roulette
PENELOPE	Penetration and ENErgy LOss of Positrons and Electrons

1 Introduction

Elekta Instrument AB is a medical technology company whose primary product is the Leksell Gamma Knife, shown in Figure 1.1. The Gamma Knife is a radio surgery device primarily used for treatment of brain tumours, intra cranial disorders and vascular diseases. To, among other things, improve the flexibility of the Gamma Knife, Elekta intends to add a Cone Beam Computed Tomography (CBCT) system to the Gamma Knife.

The goal of the work in this thesis has been to improve the image quality of the CBCT.

1.1 Leksell Gamma Knife

The Leksell gamma knife was invented at the Karolinska Institute in Stockholm, Sweden, 1967 and is named after one of its creators, Lars Leksell. The latest instalment of the Gamma Knife is called the Leksell Gamma Knife, Perfexion.

The Gamma Knife is a gamma-radiation based radio-surgery method, where 192 beams of radiation from cobalt-60 sources are focused on the target. The surrounding tissue thus suffers only minor damage in comparison to the area of interest.

In order to deliver an optimal dose to the correct position, the Gamma Knife needs accurate positioning. The current method is to physically attach a special frame to the patient's head and image the patient in a 3D MRI-scan. This 3D-scan gives a precise coordinate system of the patients head. However, this procedure is time consuming and to be able to perform several treatments it would have to be redone every time. Many patients also dislike the prospect of having a frame screwed to their heads.

To make this procedure simpler, the new Gamma Knife model will be delivered with a built in CBCT scanner. This scanner can be used to create a 3D image of the patients head. This can then be mapped to the geometry used in treatment planning where the doctor has decided which areas to treat.

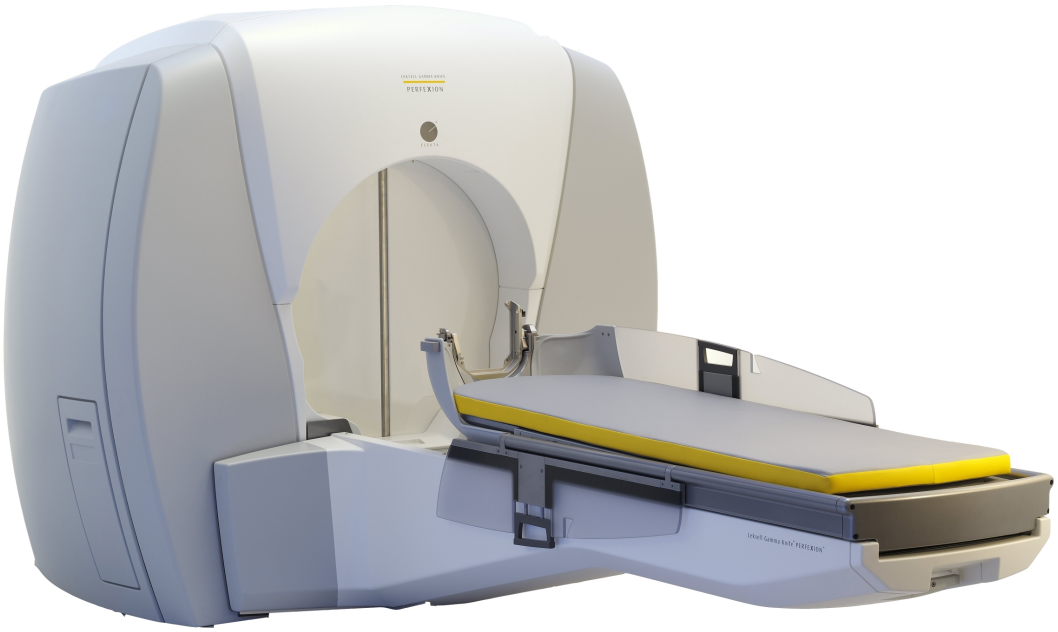


Figure 1.1: The Leksell Gamma Knife, Perfexion. Published with permission from Elekta AB.

1.2 Cone Beam Computed Tomography

CBCT is a modern medical imaging method. During a CBCT scan, the scanner revolves around the patient's head taking a large set of X-ray images. Using these images and computational methods, a 3D volume of the x-ray attenuation coefficients are obtained. CBCT is most popular in implant dentistry and radiotherapy, but has proved useful for other applications. Two of the appealing characteristics are the fast scan time and relatively small scanner size. The design of the CBCT used is shown in figure 1.2, while a schematic view showing the relevant parts is shown in Figure 1.3.

X-rays are used because they have an attenuation length of 2-6 cm in biological matter, which is suitable for clinical applications. The X-ray photons are generated in an X-ray tube (source), and are sent along a path towards the detector. On their way, they may interact with matter by being scattered or absorbed. Some materials, such as bone, have a higher chance of interacting with the photons, and thus fewer photons will be transmitted through them. The detector detects the transmitted (primary) and scattered photon and produces an image.

The most common algorithm used to reconstruct the 3D volume is the Filtered Back-Projection algorithm, developed by Feldkamp in 1984. The algorithm works by projecting the filtered signal backwards from the detector through the volume

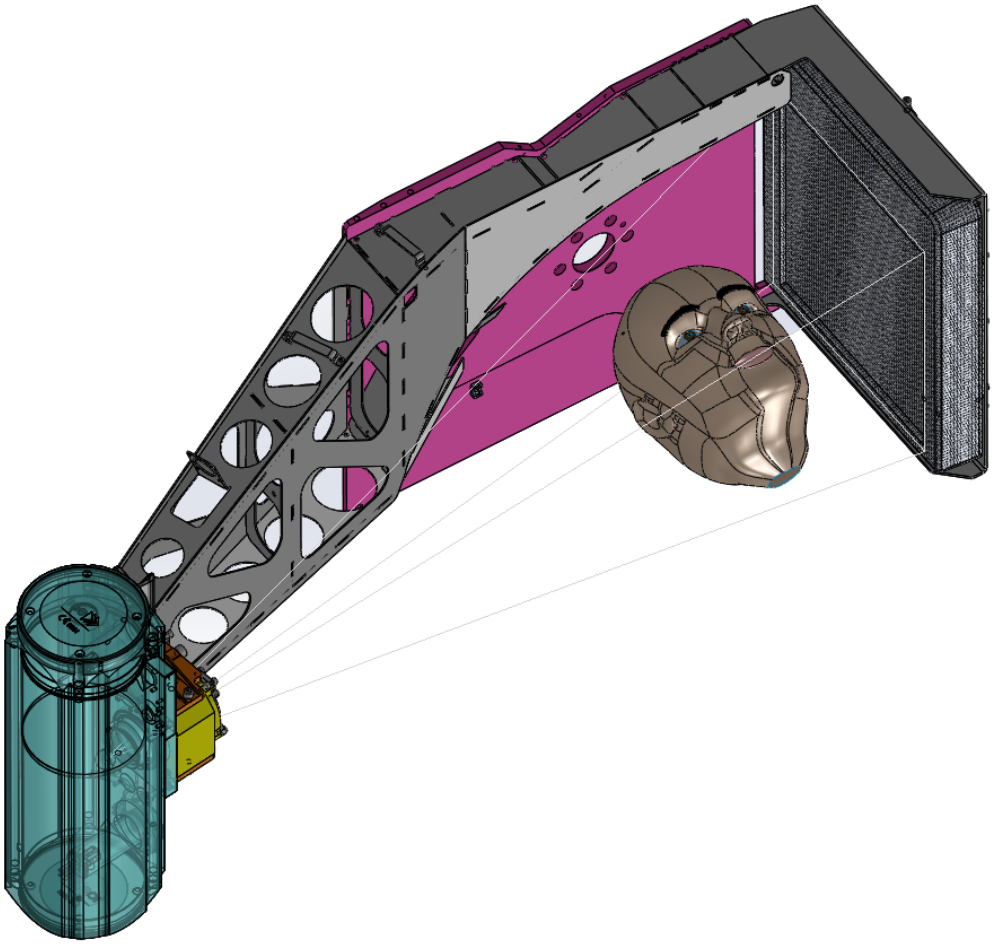


Figure 1.2: Illustration of the CBCT system with a patient. Published with permission from Elekta AB.

to estimate the attenuation at each point in the volume. A full description of the algorithm is available in [1].

1.2.1 Scatter Artefacts in CBCT Images

If the detector only detected the primary photons, the image would be an accurate representation of how much radiation is attenuated along different lines through the phantom. However, due to the scattered photons, this image loses contrast. The problems caused by this are illustrated in Figure 1.4.

The main goal of the work in this thesis was to write a code and related methods that can be used to remove the scattered photons from the X-ray images. The chosen

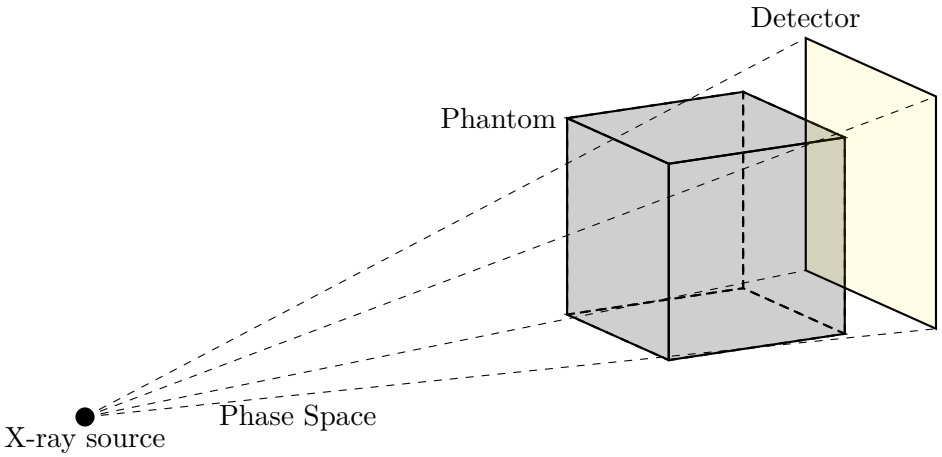


Figure 1.3: Schematic drawing of the CBCT setup showing the main components. The *phantom* is the volume being imaged. The *phase space* is the set of possible photon paths from the X-ray source. Most of these paths are aimed towards the detector. This is illustrated by the dotted lines.

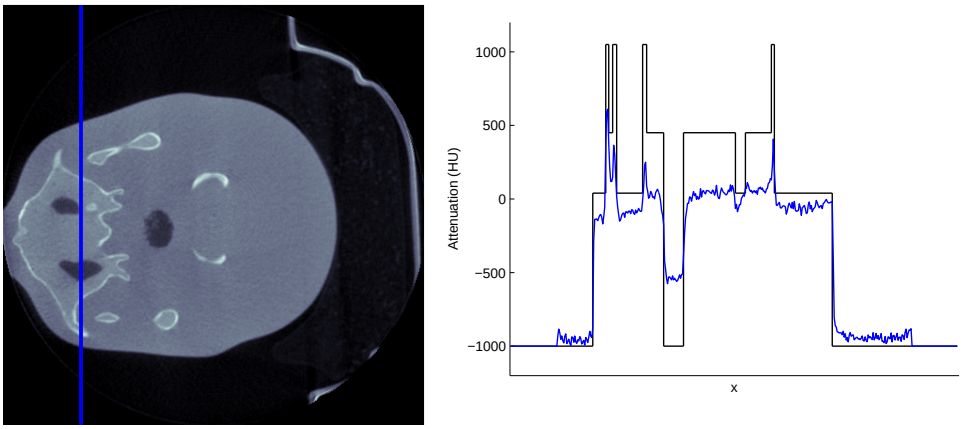


Figure 1.4: Illustration of the artefacts created by scattered photons in the CBCT reconstruction of an artificial (plastic) head. Shown in a traverse cross section in (a). In figure (b), the values along the blue line in (a) are shown. Also shown is the exact values in black. The values shown are in Hounsfield (HU) units, which is an linear scaling of the X-ray attenuation coefficient. We see that the contrast is greatly reduced and that the calculated HU values are far from the correct values. The noise present in the image is not due to scatter but is so called quantum noise caused by the relatively small number of X-ray photons that hit each pixel in the detector.

method is the method developed by Wiegert [2] around 2006. In this method, a first reconstruction of the volume is performed. Using this volume, a large number of photons are simulated using the Monte Carlo (MC) method, and the resulting scatter contribution is calculated. This calculated scatter is then subtracted from the measured images.

The MC method is a method were many samples, in this case individual photons, are taken from a distribution. The sought after quantities, such as the distribution of the scattered photons on the detector, can then be calculated by averaging. This method is regarded as accurate, but it converges relatively slowly.

In Wiegerts work, the simulation of the scatter took 20 days for one scan. Since then, significant technological improvements have been made, and the goal of this thesis is to find a method to perform this simulation in less than five minutes, thus allowing scatter correction for clinical applications.

1.3 Scientific Computation on GPUs

General-Purpose computing on Graphics Processing Units (GPGPU) is a relatively new phenomenon offering developers very high computational power at moderate costs. The idea is to use the hardware developed for commercial graphics applications, such as games, and use it for scientific computations.

Computer graphics has a parallel structure with each pixel calculated being largely independent from the other pixels. This allows a very large number of processing cores to work in parallel. To facilitate more processors on the chip, Graphics Processing Unit (GPU)s sacrifice the ability for each thread to work independently. Instead GPUs use a Single Instruction, Multiple Data (SIMD) architecture. In this architecture, multiple processor performs the same set of instructions, but on different data.

This very parallel processing structure has applications in scientific computing, since many problems are highly parallel. The computational problem in this thesis, MC photon transport, is an example of such a problem.

There are two main frameworks used for GPU computing today, Open Computing Language (OpenCL) and Compute Unified Device Architecture (CUDA). OpenCL is an open source framework, while CUDA is proprietary and owned by Nvidia, the largest supplier of GPUs today. In this work, we have used CUDA, primarily because it is already used in other company products.

1.4 Layout of thesis

In the following chapter, we will delve deeper into the problems related to scatter artifacts in CBCT and present a solution.

In chapter 2, we will discuss the tools used, starting with introducing the CUDA GPU framework. The physics of interest to the problem will also be introduced, and we will discuss what parts are the most relevant to account for. Finally, we will investigate the methods used by other authors to solve these problems and methods they have used to make the program faster without losing accuracy.

In chapter 3 we will look at the implementation in detail. We will begin with discussing how to generate and store the geometry and how to assign properties to the various materials present. Then, we will investigate the implementation of the physical model, with emphasis on the modelling of photon interactions. At the end, some coding details and tricks will be discussed.

In chapter 4 the results of the work will be presented. We first compare the code with another well known code called PENELOPE, then we test the code by simulating a real experiment. Finally, we test the effect of removing the calculated scatter from real images and see how the reconstruction is improved.

In chapter 5 we discuss the results by and problemize what may have gone worse than expected.

In the final chapter, we conclude the findings of the thesis with a summary, and discuss interesting directions of further research in this area.

2 | Background

In this chapter the background of the thesis will be presented. First, a brief introduction to the methods used will be given, with a brief overview of scientific computing on GPUs using Nvidia's CUDA tools. Some of the specific issues related to this approach will also be discussed. The MC method will also be presented and its rate of convergence discussed. The problem of photon transport in matter will also be presented.

Finally, the MC method for photon transport in matter will be presented in depth, with discussions on other relevant work in the area. Many of the more complex sub-problems will also be discussed. Various methods to speed up the method such as variance reduction methods, pre-calculations and filtering methods will be introduced.

2.1 CUDA

The CUDA computational model is a multi-level model. Each processing thread belongs to a warp, which is a set of threads sharing an instruction queue. A set of warps forms a block, and the blocks are laid out in a grid. To enumerate the threads, each thread has a three dimensional blockId, and a three dimensional threadId within the block.

The main difference between CUDA programming and Central Processing Unit (CPU) programming is that outer loops are replaced by calls to device (GPU) side functions. For example, if we want to perform some work on a 2D array of data on a CPU, the code would look like Algorithm 1.

Algorithm 1: CPU array-function

```
1 for i = 1 : n do
2     for j = 1 : m do
3         out(i,j)=complicated_function(in(i,j))
```

In CUDA, we first write a device side function, such as Algorithm 2. This function specifies the work each thread will perform. This method is then called from the host (CPU) side, as in Algorithm 3. The CUDA specific <<<1,blockDim>>> code

ensures that $n \times m$ threads are created with the correct indices. If the number of threads are larger than the maximum number that the hardware can handle, work is queued and the threads perform their work in a successive manner.

Algorithm 2: CUDA device side

```
1 out(i,j)=complicated_function(in(threadId.x,threadId.y))
```

Algorithm 3: CUDA host side

```
1 dim3 blockDim(n, m)
2 deviceSide<<<1,blockDim>>>(in,ref out);
```

Since the CUDA framework uses a SIMD architecture, the code is very sensitive to *branching*. Branching is any occasion of `if` or `switch` or other statements where the code may take different paths. If a single thread in a warp needs to take a different path from the other threads, all threads have to wait for that one thread to finish before they can continue.

The memory structure on an CUDA GPU is also optimized for parallel execution. This is in contrast to CPU memory which is optimized for serial code. Like a CPU, the GPU has a large *global memory*. This memory is often in the order of one to several GB. The global memory has three main subdivisions: the general memory (usually denoted global memory), where any kind of data is stored; the *texture memory*, which is optimized for typical texture access patterns; and the *constant memory*, which is constant in time and can therefore be heavily cache optimized. Access to the global memory is optimized for parallel access, and to achieve optimal performance, continuous blocks of 64 bytes should be read.

Each block of threads also shares an on chip memory. This memory is usually 64 kB and has two subdivisions, *L1 cache* which is used to cache global memory accesses, and a general *shared memory* for general storage. The shared memory is significantly faster to access than global memory. Finally, each thread has its own *registers*, access to the registers is extremely fast.

This memory structure is illustrated in Figure 2.1, which shows the main components of the CUDA architecture. Utilizing these different memory levels optimally is key to a successful GPU algorithm.

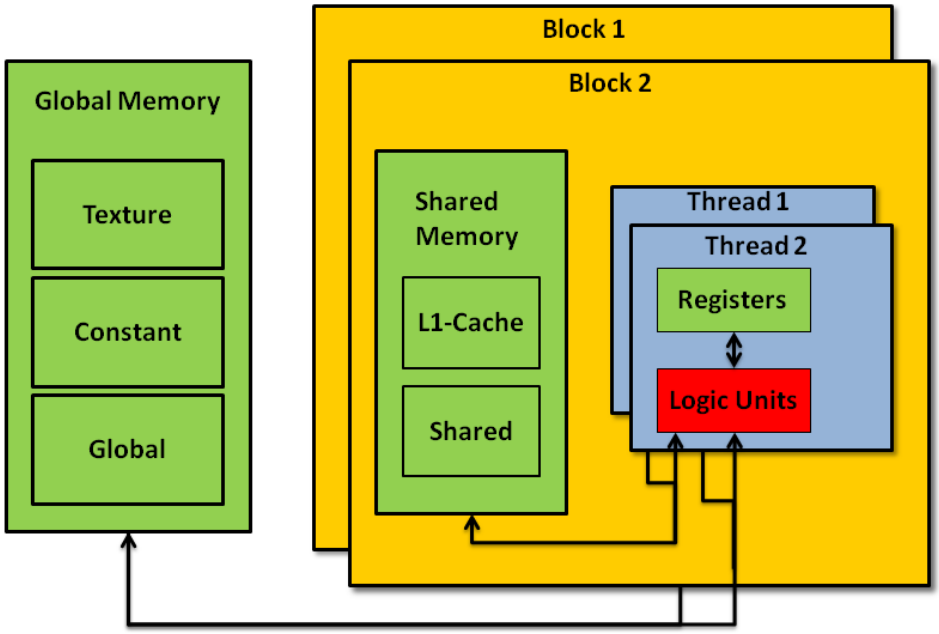


Figure 2.1: Schematic view of the CUDA thread and memory model, showing a simplified model with two blocks with two threads each.

