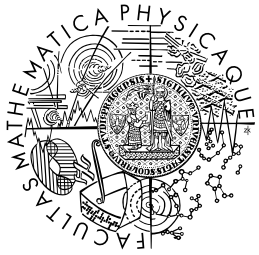


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

**MASTER THESIS**

Bc. Pavel Mikuš

# Simulating image formation in an electron microscope by electron tracing

Department of Software and Computer Science Education

| | |
|---|---|
| Supervisor of the master thesis: | Mgr. Tomáš Iser |
| Consultant of the master thesis: | Mgr. Lukáš Maršálek |
| Study programme: | Computer Science |
| Study branch: | Computer Graphics and Game Development |

Prague 2020

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date .....  .....

Author's signature

I would like to express my tremendous gratitude to all who made the realization of this thesis possible, especially to: Mgr. Lukáš Maršálek for the topic idea and passion for this subject, Mgr. Václav Alt for providing the fundamental physical blocks of the simulation, Mgr. Eva Havelková for her patience and critical eye when discussing mathematical obstacles, and my supervisor Mgr. Tomáš Iser for his careful and accurate guidance. I would also like to thank the Eyen SE company for providing me with resources and knowledge of the subject. Finally, special thanks belong to my parents, family, and people close to me, for the amazing help and support both in and outside my studies.

Title: Simulating image formation in an electron microscope by electron tracing

Author: Bc. Pavel Mikuš

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Tomáš Iser, Department of Software and Computer Science Education

Consultant: Mgr. Lukáš Maršálek, Eyen SE

Abstract: Cryogenic electron microscopy (cryo-EM) is an evolving field allowing molecular visualizations with picometer resolutions. Images are acquired by shooting electrons through molecular samples and detecting the scattered electrons. From such data, 3D shapes of the molecules can be inversely reconstructed. Currently, describing and simulating the cryo-EM image formation is based either on naive transmittance models or complicated wave-function formalisms.

In this thesis, we explore the possibility of simulating cryo-EM image formation via Monte Carlo electron tracing. We combine a delta-tracking algorithm with an electron elastic differential cross-section function and Rutherford formulae to derive two Monte Carlo estimators. The derived models are implemented in a high-performance C++/CUDA environment and compared with other common models. Our particle-based simulated images show considerable similarity to the wave-based state-of-the-art multislice model. We also evaluate our models on class averages of real measurements. Both of our proposed models have significantly higher normalized cross-correlation scores with the measured class averages when compared to the most commonly used transmittance model. The thesis proves the viability of a particle-based Monte Carlo simulation of electron microscope images and provides insight into the processes in cryo-EM. The efficient GPU implementation and high real data cross-correlation demonstrate the potential of our models to replace the transmittance model commonly used for molecular structure reconstructions.

Keywords: cryo-electron microscopy delta-tracking electron tracing elastic scattering

# Contents

# Acronyms

**CPU** central processing unit, central processor.

**cryo-EM** cryogenic electron microscopy.

**CTF** contrast transfer function.

**DCS** differential cross section.

**FFT** fast Fourier transform.

**GPU** graphics processing unit.

**PDF** probability density function.

**RTE** radiative transfer equation.

**SPA** single particle analysis.

# List of Symbols

$\sigma$ total cross section of elastic scattering.

$\bar{\mu}$ majorant scattering coefficient of elastic scattering.

$V$ Coulomb potential map.

$\hat{V}$ Fourier transform of the Coulomb potential map.

$V_{c,r}$ localized Coulomb potential map section with center $c$ and radius $r$.

$\hat{V}_{c,r}$ Fourier transform of the localized Coulomb potential map section with center $c$ and radius $r$.

$T$ transmission term - fraction of particles that remain unscattered.

# Preface

## Introduction to cryo-EM

Cryogenic electron microscopy (cryo-EM) is an evolving field that allows visualization of three-dimensional structures of living organisms at nearly atomic resolutions. Structural information of various molecules and proteins at all scales is necessary to understand how different components of the living organisms interact and what their purpose is. The use of electron microscopes and the ability to reconstruct large macromolecules at atomic levels revolutionized molecular biology [1].

### Electron microscope

The first electron microscope was constructed in the 1930s by Ernst Ruska. Compared to the standard light microscope, an electron microscope has superior resolution due to the very short wavelength of high energy electrons (about 2-5 pm) compared to the wavelength of the visible light (400-700 nm). The light microscope can achieve resolution of only 0.2 µm, while the state-of-the-art electron microscopes have resolution of 50 pm = 0.000 05 µm [2]. In theory, electron microscopes can achieve even smaller resolutions. However, there are limitations in the lens aberrations and small numerical aperture. Figure 1 shows a generic schema of an electron microscope. In the following paragraph, we provide a high level description of the microscope components adopted from [3]:

**Electron source** Electron source is usually an electron gun - a sharp piece of metal with high current going through it.

**Condenser lens** In an electron microscope, lenses are electromagnetic coils. The stronger the magnetic field, the shorter the focal length. Condenser lens focuses the emitted beam of electrons onto the sample.

**Condenser aperture** Limits the cone angle of the electrons that can interact with the sample. It works in conjunction with the condenser lens to eliminate high angle electrons.

**Sample** Sample consists of a tiny lattice covered by a water-based solution with the target molecule. The sample is frozen to cryogenic temperatures before insertion into the microscope.

**Objective lens** Objective lens is the strongest and most important lens of the whole system. It generates a highly magnified first intermediate image, which determines the resolution and quality of the final image.

**Objective aperture** Similarly to the condenser aperture, the objective aperture limits which electrons contribute to the final image.

**Selected area aperture** Selected area aperture is a second aperture after the sample. It selects which part of the sample is projected.

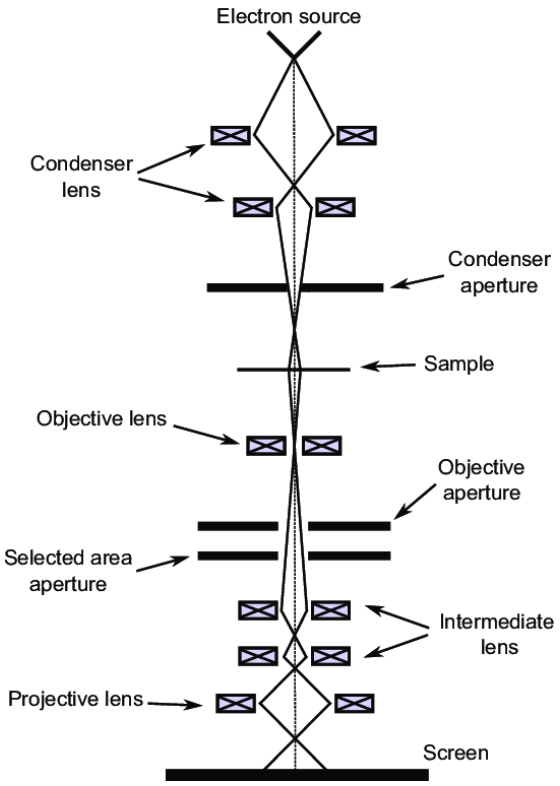



Figure 1: The schema of an electron microscope. Individual components are briefly described in the Electron microscope section. Image was taken from [4].

**Intermediate lens and projective lens** Final set of lenses, which further magnifies the first intermediate image and finally projects it on the screen of the detector.

## Data acquisition

Electron microscopes are utilized in a variety of scientific fields. In this thesis, we focus on single particle analysis (SPA) for the description of the electron microscope utilization. However, our simulation algorithm is based on general electron interactions and thus has a potential to be usable in other fields as well. In the following paragraphs, we describe the general idea of SPA and the role of the electron microscope.

During image acquisition, electrons are emitted from the electron gun in various directions. The first pair of lenses bends the electrons into a coherent beam and focuses them on the sample. Those electrons that were not succesfully focused are stopped by the condenser aperture. After interacting with the sample, the electron beam is transmitted through several lenses to achieve a high magnification factor, and finally measured on the detector. Electrons that scattered too much away are stopped by the objective aperture.

High energy electrons have very short wavelengths and thus can provide a very fine resolution. This requires that the microscope operates in a high vacuum, otherwise the electrons would interact with air molecules, adding a substantial amount of noise to the projected image. However, biological samples are not stable in a vacuum environment. Other difficulties arise in a form of a constant

thermodynamic movement of the sample molecules and radiation damage from the electrons. For these reasons, after applying the purified molecules to a metal grid, the sample is rapidly frozen in liquid ethane [5]. This protects the sample from the vacuum and prevents the movement.

During illumination by the electron beam, 2D projections of the sample are acquired. The acquired images are called micrographs and each micrograph contains hundreds of projections of the target molecule from various directions, as shown in Figure 2. However, the micrographs (example is shown in Figure 3) suffer from a low signal to noise ratio due to low suitable radiation dose. Further software processing is required to extract the desired information from the images.

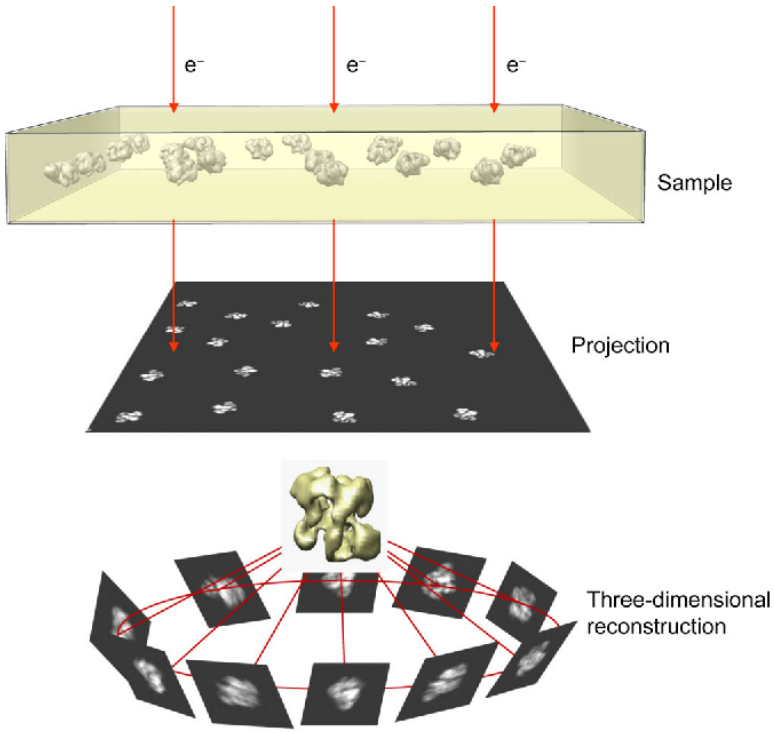



Figure 2: Image displays a sample with many instances of the target molecule. The sample is then projected on the screen of the microscope detector, forming a micrograph. The next step is then software processing of the micrographs and 3D reconstruction of the target molecule. Image taken from [6].

## Reconstruction

After a sufficient amount of data has been acquired, it needs to be computationally processed. First, the projections are picked from the micrographs using templates and other classification algorithms. Next, the new dataset of molecule projections is used in an iterative process with two steps taking turns. One step is projecting the reconstructed potential, computing the residuum, and updating the potential map. The second step is direction and shift estimation of the projections, as we do not know these properties from micrographs. For further details about the process see [8].

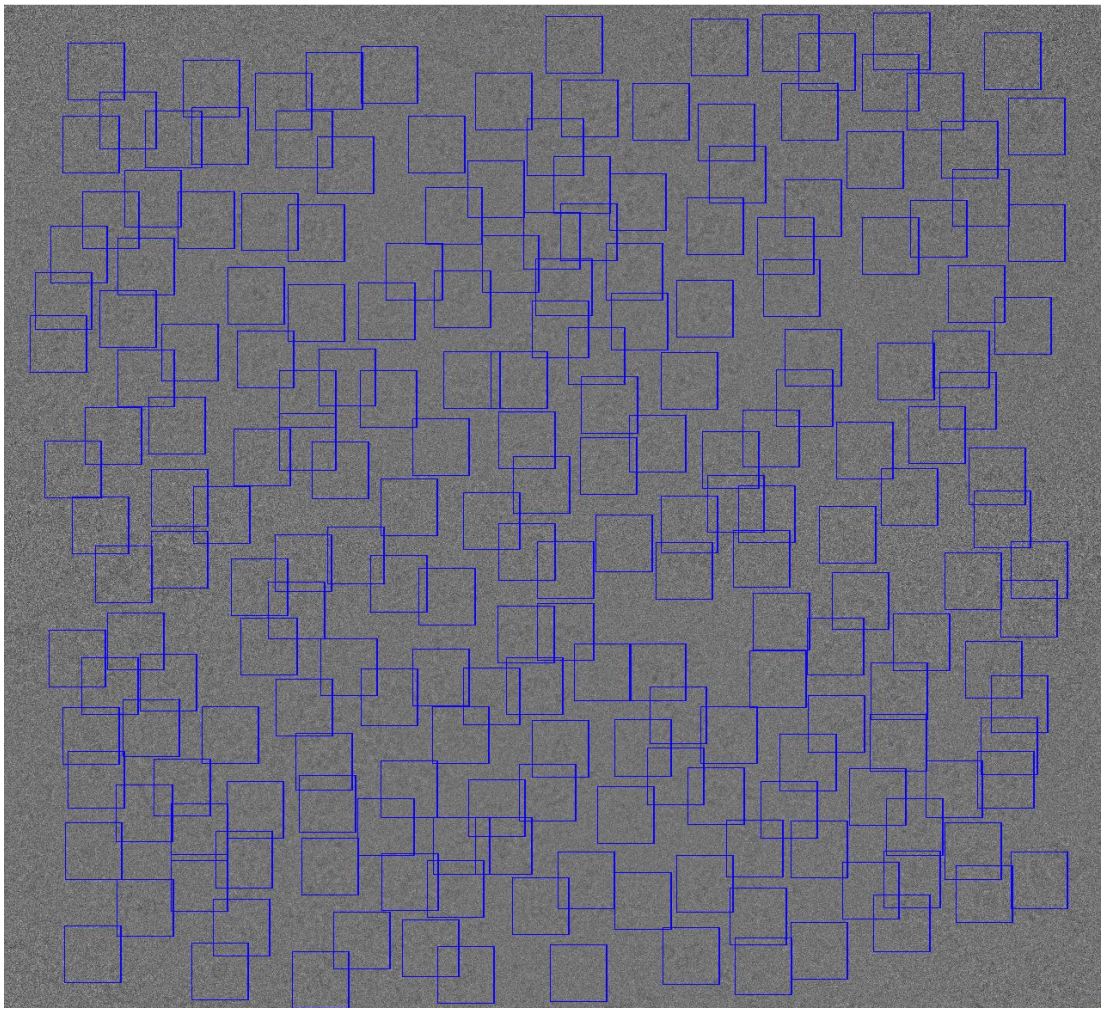


Figure 3: An example of a micrograph with picked projections of the target molecule. The noise level is very high because of the low electron dose. Higher doses would destroy the molecules, yielding invalid images (Dataset 10013 from EMPIAR [7])

## Modeling the projections

In order to simulate the projections of the sample in the electron microscope, two different models are commonly used: the transmittance model and the multislice model.

### Transmittance model

The transmittance model is the most common model used during the reconstruction process [9]. It takes the potential map as an input and generates its 2D projection. It does so by tracing a parallel beam of rays from a given direction, integrating the potential map along each ray. The process is illustrated in Figure 4 and its result approximates the transmittances through the sample along the direction. This model is also known as the X-ray transform.

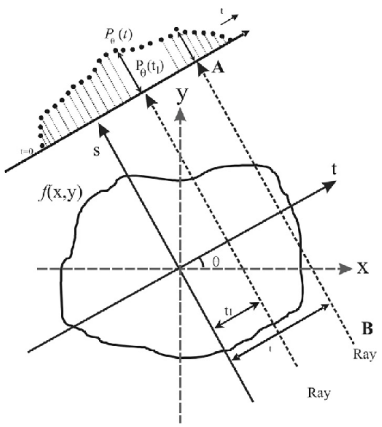



Figure 4: Illustration of the transmittance model projection in 2D. TThe projection P is generated by integrating volume f over a set of parallel rays. Image taken from [10].

### Multislice model

The multislice model is the state-of-the-art model for electron microscope simulation. It utilizes an electron wave formalism to simulate the behavior of an electron within a potential map. In the first step, the potential map is divided along the projection axis into several slices that are then sequentially processed. The thinner the slices, the more precise the method, as it is derived on a basis of infinitely thin slices. The next step is the electron wave propagation through the slices, where at each step the wave is attenuated by the processed slice. For details see [2]. The iterative process is depicted in Figure 5. To our knowledge, this model is not currently used for the structural reconstructions.

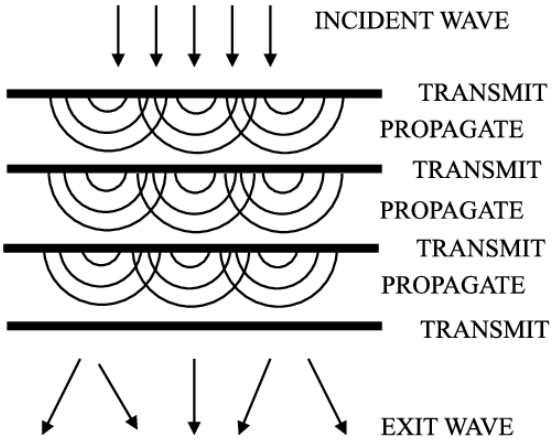



Figure 5: Illustration of the iterative process of the multislice model. The input volume is divided into a number of slices marked by the bold lines. A wave function is propagated slice by slice through the volume, performing forward and inverse Fourier transforms in each step. Image taken from [11].

## Thesis objectives

In this thesis, we aim to derive and implement a new model for simulating electron microscope projections. Our model utilizes an electron tracing simulation based on the particle-like behavior of the electrons. We focus on the simulation of electron-specimen interactions and a correct image formation, neglecting some of the processes occuring during image acquisition, such as radiation damage. The precise specification of the simulation setup is described in the first chapter.

To our best knowledge, the particle-based approach to the simulation has not yet been implemented. The purpose of this thesis is two-fold. First, it provides an intuitive insight into the processes within the electron microscope from a different perspective than the traditional wave function approach. Second, our implementation can optionally replace the transmittance model popular with cryo-EM reconstruction methods and provide a testing tool for generating artificial projections.

## Thesis outline

In the first chapter, we define our simplified simulation setup and characterize the electron interactions within the sample. We formulate the image formation problem as a radiative transfer equation (RTE), for which we propose scattering models. We provide a formulation of the Monte Carlo estimators that solve the presented RTE.

The second chapter is dedicated to several problems that arise during the implementation of the proposed models. We describe the used solutions and their impact on the simulation.

The third chapter describes the implementation of the accompanying library. The description follows the same order as the code execution in the provided test program. We explain how we reused the same ray-tracing algorithm core for different model implementations.

In the fourth chapter, we provide examples of the rendered projections. We visually compare both our models with the multislice model. We also evaluate the quality of our simulated projections with the measured class averages and compare the results to the transmittance model.

*"What you aim at determines what you see."*

—Jordan B. Peterson

# 1. Electron elastic scattering image formation

In this chapter, we describe the algorithm for image formation via electron tracing. As mentioned in Thesis objectives, our algorithm aims to encompass the same extent of functionality as Transmittance model and Multislice model, namely without the additional effects of optics, detector, radiation damage, and inelastically scattered electrons.

In the following section, we characterize electron interactions with the sample. Then, we define our simplified setup of the simulation. Finally, we describe the image formation as RTE for which we derive a Monte Carlo estimator with two viable scattering models.

