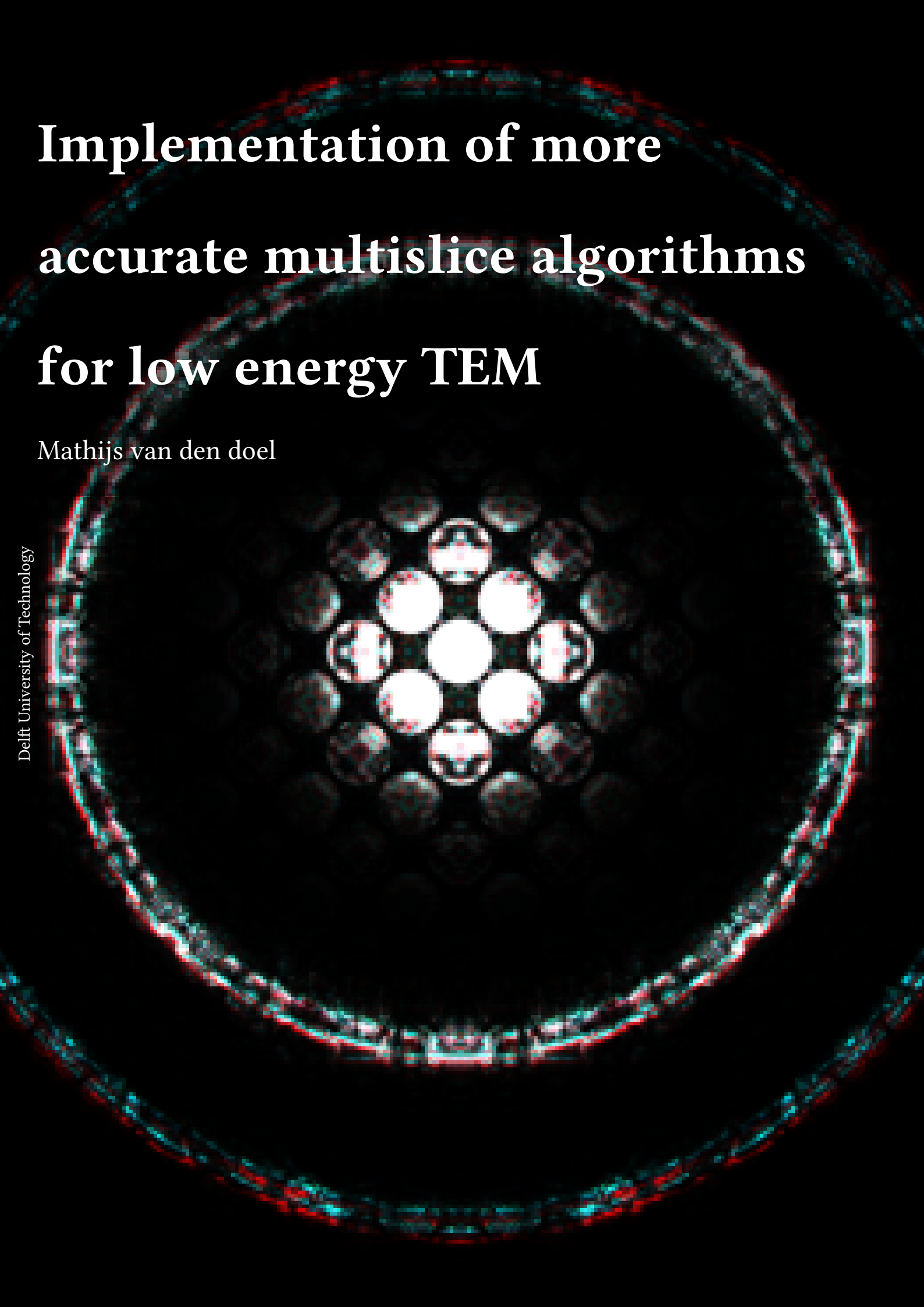
Implementation of more accurate multislice algorithms for low energy TEM

Mathijs van den doel



Delft University of Technology

Implementation of more accurate multislice algorithms for low energy TEM

Master Thesis

Bachelor of Science
Applied Physics
Delft University of Technology
Faculty of Applied Science

Author	Mathijs van den doel
Supervisor	Dr. Georgios Varnavides *Delft University of Technology*
Committee Members	Prof. Dr. Jacob Hoogenboom *Delft University of Technology* Dr. Ali Gheidari *Delft University of Technology*
Defense date	Friday December 19, 2025
Date	11-3-2026