2.2 Monte Carlo Method

The MC method dates back to the second world war and the Manhattan Project, where it was conceived by Stanislaw Ulam as a way to calculate neutron diffusion in radiation shielding. Ulam observed that while the equations governing neutron diffusion were well known, analytic solution methods proved fruitless. The method he instead decided to use was to stochastically simulate multiple neutron paths through the material and average the results to get an estimate of the shielding effect.

In the MC method, a large number of samples are taken from a random distribution governing the problem investigated. For example, if we want to calculate π, we could sample N points in the square $[-1, 1] \times [-1, 1]$ and calculate the fraction of points with norm ≤ 1. This fraction is then an estimate of $\pi/4$.

The MC method converges slowly. For the example above, the distribution of the result for one point will be Bernoulli distributed with $p = \pi/4$. The average of N estimates will thus be Binomially distributed[1]. This gives us the standard error σ

[1]More exactly, $B(N, \pi/4)/N$ distributed

$$\sigma = \sqrt{\frac{\pi(4-\pi)}{16N}} \propto \frac{1}{\sqrt{N}}$$

This result is very general and applies for any case where the point wise distribution is of finite variance, due to the central limit theorem. This convergence is relatively slow, and the method may therefore require a very large number of data points to converge.

For example, the detector used in the CBCT has 720×780 pixels, the probability p for a photon to hit a specific pixel is thus (ignoring interactions and assuming all photons hit the detector) $\approx 1/(720 \times 780)$. The relative standard error of the result is then obtained by division with the mean (μ)

$$\sigma/\mu \approx \sqrt{\frac{p(1-p)}{Np^2}} \approx \sqrt{\frac{720 \times 780}{N}}$$

To achieve a relative standard error of approximately 1%, we can make a rough approximation assuming uniform distribution and independence of all pixels. Then, the number of photons needed (N) is:

$$N \approx \frac{720 \times 780}{0.01^2} \approx 10^{10} \tag{2.1}$$

This is prohibitively large if we require the simulation to take less than a few minutes, and steps needs to be taken to reduce this number for a real time application.

2.3 Photon Transport in Matter

Photon transportation in matter is an old problem, which has been studied in many shapes and forms. A general mathematical model for photon transport is the radiative transfer equation. This equation is highly complicated to solve analytically for even slightly complicated geometries.

When photons travel through media it may interact with the material in ways such as bouncing of molecules or electrons, or by interacting in more complex ways. The 4 major types of photon-matter interactions are listed below.

- Photoelectric absorption
- Compton (inelastic) scattering
- Rayleigh (elastic) scattering
- Pair/triplet production

The probabilities for these different interactions can be calculated using the attenuation coefficient μ. μ is defined as the differential interaction probability

$$\mu = \frac{\mathrm{d}P_{\text{Interaction}}}{\mathrm{d}x}$$

Where $\mathrm{d}x$ is a infinitesimal movement of the photon and $\mathrm{d}P_{\text{Interaction}}$ is the chance that an interaction occurs. In the literature however, it is more common to use μ divided by the density ρ. This is called the mass attenuation coefficient, and is usuallu denoted μ/ρ. The mass attenuation coefficient is popular because it allows easier calculations for compound materials, and allows simple rescaling by density. Values of μ/ρ for various materials have been extensively tabulated by Hubbell *et al.*[3] and are available through the NIST XCOM webpage[4]. Typical values of μ/ρ for water and bone are shown in Figures 2.2 and 2.3. For typical x-ray energies, $E \leq 0.1$ MeV, Pair/Triplet production is irrelevant. At lower energies, $E \leq 0.01$ MeV the photoelectric effect is dominant.

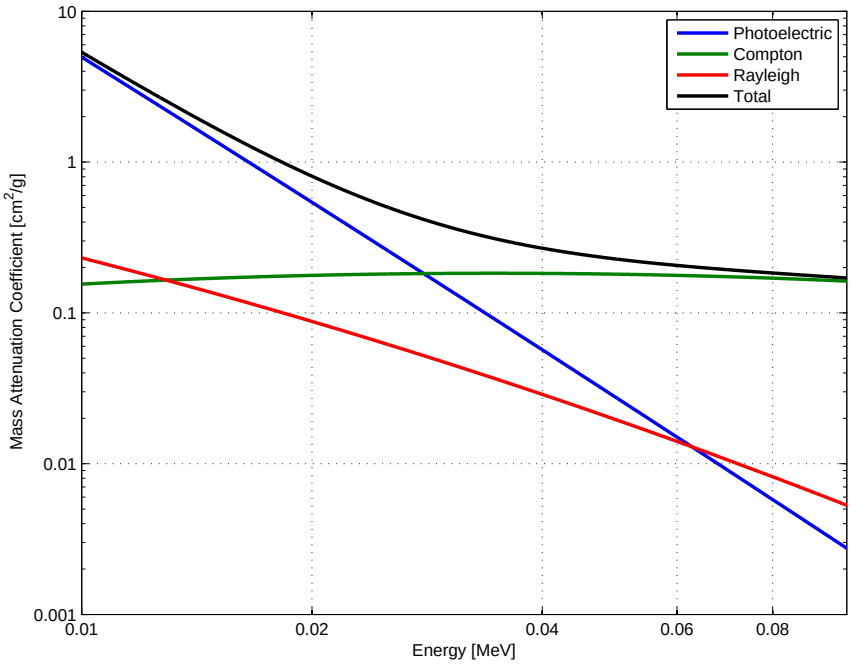


Figure 2.2: Mass Attenuation Coefficient of water at various energies.

After an interaction the direction and energy of the photon will change. The probability of each combination of direction and energy is proportional to the Differential Cross Section (DCS)

$$\frac{\mathrm{d}^2\sigma}{\mathrm{d}\Omega \mathrm{d}E'} = \frac{N_{out,E'}}{N_{in,E}}$$

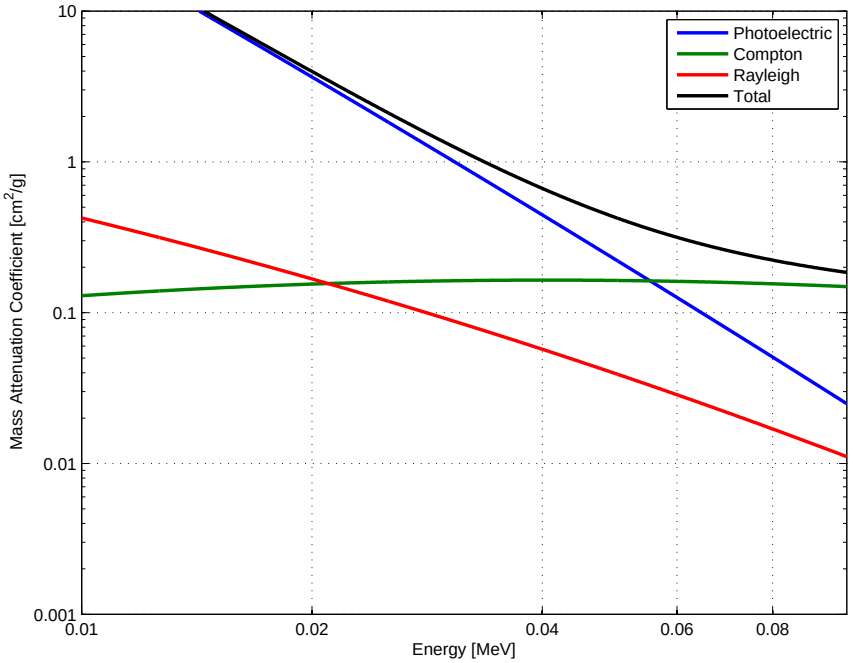


Figure 2.3: Mass Attenuation Coefficient of bone at various energies.

where $N_{in,E}$ is the number of photons travelling in the initial ($\theta = 0$) direction with energy E and $N_{out,E'}$ is the number of photons leaving in the direction of the solid angle $d\Omega$ with energy in the interval $[E', E' + \mathrm{d}E]$. The $\mathrm{d}E'$ term thus accounts for the possible change in energy.

In this thesis we will be working with non-polarized X-rays, and thus all distributions will be rotationally symmetric around the $\theta = 0$ axis, and the expression can be simplified to

$$\frac{\mathrm{d}^2\sigma}{\mathrm{d}\theta \mathrm{d}E'} = 4\pi \sin\theta \frac{N_{out,E'}}{N_{in,E}}$$

where

$$\mathrm{d}\theta = \frac{\mathrm{d}\Omega}{4\pi \sin\theta}$$

We will also find that for the Rayleigh interaction, $E' = E$, and for the Compton interaction, neglecting Doppler broadening, $E' = f(E)$ for some function f. In these cases, the DCS can be written in its energy-independent form as

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\theta} = \int_0^\infty \frac{\mathrm{d}^2\sigma}{\mathrm{d}\theta \mathrm{d}E'} \mathrm{d}E' = 4\pi \sin\theta \frac{N_{out}}{N_{in}}$$

This is the form that will be used in the rest of this thesis. We will now discuss the different types of interactions in more depth.

Photoelectric absorption is an interaction where a photon interacts with an atom and its energy deposited in an electron. This electron will then continue its way through the material and interact with the matter, gradually losing energy. There are two ways for the electron to lose energy, inelastic collision with nuclei, resulting in an energy loss by heating the material, and photon emission by Bremsstrahlung. The energy loss due to these interactions is usually measured as the electron *stopping power* of the material. The stopping power is a measure of how much energy an electron loses per unit length travelled in the material.

The electron stopping power of water is shown in Figure 2.4. We see that the electron loses approximately 10MeV/cm in total, and that almost all of the energy is lost by inelastic collisions, with only a negligible amount re-emitted as photons. The mean path that the electrons will travel before stopping is thus very short, approximately 10^{-3}cm. Because of this, we can with high accuracy assume that the energy of the electron is deposited locally as thermal energy in case of a photoelectric absorption.

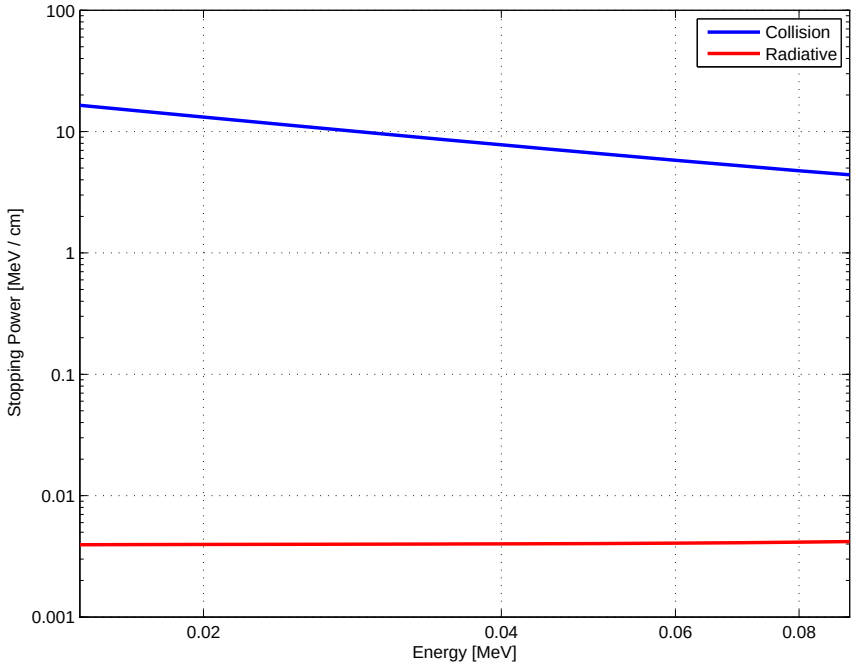


Figure 2.4: Electron Stopping Power of water at various energies. Data obtained from NIST ESTAR [5].

Compton scattering is an interaction where a photon collides with an electron in the material. The photon loses some energy, depositing it to the electron, and both the electron and photon is then scattered in new directions. As with photoelectric absorption, the electron is swiftly absorbed.

To calculate the angular and energy distribution of the scattered photons, we start by assuming that they are scattered by stationary free electrons. Using quantum physics[2] one gets the Thompson DCS

$$\frac{d\sigma_T}{d\Omega} \propto 1 + \cos^2\theta$$

Because of the energy lost to the electron, the energy of the photon will change. The energy of the scattered photon can be calculated from the conservation of energy and momentum, yielding

$$\frac{E'}{E} = \frac{1}{1 + \frac{E}{E_0}(1 - \cos\theta)}$$

This theory does however not account for relativistic effects. If we account for these, we get the Klein-Nishina DCS

$$\frac{d\sigma_{KN}}{d\Omega} \propto P^2(P + P^{-1} + \sin^2\theta)$$

where

$$P = (1 + E/E_0 \cdot (1 - \cos\theta))^{-1}$$

and E_0 is the electron rest energy $m_e c^2$. This distribution is displayed for relevant energies in Figure 2.5.

The Klein Nishina DCS is popular in many applications that aim for speed or ease of implementation, and it is quite accurate at higher energies. There are however two notable errors with this approximation. First, we ignore the binding energy of the electron. This binding energy is in the order of 100eV for water and bone [5]. Further, we ignore the momentum of the electron travelling around the nucleus which causes Doppler broadening.

The authors of PENELOPE[6] argues that accounting for binding effects gives a significantly better approximation at energies in the order of tens of keV. If we ignore binding effects, we get a significant overestimation of the number of photons scattered in the initial direction.

Accounting for Doppler broadening is less important in lighter nuclei since the electrons carry less momentum. Doppler broadening is also significantly harder to account for since the full electron structure is needed. Because of this, we ignore Doppler broadening in this thesis.

[2]This result is derived in i.e.
`ocw.mit.edu/courses/physics/8-07-electromagnetism-ii-fall-2005/readings/radiation.pdf`

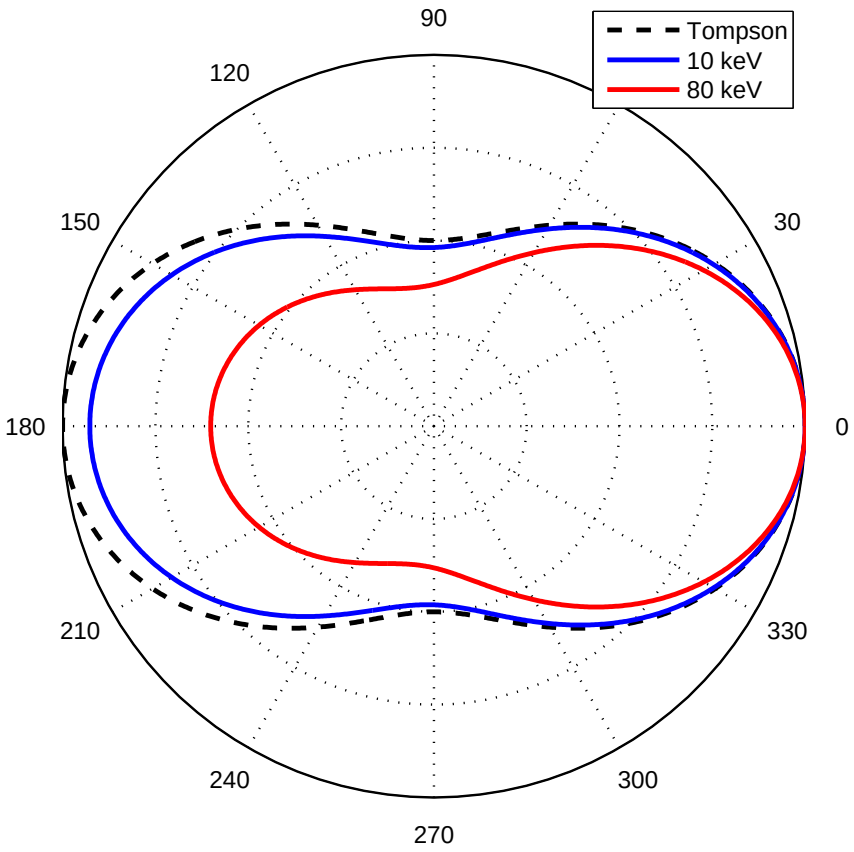


Figure 2.5: Klein Nishina distribution at relevant energies shown as a polar plot in normalized units. The Thompson distribution is also shown for comparison. At higher energies the relativistic effects are important.

If we account for the electron binding energy by means of the theory presented by Waller *et al.*[7], we get the Waller-Hartree DCS

$$\frac{d\sigma}{d\Omega} \propto \frac{d\sigma_{KN}}{d\Omega} S_M(x)$$

where S_M is the molecular incoherent scattering function, and x is the dimensionless momentum transfer, defined as

$$x = 20.6074 \frac{E}{E_0} \sqrt{2 - 2\cos\theta}$$

The molecular incoherent scattering function S_M can be approximated using the independent atom approximation:

$$S_M(x) = \sum_{\text{atom} \in \text{molecule}} S_A(x, Z)$$

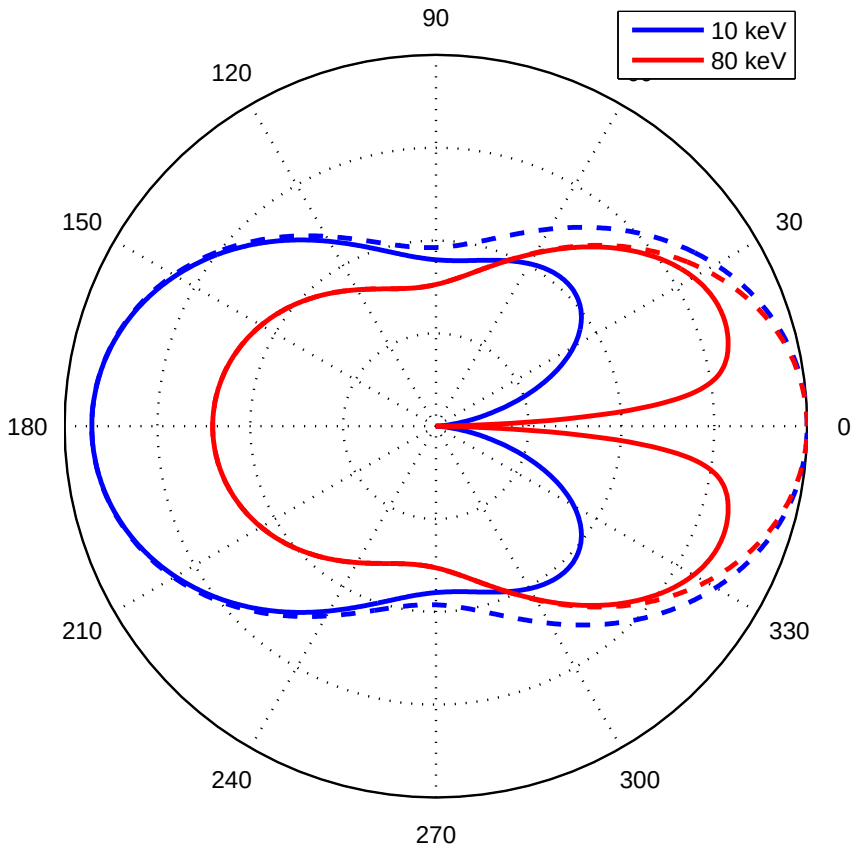


Figure 2.6: Compton scatter in bone at relevant energies. The Waller-Hartree result is given in solid lines while the dotted lines represent the Klein-Nishina approximation. Due to the binging effects, the distributions become more backwards oriented. This is especially true at lower energies.

where Z is the atomic number. Values of $S_A(x, Z)$ for various atomic numbers were obtained from xraylib[8], an open source project with extensive and easily accessible tables of X-ray related data. The source of the data is Cullen *et al.*[9].