## 1.1 Electron-sample interactions

Electrons interact with the Coulomb potential of the target molecules and the surrounding vitreous ice molecules. The image obtained from the electron microscope is a form of projection of this potential onto the detector. The commonly used energies for electron acceleration are from a few keV to a few MeV. It corresponds to one Volt unit multiplied by the electron's elementary charge. For lower acceleration energies, the emitted electrons become indistinguishable from the sample electrons, while for energies higher than a few MeV, relativistic effects and knock-on damage become significant [12]. The interaction event can be classified either as elastic or inelastic scattering. Figure 1.1 shows both types of interaction and other secondary effects. Simply said, the elastic scattering carries information while the inelastic scattering damages the sample. In the following subsections, we provide closer look into the events, for detailed info see [13].

### 1.1.1 Elastic scattering

Elastic interactions are characterized by identical initial and final quantum states of the molecule atoms. In other words, very little energy has been transferred from an electron to atoms of the sample during the interaction. Due to the large mass of the molecule compared to the mass of the electron, the average energy lost by the electron during elastic scattering is a tiny fraction of its energy and can be safely neglected. This is equivalent to assuming that the target has infinite mass [14]. The elastically scattered electrons transfer the structural information about the molecule onto the detector. The elastic scattering occurs more often in the cryo-EM setup than inelastic, and its effects are less damaging. For these reasons elastic scattering events are the main contributors to the signal in the final image [2].

### 1.1.2 Inelastic scattering

During inelastic scattering, the energy of the electron is deposited into the sample. This causes excitations and ionisations of the atoms, loss of energy of the

electron, and larger deflection angles compared to the elastic scattering [14]. Inelastic scattering contributes heavily to the radiation damage and noise in the final image [5]. Inelastically scattered electrons also contribute in the form of amplitude contrast. However, their contribution is low and blurry because the electrons lose their coherency [2]. For these reasons they are filtered out and do not reach the detector. The transferred energy and caused damage may also result in the secondary effects of interaction displayed in Figure 1.1.

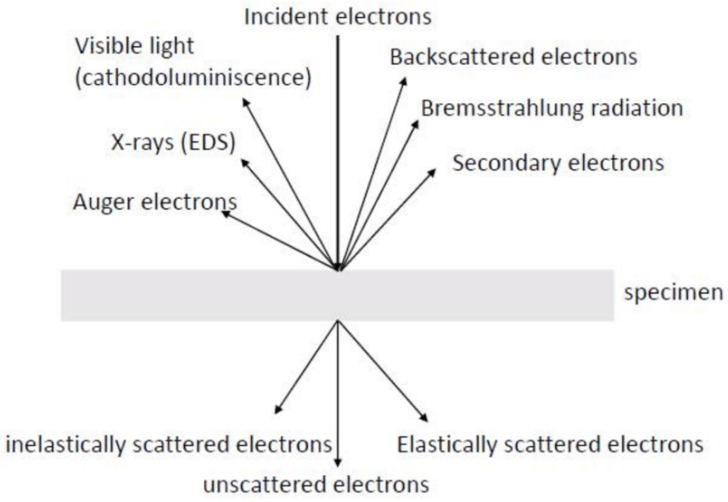



Figure 1.1: The image shows various radiation types that can appear after the scattering event. Many of these types are secondary effects of the inelastic scattering and the subsequent radiation damage. Image taken from [15].

## 1.2 Simplified image formation process

In this section, we describe the implemented formation process and simulated electron microscope components. Figure 1.2 illustrates the simplified setup of our simulation. We do not simulate the effects of lenses and apertures, and we do not explicitly model the noise caused by the vitreous ice. We mainly focus our attention on the elastic scattering of electrons, neglecting all other interactions.

### 1.2.1 Emitter

We simplify the emitter and the lenses that focus electrons on the sample by a virtual plane emitting beam of parallel electrons in a perfectly orthogonal direction. This corresponds to an ideal situation, in which the electrons in the beam are perfectly parallel before they hit the sample.

### 1.2.2 Input

The input to our simulator is a 3D representation of the Coulomb potential map of the molecule, because electrons interact elastically with the Coulomb potential of the sample. The Coulomb potential map can be either outcome of some cryo-EM reconstruction, artificially generated potential from PDB files or potential map

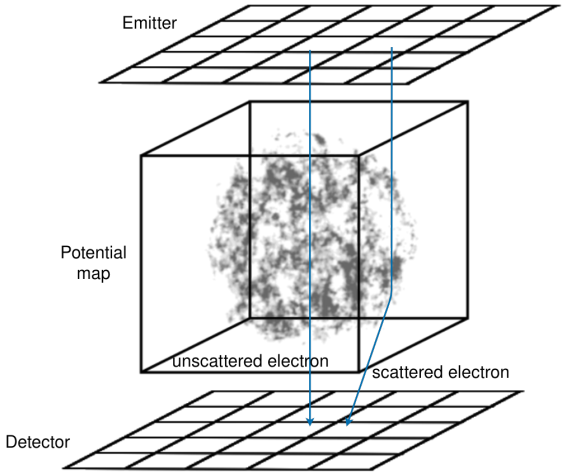



Figure 1.2: Schema of the simplified image formation process. Electrons are emitted perpendicular to the emitter plane. They elastically interact with the potential map of a single molecule. Finally, all emitted electrons are counted on the detector plane.

acquired by some other method. Similarly to other simulation models, we cannot rely on the physical plausibility of the values in the potential map, otherwise the applicability of this simulation would be severely limited. The impact of this input property is explored in Subsection 2.3.2.

### 1.2.3 Detector

In our simulation setup, it is not essential to model the lenses because our virtual detector does not need magnification. We simulate a perfect counter detector, which counts each electron impact in a pixel of the detector and the pixels can be as small as we choose. However, to compare the output images with the real measured projections, we model a part of the defocus effect through this component. In a real microscope, defocus is the distance of the sample from the focal point of the objective lens, as shown in Figure 1.3. The distance has a great impact on the acquired projections. As in a classic light microscope, defocus causes blurring of the image. But it also strongly influences the spread radius of the electrons on the detector plane, which in turn strongly changes the acquired image, see Figure 4.8 for examples. To approximate the electron spread effect, we move the detector away from the sample by the defocus value. When we refer to the defocus in the following text of the thesis, we mean by it the distance of the detector.

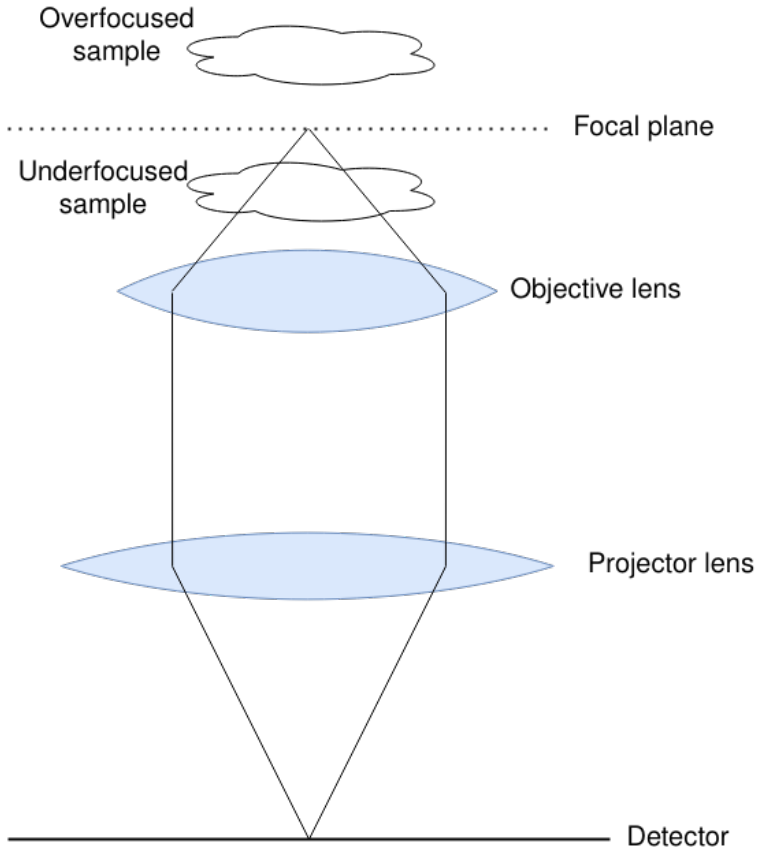



Figure 1.3: The figure shows an overfocused and underfocused sample in a real electron microscope. The distance of the sample from the focal plane determines the spread radius of the scattered electrons on the detector plane.

## 1.3 Electron elastic cross section

For this thesis, we limit the electron interactions to elastic scattering only, as explained in Subsection 1.2.2. The elastic scattering occurs when the electron is somewhat deflected from its current path by the Coulomb potential of the sample. Similarly to other particle-matter interactions, we can describe the phenomena with a differential cross section (DCS) equation [16]. In this section, we provide two means of estimating the differential and total cross-section for electron elastic scattering, but before we introduce the electron differential cross section (DCS), we describe the cross-section value.

### 1.3.1 Cross-section

A cross-section is a quantification of scattering effects of atoms on an electron. Differential cross section is a hypothetical area that captures the amount of flux of electron for the given incoming and outgoing directions. If we multiply the DCS by the flux of incoming electrons, we get the number of electrons per second that are crossing the DCS area. It ends up being the same as the number of electrons scattered into the given direction. In a way, the differential cross section can be interpreted as a size of an area that will scatter the particles into the given direction, and total cross-section $\sigma$ as an area that the electron must hit for the

scattering to occur.

### 1.3.2 Electron elastic differential cross section

The elastic scattering DCS is directly tied to the Coulomb potential of the atoms. The relation of the electron elastic DCS to the Coulomb potential $V$ is given by:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}(p' \leftarrow p) = (\frac{m}{2\pi})^2 |\hat{V}(q)|^2, \tag{1.1}$$

where $\Omega$ is a solid angle, $\hat{V}$ is a Fourier transform of the potential, $m$ is the electron mass, $p$ is the original momentum of the electron, $p'$ is the new momentum after scattering and $q = p' - p$ [16].

Elastic scattering is characterized by a negligible loss of energy of the electron from which follows that $|p'| = |p|$. We can make the observation that the relevant values of the Fourier transform of the potential are located on a sphere with radius $|p|$ and center in $-p$, see Figure 1.4. From now on, we might use the symbols $p$ and $p'$ both as directions and as momentums of the electron. It should be clear from the context whether the magnitude property of the term is important or not. Keep in mind that the electron does not lose any of its energy, thus the magnitude of the momentum remains constant during simulation.

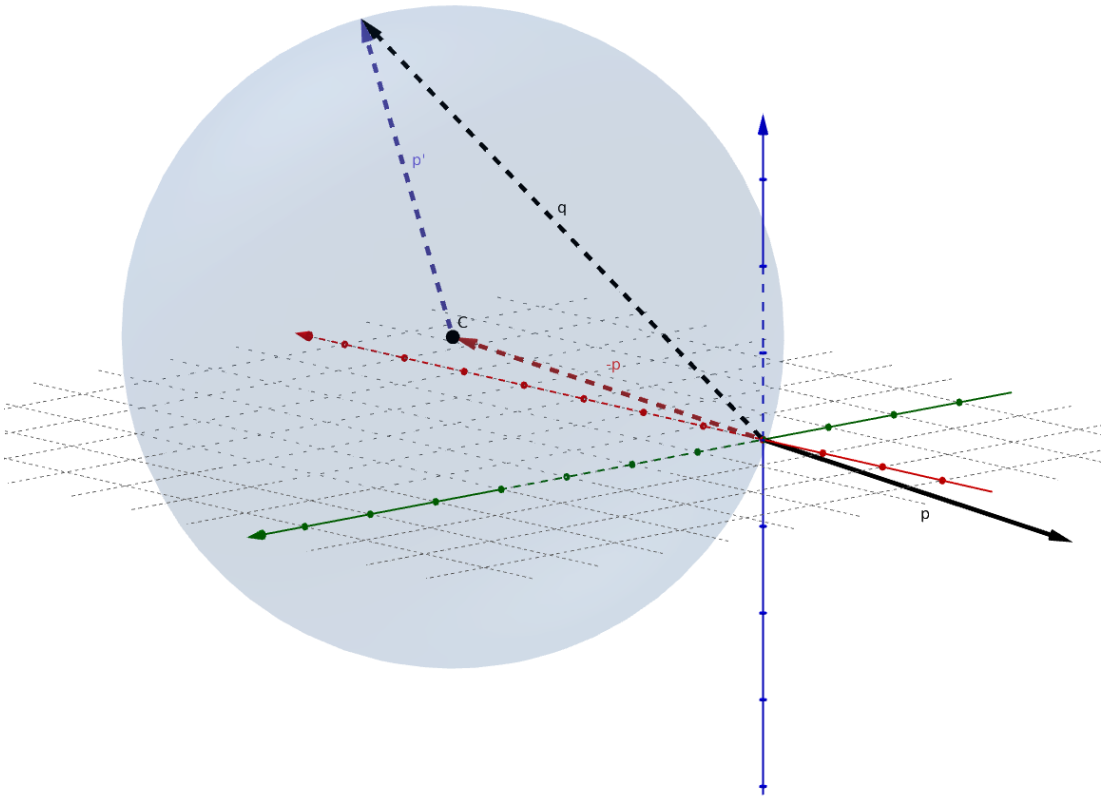
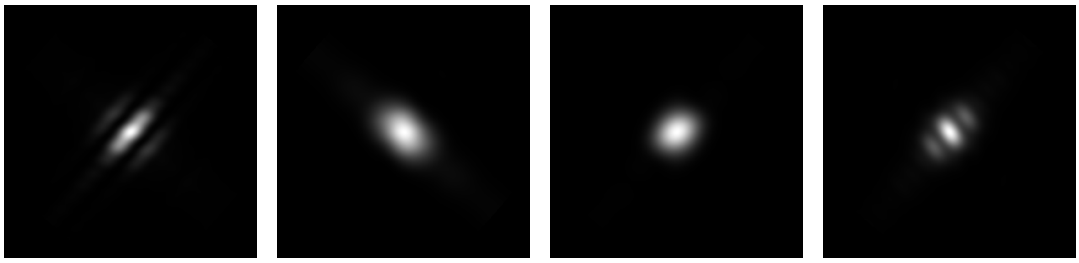



Figure 1.4: Visualization of the sphere appearing in Equation (1.1). Vector $p$ is the original momentum of the electron and vector $p'$ represents one of possible new momentums of the electron. The sphere then displays all possible values for expression $q = p' - p$. Image created via geogebra tool [17].



(a) (b) (c) (d)

Figure 1.5: Examples of the DCS scattering computed from the Equation (1.1) model. The images show DCS scattering directions on different positions within the potential. The original direction points into the middle of the image and the pixel in the middle of each side represent a scattering angle of 6 degrees. It can be viewed as a projection of the values on the cap of the sphere (see Figure 1.4), centered around the $p$ vector.

#### Total cross section and mean free path

By integrating Equation (1.1) over all scattering directions

$$\int_{\Omega} \frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}(p' \leftarrow p)\mathrm{d}\Omega, \tag{1.2}$$

we compute the total cross section $\sigma$ of the elastic scattering. The total cross section value $\sigma$ is also tied with the mean free path $\lambda$ via the following formula:

$$\lambda = \frac{1}{\rho_n \sigma}, \tag{1.3}$$

where $\rho_n$ is the number density of scattering centers, in our case the number of atoms [18]. This formula becomes useful later, as it fits nicely into the implemented delta-tracking algorithm.

#### Frame size of the Coulomb potential

In Equation (1.1), we have introduced the dependency of the cross-section on the Coulomb potential of the sample. However, it is undesirable to use the potential map of the whole sample. The first Born approximation assumes electron incoming from infinity, interacting with an isolated potential, and observing the scattering at infinite distance [16]. In other words, it assumes an isolated interaction. We reflect this idea by limiting the area that can affect the cross-section computation.

We tested different sizes of the surrounding area used to compute the DCS. In theory, the Born approximation should work the best for very low radii of the area, around 2 to 4 Bohrs [16]. From now on, we will denote the localized Coulomb potential with center $c$ and radius $r$ as $V_{c,r}$ and the corresponding Fourier transform as $\hat{V}_{c,r}$.

### 1.3.3 Screened Rutherford formulae

Screened Rutherford formuale are a mathematical model that describes DCS of an elastic electron scattering for a single atom of any chemical element [19]. The

model provides analytical forms for both the electron differential cross section and total cross section:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega} = Z^2 r_e^2 \left(\frac{1-\beta^2}{\beta^4}\right) \frac{1}{(1-cos\theta+2\eta)^2} \tag{1.4}$$

$$\sigma = Z^2 r_e^2 \left(\frac{1-\beta^2}{\beta^4}\right) \frac{1}{2\eta(\eta+1)} \tag{1.5}$$

where $\theta$ is the scattering angle between the incident and the outgoing electron, $Z$ is the atomic number of the atom, $r_e$ is the classical electron radius, $\beta$ is the ratio between relativistic speed of the electron and the speed of light and $\eta$ is the screening parameter, which reduces DCS at small scattering angles. The Rutherford formulae are dervied from the first Born approximation and thus are valid only for low atomic numbers and high energies [20]. Fortunately, this fits the situation within the considered cryo-EM. In our implementation, we use Rutherford formulae as a fast but less precise alernative to the Equation (1.1).

**Screening parameter**

A number of screening parameters has been derived for the Rutherford formulae, their overview can be found in [21]. In our software, we have used the Moliere screening parameter as described in [20]. The implemented Moliere screening parameter formula looks as follows:

$$\eta_m = \frac{Z^{\frac{2}{3}}}{4} \left(\frac{\alpha}{0.885}\right)^2 \frac{1-\beta^2}{\beta^2} \left[1.13 + 3.76 \left(\frac{\alpha Z}{\beta}\right)^2\right], \tag{1.6}$$

where $\alpha$ is the fine structure constant. We have compared the quality of the Rutherford approximation with the chosen Moliere sceening parameter by comparing it with a measured data from NIST database [22] for Sulfur. Figure 1.6 shows comparison of the differential cross sections values from NIST database and from Rutherford formula for low scattering angles. There is a slight difference in magnitude which influences the determination of the mean free path described in Subsection 2.3.2. However, the magnitude difference does not influence the scattering direction probabilites. The Rutherford approximation seems to fit the measured data very nicely.

## 1.4 Electron tracing

In Section 1.3, we have introduced equations to compute the scattering directions of the electron. The proposed problem to find the image formed by elastically scattered electrons can be formulated as particle transport in volume and thus can be solved via Monte Carlo (MC) electron tracing. Numerous MC algorithms are designed to solve the general problem of particle transport in volume [23].