Abstract

Accurate interpretation of data on the atomic scale obtained by S/TEM requires robust simulation methods. The most common method to achieve this is the multislice. The conventional multislice (CMS) algorithm relies on scattering physics assumptions which break down at lower accelerating voltages. This thesis presents an implementation of two improved versions of the multislice algorithm into the TEM simulation package abTEM written in Python. These are the Propagator Corrected Multislice (PCMS) and the Fully Corrected Multislice (FCMS). The former uses higher order terms to more accurately describe the spherical curvature of the Ewald Sphere, contrary to the parabolic approximation used in CMS. The Fully Corrected MS starts from the same Schrödinger equation as the other methods but makes fewer approximations concerning electron energy and simultaneously accounts for backscattering effects. All three methods are compared for a sample of SrTiO3 for various thicknesses and electron energies. The obtained results agree with previous literature, highlighting that the effects are negligible for high electron energies (~200 keV) and start to appear around (~50 keV). The Propagator corrected and Fully corrected MS agree until the energy drops approximately below 10 keV, where they begin to diverge. Additionally, the reconstructed backscattered signal follows the expected theoretical dependence on the square of the electron wavelength, showing significantly weaker signals at higher energies.

Keywords

Jupyter Book, TU Delft, Bachelor eindproject, BEP, Bachelor thesis

Contents

Introduction

Transmission electron microscopy (TEM) is a powerful method for imaging the atomic scale D. B. Williams and C. B. Carter [1], leveraging the much smaller wavelength of electrons compared to ordinary light, where diffraction causes a limit to resolution.

Computer simulation of the scattering process can be a powerful tool to interpret the complex scattering patterns of the electron beam. To simulate the complex interactions of the electrons with the sample, various techniques exist. This paper will focus on the multislice algorithm J. M. Cowley and A. F. Moodie [2], which works by dividing potential of the sample into many discrete slices and calculating the scattering for each slice, followed by a propagation through vacuum. The multislice algorithm, in its conventional and most common form, works by assuming that the electrons move very fast. While this is a valid assumption to make since commonly, high accelerating voltages are used for TEM, there are situations where lower voltages are required. For example if the sample is radiation sensitive and higher energy beams would damage the sample [3], [4]. Another reason for simulating with lower voltages is the use of STEM in SEM, where a TEM measurement is performed inside a SEM which typically operates at a lower voltage C. Sun, E. Müller, M. Meffert, and D. Gerthsen [5]. These situations have sparked an increased interest in low voltage TEM. Therefore, being able to accurately model TEM even for these lower voltages is of great importance.

In order to accurately simulate this, better models are required. Two recently-proposed extensions to the multislice method will be implemented into python and added to the python package abTEM: the propagator corrected multislice (PCMS) and the fully corrected multislice (FCMS) W. Q. Ming and J. H. Chen [6]. abTEM is an open-source software written in Python which makes performing multislice simulations very easy. The improved methods will be benchmarked compared to the conventional multislice (CMS).

CMS, as it exists in abTEM, assumes that all electron scattering happens in the forward direction, meaning every electron entering the sample will exit it on the other side. In reality however, there is a chance of the electrons scatter- ing back to the electron source. This thesis adds this functionality to abTEM as well as the ability to reconstruct the total backscattered wave which might be valuable to SEM simulation P. Schweizer, P. Denninger, C. Dolle, and E. Spiecker [7].

Theory

Transmission electron microscopy

Transmission electron microscopy (TEM) is a wave-based microscopy technique, similar to light microscopy, where the illumination by photons is replaced by electrons. These electrons, like photons, have a certain wavelength derived from the de Broglie wavelength, which states that the wavelength of a particle is inversely proportional to its momentum.

$$\lambda = \frac{h}{\gamma m_0 v}$$

Therefore, the wavelength of electrons can be orders of magnitude smaller than their photonic counterpart. When the object of interest approaches the scale of the wavelength with which it is imaged, diffraction causes the individual points to blend together. Thus with a smaller wavelength, a higher resolution can be achieved when imaging with electrons.

A TEM consists of a source of free moving electrons followed by a series of electromagnetic lenses which focus the beam on the specimen D. B. Williams and C. B. Carter [1]. These electrons are accelerated using an electrostatic potential known as the accelerating voltage. The beam passes through an aperture which controls the maximum angle in the beam as well as the intensity of the electron current reaching the detector.