In Figure 2.6 the angular distributions with and without accounting for binding effects is shown, we see that the binding effects are significant, especially at lower energies.

Rayleigh scattering is when the photon elastically bounces off an atom. As with Compton scatter, we can obtain the distribution of the scatter using only quantum mechanics assuming stationary and free point atoms. We get the Thompson DCS

$$\frac{d\sigma_T}{d\Omega} \propto 1 + \cos^2\theta$$

As in Compton scattering, the energy of the scattered photon can be calculated using energy and momentum conservation as

$$\frac{E'}{E} = \frac{1}{1 + \frac{E}{m_A c^2}(1 - \cos\theta)}$$

In this case however, we use the mass of the atom m_A instead of the electron mass m_e in the denominator. Since the mass of an atom is so large this expression is very closely equal to unity. Because of this, we can assume that $E' = E$.

The point-sized atom approximation does however fail at X-ray energies. For example, at energies above $\approx$ 25keV the wavelength of the X-ray becomes smaller than the Bohr radius. Accounting for the full electron structure according to the method of Born[10] gives the DCS

$$\frac{d\sigma_R}{d\Omega} \propto (1 + \cos^2\theta)|F_M(x)|^2$$

where $F_M(x)$ is the molecular form factor and x is the dimensionless momentum transfer. The molecular form factor can be calculated using the independent atoms assumption

$$|F_M(q)|^2 = \sum_{\text{atoms}\in\text{molecule}} |F_A(x, Z)|^2$$

where $F_A(x, Z)$ is the atomic form factor and Z is the atomic number. Values of these form factors are available in the literature, and Hubbell *et al.*[3] has made an accurate and comprehensive tabulation. This data was accessed from xraylib[8]. The angular distribution of the scattered photon is displayed in Figure 2.7. We see that the angular distribution is very strongly forward oriented, especially in the higher energy range.

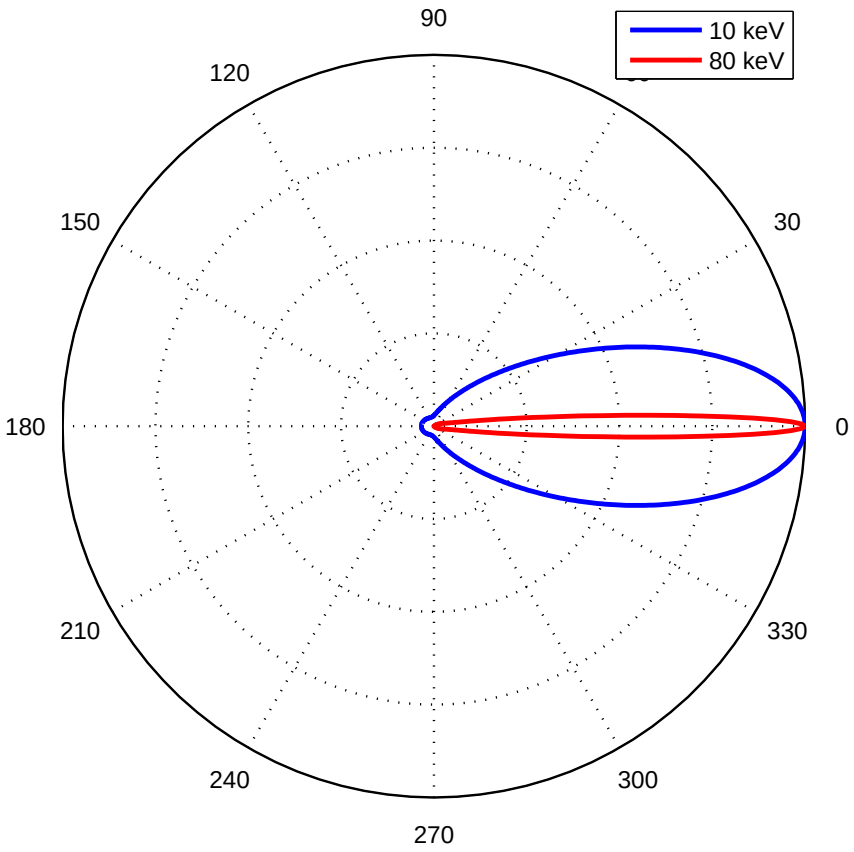


Figure 2.7: Rayleigh scattering distribution in bone for representative energies.

2.4 The Monte Carlo Method for Photon Transport

In the Monte Carlo method for photon transport, individual photons are transported stochastically through the material and the sought after quantities such as deposited energy or detector response is calculated as the average over a very large set of photons.

The photons are first generated at some point, for example in the X-ray tube. From this position, they are moved forwards a small distance. Using the material parameters of this position, we test (randomly) if the photon experiences an interaction. If it does, we sample which type of interaction happens, and simulate that interaction. After the interaction, the energy and direction of the photon is changed. This process is repeated until the photon is absorbed or leaves the geometry.

The Monte Carlo method is the most widely used method for photon transport. This is primarily because Monte Carlo methods are widely accepted as the most

accurate method for photon transport in complicated media, since no assumptions has to be made to simplify an analytic solution. Monte Carlo methods are also useful for their relative ease of implementation since only the raw physical model of transportation of a single photon is needed, with no extra continuum assumptions, dependencies or boundary conditions.

Lately the method has also grown in popularity due to its parallel structure, allowing easy implementation in parallel architectures such as supercomputers or GPU's.

2.4.1 Related Work

Several well known codes exists for MC photon transport, among these are the well maintained PENELOPE[6] code, this code has been verified to high accuracy for a wide range of photon energies. It is however slow, simulating in the order of 10^5 photons/minute, it would thus take approximately two months to reach 1% accuracy according to Equation 2.1. Another highly used code is the EGSnrc code, this code is likewise known to be highly accurate, but as with PENELOPE code it is too slow for this application.

There has recently been significant research into GPU-MC methods, such as CUDAMCML[11] developed at Lund University. This code is however also comparatively slow, and only works on layered geometries. Other GPU-MC codes include mcgpu[12] developed at the U.S. Food and Drug Administration. This code is GPU based simplification of PENELOPE and is relatively fast, at approximately 10^8 photons/minute. The code is not optimally fast since it is not correctly optimized for GPU computation. For example, it uses rejection sampling, non-coaleced memory accesses, and limited use of pre-computation.

Hissoiny has developed an GPU-MC code called GPUMCD[13] for dose calculations which has been acquired by Elekta. Dose calculations are made in the order of several MeV, where pair production is important, and electron transport needs to be considered. Further, the Rayleigh and photoelectric interactions are less important than at lower energies. Nonetheless, this code is relatively fast, and with slight modifications it runs at almost 10^9 photons per minute. The code also has other positive parts for this project, it is well documented, and integrated into other company code. This code was thus selected as a base for further investigation.

2.4.2 Variance Reduction Techniques

Variance reduction techniques are techniques that can be used to reduce the statistical error of the result without introducing any bias to the solution. Variance reduction techniques thus differ from approximation based speed-up techniques in that the result will converge exactly.

A simple way to estimate the potential for variance reduction techniques is estimating the amount of "wasted" work. Measurements show that approximately 5% of all simulated photons will hit the detector as scatter. The other 95% provide very little information about the scatter distribution. In an optimal scenario all work done would contribute with information. We can thus estimate that variance reduction could give us an $\approx \times 20$ speed-up. In other geometries than the one studied, such as dose calculations where the volume of interest is smaller, variance reduction could possibly provide even higher speed-ups.

Another form of wasted work is taking a step forward without simulating any interaction. If the step size is short, or the interaction probability low, we may need to take several steps before simulating an interaction. Some variance reduction techniques such as the Woodcock Tracing technique may seek to increase the step size or interaction probability without changing the overall result.

Several authors, such as Kawrakov *et al.*[14] and Mainegra-Hing *et al.*[15] have investigated variance reduction techniques for CPU's with promising results. Sometimes achieving an speed-up of up to $\times 60$.

A common method in variance reduction methods is to assign a numerical weight to all photons. This weight can then be manipulated as the photon is transported. When the photon is scored at the detector, the result is then scaled by the weight of the photon.

We will now discuss the variance reduction methods that are most common in the literature.

Woodcock Tracing

Woodcock tracing[16] is a probabilistic method for ray tracing, where one calculates the minimum Mean Free Path (MFP) in the entire volume, and samples a step length from the exponential distribution using this MFP. The photon is then transported by this distance in its current direction. If the photon lands in a material where the MFP (s_{cur}) is larger than the minimum MFP (s_{min}), then, with probability $P = 1 - \frac{s_{min}}{s_{cur}}$, no interaction is simulated. Otherwise an interaction is sampled according to the current medium. This can provide a noticeable speedup.

Kawrakov *et al.*[14], reports a 20% efficiency gain for coarse geometries, and better results for fine geometries. Kawrakow's tests were however performed on a CPU.

Mean Free Path Transformation

In an MC simulation lots of work is performed on the near side of the volume, where most particles interact, and little work is performed on the far side. Ideally work would be spread more evenly over the volume. Rogers *et al.*[17] has a way to

mitigate this problem. The method involves scaling the path length of the particle by a factor dependent on the direction of the particle. To ensure convergence one also has to scale the particle weight to compensate for this.

The method is further explored by Mainegra-Hing *et al.*[15], where a direction independent scaling is explored.

Kawrakow *et al.*[14] does however note that the MFP transformation method is only efficient at certain energies larger than 6 MeV, and notes that even then the gains are marginal. Because of this MFP transformation methods were not further investigated for this problem.

Forced Detection

A method outlined in Mainegra-Hing *et al.*[15]. Whenever a photon points towards the detector after an interaction, it is attenuated through the geometry using exact ray-tracing, and finally scored.

Russian Roulette

For many photons in the simulation, it often becomes obvious when it is leaving the geometry, such as when it is on the edge of the head and travelling away from the detector. For convergence reasons, we cannot ignore these photons, but we can save work by Russian Roulette. The method, outlined in Mainegra-Hing *et al.*[15], is a method where if a particle will miss the detector with a high probability, determined by some estimate, the particle is "killed", removed from the simulation with some probability P. If the particle survives, its weight is scaled by $1/(1-P)$, this ensures correct convergence of the algorithm.

Russian Roulette can be seen as a form of importance sampling, where we sample according to the chance that the particle will hit the detector and thus influence the final result.

Russian Roulette is expected to cause significant warp splitting, and may need special care for GPU implementation.

Splitting

Splitting is a method discussed in Mainegra-Hing *et al.*[15]. Splitting can be seen as the inverse of Russian Roulette. For some photons, the probability that the photon will hit the detector is significantly higher than average, and it will give a large contribution to the result.

For example, in a normal human CBCT scan, only $\approx 3\%$ of the photons will pass through the head without interacting at all. If a particle interacts at the far side

of the head, close to the detector, it will thus provide valuable insight into the scattered radiation from this specific area. To get the most out of this, we can "split" the particle, simulate the interaction N times, and continue the ray-tracing for each alternative. We have to scale the weight of the photons by $1/N$ to ensure convergence.

Mainegra-Hing *et al.*[15] have shown that this method can provide speedups with a factor over 100 on a CPU. There has however been limited research into this method on GPUs

2.4.3 Pre- and Post-Processsing

The spatial frequency of the scatter is low, and investigations also show that the scatter distribution varies relatively smoothly when the detector is rotated, and that the differences between patients is moderately small. These properties can be used to significantly reduce the number of photons needed. Some of these will now be discussed.

Post Processing Methods

An efficient filtering method will most likely be an important part of an efficient solver. There are two primary ways to perform this filtering, spatial and angular, a combination of these will most likely provide the best results.

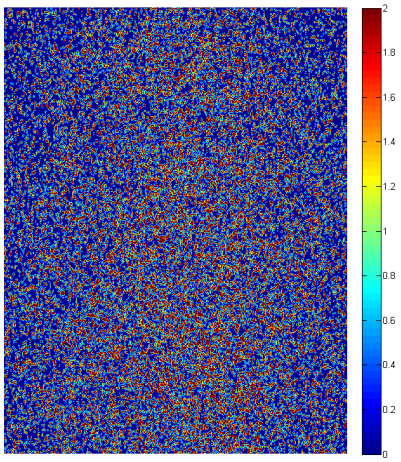
Since the spacial frequency is so low, a simple low-pass filter gives good results. An example of this is shown in figure 2.8.

To use the low variance of the scatter with respect to rotation, the most simple method would be to calculate the scatter for a relatively low number of angles, and then interpolate the result in between. This method was chosen in this thesis for its simplicity.

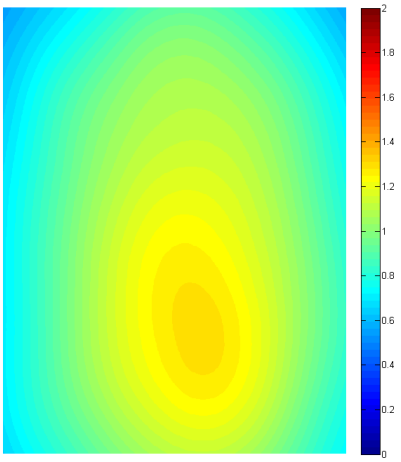
For better results, we could implement the method of Bootsma [18]. His method is to calculate the scatter for a low number of angles. Using this data, he calculates the three dimensional Fourier transform, and applies a filter.

Pre-Computation

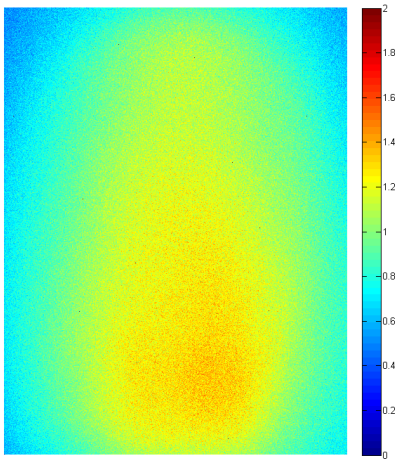
Since the scatter is similar between patients, we could perform many high accuracy simulations of photon scatter and create a set of basis functions for the scatter. This can then be used to filter data in on-line simulations. A simple method would be to construct a singular vector basis that the new simulation can then be projected on.



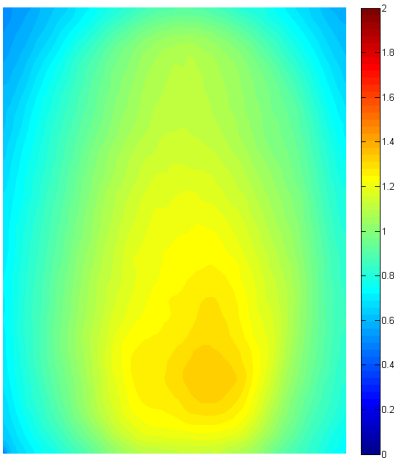
(a) Data for 10^6 photons. Slightly filtered for visibility.



(b) Lowpass filtered result with 10^6 photons.



(c) Raw data with 10^{11} photons.



(d) Lowpass filtered result with 10^{11} photons.

Figure 2.8: Comparison of the calculated scatter distributions at the detector as calculated with 10^6 and 10^{11} photons, before and after Gaussian low-pass filtration. Given in mean-normalized units. While the difference is very significant in the unfiltered images, we see that the result from 10^6 photons is close to the 10^{11} photon result after the filtering.

This method has the advantage that higher frequency components can be retained to higher precision than regular filtering.

This method is however not widely discussed in the literature. This may be because of the prohibitively long run times of building such a database, the need for large sets of accurate phantoms, or some other reason. Because of this, the method was not further investigated.

2.4.4 Random Number Generation

The accuracy and speed of the simulation is highly dependent on the random number generator used. A good Random Number Generator (RNG) should have a long period, a low memory use, and be fast. There are several well known RNGs. The Mersienne Twister (MT) algorithm[19] is perhaps the most well known highly accurate RNG, used in many popular programs, including MATLAB. MT is known to be well distributed, and is relatively fast on CPUs. However, it uses a 2492 byte state vector. This is prohibitively large for GPU implementations since it would have to be stored in global memory.

There is a modification of the MT algorithm suitable for GPUs developed by Saito *et al.*[20]. It is implemented in the CU(DA) Random Numbers (CURAND) library. This implementation has a limitation at 51200 threads.

Marsaglia introduced the Xorshift[21] algorithm in 2003. It has a shorter period than the MT algorithm, but has a smaller state vector and a faster execution. It is implemented with a slight modification as the XOR-shift with Weyl sequence (XORWOW) algorithm in CURAND.

Multiply with Carry is another algorithm introduced by Marsaglia, G.[22]. It has somewhat statistically worse performance than the Xorshift algorithm, but is slightly easier to implement. This method was preferred by Hissoiny et. al. in their work GPUMCD [13].

3 | Method

In this chapter the methods used will be described, and motivated. First, a brief outline of where the algorithm will be used will be presented, then the method for simulating photons. After that the variance reduction techniques and filtering methods will be presented. Finally the GPU specific code optimizations implemented will be described.

3.1 CBCT Volume Reconstruction with Scatter Reduction

To place this thesis work in context, we will briefly outline the overall algorithm the work will be used in. The big picture algorithm is visualized in Figure 3.1.

In a clinical setting, we first take a sequence of x-ray images at different rotation an-

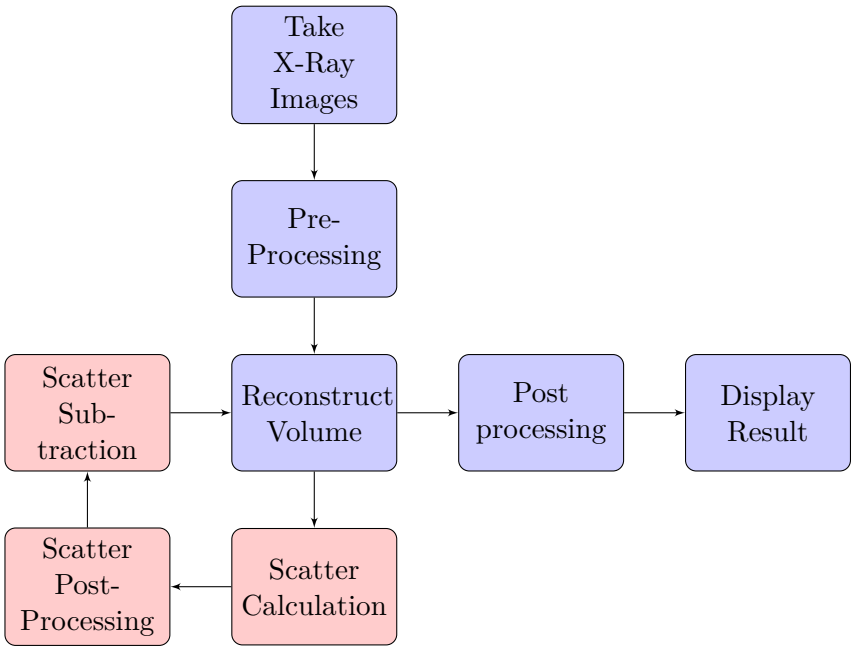


Figure 3.1: Flow chart of the CBCT reconstruction. The extent of this thesis work is marked in red.

gles around the patients head. These images are then pre-processed in a sequence of steps. These steps include stages such as removing pixel variance and compensating for the different angle of attack for the different pixels.