### 1.4.1 Problem formulation

Within each detector pixel, we are interested in the number of electrons incoming from all directions. The problem can be formulated as an integral radiative

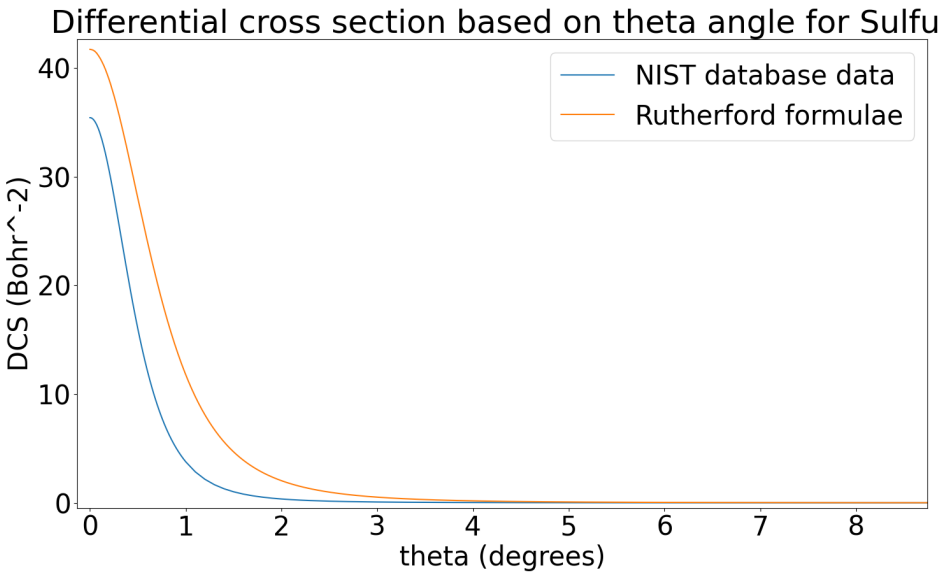



Figure 1.6: The figure shows a comparison of the Rutherford model computed differential cross sections and measured differential cross sections for Sulfur from NIST database [22]. The values fall off quickly to zero as we approach larger scattering angles. This corresponds with the DCS visualization images in Figure 1.5 generated via the model described by Equation (1.1).

transfer equation (RTE) described in [23]. Let $E(x, \omega)$ be a number of electrons incoming to a point $x$ from a solid angle $\omega$. We can modify the standard RTE with properties of our model. We intentionally leave out the absorption term, as our model does not support absorption. Moreover, we replace the scattering function with one of the electron scattering functions described later in Subsection 1.4.4:

$$E(x, \omega) = \int_0^\infty T(x, y) \left[ \mu(y) \int_\Omega f_E(\omega \leftarrow \bar{\omega}) E(y, \bar{\omega}) \mathrm{d}\bar{\omega} \right] \mathrm{d}y, \tag{1.7}$$

where $T$ is a transmission between $x$ and $y$, $f_E$ is an electron scattering function and $\mu(y)$ is the scattering coefficient at point $y$. The $T$ term corresponds to the probability that the particle will not encounter any collision on the path between points $x$ and $y$. We describe the used scattering functions in Subsection 1.4.4.

### Scattering coefficient

The scattering coefficient $\mu$ is defined by the relation $\mu = \sigma \rho_n$, where $\rho_n$ is the number density in $\mathrm{m}^{-3}$ describing how many matter particles there are in one volume unit. We also know the relation between the Coulomb potential and the total scattering section $\sigma$ from Equation (1.2). Combined together with the localized version described in Section 1.3.2, it yields:

$$\mu(c) = \rho_n(c, r) \int_\Omega (\frac{m}{2\pi})^2 |\hat{V}_{c,r}(q)|^2. \tag{1.8}$$

Assuming an infinitely small radius of the Coulomb potential $V_{c,r}$, the term $\hat{V}_{c,r}$ becomes a Fourier transform of a constant, which is a Dirac delta function:

$$\mu(c) = \rho_n(c) \int_\Omega (\frac{m}{2\pi})^2 |V(c)\delta_0(q)|^2,$$

where $V(c)$ is a Coulomb potenial at center $c$. Now, we can bring out the constant terms from the integral and use the fact that integration over Dirac delta function is 1 to get a relation of the scattering factor and the Coulomb potential $V$:

$$\mu(c) = \rho_n(c)(\frac{m}{2\pi})^2 V(c)^2. \tag{1.9}$$

From our input, we cannot deduce the number density $\rho_n$ because we do not know which elements generated the Coulomb potential at the given point $c$. Therefore, we neglect the effects of this value and set $\rho_n = 1$. This likely increases a scattering ratio of the less dense portions of the molecule and similarly reduces the scattering ratio of denser portions.

### 1.4.2 Null-collision algorithms

In Equation (1.7), the radiative transfer equation (RTE) has been established. It must be taken into account that the Coulomb potential map of the molecule is heterogeneous, which complicates the computation of the transmission. There are several approaches for the computation of the radiation transport in heterogeneous media [23]. One family of algorithms that solve the transmission estimation part of the RTE are null-collision algorithms.

The core step of the null-collision algorithms is homogenization of the heterogeneous volume by adding an artificial null volume, such that the scattering terms add up to the same constant scattering coefficient $\bar{\mu}$ for each corresponding point of the homogenized volume, see illustration in Figure 1.7. The null volume does not cause any scattering, it has Dirac delta scattering function $f_N(\omega \leftarrow \bar{\omega}) = \delta(\omega - \bar{\omega})$.

By introducing the null volume, we have artificially increased the scattering events of each particle because of an increased majorant $\bar{\mu}$ value. To compensate for that, we also proportionally decide whether the scattering happened or whether it was an artificial scattering, in which case we do not modify the particle properties. The effects cancel out and keep the physical plausibility of the RTE:

$$-\mu_n(x)E(x,\omega) + \mu_n(x)\int_{S^2} \delta(\omega - \bar{\omega})E(x,\bar{\omega})\mathrm{d}\bar{\omega} = 0,$$

where $\mu_n$ is the null portion of the scattering coefficient, brought in by the null volume [23]. The equation can be easily proved by solving the integral for the Dirac delta scattering function. Inserting this formula into the RTE Equation (1.7) yields:

$$E(x,\omega) = \int_0^\infty T_{\bar{\mu}}(x,y)\left[\mu(y)\int_\Omega f_E(\omega \leftarrow \bar{\omega})E(y,\bar{\omega})\mathrm{d}\bar{\omega} + \mu_n(y)E(y,\omega)\right]\mathrm{d}y, \tag{1.10}$$

where $\mu_n(y) = \bar{\mu} - \mu(y)$ represents the ratio of null collisions at the point $y$.

### 1.4.3 Transmission estimator

The transmission Monte Carlo estimator for the null-collision RTE given in Equation (1.10) has the following form:

$$\langle E(x,\omega)\rangle = \frac{T_{\bar{\mu}}(x,y)}{pdf(x,y)}\left[\mu(y)\int_\Omega f_E(\omega \leftarrow \bar{\omega})E(y,\bar{\omega})\mathrm{d}\bar{\omega} + \mu_n(y)E(y,\omega)\right], \tag{1.11}$$

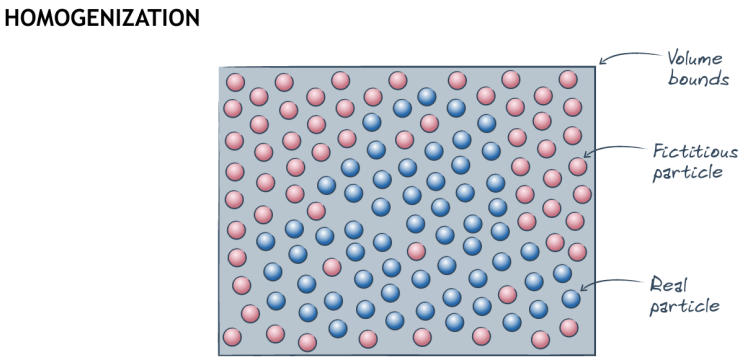



Figure 1.7: The image illustrates homogenization of the heterogeneous volume (blue). At each position within the considered volume bounds, the input volume and the null volume add up to the same majorant $\bar{\mu}$. Image taken from [23].

where $pdf(x, y)$ is the probability of picking the point $y$, or alternatively, picking a distance from $x$ to $y$. We further simplify the estimator evaluation via delta-tracking in the following subsection.

### Delta-tracking

Delta-tracking algorithm makes samples per the homogenized transmission term $T_{\bar{\mu}}$. For elastically scattered electron, the transmission term $T(t)$ corresponds to the probability that the electron will not scatter within distance $t$, i.e.,

$$T(t) = P(X > t) = e^{-\int_0^t \mu(s)ds}.$$

From that, we can derive the probability function that the electron scatters within distance $t$:

$$F(t) = P(X \leq t) = 1 - T(t) = 1 - e^{-\int_0^t \mu(s)ds}.$$

The function $F(t)$ corresponds to the cumulative distribution function for distance sampling. Before we can make it into a generator for path samples, we take advantage of the homogenized volume which has constant $\bar{\mu}$ value everywhere. This simplifies the terms:

$$T_{\bar{\mu}}(t) = e^{-\int_0^t \bar{\mu}ds} = e^{-t\bar{\mu}} \tag{1.12}$$

and so:

$$F_{\bar{\mu}}(t) = 1 - e^{-\int_0^t \bar{\mu}ds} = 1 - e^{-t\bar{\mu}}.$$

By inverting this formula, we get the recipe for generating samples with the correct distribution within the homogenized volume:

$$t = F_{\bar{\mu}}^{-1}(\xi) = -\frac{1}{\bar{\mu}} ln(\xi),$$

where $\xi$ is a uniformly distributed random number between 0 and 1. We also need the probability density function (PDF) of the generated samples, which is:

$$pdf(t) = \frac{\mathrm{d}}{\mathrm{d}t} 1 - e^{-t\bar{\mu}} = \bar{\mu} e^{-t\bar{\mu}}. \tag{1.13}$$

Let $t$ be the distance between points $x$ and $y$. By plugging the derived formulae from Equation (1.12) and Equation (1.13) into the estimator described by Equation (1.11), we can further simplify it:

$$\begin{aligned}\langle E(x,\omega)\rangle = \frac{e^{-t\bar{\mu}}}{\bar{\mu}e^{-t\bar{\mu}}}\left[\mu(y)\int_\Omega f_E(\omega\leftarrow\bar{\omega})E(y,\bar{\omega})\mathrm{d}\bar{\omega}+\mu_n(y)E(y,\omega)\right] = \\ \frac{\mu(y)}{\bar{\mu}}\int_\Omega f_E(\omega\leftarrow\bar{\omega})E(y,\bar{\omega})\mathrm{d}\bar{\omega}+\frac{\mu_n(y)}{\bar{\mu}}E(y,\omega),\end{aligned} \tag{1.14}$$

where $\mu(y)+\mu_n(y)=\bar{\mu}$. Since we want to trace only a single electron and not both the scattering and null-collision interactions at the same time, we estimate also the inner terms by randomly choosing which branch to evaluate:

$$\langle E(x,\omega)\rangle = \begin{cases}\int_\Omega f_E(\omega\leftarrow\bar{\omega})E(y,\bar{\omega})\mathrm{d}\bar{\omega} & \text{if } \varepsilon_1 \le \frac{\mu(y)}{\bar{\mu}} \\ E(y,\omega) & \text{if } \varepsilon_1 > \frac{\mu(y)}{\bar{\mu}}.\end{cases} \tag{1.15}$$

The $\varepsilon_1$ term is a random sample in range $[0,1]$ with a uniform distribution. It corresponds to a random decision whether to follow the null-scattering or real scattering function.

### 1.4.4 Scattering estimator

In the previous section we have derived a delta-tracking transmission estimator (see Equation (1.15)) for our RTE Equation (1.7). In this section, we provide two viable electron scattering functions and develop MC estimator for the scattering term:

$$\int_\Omega f_E(\omega\leftarrow\bar{\omega})E(y,\bar{\omega})\mathrm{d}\bar{\omega}.$$

The scattering functions are the Rutherford scattering function and the DCS scattering function.

#### Rutherford scattering function

We have described an analytical formulation of the electron DCS for single atom via Rutherford formulae in Subsection 1.3.3. We can turn this differential equation into a probability density function of $\cos\theta$ via normalization:

$$pdf(\cos\theta) = \frac{Z^2 r_e^2\left(\frac{1-\beta^2}{\beta^4}\right)\frac{1}{(1-\cos\theta+2\eta)^2}}{\sigma} = \frac{2\eta(\eta+1)}{(1-\cos\theta+2\eta)^2}. \tag{1.16}$$

We integrate Equation (1.16) over the interval $[-1,\cos\theta]$ in order to derive cumulative distribution function:

$$\int_{-1}^{\cos\theta}\frac{2\eta(\eta+1)}{(1-\cos\theta+2\eta)^2}\mathrm{d}\cos\theta = \frac{\eta\cos\theta+\eta}{2\eta-\cos\theta+1}. \tag{1.17}$$

Finally, by inverting the cumulative distribution function in Equation (1.17), we get a formula for sampling $\cos\theta$ proportionally to the DCS:

$$\cos\theta = \frac{2\eta\varepsilon-\eta+\varepsilon}{\eta+\varepsilon}, \tag{1.18}$$

where $\varepsilon$ is a uniformly distributed random sample between 0 and 1. Because we sample $\bar{\omega}$ proportionally to $f_E$, then $\frac{f_E(\omega \leftarrow \bar{\omega})}{pdf(\bar{\omega})} = 1$, and we can omit the term in the MC estimator. The estimation formula from Equation (1.15) simplifes to:

$$\langle E(x,\omega)\rangle = \begin{cases} E(y,\bar{\omega}) & \text{if } \varepsilon_1 \leq \frac{\mu(y)}{\bar{\mu}} \\ E(y,\omega) & \text{if } \varepsilon_1 > \frac{\mu(y)}{\bar{\mu}}. \end{cases} \tag{1.19}$$

### DCS scattering function

In this section, we derive the second viable scattering function from the DCS Equation (1.1). By normalizing Equation (1.1) with total cross section, we can derive the scattering function:

$$pdf(p') = f_E(p' \leftarrow p) = \frac{\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}(p' \leftarrow p)}{\sigma} = \frac{(\frac{m}{2\pi})^2 |\hat{V}(q)|^2}{\sigma}. \tag{1.20}$$

Note that we use the localized version of the Coulomb potential, to emphasize that the function depends on the position within the volume. If we wanted to sample this function directly, we would have to integrate its PDF to get the cumulative distribution function, and then invert it, as we did for the distance sampling. However, this approach is not viable for this scattering function.

One solution would be to somehow precompute the values and then sample the acquired discrete distribution. But there is a problem with aliasing, which the discrete distribution will very likely cause unless we do a very fine discretization, which would require a substantial amount of memory. Another problem is the function dependency on the position within the Coulomb potential map. We would have to have the precomputed samples for every position of that volume. However, we can also use the importance sampling technique, which is a common approach in computer graphics.

### Importance sampling of DCS

The idea of importance sampling is to draw samples from a distribution that is proportional to our scattering function $f_E$. This allows us to use scattering function that we cannot explicitly sample. The better the correspondence between the sampling function and the scattering function, the less variance we get in our estimations. With importance sampling, the estimator is:

$$\langle E(x,\omega)\rangle = \begin{cases} \frac{f_E(p' \leftarrow p)}{pdf_S(\bar{\omega})} E(y,\bar{\omega}) & \text{if } \varepsilon_1 \leq \frac{\mu(y)}{\bar{\mu}} \\ E(y,\omega) & \text{if } \varepsilon_1 > \frac{\mu(y)}{\bar{\mu}}. \end{cases} \tag{1.21}$$

where $f_E$ is the DCS scattering function and $f_S$ is the sampling function with the probability density function $pdf_S$. In our implementation, we used the Rutherford fromula as $f_S$ to importance sample the DCS scattering function.

# 2. Optimizations

In the previous chapter, we described the simplified microscope model and the corresponding Monte Carlo estimator for the elastic electron scattering simulation. However, we intentionally did not discuss some of the obstacles that arise during the theory to program transformation. In this chapter, we explore more in-depth implementation difficulties of the proposed models. We focus on the Fourier transform of the Coulomb potential. Next, we analyze the choice of the majorant and undesired properties of the input data. After that, we explore the mean free path and its impact on the rendering, and lastly, we inspect the implementation complications of the Rutherford scattering model.

## 2.1 Units

We use atomic units throughout our implementation, mainly the Bohr units $[a_0]$ for length. While Angstrom [Å] is usually used within the cryo-EM reconstructions and research, atomic units simplify the physics and equations that describe the behavior of an electron within the potential field and thus are more convenient for us, as explained in [16]. The relation between Bohr and Angstrom is:

$$1\ a_0\ \approx\ 0.529\,177\,210\,903\ \text{Å}.$$

## 2.2 Fourier transform of the Coulomb potential

The elastic DCS depends on the Fourier transform of the localized Coulomb potential $V_{c,r}$ as described in Section 1.3.2. To simplify the equations and implementation, we consider the location to be cube-shaped. The formal expansion of the Fourier transform of the potential in center $c$ and with radius $r$ is:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}(p' \leftarrow p) = (\frac{1}{2\pi})^2 |\hat{V}_{c,r}(q)|^2 = \\ (\frac{1}{2\pi})^2 \int_{c_x-r}^{c_x+r} \int_{c_z-r}^{c_z+r} \int_{c_z-r}^{c_z+r} V(x,y,z) e^{-2\pi i q((x-c_x)+(y-c_y)+(z-c_z))} \mathrm{d}x\mathrm{d}y\mathrm{d}z. \quad (2.1)$$

Note that we have replaced the electron mass with 1 thanks to the application of atomic units. After discretization of the integrals, we have the following formula for a discrere Fourier transform:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}(p' \leftarrow p) = \\ (\frac{1}{2\pi})^2 \sum_{\substack{x=c_x-r \\ x+=s}}^{c_x+r} \sum_{\substack{y=c_y-r \\ y+=s}}^{c_y+r} \sum_{\substack{z=c_z-r \\ z+=s}}^{c_z+r} V(x,y,z) e^{-2\pi i q\left(\frac{x-c_x}{2r}+\frac{y-c_y}{2r}+\frac{z-c_z}{2r}\right)}, \quad (2.2)$$

where $s$ is the sampling step size, which can be computed as $\frac{2r}{\#samples}$.

There exist a number of implementations of the fast Fourier transform (FFT) algorithm, even for the 3D case presented here. However, we are only interested in frequencies that lie on a sphere, as depcited in Figure 1.4. That is because for

the given momentums $p$ and $p'$ we want the value for the frequency $q = p' - p$, where the magnitude of $p'$ and $p$ is the same. We would throw away most of the frequencies acquired from full 3D FFT.

Also, to avoid Moiré patterns, the number of samples must be sufficiently high. For example, for electron with an energy of 300 keV, the magnitude of its momentum is 148.49 ${a_0}^{-1}$. The forward direction always corresponds with the zero frequency value. However, for larger scattering angles the corresponding frequencies increase rapidly. For the largest scattering angle (essentially reversing the direction of the electron) we need to compute the value of frequency of up to $2 \times 148.49\ {a_0}^{-1} = 296.98\ {a_0}^{-1}$ for this electron. Note that the frequency still has units.

To correctly compute a value of a signal with such frequency, the sampling step must be half the wavelength of the signal, which is $\frac{1}{2 \cdot 296.98} \approx 0.0017\ a_0$ in this case. Because the radius values that we intend to use range from 1 $a_0$ to 16 $a_0$, this leads to an enormous number of samples. At the same time, failing to respect this limit leads to undesired Moiré patterns in the undersampled frequencies and affects the weights of the importance sampling.

Fortunately, from Figure 1.6 and from the experiments in Figure 1.5, we can observe that vast majority of electrons do not scatter to angles wider than approximately 4 degrees. Based on this observation, we have done the following steps:

- First, we have implemented the Fourier transform as described by the Equation (2.2) above. Compared to the standard discrete FFT algorithm, this allows us to correctly represent any frequency up to the precision of the numeric type. We also do not have to compute the values for frequencies outside those that are of interest. The algorithm is simpler than FFT and can be executed within a single CUDA thread. Lastly, with importance sampling, it is sufficient to compute only a single frequency, which is faster.

- Second, we parse the number of samples per axis and the locality radius of the Fourier transform from the input parameters of the simulator. We then compute the maximum scattering angle that we can correctly represent with this setup and restrict the importance sampling accordingly.

## 2.3 Majorant choice

Before we discuss how to choose the majorant value, we first need to acquire a volume that is used for distance sampling, as described in Subsection 1.4.3. As we have declared in Subsection 1.2.2, the input to our simulator is a map representing a Coulomb potential. In Section 1.4.1, we have derived the relation between scattering coefficient and the Coulomb potential. To get a volume for rendering, we take the input potential map and transform it by the derived relation, acquiring a map of scattering coefficients.

Setting the maximum value of the generated scattering coefficients volume as majorant is not ideal. We can expect outliers in both the reconstructed data, and also artificially generated data because the equations for the atom Coulomb potential diverge to infinity at zero distance from the atom. Thus, there would be

a very high chance that the maximum value of the volume is an outlier, resulting in low scattering tendency everywhere else in the volume and low contrast in the final image.