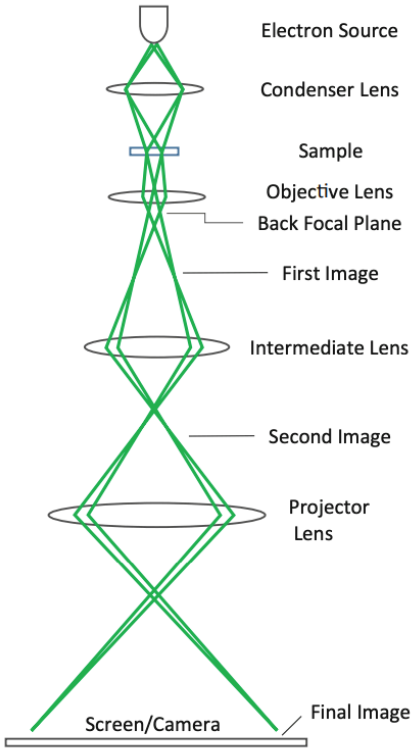


Figure 1: Diagram of a basic TEM J. M. Zuo and J. C. H. Spence [8]

This beam of electrons then passes through the sample where due to its potential the electrons undergo scattering. This scattering thus provides information of the electrostatic potential of the sample which

is ultimately the quantity of interest. The scattered electrons, after leaving the sample, are either magnified to form an image or, for all instances in this thesis, detected by a detector in the far field. This separates the electrons based on their wavevector generating a diffraction pattern. The interpretation of this diffraction pattern is where the simulation of this process is very useful since it is then possible to simulate what caused the patterns with various optical parameters.

Ewald spheres

The sample of interest is often a crystal with atoms arranged in a pattern which is periodic in real space. To understand the scattering effects of the electrons, it is useful to view the lattice in reciprocal space. Because the realspace crystal lattice is periodic, when we calculate its potential in realspace, it can be represented by a Fourier series. The different wavevectors making up the Fourier series can be plotted in reciprocal space. Assuming that the scattering happens elastically, the energy is preserved and thus also the wavelength. This means that for any scattered electron, its wavelength must be the same as that of the incident electron. Since the electron can scatter to any direction but must keep its wavelength, in reciprocal space this translates to a sphere with radius λ^{-1} whose surface represents all possible scattering directions. This sphere is called the Ewald sphere. In order for a electron to be detected after the sample, the contributions from all atoms must interfere constructively. This means that the contributions from different atoms must have a phase difference of any integer multiple of 2π. By way of Braggs law, only scattering directions with a change in wavevector equal to one of the reciprocal lattice points can meet this requirement, assuming the origin is the unscattered beam. Combining the two criteria: only lattice points in reciprocal space that land on the Ewald sphere are valid scattering direction.

Conventional multislice

The conventional multislice (CMS) algorithm is a method to simulate the scattering effects of the electron beam by the sample. To see how the multislice method works we begin by stating the problem the multislice algorithm tries to solve. The incoming electron beam can be described by its wavefunction: $\Psi(\boldsymbol{r})$, whose absolute square tells you the probability of finding an electron when measured. When this wavefunction enters the sample, it obeys the so-called time-independent, relativistically corrected Schrödinger equation.

$$\left[\frac{h^2}{8\pi^2 m}\nabla_r^2 + eU(\boldsymbol{r}) + \frac{h^2 K_0^2}{2m}\right]\Psi(\boldsymbol{r}) = 0$$

Because the CMS assumes a high velocity along the optical axis, we can write the wavefunction as:

$$\Psi(\boldsymbol{R}, z) = \phi(\boldsymbol{R}, z)e^{2\pi i K_0 z}$$

Here we have also separated the z from the full position vector $\boldsymbol{r}$. Substituting this into Eq.(2) and dropping the second derivative compared to z because it is assumed to be a slow moving envelope, as demonstrated by W. Q. Ming and J. H. Chen [6], yields:

$$\frac{\partial \phi(\boldsymbol{R}, z)}{\partial z} = i\left[\frac{\Delta}{4\pi K_0} + \sigma U(\boldsymbol{R}, z)\right]\phi(\boldsymbol{R}, z)$$

The CMS begins by dividing the sample potential into N different slices, where each slice is defined by J. Chen and D. Van Dyck [9]:

$$U_j(\boldsymbol{R}) = \frac{1}{\varepsilon}\int_{(j-1)\varepsilon}^{j\varepsilon} U(\boldsymbol{r})\,\mathrm{d}z$$

This new potential per slice can be substituted into Eq.(2) and solving this differential equation gives the solution

$$\phi_j(\boldsymbol{R}) = \exp\left[i\varepsilon\left(\frac{\Delta}{4\pi K_0} + \sigma U_j(\boldsymbol{R})\right)\right]\phi_{j-1}(\boldsymbol{R})$$

Using this formula, the wavefunction after each slice can be calculated with the wavefunction coming into that slice. Thus starting with the beam wavefunction and recursively calculating the next wavefunction for each sample slice, we end up with the final electron wavefunction, known as the exitwave.