Using the pre-processed data, an initial reconstruction of the CBCT volume is performed. This reconstruction will be relatively good, but has artefacts due to scattered photons.

Using this reconstructed volume, a MC scatter simulation is performed. The scatter is then post processed, applying filters and other compensating methods, before finally being subtracted from the images.

Using the compensated images, a new reconstruction is performed, now with higher accuracy. The volume is post-processed, and finally returned to the caller.

Since we use the reconstructed volume to correct itself, there is a chance of introducing or enhancing artefacts. However, since the scatter is very smooth, small errors in the volume should not give large errors in the result.

3.2 Geometry

The geometry used in the thesis is the actual geometry of the CBCT scanner. Two coordinate systems are used in the simulation, as shown in Figure 3.2. In the pre- and post-processing code, a coordinate system fixed on the x-ray source is used. All physics is assumed to happen inside the cubical simulation volume.

The actual physical simulation of the photons take place in the simulation volume, in this volume, a secondary coordinate system is used. This coordinate system is rotated according to the *gantry angle* θ, dependent on how the source is aligned with respect to the patients head. The coordinate system is also centred on the volume.

The method selected to represent the simulation volume was a voxel representation. Voxels are the three-dimensional equivalents of pixels, with each voxel representing a block in space.

The geometry was stored in a voxel format due to its simplicity in access, requiring only one global memory access to find the material in a position, as compared to more complex constructive quadratic geometries[1] that are used in programs such as PENELOPE[6]. One downside of voxelized geometries is that many common shapes such as spheres require high resolution to model accurately. However, thanks to the woodcock method, simulating in a high resolution geometry does not give a significant slowdown.

[1] In quadratic geometries the volume is subdivided into sub-volumes enclosed by quadratic polynomials. These include planes, parabolas and sphereoids. Using the intersections and unions of these sub-volumes, the material parameters of any point in space is defined.

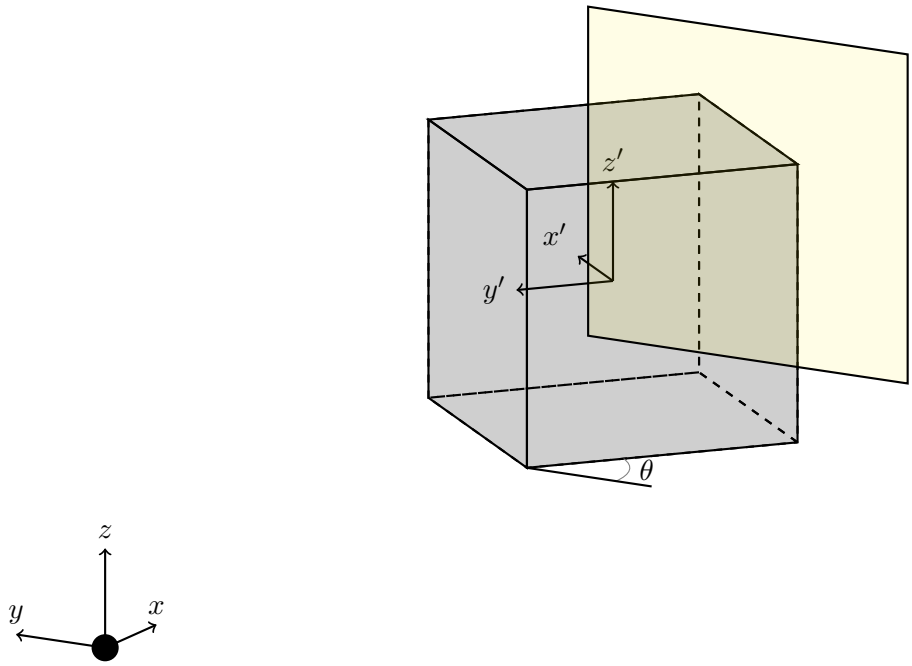


Figure 3.2: Illustration of the two coordinate systems used, shown to scale. The global coordinate system is centred on the photon source, with the x coordinate pointing towards the base of the volume and detector. The actual simulation takes place in the secondary coordinate system x',y',z'.

3.2.1 Material Model

The material model used is a crude model where the volume is approximated with three different materials, air, water and bone. These are seen as a good approximation of the materials present in the human head, further adding more materials, such as soft tissue and trabecular (spongy) bone, did not give significant improvements. This is because the errors in the CBCT raw data makes segmentation hard.

The material parameters such as density, material composition and mass attenuation coefficients were obtained from the NIST Xcom web page[4].

Material classification was performed by a threshold method. From the reconstruction we find the approximate attenuation coefficient in all voxels. All voxels with attenuation coefficient below $\mu_{\text{water threshold}}$ was classified as air, voxels between $\mu_{\text{water threshold}}$ and $\mu_{\text{bone threshold}}$ were classified as water, and all above $\mu_{\text{bone threshold}}$ as bone.

The density was calculated using two methods. The first was to approximate the density with a material specific value. This value was obtained from the NIST STAR [5] database. A second method were the density was scaled according to the

attenuation coefficient in the voxel was also tested. However, due to lack of data it was hard to find accurate numbers and this method was not used in the final implementation.

3.3 Simulating Photons

Accurate simulation of photons is crucial to an accurate estimation of the distribution of scattered photons, and significant work was spent making sure the physics of the model were accurate. The algorithm for photon simulation of one photon is outlined in Figure 3.3.

The photon is first generated at some point in the voxel geometry or at its boundary. When generated, the photon has a set of characteristics, namely position, direction, (numerical) weight and energy. We then take one step forward, and test if we have left the geometry. If that is the case, the photon is scored at the detector and the simulation is done. Otherwise an interaction is simulated. If the photon is absorbed, the simulation of the photon is done, otherwise, we start again by taking another step forward.

3.3.1 Generating Photons

The photons used in the simulation belong to a space Ω of possible photons. The space is given by

$$\Omega = \underbrace{\mathbb{R}^3}_{\text{Position}} \times \underbrace{\mathbb{R}^+}_{\text{Energy}} \times \underbrace{[0, 2\pi) \times [0, \pi]}_{\text{Direction}} \times \underbrace{\{\text{Primary}, \text{Scattered}\}}_{\text{Type}}$$

To simulate the setup we need to have an accurate probability distribution function over Ω, often called the phase space in the literature. For efficiency reasons, we also need to pick new photons from the phase space quickly. Two methods were used to generate photons.

First, the phase space was simulated to very high accuracy using PENELOPE using an model of the actual x-ray source. These photons were then stored to memory and reused in the simulation. This method has the advantage that the photons used very closely follow the actual physical probability distribution. A total of $\approx 10^8$ particles were generated in this way. Of these, approximately 10^6 photons were used in the simulation. The number was chosen to balance noise and time spent loading the phase space into memory. The detector has approximately $5 \cdot 10^5$ pixels, so the comparatively small phase space will give rise to bumps in the result as seen in Figure 3.4. These bumps will be very noticeable in the primary result, but convergence tests show that these are insignificant in the scatter since the distribution is smoothed.

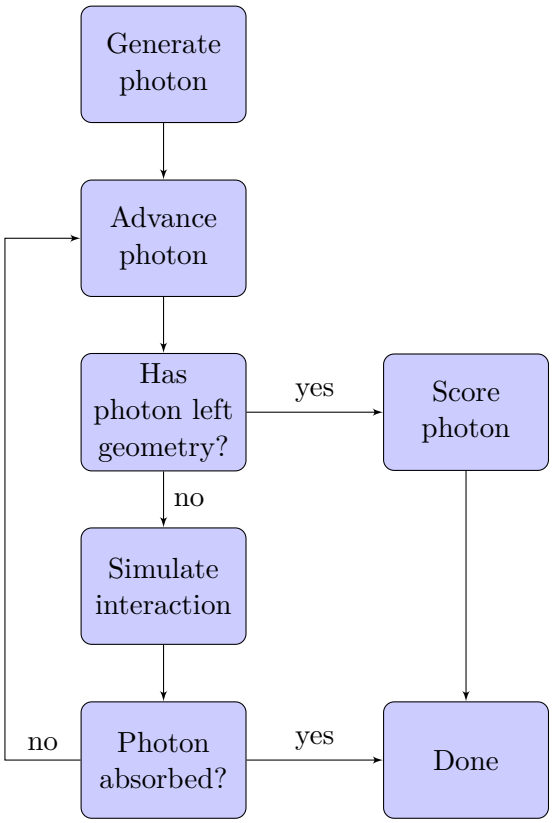


Figure 3.3: Flow chart of the photon simulation algorithm for one photon.

To speed up the loading of these photons into memory, the photons were loaded in a coalesced manner into shared memory at the start of execution. When each thread has finished its simulation, it then fetches a new photon from shared memory.

The second method used was to approximate the phase space analytically and generate photons from the approximate phase space on the fly. In the analytic approximation, Ω was significantly simplified by assuming that the photons are generated by a point source, and that none are scattered before they enter the volume, the approximate space Ω_a is thus given by

$$\Omega_a = \underbrace{\mathbb{R}^+}_{\text{Energy}} \times \underbrace{[0, 2\pi) \times [0, \pi]}_{\text{Direction}} \subset \Omega$$

Both the direction and the Energy were then fit to the measured phase space using a weighted least squares approximation, assuming independence of all the dimensions. An visualization of the measured phase space and the functions fit to approximate it is shown in Figure 3.4. The fits are very good except for the energy, where the

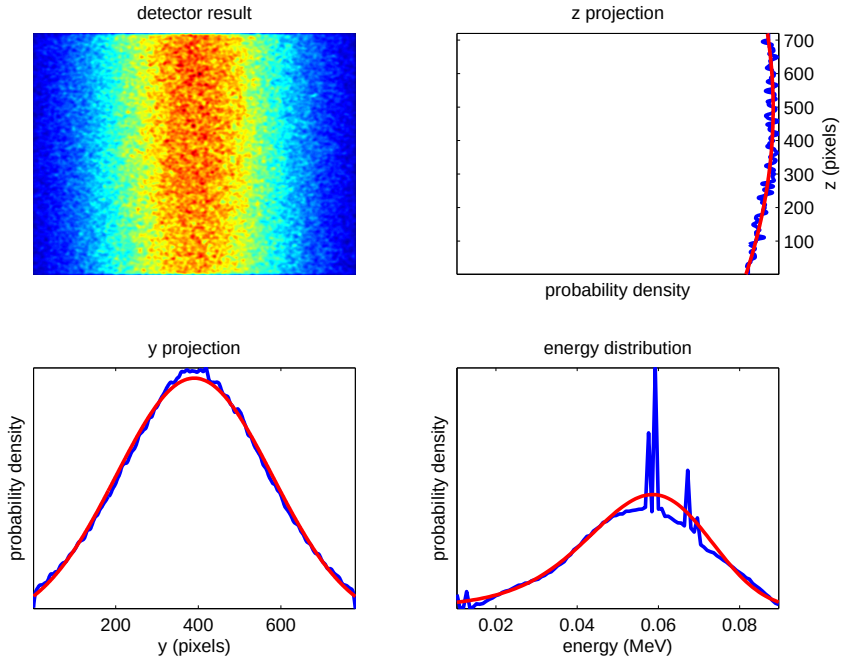


Figure 3.4: Phase space projected on detector and projections onto y and z axises, as well as the energy distribution. The measured data is coloured blue, while the fit is red.

characteristic K lines of the x-ray spectrum are not properly modelled.

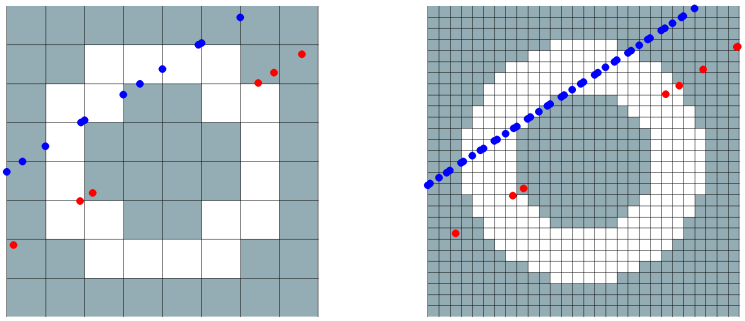
To sample from these complex distributions efficiently, they were modelled as the sum of uniformly distributed numbers. This approximation proved both fast and accurate.

After the initial photon position was generated, the photon was projected onto the simulation region along its current path, assuming no energy loss or scatter in the air. When using the sampled phase space, the photons were projected exactly onto the head to minimize time spent stepping the photon through air. This optimization gave a 70% speedup.

Testing showed that using the sampled phase space gave a slightly more accurate result, and was approximately 10% faster. The simulated phase space was thus only used in calculations where the bumps in the primary signal caused by reusing the photons were problematic.

variable	PDF $\propto$	Range
y	$\exp\left[-(y-390)^2/268^2\right]$	$y \in [0, 780]$
z	$-3.5 \cdot 10^{-7} z^2 + 3.5 \cdot 10^{-4} z + 0.90$	$z \in [0, 720]$
E	$\exp\left[-10.38E^3 + 5.15E^2 + 5.25E\right]$	$E \in [0.01, 0.09]$

Table 3.1: Approximations of the PDFs obtained from curve fitting the phase space. The direction of the photons were defined by the detector pixel the ray was headed towards.



(a) Coarse geometry (b) Fine geometry

Figure 3.5: Comparasion of voxel (blue) and Woodcock (red) ray tracing in an 2d geometry. The circles indicate possible interaction points.

3.3.2 Advance Photon

Two methods were evaluated for photon transportation, voxel tracing and the Woodcock scheme. In voxel tracing, the photon is transported to the next voxel wall, and a random number is sampled to determine if the photon underwent an interaction. In woodcock ray tracing, an exponentially distributed random number with mean value equal to the mean free path of the most dense material in the geometry is sampled and the photon transported. If the photon lands in a less dense material, the probability of an interaction is scaled down appropriately.

Woodcock ray tracing converges to the same result as voxel tracing, but may require more photons to reach convergence. Voxel tracing on the other hand may spend extra time attaining unneeded resolution, this can be seen in Figure 3.5. Here the work is approximately equal in a coarse geometry, but significant extra work is performed in the fine geometry.

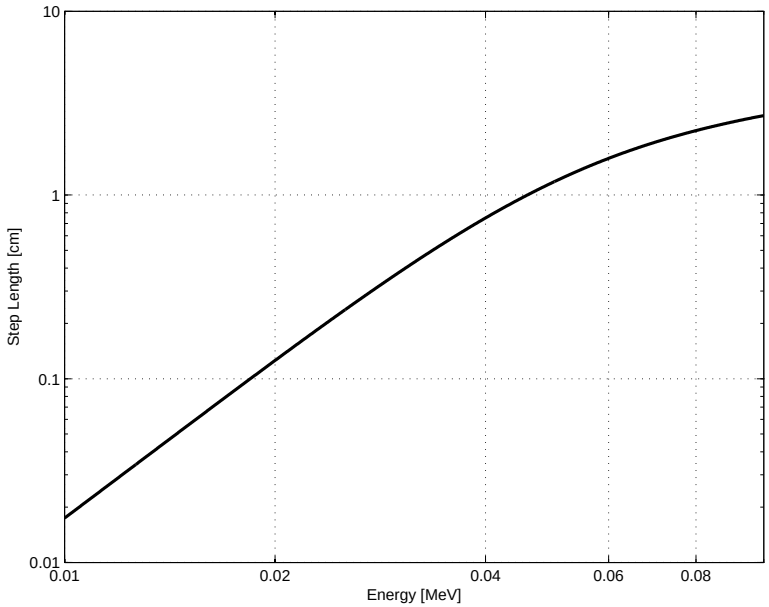


Figure 3.6: Woodcock step length assuming bone with density 2.0[g/cm^3] is the densest material.

In the woodcock step length is dependent only on the energy and the geometry, and can thus be efficiently pre-calculated before the simulation kernel is executed. The step length is then stored in texture memory and accessed using the built in linear interpolation. An example of the step length is given in Figure 3.6. For energies in the most common range, $0.4 - 0.8$MeV, the step length is of order 1-2 cm. This is significantly larger than the voxel size used, which is about 0.1 cm to be able to resolve the geometry well.

3.3.3 Score Photon

When a photon leaves the geometry a test is performed to see if the photon path intercepts the detector and if so, and what pixel it intercepts. If the photon will hit a pixel, the photon is scored at that pixel. The detector works by converting the photon energy into a current that is registered. We assume that the detector is thin and only a small amount of the energy from the photon is deposited in the detector instantly, we also assume that the deposited energy is proportional to the energy of the photon.

Since we assume that only a small amount of energy is deposited, it is reasonable to assume that the deposited energy is proportional to the path length through the detector. In Figure 3.7, a photon travelling through the detector at angle θ is depicted. We see that the distance travelled in the detector is $t' = t/\cos\theta$, and we

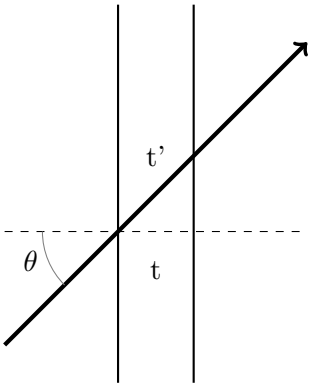


Figure 3.7: Illustration of the angular dependence of the scoring.

thus have to scale the detector response by $1/\cos\theta$.

We thus use the approximate detector response function

$$R(E,\theta) = \frac{E}{\cos\theta}$$

There are a few things we neglect in this approximation. The response of the detector also has an exponential time behaviour, and the process is thus not a true Markov process. The documentation of the detector is however not very good, and no further information was available, making the assumptions the best possible. The possibility of performing experiments with the detector were discussed, but ruled out since it was not considered part of this thesis.

The detector also has a variance on a pixel level, with some pixels being more sensitive than others. This error is calibrated by using an in-house developed method.