### 2.3.1 Element estimation

To avoid this problem, we take advantage of a chemical composition of the molecule known from e.g. spectroscopy. We assume that the largest scattering coefficient belongs to the atoms with the highest atomic number - an assumption that seems to be correct from the behavior of the Rutherford formula. We compute the percentual representation of this element in the studied molecule and choose the lowest scattering coefficient from the volume, which is still within the percentile of the largest atomic number. The percentile is set manually through the parameters and recommended values are above 0.99. For example, the $\beta$-galactosidase molecule used in the experiments has a chemical distribution shown in Table 2.1. The highest atomic number is Sulfur, so we use as the majorant a value with $1 - \frac{0.474834}{100} \approx 0.995$ percentile. This is a heuristic approach and it would not work for molecules where the highest atomic number is extensively present, so we advocate to use a reasonably high percentile.

| Element | Atomic number | Count | Percentage |
|---|---|---|---|
| Carbon- C | 6 | 20816 | 61.7759 |
| Oxygen- O | 8 | 6896 | 20.4653 |
| Nitrogen- N | 7 | 5808 | 17.2365 |
| Sulfur- S | 16 | 160 | 0.474834 |
| Sodium- Na | 11 | 8 | 0.0237417 |
| Magnesium- Mg | 12 | 8 | 0.0237417 |

Table 2.1: Element distribution of the $\beta$-galactosidase molecule.

### 2.3.2 Mean free path

Another problem tied with the majorant scattering coefficient is choosing the sampling mean free path. In classic delta-tracking, the majorant scattering coefficient also determines the sampling mean free path via the following formula $\lambda = \frac{1}{\bar{\mu}}$. However, as declared in Subsection 1.2.2, we cannot rely on the scale of the input values. Even though we have chosen a majorant scattering coefficient, plugging it into the path estimation described in Subsection 1.4.3 would return incorrect values.

To solve this problem, we re-use the element estimation. From the approach in Subsection 2.3.1, we have established which element represents the majorant scattering coefficient. From the Rutherford formula, we can compute the realistic total cross-section for the estimated majorant element. We can then use this realistic total cross-section to compute a realistic mean free path from the following formula from Section 1.3.2:

$$\lambda = \frac{1}{\rho_n \sigma}. \tag{2.3}$$

From the knowledge of the estimated majorant element, we can compute the density of the scattering centers $\rho_n$:

$$\rho_n = \frac{N_A \rho}{A},$$

where $N_A$ is the Avogadro constant, $\rho$ is the weight density of the element and $A$ is the atomic weight [18].

By associating the mean free path with a majorant scattering coefficient, instead of computing it, we essentially normalized the volume. It does not matter what the scale of the values in the input molecule is, because the sampling mean free path will always be determined by the estimated majorant element. However, it also overloads the meaning of the majorant value - now it does not influence only the noise of the final image, but also the contrast of the image, because of its association with a set mean free path.

#### Unscattered dose propagation

The typical elastic mean free paths of high energy electrons within cryo-EM are several times longer than the size of the molecule. This means that the vast majority of electrons do not interact with the specimen and contribute only to the background of the image. To save computational time, we do not simulate these electrons explicitly. Instead, we compute the unscattered dose from the cumulative distribution function for distances described in Section 1.4.3. Let $t$ be the distance from the emitter to the detector, then the probability that the electron will not scatter is:

$$P(X > t) = e^{\frac{t}{\lambda}}.$$

The detector is initialized with the unscattered dose before the simulation. To keep the estimator unbiased, we have to compensate for this by sampling the first scattering distance only within the distance $t$. For distance sampling, we use the following formula:

$$s = -\lambda \ln(1 - \varepsilon),$$

where $s$ is the sampled distance and $\varepsilon$ is a random number with uniform distribution. To restrict the sampling to distances between 0 and $t$, we restrict the $\varepsilon$ to the range $[0, 1 - e^{\frac{-t}{\lambda}}]$.

## 2.4 Rutherford scattering model

In Subsection 1.4.4, we have described the Rutherford scattering function and how to utilize it in the estimator. However, the Rutherford formulae from Subsection 1.3.3 are derived for a single atom of a known type, not for a potential map or scattering coefficients map. We tried to use the approach from Subsection 2.3.1 again and during scattering within the volume, we estimated the atomic number of the local atom based on the local scattering coefficient and the majorant scattering coefficient, for which we know the element it represents. Unfortunately, this did not work very well because the estimated scattering angles were too wide. In the experiments in Chapter 4, we show that a good approximation model is to simply use one Rutherford formula for all scattering events.

# 3. Implementation

In this chapter, we describe the implementation of the Monte Carlo simulation from Chapter 1. We utilized the C++ language and modern graphics processing units (GPUs) in order to make the simulation reasonably fast. In the description, we use the term *device memory* to refer to the memory of the GPU, and the term *host memory* to refer to the traditional central processing unit, central processor (CPU) accessible memory. In the first part of this chapter, we discuss a considered options of library or framework that could be used to simplify the GPU utilization. We also mention several other external libraries that are integrated into the developed software. We finish the chapter with a description of the functional units from which the simulator is composed.

## 3.1 Libraries

In this section we describe the chosen library *BoltView* [24] that helps us with the GPUs utilization. We also mention other GPUs libraries that we considered. In the next part, we briefly describe other libraries that we have utilized in the software.

### 3.1.1 GPU library

We have considered the following libraries/frameworks when deciding how to simplify the integration of GPU: *OptiX NVIDIA* library [25], *GVDB Voxels* [26] and *BoltView*. We have decided to use *BoltView*, which offers the most flexibility of those options. Moreover, we have a considerable experience with the framework. It also allowed us to develop the simulation in such a way that the same code can be run both on *NVIDIA* GPUs and on CPUs only, with minor code changes. In the following text, we offer a brief description of the *BoltView* and the reasons why we discarded other considered libraries.

#### *BoltView*

We have decided to use the *BoltView* library as a core of our simulator. *BoltView* library is a C++ header-only library that offers unified interface for both GPU and CPU computations. It provides data structures and implementations of meta-algorithms for structured data processing. Most of the library functionality is based around the concept of the *Image* and *View*. The instance of the *Image* class represents the actual data, stored in an n-dimensional array. Instances of *View* class are then only proxies that can access and manipulate the data of *Images* but do not own the data. The majority of the BoltView algorithms operate over the *Views*, not the *Images* themselves. The memory management is up to the developer. The usual pattern is to allocate the memory with the *Image* class and use the acquired *View* instances for actual computations.

#### *OptiX*

*OptiX* is a ray-tracing engine from *NVIDIA*. At first, it seemed to be a good choice for a particle tracing algorithm. However, after developing a demo version, we concluded that this framework restricts us more than it helps us. Perhaps the greatest issue with OptiX is its limited interface when it comes to the device functions that control the ray. The code that is supposed to run on the device has to be written in C language and compiled separately into the ptx format. During the linking time, the device code is connected with the OptiX via C bindings.

It became clear from the design that *OptiX* is intended for triangular mesh rendering, not for volumetric simulations. We did not need its optimized data structures for fast ray-triangle intersection computation, which is arguably one of the core features of *OptiX*. Overall, it restricted us in many ways while not contributing much to simplify the implementation of our algorithm.

#### *GVDB Voxels*

We have not tested the *GVDB Voxels* library directly. Compared to the *OptiX* this library is focused on fast volumetric rendering. However, at the time of starting the development, this library was only several months old and it seemed that there is no clear way to implement custom device code. Instead, the rendering was based on setting up the parameters and letting the framework use its general pre-implemented functions. We could not also figure out if it is possible to change the direction of the ray, which is a crucial part of our simulation.

### 3.1.2 Other external libraries

In this section, we describe libraries that are integrated within the software. Most of them deal with the parsing of input and output formats.

#### JsonParams

*JsonParams* [27] is a side project that has been developed to simplify the input parsing. It is a simple library for loading parameters from a file in JSON format. The parameters are specified in program as structures with static members. For each parameter, the developer specifies a name, type, and optionally default value. The library is easily extendable with new types if the proper parsing function is specified. We use this library to load all parameters of the simulation, including a path to the potential map.

#### MRC and STAR File formats

Source codes for loading and writing the MRC and the STAR File format has been provided by the *Eyen SE* company. The MRC is a binary format for n-dimensional numeric data and is commonly used for biological data, e.g. potential maps, electron density maps, measured projections, etc. The STAR File format is a metadata format accompanying projections data. It contains information about each projection from the MRC file, e.g. view angles, estimated defocus, magnification, and electrons energy. We have used STAR File data to setup the

simulation and compare our images with the measured images from the same view angle.

#### Potential reconstruction

For tests and experiments, we have included a library for generating Coulomb potential maps from PDB files. The source codes can be found in the folder `/src/density_reconstruction`. The source codes for the generation of a Coulomb potential for a given atom and distance were provided by Alt [16]. The algorithm accepts a PDB file and voxel size. Then it constructs a 3D array of floats representing the potential map. Next, it iterates over all atoms and distributes their potential into the map, which is then returned as a result.

#### gemmi

*Gemmi* [28] is a C++ and Python library developed for use in macromolecular crystallography. We use it mainly for parsing and working with PDB file format. The PDB format contains a list of atoms with their positions and type. The PDB files for many molecules are publicly accessible from an online database.

## 3.2 Simulator implementation

The program is written in C++ with an extensive use of template programming and the *BoltView* library. It is written in a duck typing style, a programming style extensively used also by the *BoltView* library. The style is an application of the so-called duck test - If it walks like a duck and it quacks like a duck, it is a duck. In programming, it means that the type of the objects is not that important. Instead, the members and methods of the object determine its applicability within the rest of the program.

Everything needed for the electron simulation is enclosed within a namespace *ems*. The core of the software is a header-only *ems* library. An example executable implementation that demonstrates the usage of the *ems* library is provided. Several core classes facilitate the simulation, all of which are described in the following section. The program is described according to the data flow of the `test_exe` program, from parsing the input parameters to the image formation step. Please refer to the file `/src/test_main.cu` which demonstrates these steps and is being utilized in the description of the simulator.

### 3.2.1 Data preparation

In this section, we describe several classes that are utilized for parsing parameters and loading the input data. We also describe the implementation of the simplified unit system that provides some level of compile-time type safety for physical units. It improves the readability of the code as it makes clear where which unit should be used.

### StrongType system

Here we introduce the *StrongType* class, defined in the file `/src/ems/strong_type.h`. The class serves as a thin wrapper around a numeric value with no implicit conversions. This allows us to define our units and have a limited compile-time checking of their correctness. It makes the implementation of physical equations easier and improves the readability of the code. It is not as powerful as template-based unit systems from e.g. *Boost* library, but provides basic unit type safety and is easy to use even within the CUDA environment.

Units and constants are defined in the `/src/ems/defines.h`. All units and their multidimensional versions realized with a *BoltView* class *bolt::VectorX* are defined using the *StrongType* class. Examples are *Bohr*, *Bohr3*, *Angstrom*, *Radians*, *Pixel2* etc. A *Bohr3* instance stands for a 3-dimensional vector of Bohr units. We do not use powers with the units, so many computations involving these are done with unit type removed and reassigned at the end.

### Setup

Upon the start of the test application, it first loads the parameters from the JSON file. Then it loads the potential, scattering, and total cross sections volumes. It is expected that all three volumes are cube-shaped and have the same full size and voxel size. The potential and cross-sections volume are important only for the DCS scattering function defined in Section 1.4.4. The scattering volume is however necessary both for the Rutherford scattering function and DCS scattering function. If the scattering volume is not provided, it is computed from the potential volume from the relation defined in Section 1.4.1. Please refer to Subsection 3.3.1 for the description of the scattering functions and the required volumes.

**Setup class:** *Setup* class is defined in the `/src/ems/setup.h` file. It requires the parsed parameters, voxel size of the loaded data, and a count of voxels per volume side. For this reason, it is constructed after the volumetric data are loaded. The *Setup* class instance contains:

**Interaction bounding box** The interaction bounding box, represented by class *BoundingBox*, defines a bounding box within which the electron can undergo scattering events. If any electron leaves this area, it either hits the detector or is thrown away. The box is always axis aligned and centered in zero position.

**Detector and emitter position and size** Emitter and detector are assigned their initial positions, which are along the Z-axis, aligned with the volume. The frame, represented by the class *ImageLocation* defined in file `/src/ems/locations.h`, has the same size as one side of the bounding box.

**Dose** The dose of the electrons is loaded from the parameters in count per squared Bohr. The scattered and unscattered fractions are then estimated as described in Section 2.3.2.

**Mean free path** The mean free path for the majorant element is estimated, as described in Subsection 2.3.2. The mean free path is then passed to the scattering functions if they require it.

**Max scattering event count** The max scattering event count is parameter loaded from the input file. It is the upper bound for the number of scattering events per single electron. By default it is unbounded.

These were the important members of the *Setup* class instance initialized in the constructor. If we compile the program for GPUs, the class is copied into the device memory, so restrictions of the CUDA environment must hold for this class.

#### Physical volume

After the *Setup* class instance is ready, the volumetric data are loaded into the memory. Whether it is a device or host memory is decided during the compile-time by a type defined in the file `/src/helper_func.h`. The memory type of the volumes also determines whether the simulation runs on GPU or CPU. In further text, we assume GPU enabled simulation, and we explicitly mention when a code is executed on the device. For the host, the only difference is that everything happens within the host memory.

Volumetric data are in fact loaded into the texture images of the GPU via *bolt::TextureImage* class (or interpolated *bolt::HostImage* for host version), which can be imagined as three dimensional array with hardware-accelerated linear interpolation. However, before the volumetric data can be used within the simulation, they need to be wrapped in an instance of the *PhysicalVolume* class, defined in the file `/src/ems/raycast_data.h`. The *PhysicalVolume* extends the volumetric data with the *BoundingBox*, position and size, and methods to access the data easily, e.g. indexing with world coordinates. In short, it gives the volume a physical representation within our simulator.

## 3.3 Electron microscope components

The core class of the simulation is the *ElectronMicroscope* class. However, it requires several key components for its construction, which determine the image formation algorithm. These components are a scattering function, an image formation, and optionally one or two ray observers. In this section, we describe in detail these components, and the next section is then dedicated to the simulation itself. First, we present a list of objects and expressions that might appear in the description of the components:

`BOLT_DECL_HYBRID` The `BOLT_DECL_HYBRID` macro is a function annotation, that tells the compiler that this function must be compiled for both the host and the device. It is used extensively throughout the library.

***Ray*** The *Ray* class a simple C++ struct containing position of the ray in *Bohr3* units and unit-less direction of the ray with its inverse for faster intersection computations.

**Payload** Payload is a term used for the data transported by the ray. The actual class is determined from the image formation component. One of the key members transported in this class is particle importance, called weight, which is altered during the importance sampling of the DCS scattering function.

***RutherfordFormulae*** The *RutherfordFormulae* class is defined in the file `/src/ems/screened_rutherford_formulae.h`. The class is initialized by a chemical element and electron energy. It contains implementations of the Rutherford formulae described in Subsection 1.3.3, as well as the derived sampling and PDF functions from Section 1.4.4. Most of the methods can be called from a device code.

***LocalizedFourierTransform*** The *LocalizedFourierTransform* class is defined in the file `/src/ems/localized_fourier_potential.h`. It implements a localized 3D Fourier transform as described in Section 2.2. It is initialized with a potential volume, position, radius and a number of samples. It contains a method `BOLT_DECL_HYBRID ComplexType at(const Num3& frequency, size_t& samples_count) const`, which returns a Fourier value and total number of samples for the the requested frequency.

```
1  BOLT_DECL_HYBRID
2  ComplexType at(const Num3& frequency, size_t& samples_count)
       const
3  {
4    samples_count = bolt::product(Num3::fill(
       max_per_axis_samples_count));
5    ComplexType sum { };
6    if (!(Num { 2 } / bolt::maxElement(u(step_size)) >= bolt::
       maxElement(bolt::abs(frequency)))) {
7      return sum;
8    }
9
10   using namespace bolt;
11   for (size_t x = 0; x < max_per_axis_samples_count; ++x) {
12     for (size_t y = 0; y < max_per_axis_samples_count; ++y) {
13       for (size_t z = 0; z < max_per_axis_samples_count; ++z)
           {
14         Bohr3 position = min + offset + bolt::product(
             step_size, Num3 { x, y, z });
15         auto shift_param = Num(-2 * M_PI) * bolt::sum(bolt::
             product(frequency, bolt::div(position - min, size)
             ));
16         auto shift = ComplexType { cos(shift_param), sin(
             shift_param) };
17         sum = sum + ComplexType { volume.access(position) } *
              shift;
18       }
19     }
20   }
21   return sum;
22 }
```

At line 6, it checks via the Nyquist theorem, whether the sampling is sufficient for the requested frequency. If not, it returns complex zero. This way, we avoid repeated patterns stemming from undersampled frequencies. The rest of the method is an implementation of the discrete Fourier transform.

### 3.3.1 Scattering function

The scattering function is the main component of the microscope. It is an object defined by the following interface:

```
template<typename TRandomGenerator, typename TPayload>
  BOLT_DECL_HYBRID
  Ray scatter(
    const Ray& orig_ray,
    TRandomGenerator& rng,
    Bohr& next_event_distance,
    TPayload& payload,
    Bohr max_distance = Bohr { 0 }) const
```

The goal of this function call is to process a scattering event, determine the distance to the next scattering event and return a new *Ray* object.

`orig_ray` The `orig_ray` argument contains the current position, where the scattering should happen, and the original direction of the particle.

`rng` The `rng` is a callable functor that returns random numbers in range $[0, 1]$.

`next_event_distance` The `next_event_distance` argument is passed by reference and should be assigned in this function. It determines the distance to the next scattering event.

`payload` The payload argument contains the payload of the particle. The class is determined by the image formation component, but it usually contains at least simulation importance weight of the particle.

`max_distance` This argument is used to importance sample lower distances and sets the upper bound of the `next_event_distance`. As described in Section 2.3.2, most of the particles do not scatter within the *BoundingBox* of the sample, so the `next_event_distance` is set for the first scattering event to that distance. The default value zero means unbounded maximum distance.

Note that the function is `const`, as many particles are using the scattering function at the same time during the simulation. In the following text, we describe some of the implemented scattering functions, all of which can be found in the file `/src/ems/scattering_function.h`. The first is the *NoScattering* scattering function for illustration purposes, followed by the two functions described in Subsection 1.4.4.

#### Tracing in transmittance model

The *NoScattering* scattering function is very simple object that almost conforms to the interface described above.

```
struct NoScattering {
  explicit NoScattering() {
  }

  template<typename TRandomGenerator, typename TPayload>
  BOLT_DECL_HYBRID
```

```
7   Ray scatter(
8     const Ray& orig_ray,
9     TRandomGenerator& rng,
10    Bohr& next_event_distance,
11    TPayload& payload,
12    Bohr max_distance = Bohr { 0 }) const
13  {
14    next_event_distance = Bohr { INF };
15    return orig_ray;
16  }
17 };
```

It does not change the direction of the ray and sets the `next_event_distance` variable to infinity, essentially saying that this ray does not scatter. It does not respect the `max_distance` parameter, because this scattering function is not used to simulate electron transport. It is used to facilitate the common transmittance model from Section Transmittance model. Not respecting the `max_distance` parameter causes the dose estimation to be biased, but it has no effect on the transmittance model, because the model does not respect dose.