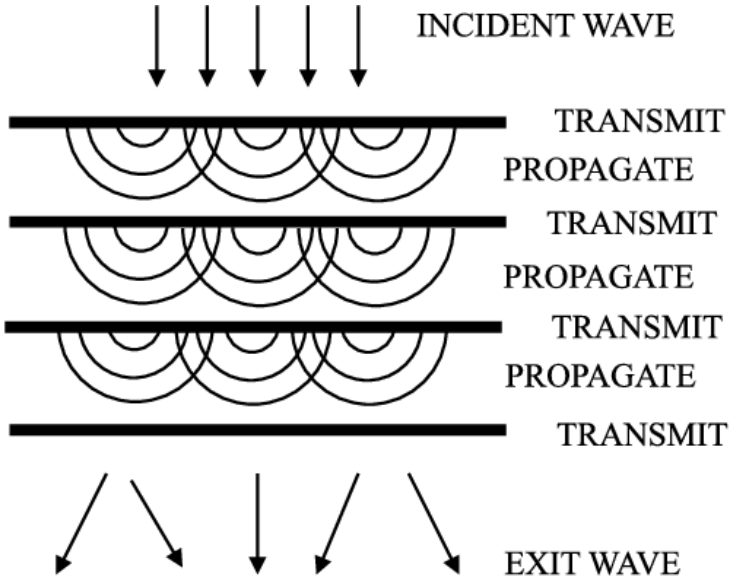


Figure 2: Diagram showing the working of the multislice algorithm D. A. Muller, E. J. Kirkland, M. G. Thomas, J. L. Grazul, L. Fitting, and M. Weyland [10]

Numerically, the multislice algorithm can be calculated both in realspace and using a Fourier transform. The latter is the more common approach, since using the fast Fourier transform (FFT) will result in a much faster calculation. If the slice thickness is sufficiently small, Eq.(6) can be rewritten as

$$\phi_j(\boldsymbol{R}) = \exp\left(\frac{i\varepsilon\Delta}{4\pi K_0}\right)\exp\left[i\varepsilon\sigma U_j(\boldsymbol{R})\right]\phi_{j-1}(\boldsymbol{R})$$

The first operator in the exponent is called the Fresnel propagator, while the second is the transmission operator. If we now apply the transmission operator in realspace by multiplying it with the incident

wave function, we are left with the transmitted wave. Then the Fourier transform is taken of this transmitted wave and multiplied by the fourier transform of the Fresnel propagator which turns into a simple multiplication.

$$\mathcal{F}\left[\exp\left(\frac{i\varepsilon\Delta}{4\pi K_0}\right)\right] = \exp\left[-\frac{i\pi\varepsilon k^2}{K_0}\right]$$

After that, the inverse Fourier transform is taken to obtain the final wave.

$$\phi_j(\boldsymbol{R}) = \mathcal{F}^{-1}\left\{\exp\left[-\frac{i\pi\varepsilon k^2}{K_0}\right]\mathcal{F}\left[\exp\big(i\varepsilon\sigma U_j(\boldsymbol{R})\big)\phi_{j-1}(\boldsymbol{R})\right]\right\}$$

Python implementation

To implement Eq.(6) numerically we start with a discretized illumination wavefunction which is a 2D complex array as well as a 3D potential array. The potential array is already a potential for each slice calculated using Eq.(5). This is handled by the abTEM J. Madsen and T. Susi [11] library with a Waves object and Potential object respectively which handles the array and the metadata like energy and wavelength and also provided a range of functions like detection with detectors and easily scanning a range of positions. Because in Eq.(6) the 2D laplacian is an operator acting on the wavefunction is in the exponent, we need to use the Taylor expansion of the exponent.