3.3.4 Simulating Interactions

If the photon is determined to undergo an interaction, the photon will change direction and change its energy. Since there are two directions the photons direction vector can be rotated, we need to determine the two Euler angles corresponding to this rotation. The angles are shown in Figure 3.8. Once these angles have been determined, the particle could be rotated by applying Rodriguez's rotation formula (3.1) for the rotation of a vector $\mathbf{v}$ around an arbitrary vector $\mathbf{k}$ by ψ radians, twice.

$$\mathbf{v}_{rot} = \mathbf{v}\cos\psi + (\mathbf{k}\times\mathbf{v})\sin\psi + \mathbf{k}(\mathbf{k}\cdot\mathbf{v})(1-\cos\psi) \tag{3.1}$$

For fast computation, we want to only perform one rotation on the vector. To achieve this, we can pick the $\mathbf{k}$ vector appropriately and do some pre-calculation. From Figure 3.8 we see that we can decompose the rotation as first rotating the

vector θ radians around an arbitrary vector in the $x'y'$ plane, and then an rotation of φ radians around the $\hat{\mathbf{z}}'$ axis. The reason we can pick an arbitrary vector in the $x'y'$ plane is that the angular distributions studied are homogeneous in φ. We chose the vector in the $x'y'$ plane as

$$\mathbf{k}_{xy} = \frac{\hat{\mathbf{z}}' \times \mathbf{v}}{||\hat{\mathbf{z}}' \times \mathbf{v}||}$$

To reduce the number of calculations needed in runtime, we can also include the φ rotation into our choice of $\mathbf{k}$. We do this by rotating the $\mathbf{k}_{xy}$ vector around the $\hat{\mathbf{z}}'$ axis by φ radians. We obtain

$$\mathbf{k} = \mathbf{k}_{xy}\cos\varphi + (\hat{\mathbf{z}}' \times \mathbf{k}_{xy})\sin\varphi + \hat{\mathbf{z}}'(\hat{\mathbf{z}}' \cdot \mathbf{k}_{xy})(1 - \cos\varphi)$$

Inserting this into the Rodriguez's rotation formula we arrive at

$$\begin{aligned}
\mu'_x &= \mu_x \cos\theta + \sin\theta(\mu_x\mu_z\cos\varphi - \mu_y\sin\varphi)(1-\mu_z^2)^{-1/2} \\
\mu'_y &= \mu_y \cos\theta + \sin\theta(\mu_y\mu_z\cos\varphi + \mu_x\sin\varphi)(1-\mu_z^2)^{-1/2} \\
\mu'_z &= \mu_z \cos\theta - \sin\theta\cos\varphi(1-\mu_z^2)^{1/2}
\end{aligned}$$

where (μ_x, μ_y, μ_z) is the direction (unit) vector, parallel to $\hat{\mathbf{z}}'$. We see now why the choice of $\mathbf{k}_{xy}$ was good. It reduces the number of calculations noticeably by eliminating the last part of Rodriguez rotation formula.

One issue with this formula may be numerical instability due to division with $(1-\mu_z^2)$ which may be very close to zero. It is however only performed relatively rarely since few particles will be travelling in the $\hat{\mathbf{z}}$ direction. We also performed some tests with a more complicated method that avoids dividing by numbers close to zero, we found that the results were indistinguishable from this more simple formula, but the method was slightly slower.

Compton Scattering

When a photon undergoes a Compton interaction, a part of the photons energy is deposited locally and the photon is given a new energy and a new direction. Three methods to simulate this were tested for computational efficiency and physical correctness.

The first way was to sample the Klein-Nishina distribution using rejection sampling. Rejection sampling is favoured in many of the well known codes, such as PENELOPE[6]. However, as shown in Appendix A, rejection sampling scales badly in GPU's.

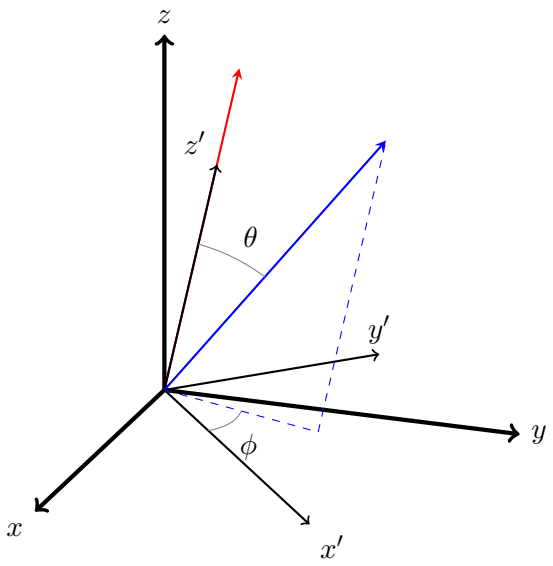


Figure 3.8: Illustration of euler angles θ and ϕ for a particle initially traveling in the z' (red) direction. The new direction is given in blue.

A deterministic method was thus also tested. Everett *et al.*[23] gives a method for sampling the Klein-Nishina distribution at all energies above 1keV with a relative error $\leq 2\%$. This method was implemented and tested.

Finally, a more physically advanced model taking binding energy into account was tested. Since using the binding energy will require a global memory access to find the value of the incoherent scattering function. We selected a third way of sampling the distribution, by using a pre-calculation intensive method to speed up the actual computation.

The inverse-Cumulative Distribution Function (CDF) of the distribution of $\cos\theta$ for each material and for a range of energies was pre-calculated in MATLAB and stored in a table. An example of such a table is given in Figure 3.9. This distribution was then stored in texture memory and sampled using the built-in linear interpolation.

Testing showed that the tabulated inverse-CDF method was both more physically accurate, and gave a modest $\approx 10\%$ speed-up over the other methods. It was thus selected as the best method.

Rayleigh Scattering

In Rayleigh scattering the photons energy is conserved, but it is given a slightly different direction, dependent on the energy and material. For Rayleigh scattering, no simple approximation that is physically valid is available, and we need to use

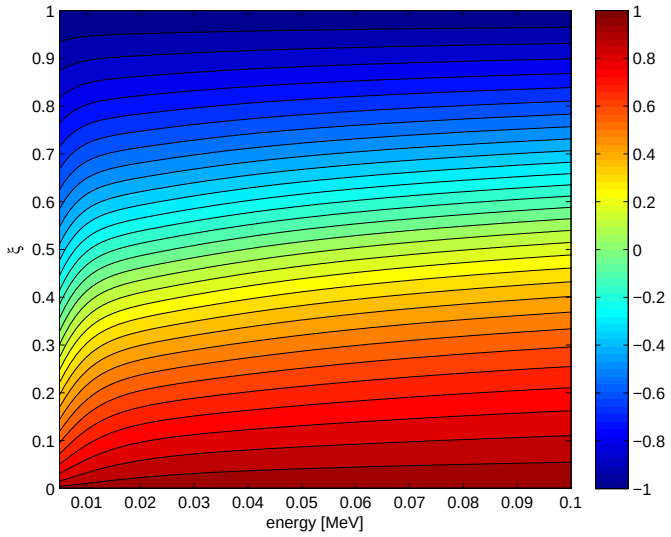


Figure 3.9: Compton inverse CDF table for water. The table is colored by $\cos\theta$. It is sampled by selecting a random number $\xi \in [0, 1]$ and sampling the table for the appropriate energy.

the full model.

As with Compton scattering, we need global memory access to find the atomic form factor, because of this we opted to use the tabulated inverse CDF method described in 3.3.4.

Photoelectric Effect

When the photon experiences a photoelectric effect the energy is transferred to an electron. As discussed in 2.3, the photons energy is quickly absorbed with high probability. We approximate this by terminating the photon on the spot.

3.3.5 Energy Cut-off

From Figure 3.6, we see that for low energies, the step length becomes very small, and the photoelectric effect starts to dominate. To speed the code up and avoid simulating photons that will very likely be absorbed anyway, we set a cut-off energy. All photons with energies below the cut-off are absorbed instantly.

To select the cut-off, we observed how far the photons will travel on average before they have been absorbed. This is shown in Figure 3.10. We see that if we set the cut-off energy to 0.01 MeV, we will not change the result significantly since almost

all cut-off photons would be absorbed within 1 cm in water, and 1 mm in bone. We also note that we will not significantly change the type of interactions that will occur, since the photoelectric effect dominates at these energies.

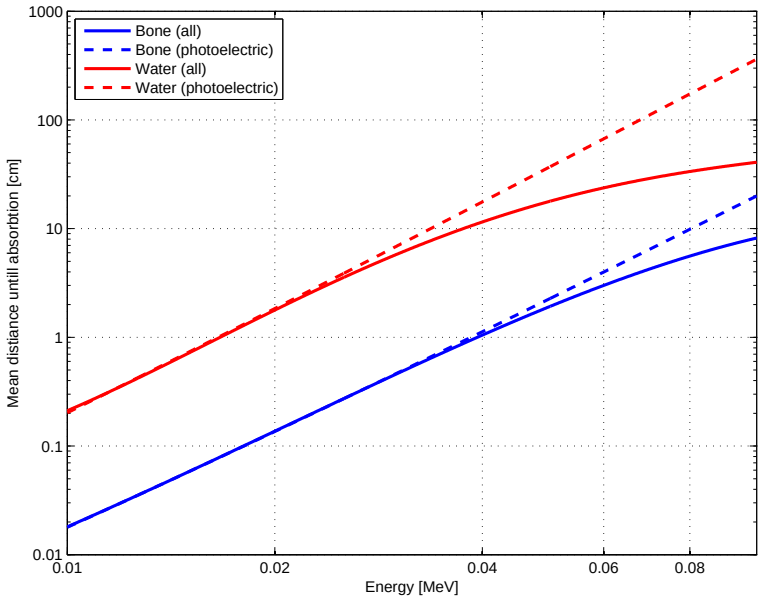


Figure 3.10: Mean travel distance until absorption in water and bone assuming all effects and only photoelectric effects. The photoelectric effect dominates at lower energies.

This effect was also investigated in practice in a head geometry. Several calculations were performed with different cut-offs set, and the relative error and runtime were investigated. The results are shown in Figure 3.11. As can be seen, setting the cut-off at 0.01 MeV gives a good margin, while having no significant effect on the result.

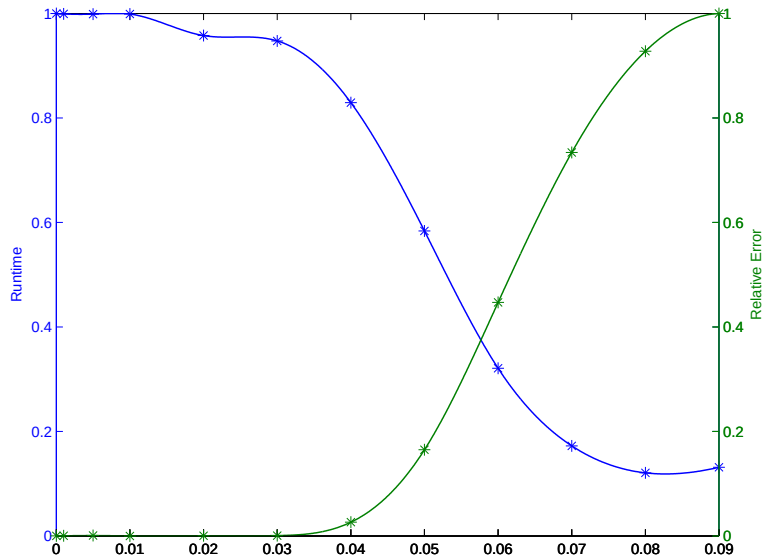


Figure 3.11: Investigation of the impact of the cut-off energy on runtime and error in a head geometry.

3.4 Variance Reduction

Several variance reduction methods were tested, with focus on the methods that had given the best results in other research. The most popular methods are Splitting and Russian Roulette, with notable results also reported for the forced detection technique.

3.4.1 Splitting

Splitting was implemented using a two-stage procedure. In the first step, photons were transported through the volume until they experienced an interaction. The photon data, as well as all information needed to simulate the interaction was then stored in shared memory. When a sufficient amount of photons were accumulated in shared memory, they were moved in a batch to global memory. If a photon did not experience any interaction and hit the detector it was scored as a primary photon.

In the second step, the split photons were simulated. Each photon was split into N_s photons, each with weight $1/N_s$. To improve thread coherency, N_s was chosen to be a multiple of the warp size, so each warp starts at the same position. This also helps memory access since multiple threads accessing the same memory is a so called Broadcast, which is faster. These photons were then traced through the volume in the usual way.

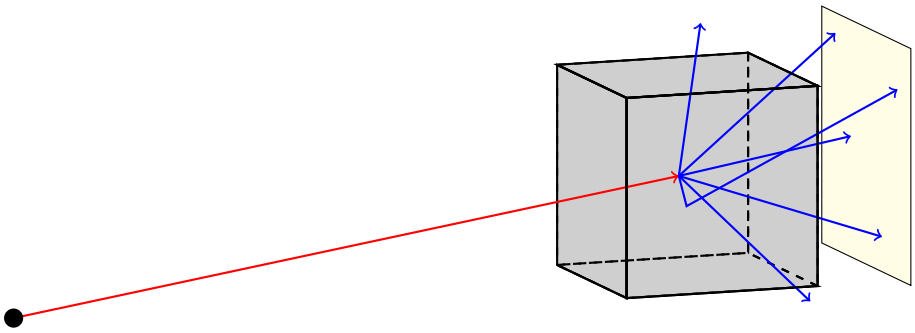


Figure 3.12: The splitting procedure. The red path represents the photon as it travels from the source. It then undergoes an interaction at the center of the volume. At the interaction point, we create several copies of the photon and simulate the interaction for each of them. The weight of the split photons is then scaled such that the total weight is conserved.

Position Dependent Splitting

If we want to minimize the variance of the result we could improve the splitting technique by changing the splitting parameters depending on where in the phantom the interaction occurs. If we approximate the spectrum to be mono energetic, a relatively strong assumption, we can calculate the variance of the result in one pixel.

If the chance that a photon in voxel i will hit pixel j is p_{ij}, then the standard error in pixel j from n_i photons passing through voxel i will be given by a normalized binomial distribution with variance:

$$\sigma_{ij}^2 = \frac{p_{ij}(1-p_{ij})}{n_i}$$

Assuming independence, we can thus estimate the total variance of pixel j by summing over all the N_v voxels scaled by the expected contribution, which can be assumed to be proportional to the number of photons expected to interact in the voxel P_i^{Interact}.

$$\sigma_j^2 = \sum_{i=1}^{N_v} P_i^{\text{Interact}} \frac{p_{ij}(1-p_{ij})}{n_i}$$

By summing over all N_p pixels we thus get the total variance of the detector result.

$$\sigma^2 = \sum_{j=1}^{N_p} \sum_{i=1}^{N_v} P_i^{\text{Interact}} \frac{p_{ij}(1-p_{ij})}{n_i}$$

To minimize the variance, we have to solve the optimization problem

$$\begin{aligned} \underset{n_1,n_2,\ldots,n_{N_v}}{\text{minimize}} \quad & \sum_{j=1}^{N_p}\sum_{i=1}^{N_v} P_i^{\text{Interact}} \frac{p_{ij}(1-p_{ij})}{n_i} \\ \text{subject to} \quad & \sum_{i=1}^{N_v} n_i = N \end{aligned}$$

This problem can be solved using the method of Lagrange multipliers and we find that

$$n_i \propto \sqrt{P_i^{\text{Interact}} \sum_{j=0}^{N_p-1} p_{ij}(1-p_{ij})}$$

Since the probability of a photon hitting any single pixel is small, we can assume $p_{ij} \ll 1$, the expression thus simplifies to

$$n_i \propto \sqrt{P_i^{\text{Interact}} \sum_{j=0}^{N_p-1} p_{ij}}$$

This result can be given a simple interpretation by observing that

$$\sum_{j=1}^{N_p} p_{ij} = P_i^{\text{Hit}} = \text{Probability that particle from voxel } i \text{ will hit the detector}$$

The number of simulated interactions in each voxel should thus be proportional to the square root of this probability that the photons will hit the detector. However, since we can only adjust the splitting parameter, N_s, we have to scale N_s by the probability P_i^{Interact} that an interaction will occur in voxel i. We thus find that an estimate of the optimal splitting parameter

$$N_s \propto \sqrt{\frac{P_i^{\text{Hit}}}{P_i^{\text{Interact}}}} \tag{3.2}$$

To approximate this probability, the splitting parameter was varied by assuming that the volume studied was a water wall of thickness 20 cm, approximating a human head.

The method was implemented by first performing a calculation in a pure water cube. A simulation was performed and the two values P_i^{Hit} and $P_i^{Interact}$ were calculated

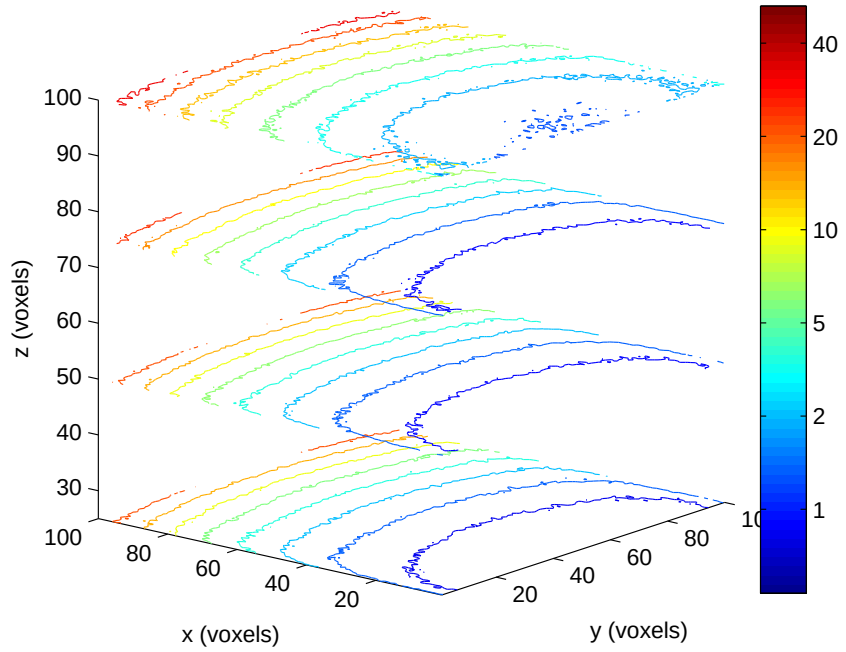


Figure 3.13: Logarithmic plot of the optimal splitting parameters in a pure water geometry.

for each voxel. From this, the positional dependence of the splitting parameter N_s was calculated according to 3.2. The volumetric data is presented in Figure 3.13.

We see that the parameter is largely z independent, and approximately y independent. Because of this, we approximated N_s with

$$N_s^{\text{approx}} = N_0 e^{x/(5.4[\text{cm}])}$$

where N_0 is a tuning parameter. This approximation is shown in Figure 3.14. We see that the approximation agrees well with the calculated values except for small deviations close to the edges.

3.4.2 Russian Roulette

Two Russian Roulette (RR) methods were tested, one positional/directional method where the photons were killed if they were leaving the volume, and one spectral, where the phase space was pre-processed and photons with a low chance to contribute were killed.