### Rutherford scattering function

The *RutherfordSimpleScattering* class represents the Rutherford scattering function. The body of the scatter function looks like this:

```
1  {
2    Num random_sample = rng();
3    if (max_distance > Bohr { 0 }) {
4      Num bound = Num { 1 } - exp(-max_distance / mean_free_path);
5      random_sample *= bound;
6    }
7
8    next_event_distance = sampleNextEventDistance(random_sample);
9
10   auto scattering_scattering_coefficient = scattering_volume.access(
         orig_ray.origin);
11   auto ratio = scattering_scattering_coefficient / majorant;
12   if (rng() < ratio) {
13     Num ruther_pdf { };
14     auto new_dir = sampleDirection(orig_ray.dir, rng(), rng(),
           ruther_pdf);
15     return Ray { orig_ray.origin, new_dir };
16   } else {
17     return orig_ray;
18   }
19 }
```

The first conditional block (line 3) handles the `random_sample` bounds according to the dose propagation from Section 2.3.2. It is followed by the next event distance estimation (line 8). The rest of the code illustrates the implementation of the delta tracking described in Section 1.4.3. If a random number is smaller than the ratio of the local scattering coefficient and the majorant (line 12), it is a real scattering event, otherwise, it is a null scattering event and we continue in the same direction. If it is a real scattering event, we use the derived sampling function from Section 1.4.4 to sample a new direction (line 15). The atomic number

used in the Rutherford sampling function has been parsed from the parameters file.

#### DCS scattering function

The DCS scattering function is represented by the class *RutherElectronScattering*. The Ruther portion of its name refers to the applied importance sampling via Rutherford formulae. The body of its scatter function is nearly identical with the Rutherford scattering function described above, so we show only the different part.

```
1 if (rng() < ratio) {
2   Num ruther_pdf { };
3   auto new_dir = sampleDirection(orig_ray.dir, rng(), rng(),
        ruther_pdf);
4   auto electron_pdf = getPdfForDir(orig_ray, new_dir,
        total_cs_volume.access(orig_ray.origin));
5   payload.weight *= electron_pdf / ruther_pdf;
6   return Ray { orig_ray.origin, new_dir };
7 }
```

We sample a new direction using Rutherford formulae (line 4). We do not estimate an atom, but we use the atom estimated from the majorant value. Then we compute the PDF of this direction (line 5) from Equation (1.20) and use both the sampling PDF and the Rutherford PDF to weight the electron contribution (line 6).

```
1 BOLT_DECL_HYBRID
2 Num getPdfForDir(const Ray& orig_ray, const Num3& new_dir, Num
      total_cross_section) const
3 {
4   auto localized_fp = ems::makeLocalizedFourierTransform(
5     potential_volume,
6     orig_ray.origin,
7     cross_section_frame_size,
8     max_fourier_samples_per_axis);
9
10  auto value = computeSampleValue(localized_fp, orig_ray.dir,
        new_dir);
11  return value / total_cross_section;
12 }
```

The PDF computation is done via the class *LocalizedFourierTransform*. Note that the normalizing total cross section is not computed during scattering, but precomputed in the `total_cs_volume`, otherwise the simulation would be extremely slow.

### 3.3.2 Image formation

The image formation component is responsible for the interpretation of the electrons hitting the detector. It is defined by the following interface:

```
1 using Payload = *;
2
3 template<typename TProjectionView>
4   BOLT_DECL_HYBRID
```

```
5   void operator()(
6     const TProjectionView& projection,
7     const IPixel2& projection_pixel,
8     const Payload& payload,
9     const Ray& ray_deviation,
10    const Bohr2& pixel_size) const
```

The first line is a definition of the payload class, which is then used throughout the simulator. Note that the payload class must have a constructor with no arguments. The content of the class can be modified by the three customizable components of the microscope: the scattering function, the image formation, and the ray observers. It is up to the programmer to make sure that all those components are sensibly modifying the electron payload.

The image formation component must be callable with the listed arguments. The object is called each time an electron hits the detector. Passed arguments have the following meaning:

**`projection`** The `projection` argument is an instance of the *bolt::View* class that contains the final image.

**`projection_pixel`** The `projection_pixel` argument contains coordinates of a pixel that has been hit by the electron. It is always within bounds of the `projection` view.

**`payload`** This argument contains the payload of the electron.

**`ray_deviation`** The `ray_deviation` contains exact position of the impact and a direction under which the impact has occurred. However, the direction is related to the detector in such a way that perpendicular impact has always a direction of $[0, 0, 1]$.

**`pixel_size`** The `pixel_size` argument contains a pixel size of the detector.

The class is supposed to modify the `projection` data to record the electron impact. However, as with other components, this member function can be executed by many threads at the same time. Therefore the modifications must be thread-safe, otherwise, race conditions are to be expected.

### Counter image formation

All of the implemented image formation components can be found in a file `/src/ems/image_formations.h`. Here we present the *CounterImageFormation* class that is used in all examples in the provided test program and works in accordance with the description in Subsection 1.2.3.

```
1 struct DistancePayload {
2   Num weight { 1 };
3   Bohr distance { 0 };
4 };
5
6 struct CounterImageFormation {
7   using Payload = DistancePayload;
8
9   template<typename TProjectionView>
```

```
10   BOLT_DECL_HYBRID
11   void operator()(
12     const TProjectionView& projection,
13     const IPixel2& projection_pixel,
14     const Payload& payload,
15     const Ray& ray_deviation,
16     const Bohr2& pixel_size) const
17   {
18 #if defined(__CUDA_ARCH__)
19     atomicAdd(&projection[u(projection_pixel)], payload.weight);
20 #else
21     projection[u(projection_pixel)] += payload.weight;
22 #endif
23   }
24 };
```

We use the *DistancePayload* class as our payload. While we do not use the distance member of the class in the provided test program, it has been used for debugging and other experimental image formation models. The *CounterImageFormation* class is quite simple, for each electron impact, it adds its weight to the pixel value. Note that there are two versions for the device and the host. The host is not thread-safe, as the current version of the software does not enable multithreading on the host.

### 3.3.3 Ray observers

Ray observers provide a means to modify the payload of an electron when it moves through the volume bounding box defined in *Setup* class. They are an optional component. They are not used for the simulation of electrons in our test program, but they facilitate a simulation of the transmission model. The ray observers are defined by the following interface:

```
1 template<typename TPayload>
2   BOLT_DECL_HYBRID
3   void operator()(TPayload& payload, const Ray& ray, const Bohr&
        step) const
```

It is again a callable object and the same rules apply as in the previous components - mainly that the method can be called by many electrons at once. The arguments have the following meaning:

**`payload`** This argument contains the payload of the electron.

**`ray`** The `ray` argument represents the observed ray. It provides an origin of the observation and direction. Each electron is observed on the path between scattering events and the in the segments from the emitter and to the detector.

**`step`** The step argument contains the distance for which the electron should be observed. Together with the `ray` argument, it gives a full description of the observed path.

**Weighting ray observer**

Here we provide an example of a ray observer. This observer implements the transmittance model together with the *NoScattering* scattering function. We have intentionally left out the body of the `getWeightedDistance` method, as it is unnecessary for the illustration of an observer and quite long.

```
template<typename TVolumeView>
struct WeightingVolumeRayObserver {
  BOLT_DECL_HYBRID WeightingVolumeRayObserver(PhysicalVolume<
      TVolumeView> physical_volume, float outside_weight) :
    physical_volume(physical_volume),
    outside_weight(outside_weight),
    min_step_size(bolt::minElement(physical_volume.getSizeOfVoxel())
        / 100.0f)
  {
  }

  template<typename TPayload>
  BOLT_DECL_HYBRID
  void operator()(TPayload& payload, const Ray& ray, const Bohr&
      step) const {
    payload.distance +=Bohr {getWeightedDistance(ray, step)};
  }

private:

  BOLT_DECL_HYBRID
  float getWeightedDistance(const Ray& start_ray, const Bohr& step)
      const {
    ***
  }

  PhysicalVolume<TVolumeView> physical_volume;
  float outside_weight;
  Bohr min_step_size;
};
```

The observer adds a weighted distance to the payload distance member. It computes the weighted distance within the `getWeightedDistance` as a sum of traversed voxels times the value within that voxel, essentially computing the path integral.

## 3.4 Electron microscope simulator

After we have described the components of the simulator, we can construct an instance of the class *ElectronMicroscope* defined in the file `/src/ems/ems.h`. The class has a following helper function for construction which enables the C++ template argument deduction:

```
template<typename TScattering, typename TImageFormation, typename
    TRayObserverOne = NullRayObserver,
  typename TRayObserverTwo = NullRayObserver>
ElectronMicroscope<TScattering, TImageFormation, TRayObserverOne,
    TRayObserverTwo> constructElectronMicroscope(
  const ElectronMicroscopeSetup setup,
```

```
5    TScattering scattering,
6    TImageFormation image_formation,
7    TRayObserverOne observer_one = { },
8    TRayObserverTwo observer_two = { })
9  {
10   return ElectronMicroscope<TScattering, TImageFormation,
         TRayObserverOne, TRayObserverTwo> { setup, scattering,
         image_formation,
11       observer_one, observer_two };
12 }
```

The constructor of the simulator class requires an instance of the *ElectronMicroscopeSetup* class and the components described in the previous section - one scattering component, one image formation component, and optionally one or two observer components. Each of those components is a templated argument. It can be any object that follows the interfaces described in the previous section - mainly that they will potentially operate on the GPU device, and each of the interface methods may be called from multiple threads at the same time. Note that we do not pass any volumetric data into the microscope directly. This is because the components themselves should carry the pointers/views of the data they require. The microscope class works only with the *BoundingBox* instance initialized in the `setup` argument.

### 3.4.1 Rendering

The instance of the *ElectronMicroscope* class offers a single public method `render`, which performs the simulation and writes the result into the provided memory. However, before we describe the `render` method, we introduce a function to create a *BoltView* image of random generators, which is necessary to run the simulation.

```
1  template<bool TRunOnDevice, typename TSizeType>
2  auto createRandomGeneratorsImage(TSizeType size, unsigned seed) {
3    if (seed == 0) {
4      seed = std::random_device { }.operator ()();
5    }
6    return std::move(detail::createRandomGeneratorsImageImpl<
         TRunOnDevice>::run(size, seed));
7  }
```

The view derived from this image is then used during the simulation to generate random numbers for each of the electrons. Special generators view is needed because on the device, the standard random generators are not present, and the *CUDA* random generators are good for a one-time generation of a fixed count of numbers. The created image of random generators contains a pseudorandom generator in each pixel, and all of the pixels are independent, meaning that the simulation can be run in as many threads as there are pixels in the image.

The `render` method of the *ElectronMicroscope* class has the following definition:

```
1  template<typename TProjectionView, typename TGeneratorsView>
2  Detector render(
3    TProjectionView projection,
4    bolt::Quaternion<Num> orientation,
5    const Pixel3& subpixel_shift,
```

```
6     const Bohr& detector_distance,
7     TGeneratorsView generators)
8 {
9     Num unscattered_dose_per_pixel;
10    auto emitter_detector = constructEmitterAndDetector(
11      t<IPixel2>(projection.size()),
12      t<IPixel2>(generators.size()),
13      orientation,
14      subpixel_shift,
15      detector_distance,
16      unscattered_dose_per_pixel);
17    Emitter emitter = emitter_detector.first;
18    Detector detector = emitter_detector.second;
19    auto raytracer = constructRayTracer(*this, projection, generators,
          emitter, detector);
20    bolt::fill(projection, typename TProjectionView::Element {
          unscattered_dose_per_pixel });
21    EmsIndexView<2, TProjectionView::kIsDeviceView> emitting_view { u(
          emitter.size) };
22    bolt::forEach(emitting_view, raytracer);
23    return detector;
24 }
```

The arguments have the following meaning:

**`projection`** The `projection` is preferably an instance of a 2D *BoltView* view object. The number of pixels determines the pixel size of the virtual detector because the *ImageLocation* is already determined by the electron setup. The memory type of this object (host or device) also determines whether the simulation is executed on the host or the device. The rendered image is stored in this view.

**`orientiation`** The `orientation` argument is a quaternion describing the orientation of the microscope with respect to the simulation *BoundingBox*. During the simulation, the *BoundingBox* remains static, but the emitter and detector are rotated according to this parameter.

**`subpixel_shift`** This argument shifts the emitter and detector, which visually shifts the projection of the sample.

**`detector_distance`** The detector distance value corresponds to the defocus effect described in Subsection 1.2.3. The larger is the distance, the greater the electron spread can be observed in the rendered image.

**`generators`** The `generators` a 2D *BoltView* view of a generators image described previously. Each pixel of this object must provide a callable object that returns a uniform random number in the range $[0, 1]$. The memory type of this object should correspond to the memory type of the `projection`. The size of this view determines the size of an emitter pixel, similarly to how the `projection` determines the size of the detector pixel. The number of emitter pixels also determines the number of possible threads of the simulation.

The method first constructs and instance of the *Emitter* class and the *Detector* class, both if which are defined in the file `/src/ems/rayast_data.h`. They

connect together the *ImageLocation* class instances for emitter and detector with the views to give them physical representation, similarly to what *PhysicalVolume* class does. The variable `unscattered_dose_per_pixel` is also initialized during the construction and then written into the result projection (line 20).

The simulation of the electrons is executed on line 22. For each emitter pixel, a `raytracer` object is executed. The code within the `raytracer` is executed either on the device or the host, depending on the configuration. In the following text, we describe the *RayTracer* class which represents the core of the simulation. After the call to the `render` method is finished, the final image is stored in the provided `projection` view.

### Ray tracer

The *RayTracer* class is defined in the file `/src/ems/ray_tracer.h`. For the execution on device, a copy of the instance of this class is passed to each executed thread, along with a corresponding emitter pixel index. The construction function is defined as:

```
1  template<typename TElectronMicroscope, typename TProjectionView,
       typename TGeneratorsView>
2  BOLT_DECL_HYBRID
3  RayTracer<TElectronMicroscope, TProjectionView, TGeneratorsView>
       constructRayTracer(
4    const TElectronMicroscope& microscope,
5    TProjectionView projection,
6    TGeneratorsView generators,
7    const Emitter& emitter,
8    const Detector& detector)
9  {
10   return RayTracer<TElectronMicroscope, TProjectionView,
         TGeneratorsView>(microscope, projection, generators, emitter,
         detector);
11 }
```

It takes the following arguments:

**`microscope`** This argument contains an instance of the *ElectronMicroscope* class in which the simulation occurs. The microscope provides a simulation bounding box and the components that control the simulation of the electron.

**`projection`** A view that serves as result storage.

**`generators`** A view that provides a random generator for this emitter pixel.

**`emitter`** The class instance provides methods to get a position and direction of an initial electron from the emitter pixel coordinates. It also contains a count of electrons that need to be simulated for each emitter pixel.

**`detector`** This is a physical representation of the `projection`. It has a pixel size, position, and rotation. It is used to compute the impact location of the electrons that hit the detector.

In the `render` method of the *ElectronMicroscope*, for each emitter pixel a *RayTracer* instance is executed in the following method:

```
1 BOLT_DECL_HYBRID
2 void operator()(const bolt::Int2& origin_pixel) const {
3   tracePixelRays(t<IPixel2>(origin_pixel));
4 }
```

Which is only a wrapper function for the `tracePixelRays` function, that iterates over all electrons of the pixel and traces them:

```
1 BOLT_DECL_HYBRID
2 void tracePixelRays(const IPixel2& origin_pixel) const {
3   for (size_t sample = 0; sample < emitter.samples_per_pixel; ++
        sample) {
4     Payload payload { };
5     Bohr3 origin = emitter.getSourceLocation(origin_pixel, getRnd(
          origin_pixel), getRnd(origin_pixel));
6     Ray ray { origin, emitter.init_ray_direction };
7     traceSingleRay(ray, origin_pixel, payload);
8   }
9 }
```

As described before, the `emitter` provides a number of electrons that should be simulated for the current pixel, as well as the initial position and direction of the electron. The following two functions demonstrate, how the generators view is used to generate random numbers:

```
1 BOLT_DECL_HYBRID
2 typename TGeneratorsView::AccessType getGenerator(const IPixel2&
    origin_pixel) const {
3   return generators[u(origin_pixel)];
4 }
5
6 BOLT_DECL_HYBRID
7 float getRnd(const IPixel2& origin_pixel) const {
8   return getGenerator(origin_pixel)();
9 }
```

The main tracing body of a single electron is in the method `traceSingleRay`:

```
1 BOLT_DECL_HYBRID
2 void traceSingleRay(const Ray& start_ray, const IPixel2&
    origin_pixel, Payload& payload) const {
3   size_t event_count = 0;
4   Bohr next_event_distance;
5   Ray ray = start_ray;
6   ray = microscope.scattering.scatter(
7     ray,
8     getGenerator(origin_pixel),
9     next_event_distance,
10    payload,
11    microscope.setup.distance_from_emitter_to_detector);
12
13  Bohr detector_plane_distance;
14  IPixel2 detector_pixel;
15  bool valid_detector_pixel;
16  valid_detector_pixel = detector.getIntersectedPixel(ray,
        detector_pixel, detector_plane_distance);
17
18  while (detector_plane_distance > next_event_distance &&
        event_count < microscope.setup.max_scattering_event_count
```

```
19      && willIntersectVolume(ray)) {
20      observe(payload, ray, next_event_distance);
21      ray = { ray.origin + ray.dir * next_event_distance, ray.dir };
22  //    LOG("previous ray: " << toString(ray.dir));
23      ray = microscope.scattering.scatter(ray, getGenerator(
            origin_pixel), next_event_distance, payload); //Warning:
            next_event_distance is changed here
24  //    LOG("new ray: " << toString(ray.dir));
25      event_count++;
26      valid_detector_pixel = detector.getIntersectedPixel(ray,
            detector_pixel, detector_plane_distance);
27    }
28
29    if (valid_detector_pixel) {
30      observe(payload, ray, detector_plane_distance);
31      ray = { ray.origin + ray.dir * detector_plane_distance, ray.dir
            };
32
33      Frame emitter_frame { emitter.location.xDir(), emitter.location.
            yDir(), emitter.init_ray_direction };
34
35      auto ray_deviation = emitter_frame.toLocal(ray.dir);
36      microscope.image_formation(projection, detector_pixel, payload,
            Ray { ray.origin, ray_deviation }, detector.pixel_size);
37    }
38  }
```

In the first several lines, variables used in the simulation are declared. On line 6, the `ray` is attenuated by the microscope scattering function for the first time. It is not there to change the direction but to initialize the `next_event_distance` variable. Also note, that this is the only place where the optional argument `max_distance` of the scattering function is used and it is given the value of `microscope.setup.distance_from_emitter_to_detector`. Beware that this distance is not affected by the defocus shift that modified the position of the detector.

In the `while` cycle defined on line 18, the electron progresses through the volume. First, observers are called on it (line 20), then the electron is shifted to a new position based on the `next_event_distance` value. This is followed by a scattering event on line 23. Lastly, a detector pixel, towards which the electron aims, is updated. The boolean variable `valid_detector_pixel` describes whether the detector pixel is within the detector boundaries or lies on the detector plane but outside the detection area. This cycle is repeated as long as the following conditions hold:

- The `next_event_distance` is shorter than the distance to the detector plane.
- The event count is within the limit.
- The ray aims towards or is within the microscope *BoundingBox*.

If any of those conditions are not fulfilled, the electron has either hit the detector or has scattered away from both the sample and the detector. In a case, that the electron hits the detector and it is a valid detector pixel, the image component is executed with the appropriate arguments (line 36).

# 4. Experiments

In this chapter, we study the quality of the image formation algorithm. In the first section, we describe the potential map used in the experiments. In the next section, we study the behavior of the scattering functions for various parameters. In later sections, we show the simulated projections and compare them to the projections of the transmittance and the multislice models. In the last section, we evaluate the correctness of the simulation via normalized cross-correlation with the measured dataset.

## 4.1 Data

For experiments, we use the $\beta$-galactosidase molecule, determined by cryo-EM at an average resolution of 2.2 Å [29]. Both the reconstructed map and the atomic model files of the molecule are publicly available. We use the atomic model from the PDB file format and reconstruct the potential map in our software, to make sure that we have a map of the Coulomb potential of the molecule. Figure 4.1 displays the atomic model of the molecule and Figure 4.2 shows the Coulomb potential of the molecule generated from the atomic model.

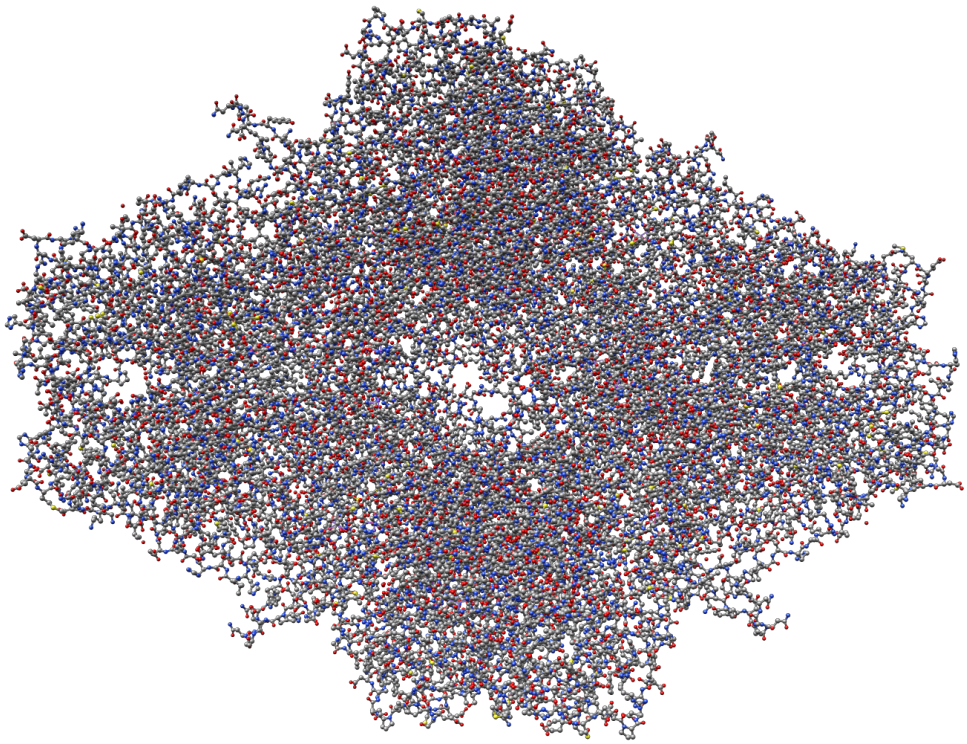


Figure 4.1: Visualization of the publicly available atomic map of the $\beta$-galactosidase. Rendered in the UCSF Chimera tool.

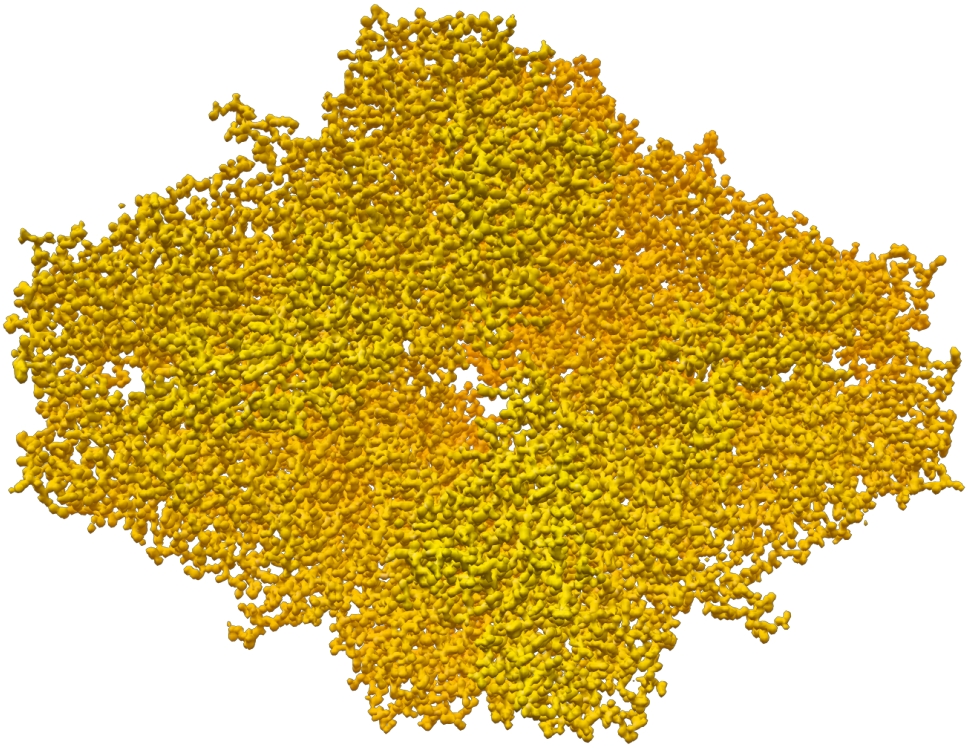


Figure 4.2: Visualization of the generated potential map of the $\beta$-galactosidase used in experiments. Rendered in the UCSF Chimera tool.

## 4.2 Scattering function

In this section, we illustrate the behavior of the implemented scattering functions: the DCS scattering function, and the Rutherford scattering function. For both functions, we plot their PDF for various positions and settings. For these experiments, we used a generated potential map with a voxel size of 0.6375 Å and the electron energy of 300 keV, which is the same energy used for the acquisition of the $\beta$-galactosidase and commonly used in cryo-EM.

### 4.2.1 DCS scattering function

With the DCS scattering function, we are mainly interested in the influence of the size of the considered potential location - the radius term $r$ in the localized potential Fourier transform. In Figure 4.3, we show the differential cross-section values for a cone of angles around the forward direction. While there are some changes in the shape, the preferred direction is always the forward direction and the directions of interest are within 4 degrees from the forward direction, which fits the theory that elastic scattering angles are very small [2]. For this reason, we use only as many samples as required to correctly reconstruct scattering angles up to 4 degrees, see Section 2.2. For higher scattering angles, we approximate the DCS with zero. The bias introduced by this cut-off of angles is expected to be small.

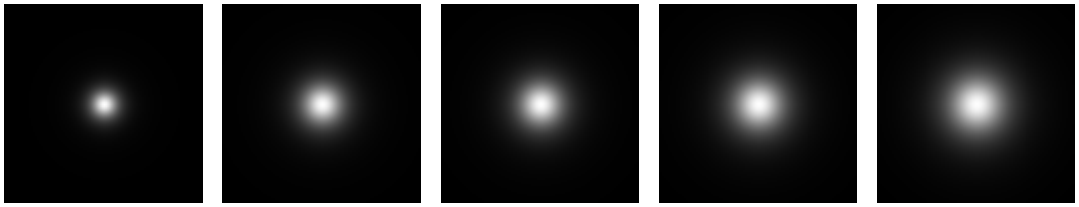
(a) r=$2a_0$ (b) r=$4a_0$ (c) r=$8a_0$ (d) r=$16a_0$ (e) r=$32a_0$

Figure 4.3: The figure displays the PDF of the DCS scattering function around the forward direction represented by the pixel in the middle, for different radii $r$ of the molecule potential. The spread angle in the image is 6 degrees, meaning that a pixel in the center of each side of the image represents a direction with a scattering angle of 6 degrees. Very low scattering angles are to be anticipated and match the theory.

### 4.2.2 Rutherford scattering function

The Rutherford scattering function is defined for a single atom of a known atomic number and does not depend on the potential values. Nevertheless, it makes a good analytical alternative to the DCS scattering function. We found two applications of this function. We use it as a fast analytical approximation of the DCS scattering function and compare those functions in the image formation simulation. We also use the function to importance sample the DCS function thanks to their similarity.

In Figure 4.4 we display the Rutherford PDF for different elements which are present in the $\beta$-galactosidase and common in other biological molecules. The image shows that the scattering function for Hydrogen (atomic number 1) is visually most similar with the DCS scattering function examples shown above. The suitability of this atomic number is later confirmed by the cross correlation experiments with real data (Figure 4.14).



(a) H, Z=1 (b) C, Z=6 (c) O, Z=8 (d) Mg, Z=12 (e) S, Z=16

Figure 4.4: The figure displays the PDF of the Rutherford scattering function around the forward direction, represented by the pixel in the middle for various elements and their atomic numbers Z. The spread angle in the image is 6 degrees, meaning that a pixel in the center of each side represents a direction with the scattering angle of 6 degrees. Hydrogen visually fits the DCS the most, compare to 4.3. The suitability of the lowest atomic number is later confirmed by the experiments (Figure 4.14).

## 4.3 Projections

In this section, we show projections of the molecule with different simulation setups and later compare them to the projections from the transmittance and multislice model. Note that all images are normalized to increase the perceived contrast. This may cause the diversions in the color of the background. The setup is similar as before, we used a generated potential map with a voxel size of 0.6375 Å and the electron energy of 300 keV.

### 4.3.1 Electron dose

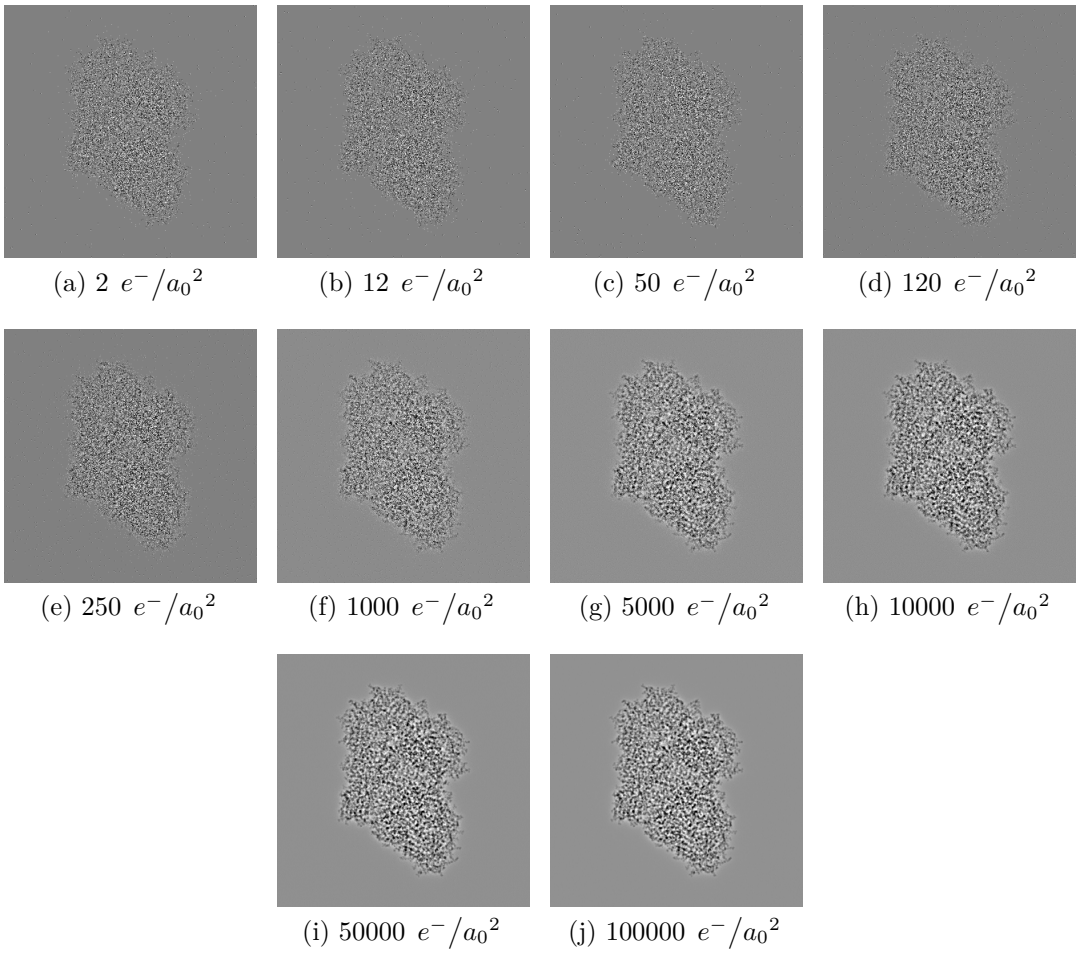
In Figure 4.5 we show projections with the DCS scattering function at an increasing electron doses. As expected, with the increasing dose the noise within the image recedes. For illustration, the dose used to acquire the $\beta$-galactosidase data is 45 $e^- \big/ \text{Å}^2$ which corresponds to approx. 12 $e^-/{a_0}^2$. However, the estimated mean free path for the majorant element, which is Sulfur for the $\beta$-galactosidase, is approx. 3800 $a_0$, while the length of the volume is about 700 $a_0$. This means that only 2 electrons per Bohr squared undergo a scattering event and many of these events will result in null scattering, essentially contributing no information to the image. The real projections contain noise from other sources (scattering from the air and water molecules, detector noise, variances in the electron beam density, etc.) and it is not possible to directly compare them, as shown in Section 4.4. Thus, we cannot verify the validity of the noise of our model caused by undersampling.

### 4.3.2 Fourier radius influence

In Figure 4.6 we illustrate how the projections differ for various radii of the Fourier transform of the potential in the DCS scattering function. The projections differ more than was anticipated. Note that the radius does not influence where the electrons scatter, only in which directions. With an increasing radius, low-frequency effects of the scattered electrons seem to be exchanged for higher-level clusters of scattered electrons. It may be explained by the fact that for larger scattering radii, a larger portion of the molecule influences the new direction, leading to the scattered electron carrying lower frequency information, but it is only a speculation. We also evaluate the different radii in comparison with the measured data in Figure 4.13. However, these measurements do not give a clear answer to which of the radii approximates the real behavior of the electron most correctly, it remains an open question.

### 4.3.3 Rutherford scattering influence

Figure 4.7 displays the simulated projections with the Rutherford scattering function for varying atomic numbers. We later show that this approximation works quite well with the lowest atomic number. We have used a dose of 100 000 $e^-/{a_0}^2$. The image shows that for increasing atomic number the halo effect around the molecule becomes larger. This is due to the scattered electrons having tendency to scatter into larger angles for heavier atoms, as shown in Figure 4.4.



(a) 2 $e^-/{a_0}^2$ (b) 12 $e^-/{a_0}^2$ (c) 50 $e^-/{a_0}^2$ (d) 120 $e^-/{a_0}^2$

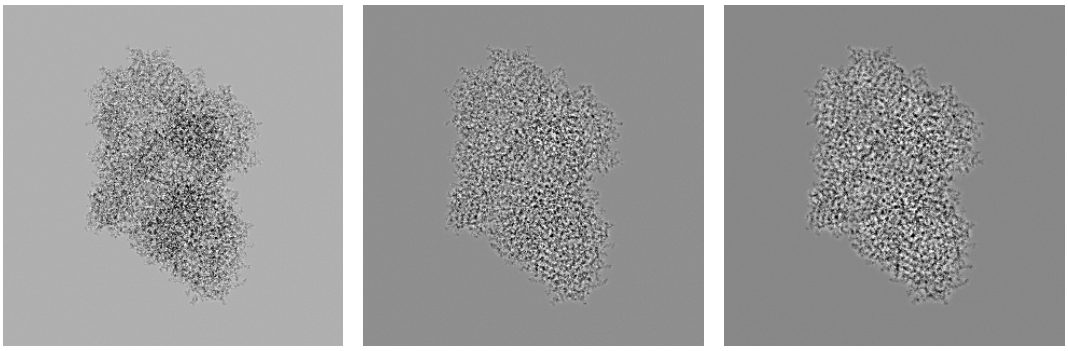
(e) 250 $e^-/{a_0}^2$ (f) 1000 $e^-/{a_0}^2$ (g) 5000 $e^-/{a_0}^2$ (h) 10000 $e^-/{a_0}^2$

(i) 50000 $e^-/{a_0}^2$ (j) 100000 $e^-/{a_0}^2$

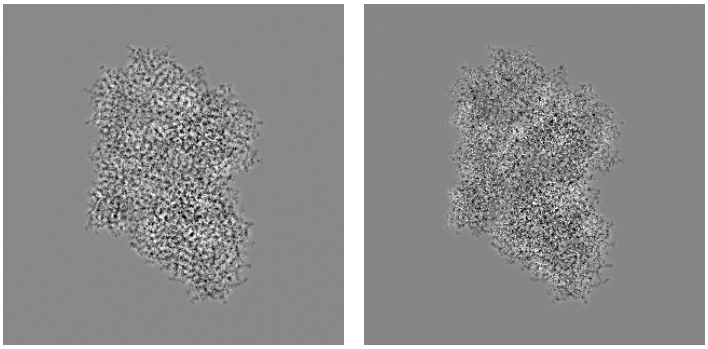
Figure 4.5: The figure shows the influence of the electron dose on the acquired image. The dose of 12 $e^-/{a_0}^2$ corresponds to a realistic dose of 45 $e^-/\text{Å}^2$, which has been used to acquire the real $\beta$-galactosidase data. The images are normalized which can cause diversions in the backgroud color. As anticiapted, the percieved amount of noise is reduced with increased electron dose.

### 4.3.4 Defocus influence

In this subsection, we show the influence of various defocus distances for the Rutherford scattering model, however, the effect is nearly identical for the DCS model. As described in Subsection 1.2.3, we model the defocus only as a distance of the detector from the sample, because our model does not contain any lenses. As shown in Figure 4.8, the important effect of this parameter is the area that the scattered electrons cover on the detector. Again a dose of 100 000 $e^-/{a_0}^2$ has been used to produce the images. In the $\beta$-galactosidase dataset, the estimated defocus values range between 3000 Å and 40 000 Å. In the figures, we show a similar range of defocus values. We have also included the projection at a very low defocus of 192 Å to illustrate the vast difference. Our model does not allow smaller defocus than half of the size of the input volume. Also note that for our model a value of zero defocus would mean loss of all information for the potential in focus. A similar problem is also present in the real microscopes and is the reason why the measurements are done in defocus.



(a) r=1$a_0$ (b) r=2$a_0$ (c) r=4$a_0$



(d) r=8$a_0$ (e) r=16$a_0$

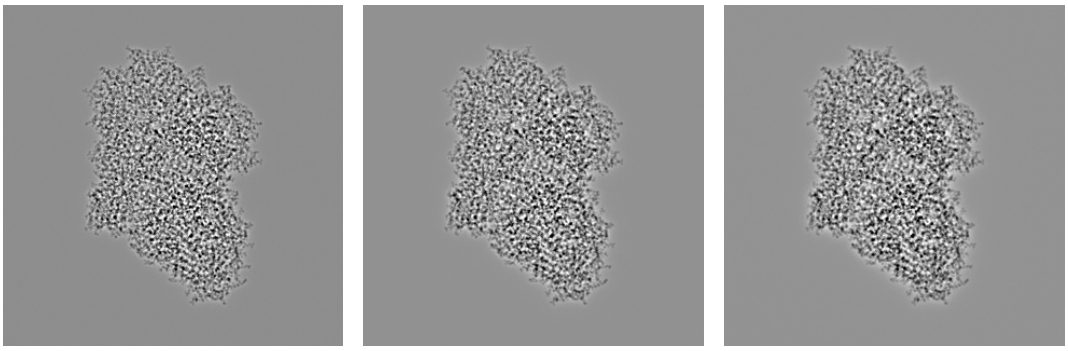
Figure 4.6: Projections from the DCS scattering model with different radius of the scattering. The radius of 1 $a_0$ causes the scattered electrons to be mostly weighted down to zero, corresponding to tiny scattering angles. With an increasing radius, low frequency effects of the scattered electrons seem to create more centered clusters of scattered electrons. The effect was not anticipated and more experiments are needed for explanation.