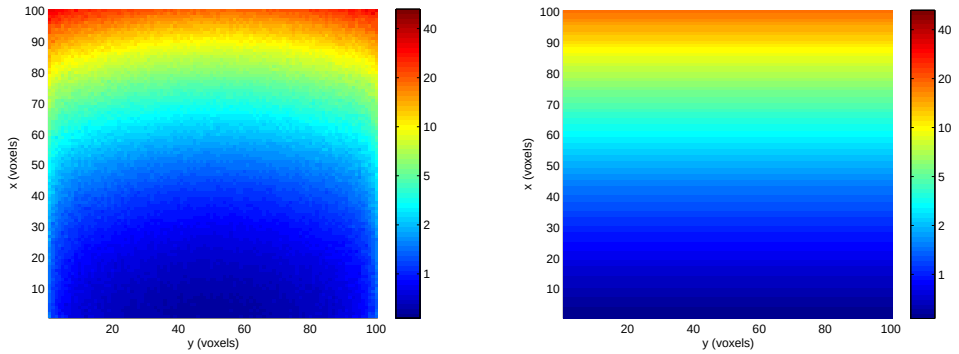


Figure 3.14: Comparasion of the calculated optimal splitting parameter (left) and the exponential approximation (right) for a traverse slice in the middle of the volume.

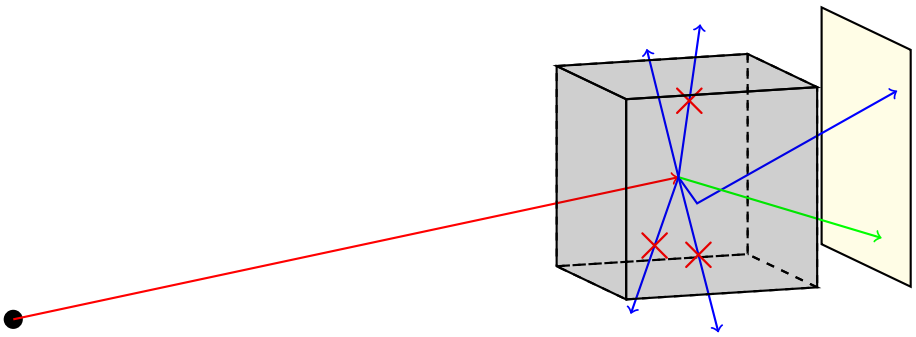


Figure 3.15: Russian Roulette method. The red line is the path of the photon before any interaction. An interaction then occurs in the center of the volume. The blue lines represent possible photon paths after the interaction that do not point towards the detector. The crossed over paths are killed photons. In this example, one blue photon interacts again and then hits the detector and one misses the detector, while three of the photons that would have probably missed the detector were killed. The green path is aimed at the detector and will not experience the roulette.

Positional and Directional

In the positional/directional RR method, the photons were killed if they were likely to leave the volume and miss the detector. This was determined by checking after each interaction if the current path intercepts the detector, as done by Mainegra-Hing*et al.*[15]. If the photon path does not intercept the detector, it was removed with probability P. If the photon was not killed, its weight was rescaled by $1/(1-P)$.

Spectral

Another method tested was to pre-emptively perform Russian Roulette based on the energy of the photons, before they were even added to the phase space. Using calculations analogous to the ones done for the splitting parameter, but taking the energy of the photon into account, we find that the number of simulated photons with energy E should be proportional to

$$n_E \propto \sqrt{P_{Hit,E} \cdot R(E)}$$

where $P_{Hit,E}$ is the chance that the particle will hit the detector (as scatter) and $R(E)$ is the detector response function (ignoring angular dependence). In this thesis we have taken $R(E) = E$ as discussed in section 3.3.3.

Calculating the quantity $P_{Hit,E}$ is non-trivial, and a simulation method was selected. The simulation was performed in a true head geometry reconstructed from a MR scan. The probability of hitting the detector as a function of the photon energy was recorded. The running time was also recorded. The results are shown in Figure 3.16. We see that it takes significant time to simulate the low energy photons, but they have a very small chance to hit the detector.

In Figure 3.17 the calculated values of n_e are shown together win a linear fit, $n_E = c \cdot E$. We see that the linear fit agrees quite well with the calculated values at higher energies, and is an acceptable approximation at lower. We chose to use this linear fit since using the exactly calculated curve would give numerical overhead, without any guarantee that the result holds to this precision for other geometries.

Using this result we then performed Russian roulette on the phase space by killing the photon with probability $1 - n_E$. If we keep the photon, we scale its weight by $1/n_E$ to ensure that the calculated result is correct.

3.4.3 Forced Detection

The forced detection technique was implemented in a two step procedure. In the first step, photons are transported through the material in the usual manner until they experience an interaction.

After the interaction, the second step of the code is engaged. One photon is created for each pixel in the detector. The weight of the photons are scaled according to the probability that the path of the photon would point towards the pixel after the interaction. Each such photon is then attenuated exactly as it travels through the medium, in each step reducing its weight according to the probability of an interaction. When the attenuated photon hits the detector, it is scored in the target pixel.

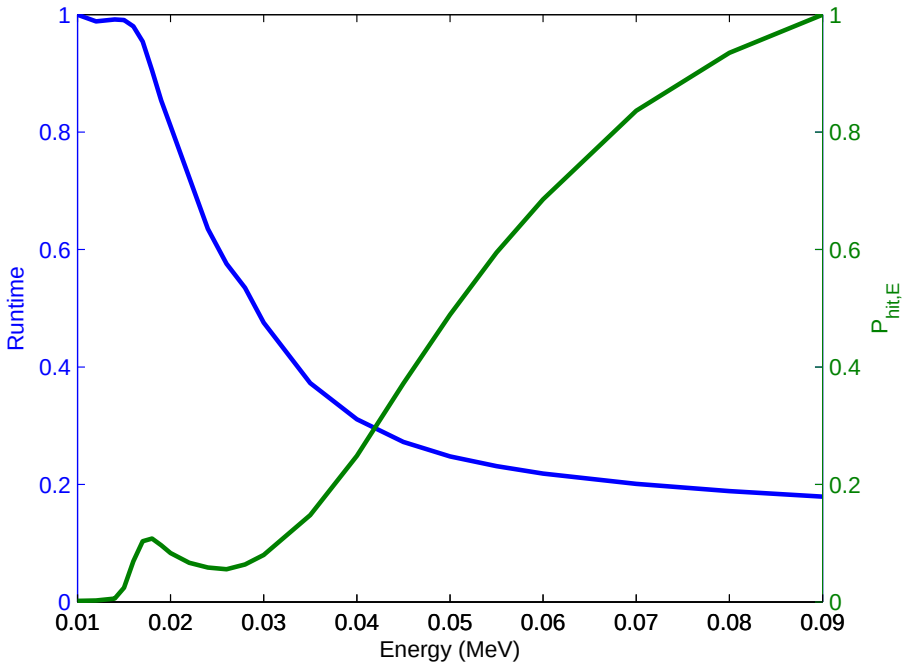


Figure 3.16: Calculated $P_{Hit,E}$ and runtime in a head geometry for a range of energies. Values are given in a normalized scale.

After second step is done, the code continues with step one and transports the photon further. If it experiences an interaction it once again enters the second step. If it has not experienced any interactions and hits the detector, it is scored as a primary photon. If it has interacted, it is not scored when it hits the detector. This is because it has already been scored in the second step of the code.

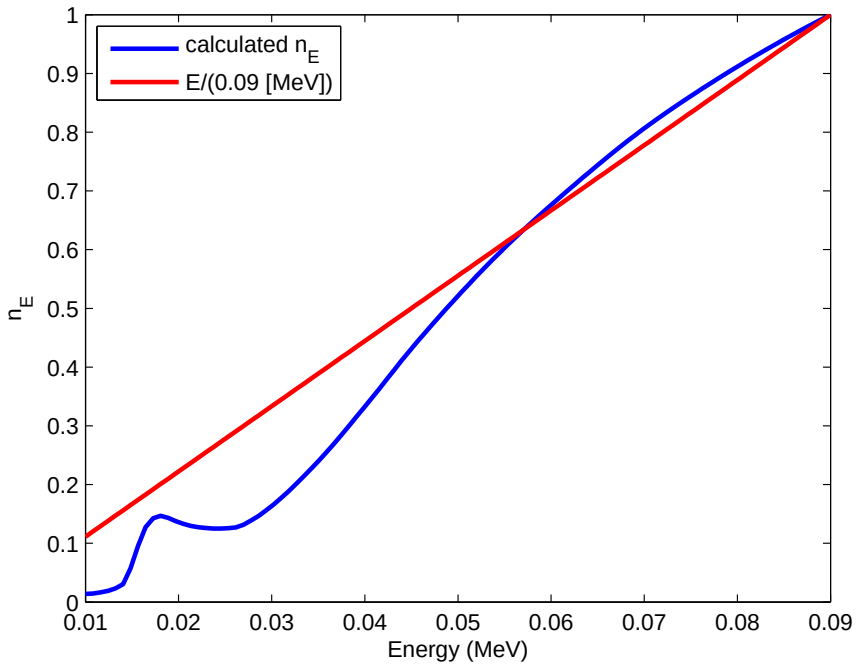


Figure 3.17: Calculated n_E with a linear approximation.

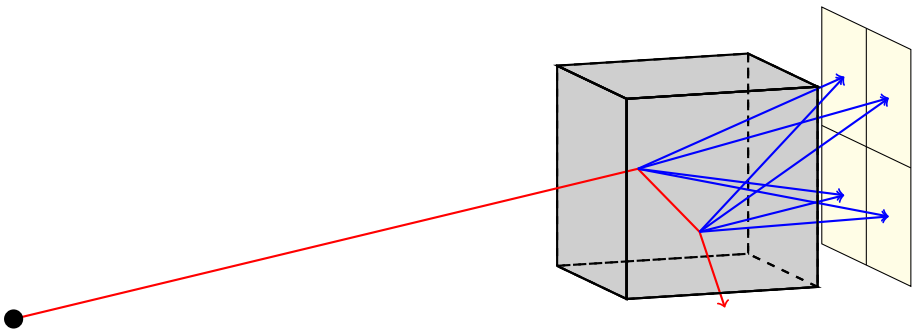


Figure 3.18: Forced detection method illustrated for a 4 pixel detector. A particle (red) is generated at the source and travels through the volume, where it interacts in two positions and then misses the detector. From the interaction positions the photons are attenuated exactly (blue) to the detector.

3.5 Filtering Methods

Since the scatter has very low spacial frequency it can be filtered aggressively. For simplicity, we decided to use a linear filter.

To decide what filter to use, several scatter distributions for a range of human heads were calculated to high accuracy. These were then Fourier transformed, and the frequency spectra was inspected.

We found that 95% of the energy in the scatter was contained at frequencies below 0.16 [cm^{-1}]. A Gaussian low-pass filter with a cutoff frequency of 0.45 [cm^{-1}] was then selected. This cut-off should ensure that no significant data is lost.

Since the scatter varies smoothly with the gantry angle, the scatter was only calculated for every 10 images[2], and then linearly interpolated for the images in between.

3.6 Scatter Removal

The final step is to remove the calculated scatter from the images. The removal of scatter from the images require some care. This is because the fraction of scatter in the measured image may be up to 70%. If all of the calculated scatter is removed from the image, a miscalculation in the scatter may give non-physical negative values in the image. This in turn makes the reconstruction bad.

To remedy this, we do not remove all of the calculated scatter. Instead, we only remove a fraction of the scatter, dependent on how much signal was measured at the detector. The fraction we remove should be proportional to the calculated scatter if the calculated scatter is significantly smaller than the value on the detector, and approach the detector value when the calculated scatter is large. Finally, we can never remove more than the measured detector value.

Specifically, we need a function $f(x, y)$ where x is the calculated scatter and y the measured image value that has the following properties

$$\begin{aligned} f(x,y) &\approx x \quad x/y \ll 1 \\ f(x,y) &\approx y \quad x/y \gg 1 \\ f(x,y) &\leq y \end{aligned}$$

A useful family of such functions is

$$\hat{f}_c(x,y) = y(1 - e^{-(x/y)^c})^{1/c}$$

[2]This corresponds to a $\approx 5^o$ rotation.

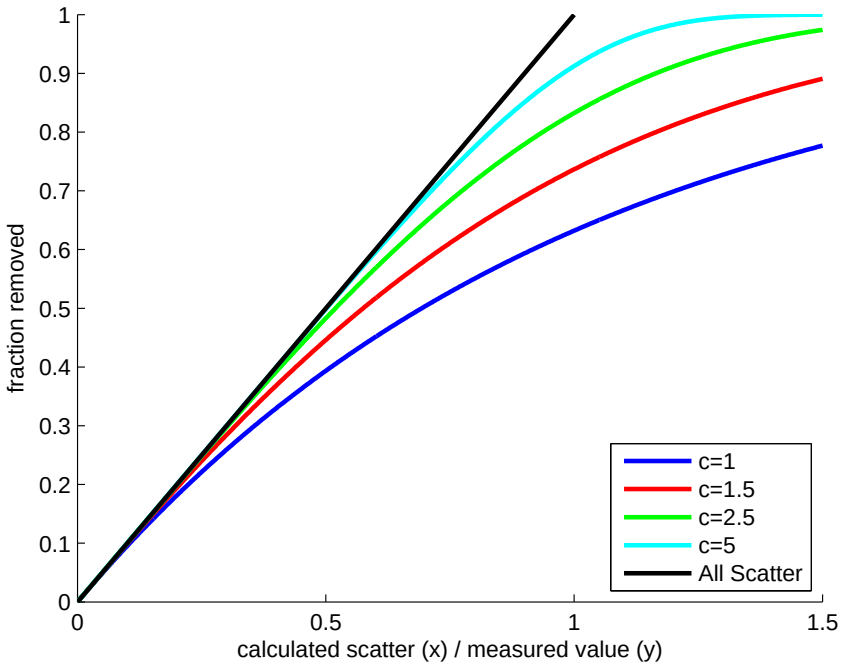


Figure 3.19: Example of the scaling function $\hat{f}_c(x, y)/y$ for some values of c and the non-scaled (black) value for comparison.

where $c > 0$ is a parameter. This family is illustrated in Figure 3.19. We see that as c increases the removed scatter gets closer and closer in agreement with the calculated value. If we are confident in our calculation, we can thus set a high c value, and if we are not, we can set a lower value. For the analysis in this thesis c was set by trial and error to 2.

Using this function, the scatter corrected images are calculated as:

$$\text{Image}_{\text{corrected}} = \text{Image}_{\text{measured}} - \hat{f}_c(\text{Scatter}, \text{Image}_{\text{measured}})$$

3.7 Code Optimizations and Details

Several optimizations were implemented in the code to make it as fast as possible, these will be outlined in this section. Many of the optimizations are specific to the GPU environment, which require some extra care to achieve optimal performance.

3.7.1 Random Number Generation

We tested the CURAND implementation of the MT and Xorshift algorithm as well as the implementation of the Multiply with Carry algorithm used by Hissony *et al.*[13] and the Linear Congruential method. We compared the algorithms using a simple test program and the Nvidia Visual Profiler to extract average run times and register use. A comparison of the algorithms is shown in Table 3.2. Note that the period is not the only or even best measure of statistical properties, but it was deemed the most relevant in this case. We found that the MT algorithm was prohibitively slow due to a large number of global memory accesses. The Xorshift algorithm however provides comparable performance to the Multiply with Carry algorithm, it does however require significantly more registers.

We decided to use the multiply with carry algorithm since it was faster, used less registers, and the statistical loss was negligible in our case. We also noted that in real runtime applications multiply with carry performed better than indicated in this table. This was possibly because of the more efficient register use which was not as noticeable in the isolated test.

The period of the multiply with carry method is also large enough for even the longest simulations.

Algorithm	numbers/s	Registers	Period
CURAND MT	0.032	3	2^{19937}
CURAND Xorshift	0.352	5	2^{192}
Multiply with Carry	0.478	1	2^{128}
Linear Congruental	1.000	0	2^{32}

Table 3.2: Comparasion of RNG algorithms. The values are normalized according to the Linear Congruential generator.

3.7.2 Memory Use and Accesses

In the CUDA programming model, global memory accesses are in general very expensive. A single global memory access takes approximately 700 clock cycles. As a comparison, a single precision multiply add takes 1 cycle, and most transcendental functions can be calculated in around 20 cycles. If the kernel is also bandwidth limited, the warp may need to wait for its global memory access to be processed, further slowing the operation down.

To compensate for the slow global memory, CUDA can allow other warps to perform work while one warp waits for its memory access to finish. For CUDA to be able to do this, the extra warps need to be able to fit into memory. There are two limiting factors to this, register use, and shared memory use. According to our calculations

in appendix B we find that to never have the warps waiting for data we need to keep

$$200 \geq \text{registers used / thread}$$
$$3000 \geq \text{shared memory used / warp}$$

We see that we are not severely limited by register use, but that the amount of shared memory is relatively small. The amount of shared memory does for example allow us to store approximately 3 photons per thread in shared memory. This is a noticeable limitation if we want to implement variance reduction methods that utilize the shared memory. For example, it makes methods such as work stealing queues hard to implement using shared memory.

3.7.3 Built-in Function Calls

The only function calls in the code to built-ins are mathematical functions and memory access functions, including atomic functions[3]. The code is compiled with the `-use_fast_math` option to speed up mathematical calculations, this comes at a small theoretical accuracy loss, but no effects of this were seen in tests. All of the code was written using single precision, which gives a significant speed-up as compared to double precision.

Care was also taken to select the appropriate built in function in each step. For example, the scattering angle φ is sampled in every step, and its cosine and sine are calculated. To reduce time calculating these, the function `sincospif(2.0f*rand(),&sinphi,&cosphi)` was used instead of separate calls to sin and cos. The use of the "pi" function also gives a slight speedup and somewhat higher accuracy.

The only write to global memory done is the scoring of the photon when it hits the detector. Here the `atomicAdd(&arr,value)` function was used. Since there are approximately 500000 pixels in the detector and at most about a thousand threads running at once there is only a small chance of collisions in this call.

3.7.4 Thread Coherence

Thread divergence is a major issue on GPUs and avoiding it gives notable speed-ups. Several measures were implemented to achieve optimal runtime efficiency. A simple measure was to ensure that all code that was equal between branches were moved to be executed in the same place. For example, the interaction functions (Compton, Rayleigh, etc.) only output the cosine and energy of the photon after the interaction, and the rotation is performed simultaneously for all threads.

[3] An atomic function is a hardware function that is used to perform operations on shared or global memory without collisions

Another optimization was to ensure that all threads in a warp finish execution simultaneously, so that no threads have to wait for others to finish. This was implemented by loading a small set of photons into shared memory and having the threads simulate the photons multiple times until the total number of simulation has reached the required number. To ensure that photons that take longer to simulate do not get under-represented in the result, the threads access the photons in an uniform manner. This optimization gave a $\approx 50\%$ speedup over each thread simulating one photon.

3.7.5 Numerical Precision

In long runs we may expect to experience problems with numerical precision due to adding many small numbers. Machine epsilon for IEEE-754 floats is $\approx 10^{-7}$, with 500000 pixels on the detector we thus need $\approx 10^{12}$ photons before numerical errors become large, this is at least two orders of magnitude higher than typical run sizes.

To avoid this problem for larger runs, such as overnight runs, the program was made to run in batches. In each batch, a detector image is calculated with $\approx 10^9$ particles, the sub-sums are then added together into a final result. This proved no noticeable performance loss, and allows the program to run approximately 10^{20} photons before numerical precision becomes an issue.

4 | Results

Here, the results of the study will be presented. First the performance of the code will be evaluated, and then we discuss the various variance reduction techniques tested to increase performance.

After that, we verify the physical accuracy of the model against PENELOPE. Finally the algorithm is tested in a bigger context by performing a volume-reconstruction using the scatter-corrected images.