### 4.3.5 Electron energy influence

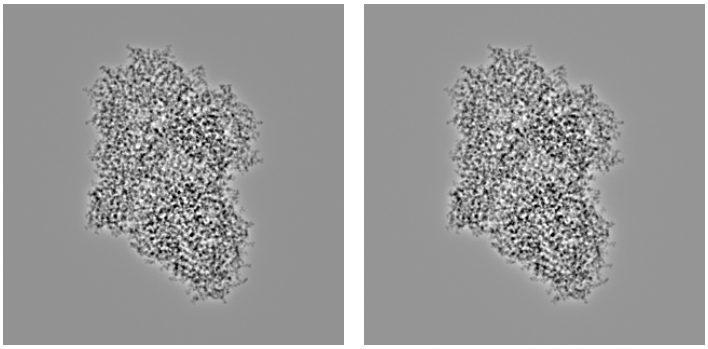
Figure 4.9 illustrates how the projections differ for various initial electron energies. We have used the DCS model for this experiment, as the energy of the electron influences not only the electron mean free path, but also the magnitude of the electron momentum which is used in the DCS evaluation. However, as can be observed in the image, the major effect is in the amount of noise stemming from longer mean free paths for higher energies, as shown in Table 4.1. If we return once more to the sphere graph from Figure 1.4, the electron energy influences the radius of the sphere, but the low scattering angles are still located around the frequencies close to zero. This means that the DCS distribution for low scattering angles probably remains similar for various energies of the electrons.

## 4.4 Comparison

In this section, we compare simulated images from our model with both the transmittance and multislice models. The measured projections from the microscope are not viable for direct comparison due to a very low signal to noise ratio, see Figure 4.10. Instead, in Subsection 4.4.2 we compare the quality of the simulation



(a) Z=1 (b) Z=3 (c) Z=8



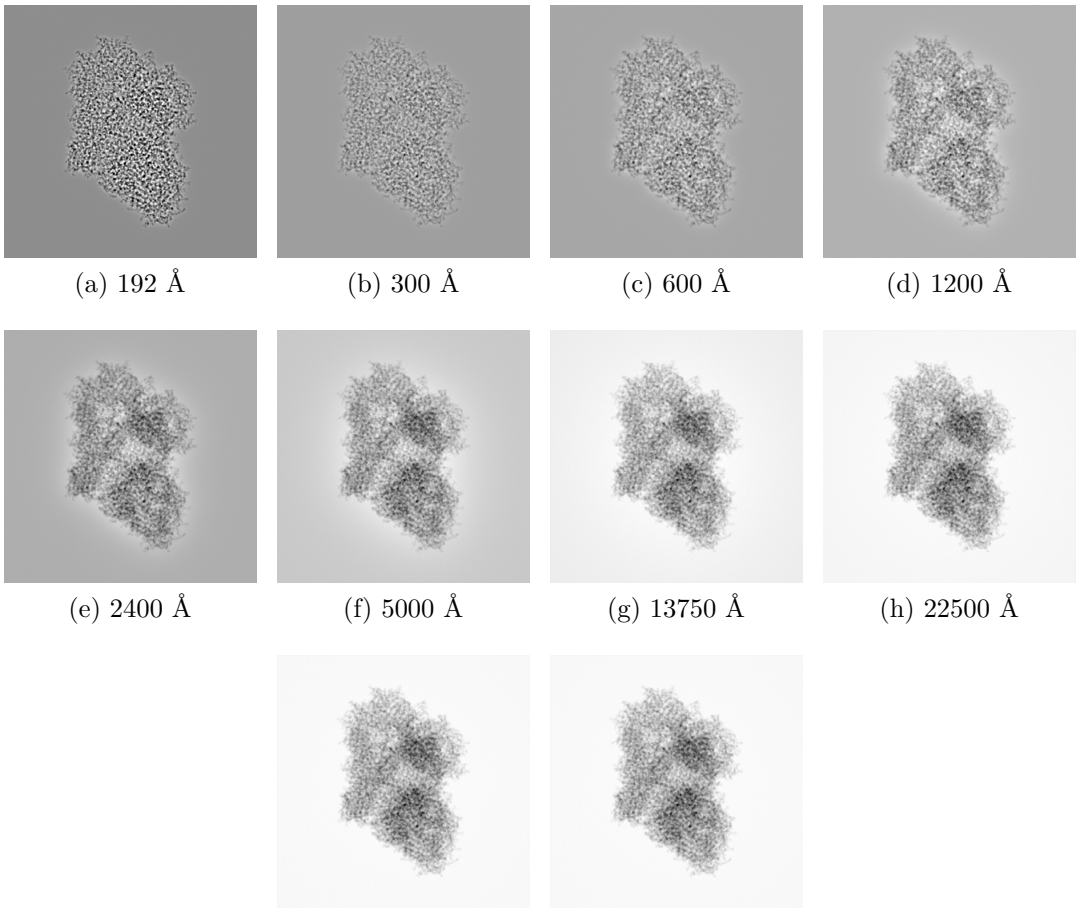
(d) Z=12 (e) Z=16

Figure 4.7: Images show projections from the Rutherford scattering model with different atomic number used in the scattering formula. For higher atomic numbers, the scattering angles became larger (Figure 4.4), leading to an increased halo distance around the molecule. The Hydrogen with Z=1 is later shown to be the best approximation when compared with the measured data.

with averages of the measured projections.

### 4.4.1 Projections

Figure 4.11 displays the projections of the $\beta$-galactosidase molecule from the same angle using different models. The multislice model shows the amplitude of the exit plane wave, without the effects of the lenses and defocus. The transmittance model does not simulate defocus at all. Our DCS and Rutherford models have the lowest possible defocus of half the size of the potential volume, which is about 160 Å. There is a relatively good visual correspondence of the DCS and Rutherford models with the multislice model. Note that while the multislice model simulates a plane wave function propagating through the potential map, our models simulate electrons as particles and we simply detect their numbers on the detector, so the similarity is remarkable. The transmittance model is not a good approximation of the image formation, at least not at low defocus. However, as shown in Figure 4.8, the defocus causes scattered electrons to cover a larger area and the transmittance model actually becomes a very good approximation. This is beacuse the contrast effect of the scattered electrons becomes weaker, and the transmittance model treats the volume as a purely absorbtion volume, in which electrons disapear during interactions.



(a) 192 Å (b) 300 Å (c) 600 Å (d) 1200 Å

(e) 2400 Å (f) 5000 Å (g) 13750 Å (h) 22500 Å

(i) 31250 Å (j) 31250 Å

Figure 4.8: The figure illustrates how the simulated images differ for various defocus values. In these figures, we used the Rutherford scattering function with the Hydrogen scattering parameter, which provided the finest fit with the measured data, as we show in Section 4.4.

### 4.4.2 Class averages

In this subsection, we compare the simulation with the measured data from the $\beta$-galactosidase dataset. As shown in Figure 4.10, the measured projections themselves are too noisy for direct comparison. Instead, we use class averages from the preprocessing step of the reconstruction - after the projections of the molecule are found and cut from the micrograph, they are classified by similarity into various classes representing projections from a similar angle. The classification was done by the RELION tool [30]. Figure 4.12 shows four averages that we used for the model quality estimation.

#### Setup

During the reconstruction process, the direction of each projection is estimated. From the RELION tool, we have four sets of projections with similar viewing angle and the corresponding average image. We simulate a set of projections with the same angles and parameters, and then compare the average of the simulated set with the average of the measured set. As a similarity metric of the averages, we

| Energy [keV] | $\lambda$ [$a_0$] |
|---|---|
| 50 | 1278.46 |
| 100 | 2049.53 |
| 150 | 2652.42 |
| 200 | 3132.67 |
| 250 | 3521.41 |
| 300 | 3840.50 |
| 350 | 4105.63 |
| 400 | 4328.31 |
| 450 | 4517.15 |
| 500 | 4678.67 |

Table 4.1: Table shows estimated mean free paths (in Bohr units) for different electron energies. Note that the distance from the emitter to detector at the lowest defocus setup, which is used in this test, is around 700 $a_0$.

use a normalized cross-correlation defined as:

$$ncc(A,B) = \frac{\sum_{x=0}^{n-1}\sum_{y=0}^{m-1} A[x,y] * B[x,y]}{\sqrt{\sum_{x=0}^{n-1}\sum_{y=0}^{m-1} A[x,y]^2 * \sum_{x=0}^{n-1}\sum_{y=0}^{m-1} B[x,y]^2}} \tag{4.1}$$

where $A$ and $B$ are the compared images with sizes $n$ and $m$.

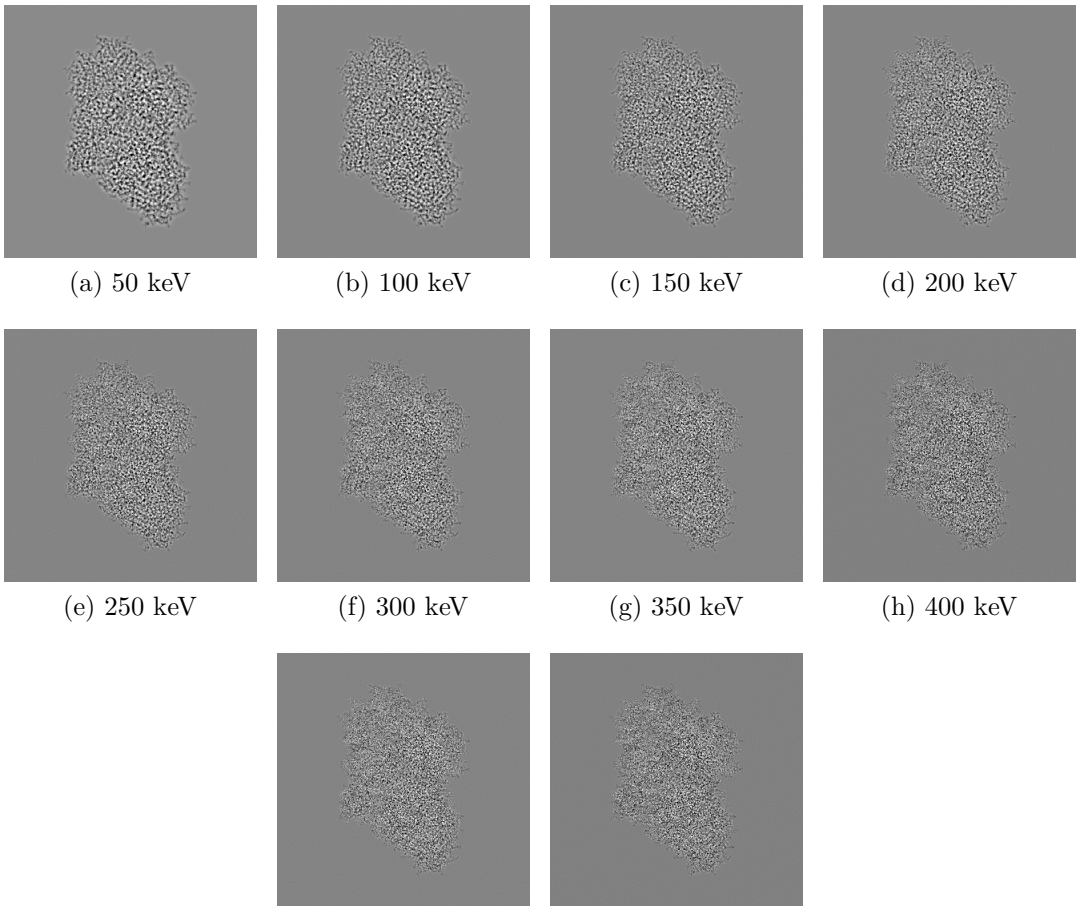
For comparison, we use both the DCS model, the Rutherford model, and the transmittance model. In the DCS model we test the influence of the radius parameter and in the Rutherford model, we modify the scattering atomic number parameter. The transmittance model is tested with one ray per pixel and with random supersampling with six rays per pixel.

We do not generate and compare the projections of the multislice model because it is difficult to set up fair conditions. The multislice approach models defocus through a contrast transfer function (CTF). However, the CTF function also models other lens properties which are not desirable in our setup, because we have CTF corrected real data averages. That means that the effect of the CTF has been suppressed and comparing CTF affected images with corrected images yields poor correlation.

Also, as described previously, the class averages generated by the RELION tool have inverted contrast. This results in a negative sign of the computed cross-correlation. In the following sections, we will compare only the absolute values of the normalized cross-correlation and ignore the sign. In real unmodified measurements, the particle projections are represented by lower values compared to the background, which is in correspondence with both the DCS and Rutherford model.

### Results

**DCS model** Figure 4.13 shows normalized cross-correlation of the DCS scattering model with each of the class averages. The figure also displays how does the localized Fourier transform radius influence the normalized cross-correlation. Three of the four class averages show similar behavior concerning the radius influence, but the behavior is not linear. Together with



(a) 50 keV (b) 100 keV (c) 150 keV (d) 200 keV

(e) 250 keV (f) 300 keV (g) 350 keV (h) 400 keV

(i) 450 keV (j) 500 keV

Figure 4.9: The figure illustrates how the simulated images differ for various electron energies. The main difference is the amount of noise in the images, which stems from different estimated mean free paths (shown in Table 4.1).

the fact that the fourth class average displays nearly inverse behavior, it is difficult to establish which radius is the best or how the correlation might progress for higher radii.

**Rutherford model** Figure 4.14 shows normalized cross-correlation of the Rutherford scattering model with each of the class averages and the effect of the scattering atomic number. There is a clear relationship between the atomic number and the normalized cross-correlation which is consistent across all classes. The figure also shows that the lowest atomic number 1 (Hydrogen) has the best fit across all classes.

**Transmittance model** With the transmittance model we tested one ray per pixel and random supersampling with six rays per pixel. The difference in the achieved cross-correlation was minimal between those two setups, with the supersampled version being always very slightly better. Thus we have used only the supersampled results and in Table 4.2 we show a comparison of the transmittance model and the other two models.

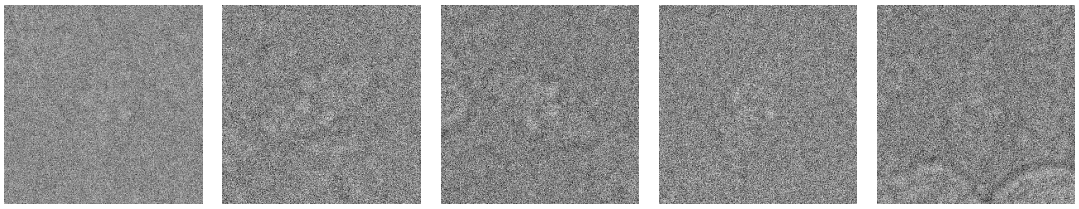
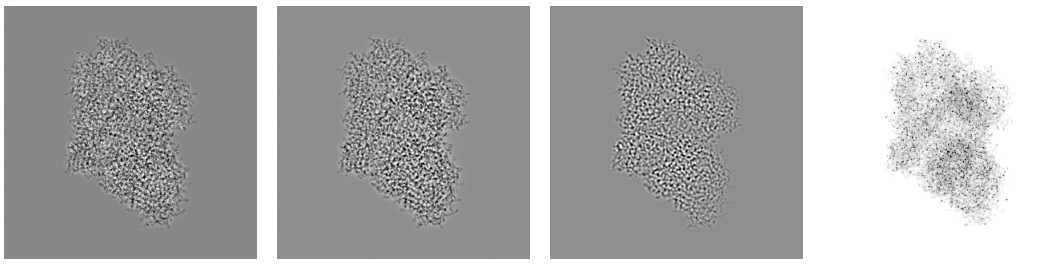


Figure 4.10: Figure shows examples of measured projections from the $\beta$-galactosidase dataset. The particle is difficult to see due to the low signal to noise ratio.

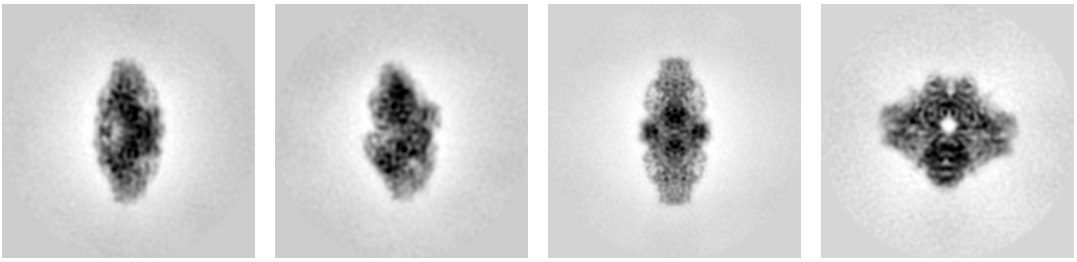


(a) DCS (b) Rutherford (c) Multislice (d) Transmittance

Figure 4.11: Figure shows the simulated projection of the molecule via different models. For the DCS model we have used the radius of 4 $a_0$. For the Rutherford model, the sampling atomic number is set to 1. The multislice projection has been generated by the Matlab simulator by Vulović [2]. The slice thickness is set to a resolution of one voxel, which is 0.7 Å. It uses a blurred potential because the Fourier transform does not handle high-frequency information well and produces sinc-like artifacts. For the transmittance model, we have used a random supersampling with six rays per pixel.

Table 4.2 shows the best cross-correlation for each model and each class. Both the DCS and Rutherford model outperform the transmittance model in the correlation with the class averages. The DCS scattering model is better than Rutherford in all but the class average 50, where the behavior of all models is distinct. We expected the DCS model to outperform the Rutherford, as it is in theory better approximation of the scattering directions. Ideally, comparison with more class averages could help to rule out the strange properties of class 50.

For illustration purposes, Figure 4.15 shows class averages and the generated averages for each class and each of the models. The positions of the images corresponds with the organization of Table 4.2.



(a) class10 (b) class38 (c) class42 (d) class50

Figure 4.12: The figure shows class averages that we used for comparison with the simulated data. There is a noticeable circle in each average, which is caused by the classification algorithm. Due to the different rotation of each projection, the area outside of the intersection of these projections is nulled. The averages are generated using the RELION software [30].

| class \ model | DCS | Rutherford | Transmittance |
|---|---|---|---|
| class10 | -0.972441 | -0.962295 | 0.922992 |
| class38 | -0.975895 | -0.963070 | 0.925164 |
| class42 | -0.975895 | -0.967548 | 0.932671 |
| class50 | -0.976039 | -0.977107 | 0.945431 |

Table 4.2: Each row of the table represents one class average and each column represents one of the models. Each entry is the best achieved normalized cross-correlation of the given model and the given class average. Clearly, DCS yields the highest correlation with the real data class averages and both DCS and Rutherford models outperform the Transmittance model.

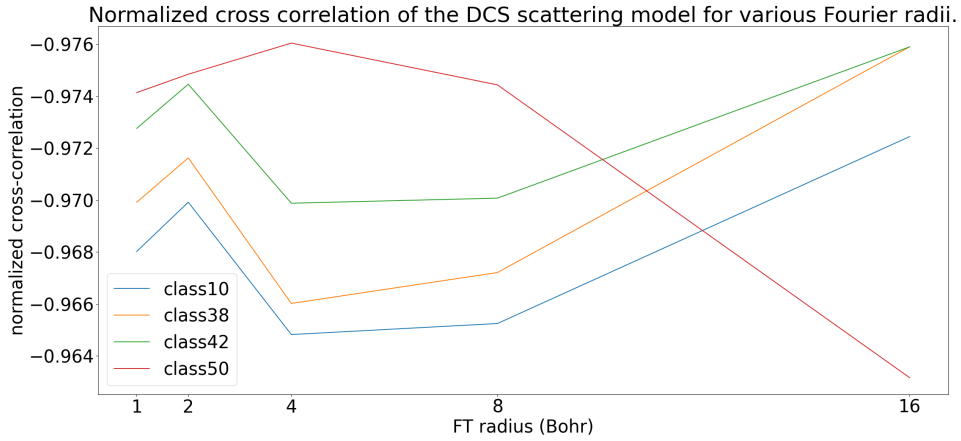



Figure 4.13: The figure shows a normalized cross-correlation of the DCS model and the class average. On the X-axis are radii used for the localized Fourier transform and each line represents correlations with one class average. The figure shows similar behavior for all class averages except for the average of class 50. It might be due to the angle of the average or higher noise levels. Also, note that the behavior is not linear - it is not clear whether any of the radii are better than others. More measurements are needed.

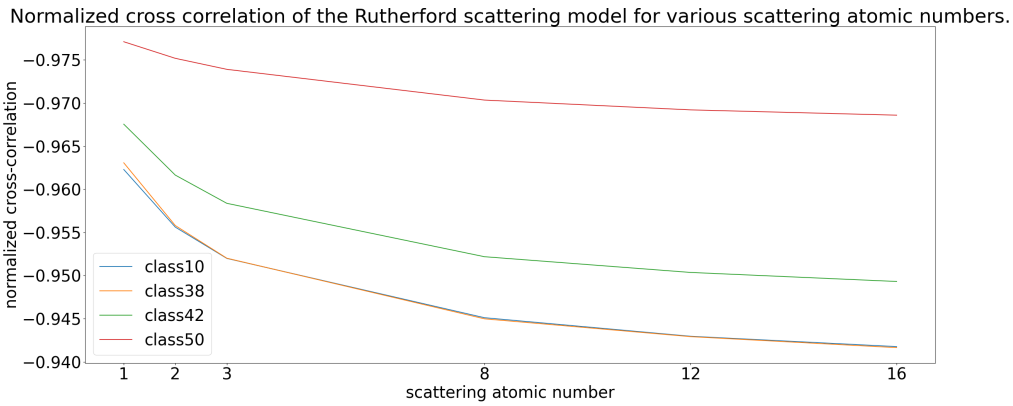
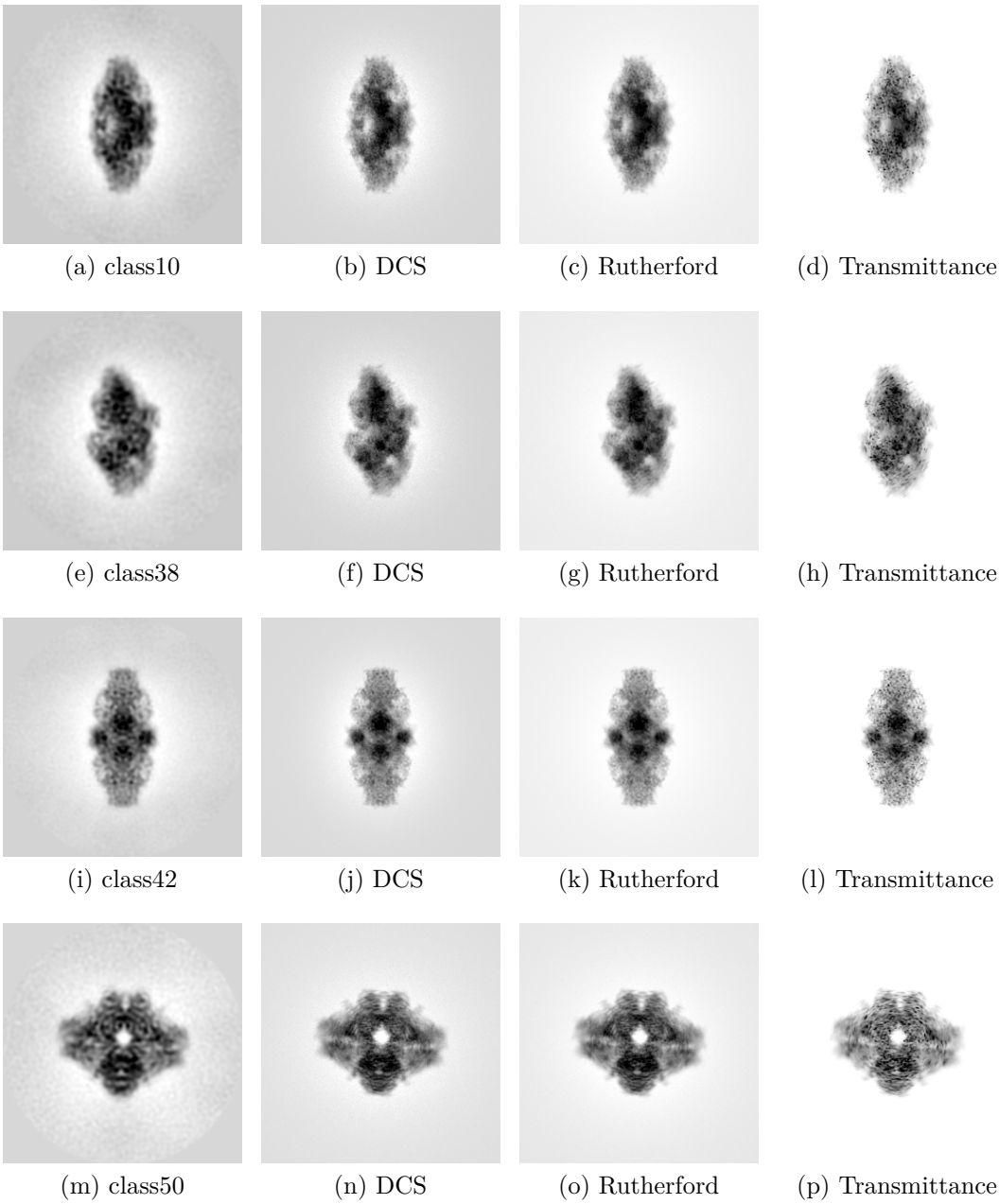



Figure 4.14: The figure shows a normalized cross-correlation of the Rutherford model and the class average. Each line represents correlations with one class average. The X-axis represents the atomic number used for the scattering of the electrons. There is a clear trend of decreasing normalized cross-correlation with increasing atomic number.



(a) class10 (b) DCS (c) Rutherford (d) Transmittance

(e) class38 (f) DCS (g) Rutherford (h) Transmittance

(i) class42 (j) DCS (k) Rutherford (l) Transmittance

(m) class50 (n) DCS (o) Rutherford (p) Transmittance

Figure 4.15: Each row of the table represents one class average and each column represents one of the models. The first column contains class averages with inverted values, to optically correspond with the real measurements. The second column is the average generated by the DCS model with the best cross-correlation with the given class average. Similarly, the third and fourth columns show the best average for the Rutherford model and transmittance model, respectively.

# Conclusion

In this thesis, we studied the applicability of a Monte Carlo particle-based simulation to electron-specimen interaction in an electron microscope, specifically with setup common in cryogenic electron microscopy single particle analysis. We started with a description of the electron microscope and its components and provided a short introduction to the field of cryo-EM SPA. We also briefly mentioned two alternative models that aim to simulate the image formation.

Then, we provided characterizations of the electron interactions with the sample and their contribution to the final image. From this, our simplified virtual microscope setup and input of the simulation were defined. We presented two scattering models of the elastic interactions, one based on the differential cross section function described by [16] and one derived from the Rutherford formulae. Next, we formulated the image formation process as a modified radiative transfer equation for heterogeneous media, a problem commonly emerging in computer graphics. To solve the RTE, we showed how to adapt a delta-tracking algorithm and create a Monte Carlo estimator for the electron transmission. To complete the estimator, we have derived an explicit scattering sampling function for the Rutherford model. We have also applied it as an importance sampling function in the DCS scattering model, which does not have explicit scattering formulation.

Extensive attention was given to the computationally demanding Fourier transform emerging in the DCS scattering model. We also considered the physical implausibility of the input data and solved it by the proposed estimation of the majorant chemical element.

The implemented simulation library is written in C++ and CUDA languages. The core of the library is a flexible ray-tracing algorithm implementation, that allowed us to implement both of the proposed scattering models and also the transmittance simulation model. The flexibility stems from the key components of the simulation: the scattering function, ray observers, and the image formation, which are all swappable for different implementations. The library builds upon the functionality of BoltView [24] and allows simple switching between CPU only and GPU accelerated program, which is very useful for debugging purposes.

We performed tests and evaluations on both simulated data and measured data from the publicly available $\beta$-galactosidase dataset. The simulations were executed on an artificially generated potential map based on atom positions from the same dataset. The low defocus projections showed good visual correspondence with the multislice model. The similarity is remarkable given the vastly different approach to the simulation. We also compared simulated class averages with the real class averages emerging in the classification step of the reconstruction algorithm. The experiments show a good visual correspondence of the simulated and measured averages. Both proposed scattering models outperform the transmittance model in terms of normalized cross-correlation with the measured class averages.

In summary, we have implemented a novel approach to the electron microscope simulation, which is based on electron tracing. The implemented model shows good results and provides insight into the image formation of an electron microscope on an intuitive particle-based level.

The fast CUDA implementation demonstrates the potential of our models to replace the transmittance model commonly used in the reconstruction process, due to their relatively low computational cost and very high cross-correlation with the measured data. One of the further research options is a thorough investigation of the localized Fourier transform radius influence in the DCS scattering model since our datasets did not show a clear pattern. The simulation can further be expanded to a full electron microscope, with all its components. Such a simulator would provide a complete particle-based insight into the noise and patterns present in the electron microscope images.

# Bibliography


[1] R. A. Crowther. *The resolution revolution: recent advances in cryoEM.* Academic Press, 2016.

[2] M. Vulović, R. B. G. Ravelli, L. J. van Vliet, A. J. Koster, I. Lazić, U. Lücken, H. Rullgård, O. Öktem, B. Rieger, I. Lazić, C. Bajaj, and A. Rand. *Modeling of Image Formation in Cryo-Electron Microscopy.* PhD thesis, Delft University of Technology, 2013.

[3] P. Dey. Electron microscopy: Principle, components, optics and specimen processing. In *Basic and Advanced Laboratory Techniques in Histopathology and Cytology*, pages 253–262. Springer, 2018.

[4] N. Marturi. *Vision and visual servoing for nanomanipulation and nanocharacterization in scanning electron microscope.* PhD thesis, Université de Franche-Comté, 2013.

[5] R. Fernandez-Leiro and S. H. W. Scheres. Unravelling biological macromolecules with cryo-electron microscopy. *Nature*, 537(7620):339–346, 2016.

[6] H. Wang. Cryo-electron microscopy for structural biology: current status and future perspectives. *Science China Life Sciences*, 58(8):750–756, 2015.

[7] A. Iudin, P. K. Korir, J. Salavert-Torres, G. J. Kleywegt, and A. Patwardhan. Empiar: a public archive for raw electron microscopy image data. *Nature methods*, 13(5):387–388, 2016.

[8] S. H. W. Scheres. A bayesian view on cryo-em structure determination. *Journal of molecular biology*, 415(2):406–418, 2012.

[9] E. Havelková. Regularization methods for discrete inverse problems in single particle analysis. Master's thesis, Charles University, Prague, 2019.

[10] M.F.L. Pereira and P. Cruvinel. A model for soil computed tomography based on volumetric reconstruction, wiener filtering and parallel processing. *Computers and Electronics in Agriculture*, 111, 2015.

[11] E. Kirkland. Image simulation in transmission electron microscopy. 2006.

[12] L. Reimer. *Transmission electron microscopy: physics of image formation and microanalysis.* Springer, 2013.

[13] M. J. Peet, R. Henderson, and C. J. Russo. The energy dependence of contrast and damage in electron cryomicroscopy of biological molecules. *Ultramicroscopy*, 203:125–131, 2019.

[14] F. Salvat, J. M. Fernández-Varea, and J. Sempau et al. Penelope-2008: A code system for monte carlo simulation of electron and photon transport. In *the Workshop Proceedings*, 2008.

[15] H. Ali. *Study of surface passivation behavior of crystalline silicon solar cells.* PhD thesis, University of Central Florida, 2017.

[16] V. Alt. Simple potential scattering and approximations. Technical report, Eyen SE, 2020.

[17] M. Hohenwarter, J. Hohenwarter, Y. Kreis, and Z. Lavicza. Teaching and learning calculus with free dynamic mathematics software geogebra. In *11th International Congress on Mathematical Education. Monterrey, Nuevo Leon, Mexico*, 2008.

[18] S. N. Ahmed. *Physics and engineering of radiation detection*. Academic Press, 2007.

[19] H. Nikjoo, S. Uehara, D. Emfietzoglou, and A. Brahme. Heavy charged particles in radiation biology and biophysics. *New journal of physics*, 10(7):075006, 2008.

[20] M. Čalkovskỳ, E. Müller, M. Hugenschmidt, and D. Gerthsen. Differential electron scattering cross-sections at low electron energies: The influence of screening parameter. *Ultramicroscopy*, 207:112843, 2019.

[21] R. Idoeta and F. Legarda. Small angle screening factors for elastic scattering of electrons. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, 83(1-2):42–46, 1993.

[22] A. Jablonski, F. Salvat, and C. J. Powell. Nist electron elastic-scattering cross-section database–version 3.1. *National Institute of Standards and Technology*, 2003.

[23] J. Novák, I. Georgiev, J. Hanika, J. Křivánek, and W. Jarosz. Monte Carlo methods for physically based volume rendering. In *ACM SIGGRAPH 2018 Courses*, New York, USA, 2018. ACM.

[24] Eyen SE. BoltView. `https://www.boltview.org/`. Accessed: 2020-11-26.

[25] Nvidia Corporation. NVIDIA OptiX™ Ray Tracing Engine. `https://developer.nvidia.com/optix`. Accessed: 2020-1-17.

[26] Nvidia Corporation. NVIDIA® GVDB Voxels. `https://developer.nvidia.com/gvdb`. Accessed: 2020-2-23.

[27] P. Mikuš. jsonParams. `https://gitlab.com/Godrak/jsonparams`. Accessed: 2020-8-23.

[28] GEMMI - library for structural biology. `https://gemmi.readthedocs.io/`. Accessed: 2020-8-23.

[29] A. Bartesaghi, A. Merk, S. Banerjee, D. Matthies, X. Wu, J. L. S. Milne, and S. Subramaniam. 2.2 Å resolution cryo-EM structure of $\beta$-galactosidase in complex with a cell-permeant inhibitor. *Science*, 348(6239):1147–1151, 2015.

[30] S. H. W. Scheres. RELION: implementation of a Bayesian approach to cryo-EM structure determination. *Journal of structural biology*, 180(3):519–530, 2012.

# List of Figures

# List of Tables

# A. Attachments

## A.1 User guide

In this section, we provide a user guide for the library and for the `test_exe` program compilation and execution.

### A.1.1 Compilation

The compilation is driven by the *CMake* tool. The `/src/CMakeLists.txt` file is present in the root folder. Following libraries are necessary to compile the executables defined in the `/src/CMakeLists.txt`:

***CMake*** *Cmake* tool in version 3.10 and higher.

***CUDA*** *CUDA* in version 10.0 and higher must be present on the machine. This also means that the program requires a *CUDA* compatible graphic card. Path to the nvcc tool must be set in the root `/src/CMakeLists.txt` file.

***cuRAND* and *cuFFT*** Additional *CUDA* APIs.

**Boost** *Boost* components `system`, `filesystem` and `program_options` in version 1.53 and higher are required.

### A.1.2 Execution

The compilation step will produce several executable files. Here we describe only the `mrcBuild` and `test_exe` executables.

#### `mrcBuild`

The `mrcBuild` executable accepts two arguments - a PDB file with the molecular description and voxel size in Å. If the voxel size is not provided, the executable will print a chemical distribution of the molecule - the percentual amount of each chemical element in the PDB file. If the voxel size is provided, the executable generates a potential map of the given PDB file and saves it in the MRC file format in the current work directory. Sensible voxel size is larger than 0.7 Å, recommended is 1.5 Å.

#### `test_exe`

The `test_exe` executable accepts one argument - name of a JSON file with execution parameters. The file looks like this:

```
1 {
2   "ems_projectionsFile" : "/data/Particles/particles.
      mrcs",
3   "ems_projectionsStarFile" : "/data/particles.star",
4
5   "ems_dose": 10000,
```

```
6   "ems_electronEnergy" : 300,
7   "ems_randomizedElectronEmission" : 1,
8   "ems_applyDefocus": true,
9
10  "ems_samplingAtomicNumber":16,
11  "ems_majorantAtomicNumber": 16,
12  "ems_majorantPercentile": 0.9952,
13
14  "ems_emitterPixelSize":[1264,1264],
15  "ems_detectorPixelSize":[316,316],
16  "ems_samples":50,
17  "ems_directions":1,
18
19  "scatterEL_crossSectionFrameRadius": 2,
20  "scatterEL_maxSamplesPerAxis": 15,
21
22  "ems_renderType":"rele",
23
24  "crossSection_FrameRadius": 2,
25  "crossSection_maxSamplesPerAxis" : 16,
26  "crossSection_sampleCount" : 50,
27
28  "ems_potentialFile" : "/data/5a1a_potential.mrc",
29  "ems_scatteringFile" : "",
30  "crossSection_crossSectionsFile" : "",
31  "crossSection_dumpPath" : "/data/5a1a_crossSections.
      mrc"
32  }
```

**ems_projectionsFile** contains path to the file with measured projections. The program uses it to estimate the projections size, if `ems_detectorPixelSize` is not set.

**ems_projectionsStarFile** describes the content of the previous file. It contains metadata of each measured projection, including their defocus and estimated angle. The simulated projections are done from the same angles as defined in this file.

**ems_dose** is dose of the electrons per Bohr squared. A value of 12 is realistic but tens of thousands are necessary for noise-free images.

**ems_randomizedElectronEmission** sets the origin of the electron within the emitter pixel - 0 means always in center, 1 means random position within the pixel and 2 means random position anywhere on the emitter.

**ems_applyDefocus** is flag whether to apply a defocus from the projections star file metadata.

**ems_samplingAtomicNumber** is atomic number for the Rutherford formula. For `sruther` render type, it sets the scattering atomic number of the Rutherford

model. For `rele`, it sets the importance sampling atomic number of the DCS scattering model.

**`ems_majorantAtomicNumber`** is an atomic number that corresponds to the largest potential value in the volume. In our approximation, it is the element with the highest atomic number present in the molecule.

**`ems_majorantPercentile`** describes which value of the potential should be considered as majorant. If there is a large noise peak in the volume, it is beneficial to set this number to lower than 1.

**`ems_emitterPixelSize`** sets the size of the emitter image. Since each emitter pixel is executed in parallel, this parameter can influence the computational performance of the execution. Multiples of the `ems_detectorPixelSize` are recommended for aliasing reasons.

**`ems_detectorPixelSize`** sets the size of the detector and in turn of the result projections. Also determines the pixel size of the detector, as the physical size of the detector always matches the size of the potential volume. If not provided, the value is parsed from the provided projections file.

**`ems_samples`** sets how many projections of the current metadata line from the projections metadata file should be generated.

**`ems_directions`** sets how many directions from the projection star file should be generated. The final amount of projections is `ems_samples`*`ems_directions`.

**`scatterEL_crossSectionFrameRadius`** sets localized Fourier transform radius for the DCS scattering model.

**`scatterEL_maxSamplesPerAxis`** sets how many samples per axis are done in the localized Fourier transform. At least eight times the value of `scatterEL_crossSectionFrameRadius` is recommended, otherwise viable directions are cut off due to the Nyquist.

**`ems_renderType`** sets in which mode does the simulation operate, three options are possible: `rele` uses the DCS scattering model, `sruther` uses the Rutherford scattering model and `int` uses the transmittance model.

**`crossSection_FrameRadius`** sets the localized Fourier transform radius during the computation of the total cross section in each voxel. Should be same as `scatterEL_crossSectionFrameRadius`.

**`crossSection_maxSamplesPerAxis`** sets how many samples per axis are done in the localized Fourier transform during the computation of the total cross section in each voxel. Should be same as `scatterEL_maxSamplesPerAxis`.

**`crossSection_sampleCount`** sets how many samples are done in each voxel for the Monte Carlo estimation of the total cross-section.

**`ems_potentialFile`** is path to the input potential file.

**ems_scatteringFile** is a path to the file with the scattering coefficients. If not provided, it is computed from the potential file.

**crossSection_crossSectionsFile** is path to the file with the precomputed total cross sections in each voxel. If not provided, the total cross sections are computed from the potential file.

**crossSection_dumpPath** is a path to which the volume with the computed total cross-section is saved if they were computed. It can save time, as computing the total cross sections is slow and unnecessary for each execution.