4.1 Performance

The raw code without variance reduction is fast, simulating approximately $3 \cdot 10^9$ photons per minute in a real world geometry. Of this, approximately 20% is spent on photon creation and scoring. About 80% of the time is thus on moving the photon forwards and simulating interactions.

To evaluate the performance, the code was profiled using the Nvidia Visual Profiler. The visual profiler reports few problems with the code, except for a 25% branch divergence overhead and a 30 % global memory instruction replay overhead. These errors are probably hard to reduce significantly further. A nonzero branch overhead is unavoidable since there are several paths that the code may take in each step, such as different interactions, the memory access patterns are also inherently non-coalesced since the photons move independently.

4.1.1 Variance Reduction Methods

Several variance reduction methods were evaluated. Their performance was measured using the efficiency ϵ, defined as:

$$\epsilon = \frac{1}{\sigma^2 T}$$

Where σ is the standard error of the result, and T the runtime. Since we know that MC methods converge with speed $\sigma \propto T^{-1/2}$. ϵ will actually be T-independent, and only depend on the methods used.

The standard error was calculated as the root mean square of the standard errors of each pixel. The standard error of one pixel was defined as:

$$\sigma_{\text{pixel}} = P_i - \hat{P}_i$$

where P_i is the measured pixel value, and $\hat{P}_i$ is a gold standard, calculated with 10^{12} particles.

The tests were performed using the head phantom geometry on a Nvidia Quadro 4000 graphics card.

Woodcock ray tracing

The woodcock ray tracing method gave a significant efficiency gain of $\approx 180\%$ when compared to the regular ray tracing method. This was due to significantly reduced warp splitting and a better arithmetic-memory quotient.

Splitting

Two methods were tested for splitting, static and position dependent. They both gave small efficiency gains before filtering, but after filtering the data, the efficiency was lost. This was likely because the smaller number of primary photons simulated per unit time introduced bias in the filtered result.

Russian Roulette

The spectral roulette method gave a small efficiency gain of $\approx 15\%$.

No other implementation of Russian Roulette was found to give any efficiency gain, instead yielding noticeable efficiency losses. This is believed to be due to extensive warp splitting and memory overhead caused by the method.

Forced Detection

The forced detection method tested gave a negative effect on the efficiency, approximately halving it.

4.2 Physical Accuracy

The total error of the algorithm is the sum of the physical and numerical errors. To achieve a low total error we thus need a low physical error.

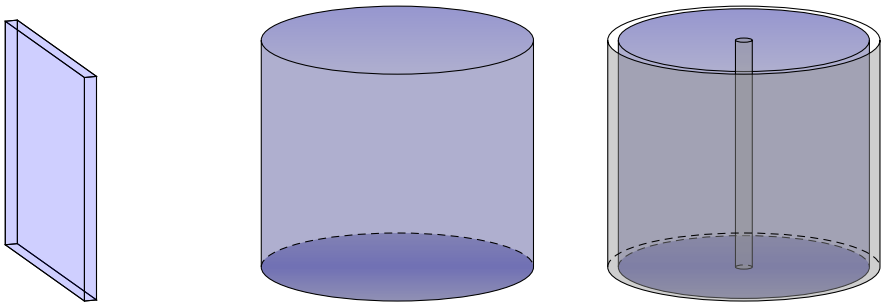


Figure 4.1: Illustration of the three test geometries used.

The physical error was measured by testing the code against the well known PENELOPE code for some simple geometries. It was also tested in a real world setting by simulating the result of an actual CBCT run.

4.2.1 PENELOPE comparison

The code was compared to PENELOPE results for 3 different geometries. These geometries were:

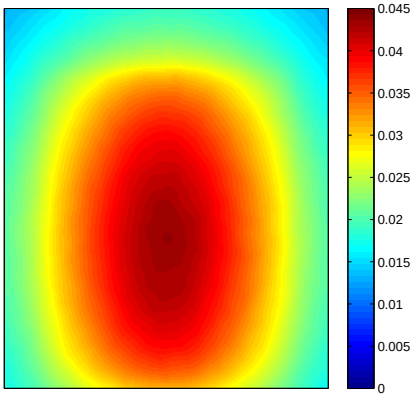
1. Thin plane of water
2. Solid water cylinder
3. Water cylinder with bone shell and core

These were intended to be a set of representative yet simple scenarios that were easily represented analytically. The geometries are illustrated in Figure 4.1. They were scaled so that they did not fill the whole simulation volume, only covering about half. This was to get good results from the edge effects of the simulation.

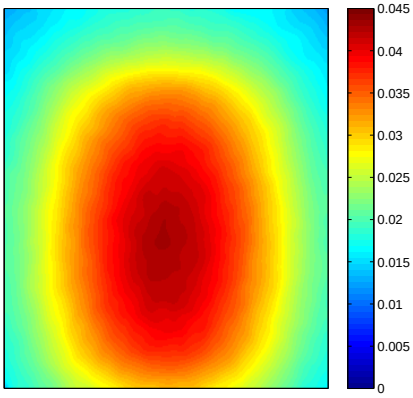
The PENELOPE test was performed used an in-house variant of the code run on a 128-core CPU cluster. Both simulations were done using the same geometry and using photons from the same phase space. To facilitate accurate comparison between the results, they were normalized according to the vacuum (gain) result.

The results are shown in Figure 4.2, and histograms of the errors are shown in Figure 4.3. The results agree with each other with high accuracy, with relative errors of approximately 2%.

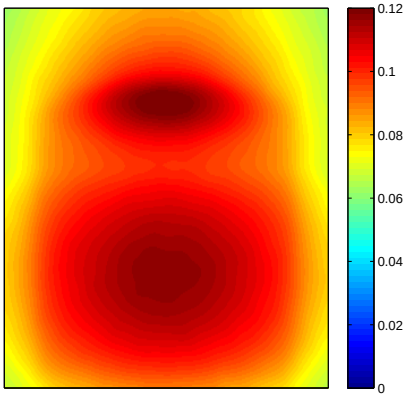
The results for the water cylinder with bone shell and core geometry is the geometry with the most noticeable error. The secondary peak at the top is approximately 5% higher in the PENELOPE calculation than in this calculation.



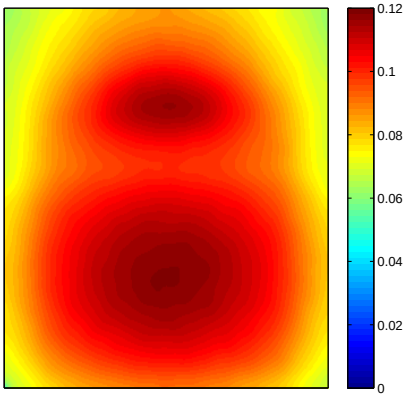
(a) Penelope, sheet



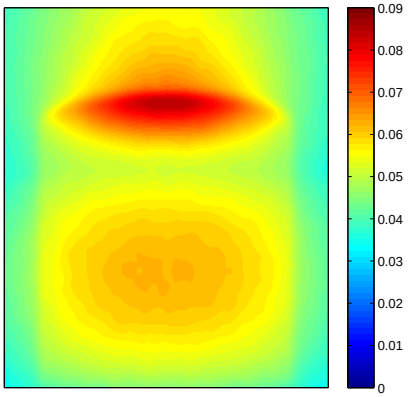
(b) This, sheet



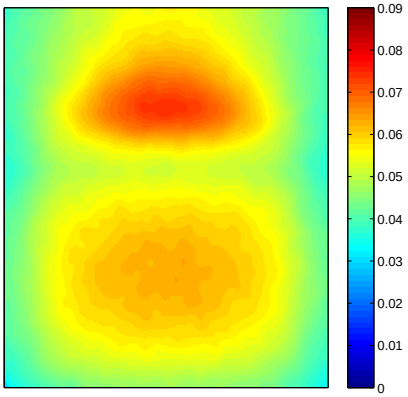
(c) Penelope, cylinder



(d) This, cylinder

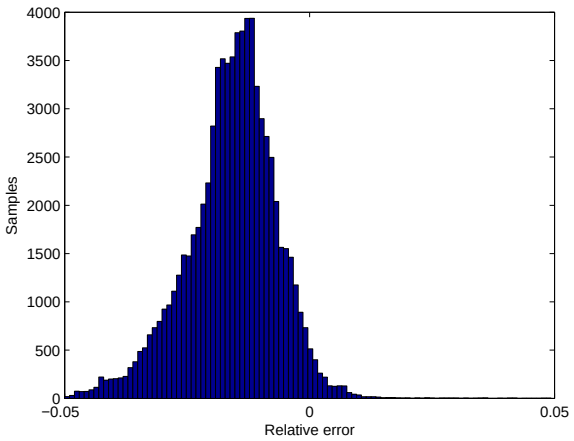


(e) Penelope, Cylinder with bone

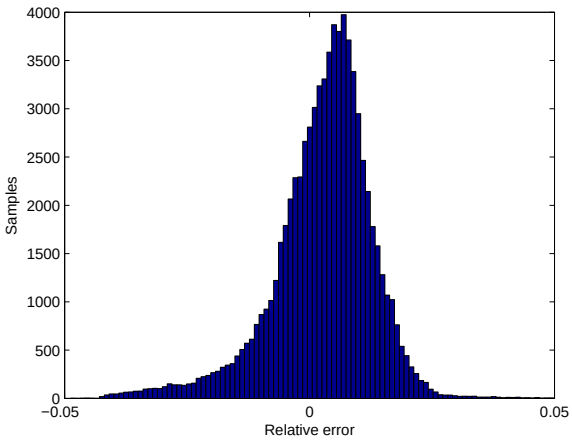


(f) This, Cylinder with bone

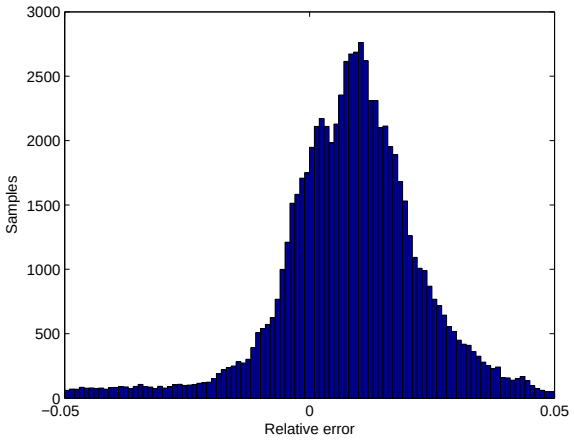
Figure 4.2: Comparison of the calculated scatter distributions using PENELOPE and this thesis code.



(a) Sheet



(b) Water cylinder



(c) Cylinder with bone

Figure 4.3: Histograms of the relative errors in the simple geometries.

4.2.2 Head phantom

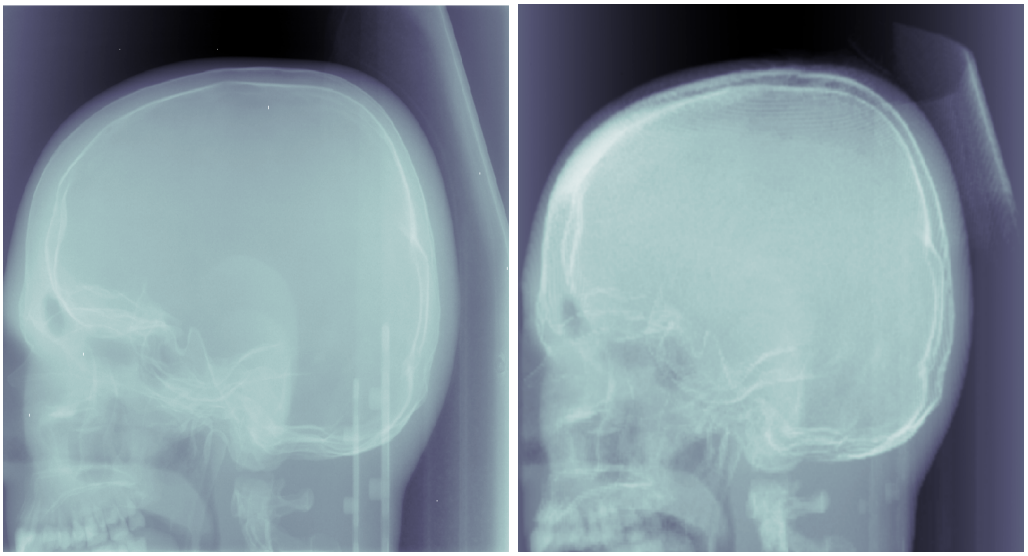
Tests were also performed on an artificial head phantom against real world data. In this test, there was no way to extract only the scattered photons, and we have to compare the full detector signal against the measured result. This tests did however allow us to test the full work flow against the real world result. The head phantom used is shown in figure 4.4.

The result of an simulation of the detector response with the head phantom is shown in Figure 4.5. We see that the total results are very similar, but with some noticeable discrepancies. Specifically, the attenuation in the forehead is grossly exaggerated in this example. This is due to the segmentation method used, which overestimates the amount of cortical (dense) bone in the forehead. There is also a noticeable underestimation of the attenuation of the pillow to the right in the figure.

These errors are more easily seen if we study the relative error of the result, as in Figure 4.7. We note that the attenuation of the facial region seems to be overestimated while the attenuation of the teeth are underestimated.

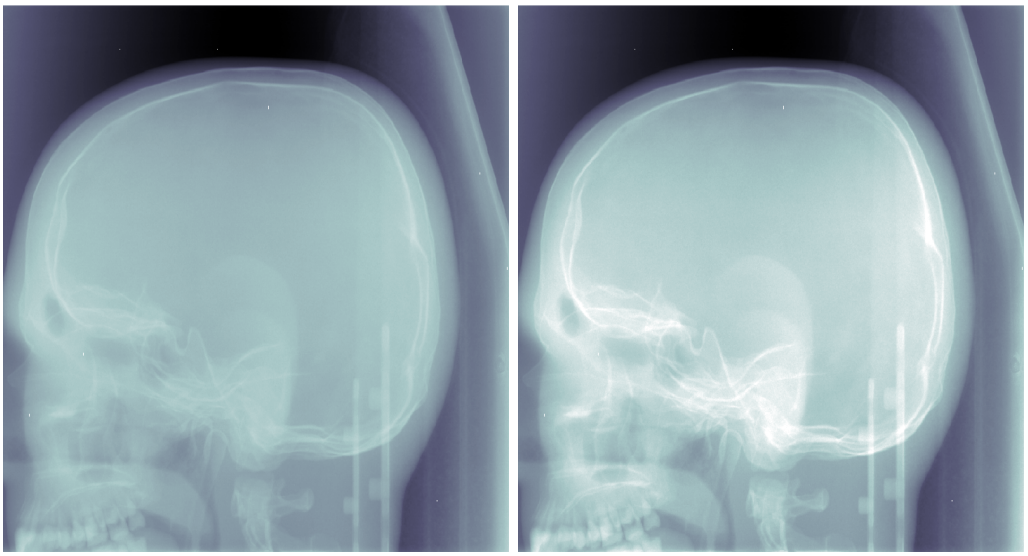
The result of subtracting the calculated scatter from a measured X-ray image is shown in Figure 4.6. Here we see a noticeable increase in sharpness and contrast due to the scatter compensation.

Figure 4.4: An image of the head phantom used in the study.



(a) Measured (b) Simulated

Figure 4.5: Comparison of the detector result with gantry angle $\theta = 0$ measured using the CBCT and simulated using this thesis' code. The values were normalized according to the gain (vacuum) result.



(a) Measured result (b) With scatter compensation

Figure 4.6: Comparison of the measured result without and with scatter compensation shown in equal color-scales.

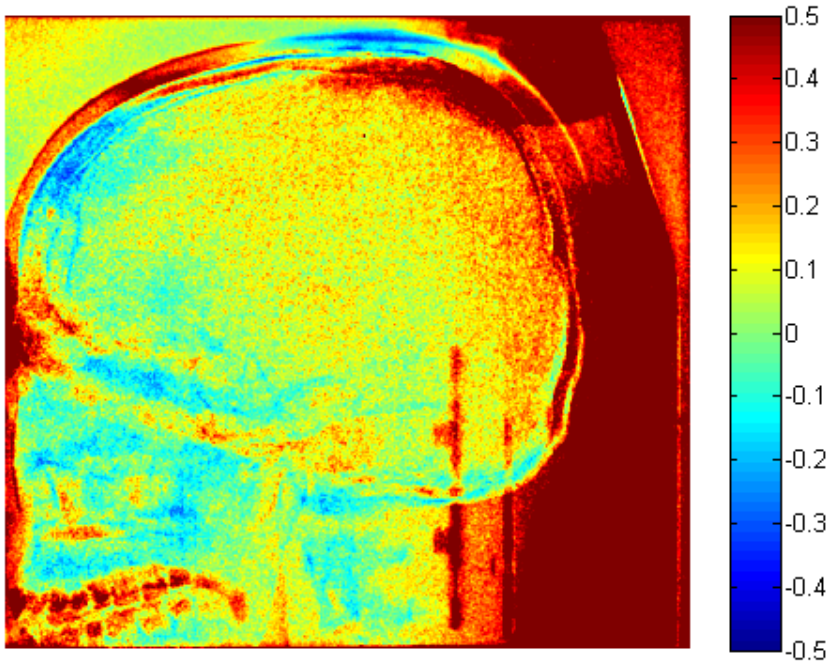


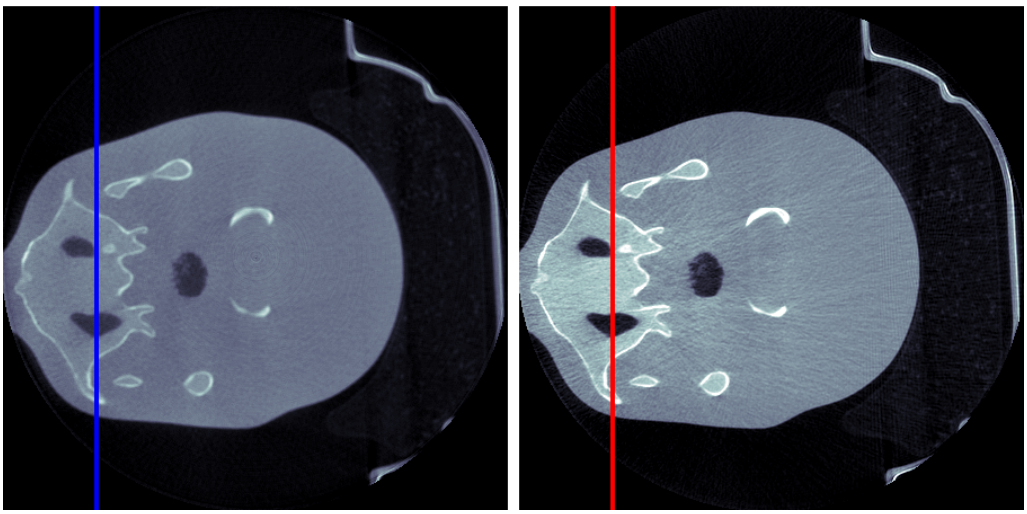
Figure 4.7: Relative error of the calculated result as compared to the detector response (calculated − measured)/measured.

4.3 Effect on reconstruction

We finally tested the scatter compensation in a real world setting by inserting the scatter compensation code into the full reconstruction algorithm. A cross-section of the reconstructed volume is shown in Figure 4.8. The scatter compensation gives a noticeably higher contrast in the reconstructed volume, and the trabecular bone in the cheekbones is notably different from the soft tissue.

The effect is especially evident if we look at the values along a line, as in Figure 4.9. Here the significantly increased contrast is clearly evident. The values are also much closer in agreement with the correct Hounsfield values, especially for the trabecular (spongy) and cortical (dense) bone.

One effect that can be seen is the slight upwards bulge of the result. The HU values are underestimated at the edges, and overestimated in the middle. This is



(a) Without scatter compensation (b) With scatter compensation

Figure 4.8: A slice of the reconstructed volume in the nasal region with the head facing to the left in the images. The results are shown with and without scatter compensation. The values along the colored lines is shown in Figure 4.9

likely due to an effect called beam hardening. Beam hardening is caused by the poly-energetic energy spectrum generated by the x-ray tube. As the x-rays travel through matter, the low-energy photons are attenuated very quickly, while the high energy photons travel further. Because of this, the assumption that the x-rays are attenuated exponentially in the material is slightly wrong.

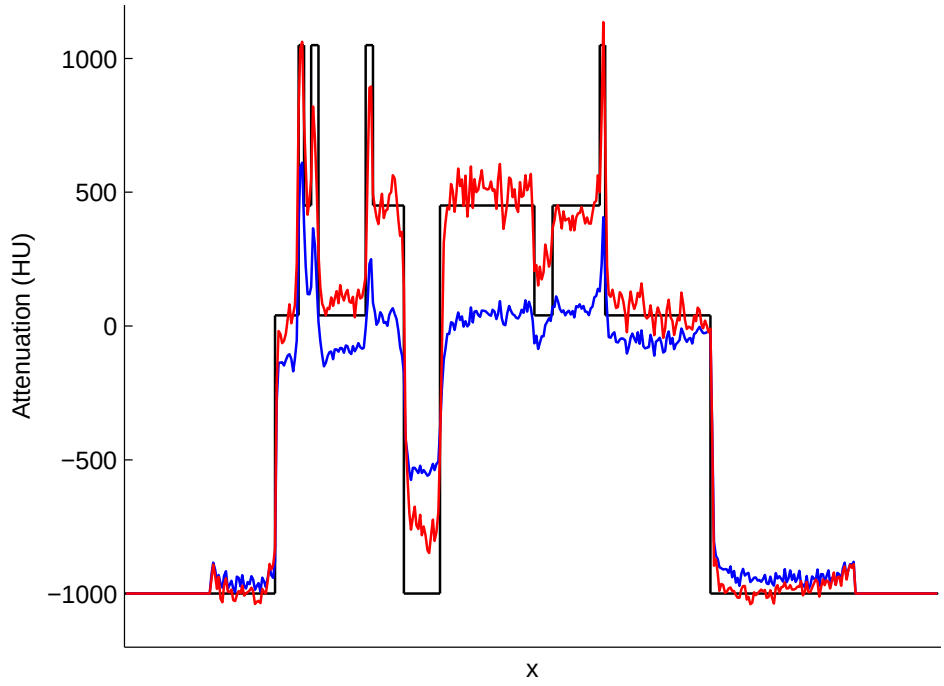


Figure 4.9: Comparison of the result along the lines in Figure 4.8, with the scatter compensated result in red and raw result in blue. Correct values given from the manufacturer are given in black. With the scatter compensation we can see that the phantom has three materials, loosely corresponding to water, trabecular (450 HU) bone and Cortical (1050 HU) bone. On average, the errors in the HU values are reduced by approximately 70%.

5 | Discussion

We will now discuss the results in depth. We will look at what went well, and try to problemize the things that did not go as expected.

5.1 Performance

Significant work was put into making the code as fast as possible, and many different methods were investigated. Overall, the resulting code is fast, but this was largely from the efficient GPU code written and not from the variance reduction techniques.

Several small things gave large performance improvements. These included reusing threads, loading the phase space batch-wise into shared memory and projecting the photons onto the head. These improvements gave an approximately 200% performance gain over the trivial implementations.

Other things such as the lookup table based interaction simulation also gave small but notable speed-ups. Other small things that improved the speed of the program was correct use of built in functions and pre-calculations.

5.1.1 Variance Reduction

The speed-ups obtained from implementation of variance reduction methods were significantly smaller than those obtained by other authors. Where many other authors report speed-ups of up to $\times 60$ or more, the largest speed-up obtained was approximately $\times 3$ from the woodcock stepping method.

This lack of speed-ups can probably be explained by the GPU's difficulties with branching code. Most variance reduction methods involve significant branching, for example, in the Russian Roulette method, one photon, and hence thread, may be killed of. There is then two options, either the thread is discontinued, but this simply means that it is performing no work. The other option is that some new work is generated for the thread, such as simulating a new photon. The generation of this new work is however a form of branching, since the other threads has to wait for the thread.

For the forced detection technique, some of the efficiency loss may be explained by the filtering methods used. Since the effect of the technique can be likened with a

filter, it does not synergize well with a post-processing filter.

5.2 Accuracy

The accuracy of the results obtained are overall promising. The comparison with PENELOPE shows that the physical modelling closely agrees with other well esteemed codes, but the errors sometimes exceed 5%. These larger errors are big enough to cause noticeable errors in the reconstructed volume. These errors could be due to PENELOPE using quadratic geometries, which is better suited for the simple test cases we investigated. There may also be small errors in the physical modelling.

When testing the code against the actual measured result, the errors were larger, in the range of 20% for most of the head, with more significant errors due to the pillow. The size of the errors show that we are close in our modelling of the physical processes, but that there are some way left to go. The most significant improvement could probably be obtained by improving the quality of the segmentation, but a better model of the detector could also be worthwhile.

5.3 Reconstruction

Using the scatter reduction in the reconstruction gave a significantly more accurate result, with all values being closer to the correct values, and with significantly higher contrast. The new signal was however somewhat more noisy, an effect that may need to be accounted for should this method be used in a commercial product. This noise is largely from the quantum noise in the input images.

The errors caused by the beam hardening effect are probably the next problem that has to be addressed after scatter compensation. One way to do this would be to calculate a conversion factor from the measured values to the expected values if the spectrum was mono-energetic.

6 | Conclusions

Overall the work was a success. The developed code and methods are able to calculate the scatter distribution at all measured angles in a real world geometry in approximately one minute of computational time. This was mostly due to the very parallel nature of the problem, and the computational power of the GPU.

We noted that variance reduction methods are not as good when implemented on GPU's as on CPU's, but that some methods, such as the woodcock stepping method, still give noticeable efficiency gains.

The results for the scatter computations were promising, but there were still noticeable errors in the calculated detector result when compared to the measured result. This suggests that we are missing some important parts.

Finally, the resulting reconstructed volume has significantly higher contrast and more accurate HU values than without scatter compensation, indicating that the calculated scatter was correct.

6.1 Further work

While a wide range of methods have been studied in the course of this work, much more could be done to make the code more efficient and accurate. We will now discuss some ideas that have come up during the work.

6.1.1 Sources of error

There seems to be a significant error associated with the segmentation of the reconstructed volume. This is probably in part because the data is bad, but also due to the very simple threshold method used. A improved method would give higher accuracy in the classification of individual voxels, but also possibly allow for more materials to be added, such as brain matter, bone marrow and fat. The modelling of the pillow could also use some extra work.

A related issue is that we disregard what happens outside of the volume. In reality, small amounts of scatter may leave the volume and be reflected back by the patients body or by some other objects. Accounting for these effects accurately may prove very challenging, but there may be approximate methods.

Another significant source of error may be the approximations made in the interaction modelling. One is the independent atom assumption, used in both the Rayleigh and Compton scatter calculations. Poludniowski *et al.*[24] reports that using the independent atom approximation for Rayleigh scattering introduces an $\approx 5\%$ error in the scatter. Accounting for this discrepancy would probably improve accuracy.

In the Compton interaction we also disregard the Doppler effect. This could give a noticeable error, from the data given in PENELOPE[6], we see that this leads to us overestimating the energy of the scattered photons, thus overestimating the detector response. Accounting for these effects would most likely come at a noticeable performance price, but may be worth doing.

The model of the detector is also relatively primitive and it has not been properly tested. A way to improve this would be to measure or calculate the detector response function in more detail than what was done in this thesis.

6.1.2 Possible Speed-ups

A simple way to reduce the amount of work performed is to not spend any work on tracing photons that will not interact. A way to do this is to first pre-calculate the convex hull of the head, possibly in an approximate manner. This can then be used for two things. If a photon is determined to miss the convex hull, which can be calculated quickly, no simulation is needed, and the photon can be transported to the detector. If it hits the hull, it can be transported to the inside. When the photon leaves the hull, we know that it will never return, and the photon can be scored at the detector.

Several variance reduction methods exist that may give speed-ups. One such method is dynamic splitting as discussed in Mainegra-Hing *et al.*[15]. Another option may be that better implementations of the variance reduction methods may give a speed-up that is higher than the ones found.

A final and significant way to increase the speed of the calculation is to use good filtering methods. Bootsma [18] discusses such a method in his doctoral thesis. His method is to exploit the fact that the scatter distribution varies smoothly with the gantry angle. He then applies a three dimensional filter to the set of all images. Implementing these filtering methods have been discussed with Bootsma, but the final implementation is outside of this thesis scope. This could probably be combined with the pre-calculation approach discussed in the background of this work.

Bibliography

[1] L. Feldkamp, L. Davis, and J. Kress, "Practical cone-beam algorithm," *J. Opt. Soc. Am. A*, vol. 1, pp. 612–619, June 1984.

[2] J. Wiegert, *Scattered radiation in cone-beam computed tomography: analysis, quantification and compensation.* PhD thesis, Philips Research Laboratories, Aachen, Germany, 2006.

[3] J. H. H. et al., "Atomic form factors, incoherent scattering functions, and photon scattering cross sections," *Journal of Physical and Chemical Reference Data*, vol. 4, no. 3, 1975.

[4] B. M.J., H. J.H., S. S.M., C. J., C. J.S., S. R., Z. D.S., and Olsen, "XCOM: Photon cross section database," 2010. available: `http://physics.nist.gov/xcom`.

[5] B. M.J., C. J.S., Z. M.A., and C. J., "ESTAR, PSTAR, and ASTAR: Computer programs for calculating stopping-power and range tables for electrons, protons, and helium ions," 2005. available: `http://physics.nist.gov/cgi-bin/Star/compos.pl`.

[6] F. Salvat, J. M. Fernandez-Varea, and J. Sempau, *PENELOPE, a code system for Monte Carlo simulation of electron and photon transport.* Facultat de Fisica (ECM and ICC), Universitat de Barcelona., 2011.

[7] I. Waller and D. Hartree, "On the intensity of total scattering of x-rays," *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, vol. 124, pp. 119–142, 1929.

[8] A. Brunetti, M. S. del Rio, B. Golosio, A. Simionovici, and A. Somogyi, "A library for x-ray-matter interaction cross sections for x-ray fluorescence applications," *Spectrochimica Acta Part B: Atomic Spectroscopy*, vol. 59, no. 10-11, pp. 1725 – 1731, 2004.

[9] D. Cullen, J. Hubbell, and L. Kissel, "Epdl97: the evaluated photon library," tech. rep., Lawrence Livermore National Laboratory, 1997. Technical Report UCRL-50400 Vol. 6 Rev. 5.

[10] M. Born, *Atomic Physics.* Courier Dover Publications, 8 ed., 1969.

[11] E. Alerstam, T. Svensson, and S. Andersson-Engels, "Parallel computing with graphics processing units for high-speed monte carlo simulation of photon migration," *Journal of Biomedical Optics*, vol. 13, no. 6, pp. 060504–060504–3, 2008.

[12] A. Badal and A. Badano, "Accelerating monte carlo simulations of photon transport in a voxelized geometry using a massively parallel graphics processing unit," *Medical Physics*, vol. 36, no. 11, pp. 4878–4880, 2009.

[13] S. Hissoiny, H. Bouchard, B. Ozell, and P. Despres, "Gpumcd: a new gpu-oriented monte carlo dose calculation platform," *ArXiv e-prints*, jan 2011.

[14] I. Kawrakow and M. Fippel, "Investigation of variance reduction techniques for monte carlo photon dose calculation using xvmc," *Phys. Med. Biol.*, vol. 45, 2000.

[15] E. Mainegra-Hing and I. Kawrakow, "Variance reduction techniques for fast monte carlo cbct scatter correction calculations," *Phys. Med. Biol.*, vol. 55, 2010.

[16] I. Lux and L. Koblinger, *Monte Carlo Particle Transport Methods: Neutron and Photon Calculations*. CRC Press, 1991.

[17] D. Rogers and A. Bielajew, *The Dosimetry of Ionizing Radiation*. Academic Press, 1990.

[18] G. Bootsma, *Physics and Computational Methods for X-ray Scatter Estimation and Correction in Cone-Beam Computed Tomography*. PhD thesis, University of Toronto, 2013.

[19] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, pp. 3–30, Jan. 1998.

[20] M. Saito and M. Matsumoto, "Variants of mersenne twister suitable for graphic processors," *ACM Trans. Math. Softw.*, vol. 39, pp. 12:1–12:20, Feb. 2013.

[21] G. Marsaglia, "Xorshift rngs," *Journal of Statistical Software*, 2003. Available at `http://www.jstatsoft.org/v08/i14/paper`.

[22] G. Marsaglia and A. Zaman, "A new class of random number generators," *Annals of Applied Probability*, vol. 3, no. 1, pp. 462–480, 1991.

[23] C. Everett and E. Cashwell, "New method of sampling the klein-nishina probability distribution for all incident photon energies above 1 kev (a revised complete account)," Mar 1978.

[24] G. Poludniowski, P. M. Evans, and S. Webb, "Rayleigh scatter in kilovoltage x-ray imaging: is the independent atom approximation good enough?," *Physics in Medicine and Biology*, vol. 54, no. 22, p. 6931, 2009.

A | Rejection Sampling

In the code, several random distributions are sampled, such as the Klein-Nishina distribution and the Rayleigh scatter distribution. In addition to that, distributions are sampled if the photons are generated artificially. Efficient sampling of these distribution is important for performance.

There exists three general methods for sampling a stochastic variable X. The first is to sample $F^{-1}(U(0,1))$ where F^{-1} is the inverse CDF of the variable. This method has an obvious aesthetic appeal, however, in many cases the inverse CDF is not available and it needs to be approximated.

The second is finding some function $g(\mathbf{x})$ such that $g([U(0,1),\ldots,U(0,1)])$ is distributed equally to X. An example of this method is the box-muller transform for calculation of Gaussian distributed random numbers.

The last method is to generate random numbers by rejection sampling. This method is presented in algorithm 4.

Algorithm 4: Rejection method for sampling arbitrary PDF.

Output: Random number distributed according to the n dimensional PDF $f_X(\mathbf{x})$ in the hypercube $[\mathbf{a},\mathbf{b}]$

```
repeat
    for i = 1 : n do
        x_i ← rand() · (b_i − a_i) + a_i
    r ← rand() · max_[a,b](f_X)
until r < f_X(x)
return x
```

Due to the architecture of GPU's, rejection sampling is inherently slow if performed on a warp or block level. This is because all threads has to wait for the last thread to finish rejection sampling. In a CPU, the number of iterations needed to find a value using rejection sampling is geometrically distributed. In a GPU the number is distributed as the maximum of n geometrically distributed variables, where n is the number of threads executing the code. The slowdown factor is illustrated in figure A.1. We see that rejection sampling should not be used unless the number of threads executing the code is small, or if the rejection probability is very small.

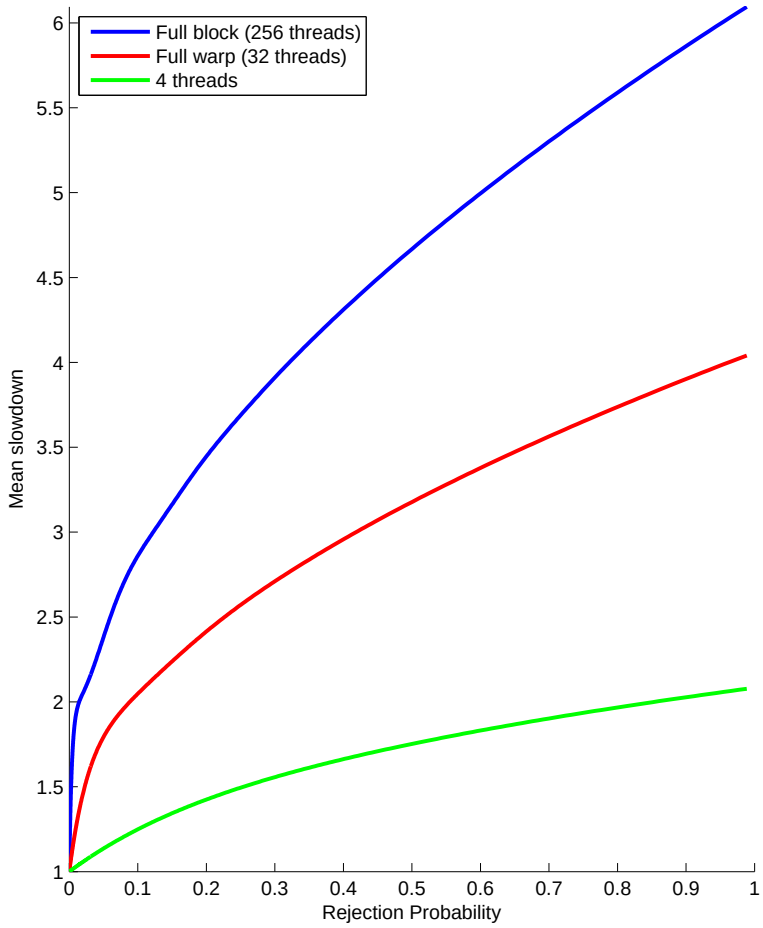


Figure A.1: Illustration of how much longer rejection sampling takes when performed by several threads as compared to a single thread.

B | Estimation of Memory Limitations

From the CUDA specifications we find that

$$\frac{\text{work cycles}}{\text{work cycles} + 700 \cdot N_{\text{memmory access}}} \leq$$
$$\leq \min\left(\frac{\text{threads / SM} \times \text{registers used / thread}}{\text{registers / SM}},\right.$$
$$\left.\frac{\text{warps / SM} \times \text{shared memory used / warp}}{\text{shared memory / SM}}\right)$$

for a CUDA device of compute capability 3.0 or 3.5, this reduces to

$$\frac{\text{work cycles}}{\text{work cycles} + 700 \cdot N_{\text{memmory access}}} \leq$$
$$\leq \min\left(\frac{\text{registers used / thread}}{1333},\right.$$
$$\left.\frac{\text{shared memory used / warp}}{21845}\right)$$

Initial tests showed that we need approximately 7000 processor cycles to simulate one photon. We also required approximately 50 global memory accesses, for things such as calculating the step size, finding the current material and getting the mass attenuation coefficients. Inserting these values gives us an order of magnitude estimate of how we need to limit the register and shared memory use

$$222 \geq \text{registers used / thread}$$
$$3500 \geq \text{shared memory used / warp}$$

TRITA-MAT-E 2014:05
ISRN-KTH/MAT/E—14/05-SE

www.kth.se