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

To get the power n of the multislice operator, we simply apply it n times. By using this method it is possible for the series to diverge R. Zekendorf [12] depending on slice thickness and sampling size. The code for the conventional multislice operator can be found in Code. 1
Then we can use the exponential series to calculate the exponent of the conventional operator applied to the incident wavefunction Code. 2.

Propagator Corrected MS

Theoretical Framework

The CMS approximates the Ewald sphere as a parabola. Therefore, it is only accurate for small scattering angles which is not a good approximation when the accelerating voltage is lower and the electrons scatter at higher angles. An improved algorithm which accounts for this, a modification of Eq.(6), was recently proposed W. Q. Ming and J. H. Chen [6].

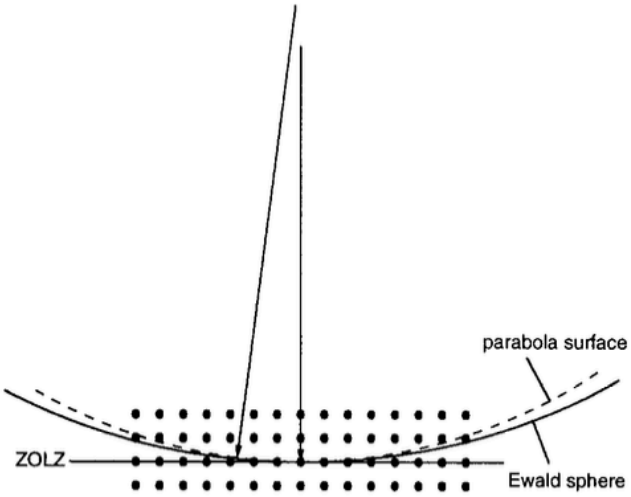


Figure 3: Approximation of the Ewald sphere by a parabola J. Chen and D. Van Dyck [9]

$$\phi_j(\boldsymbol{R}) = \exp\left[i\varepsilon\left(2\pi K_0\left[\sqrt{1+\frac{\Delta}{(2\pi K_0)^2}}-1\right]+\sigma U_j(\boldsymbol{R})\right)\right]\phi_{j-1}(\boldsymbol{R})$$

We end up with the laplace operator inside a square root which is tricky to implement numerically. Therefore, we again take the Taylor expanded form:

$$\phi_j(\boldsymbol{R}) = \exp\left[i\varepsilon\left\{\left[\frac{\Delta}{4\pi K_0}+\sigma U_j(\boldsymbol{R})\right]-\frac{1}{4\pi K_0}\left(\frac{\Delta}{4\pi K_0}\right)^2+\frac{1}{8\pi^2 K_0^2}\left(\frac{\Delta}{4\pi K_0}\right)^3-\ldots\right\}\right]\phi_{j-1}(\boldsymbol{R})$$

Here we find the CMS operator inside the square brackets in addition to a series of higher order terms to correct for the Ewald sphere curvature.

Python Implementation

Eq.(2) can be implemented numerically by utilizing the same method used for the exponent series where the higher powers of the propagator part can be calculated by applying the operator multiple times. After adding the right prefactor and taking the exponent we end up with correct result Code. 3. To calculate the exponent of this series we use the same exponential as in the previous chapter Code. 2.

Results

In order to compare the propagator corrected multislice as fairly as possible to the conventional multislice, we will perform the conventional multislice calculations in realspace. Keeping the calculations in realspace creates some artifacts compared to the fourier version, especially at low voltages (see

Figure 1), due the approximation of the laplacian by a finite difference stencil. However, since the fully corrected multislice can only be implemented in realspace, we decided to calculate the CMS in realspace as well. For the simulation a sampling interval of 0.1 Å x 0.1 Å was used and a slice thickness of 0.05 Å.

The two methods (realspace MS and PCMS) are applied to a crystal of SrTiO3 [13] with size of (1x1x24) unit cells and (1x1x48) unit cells for planewaves of different energies. The propagator corrected operator is chosen to include up to the third power correction term. To speed up calculations, the existing laplace stencil provided by abTEM was adapted to be able to utilize the GPU. All calculations were performed on a NVIDIA A40 GPU Department of Imaging Physics [14].

After calculating the exitwave we apply a PixelatedDetector provided by abTEM which gives us the intensity of the diffraction patterns. Because most electrons pass by the sample without scattering, the central pixel corresponding to the unscattered beam is very bright compared to the diffracted rays. For that reason in the next plot the central pixel is set to 0. To even further enhance visible detail the power of 0.25 is applied to the image.

24 unit cells z thickness

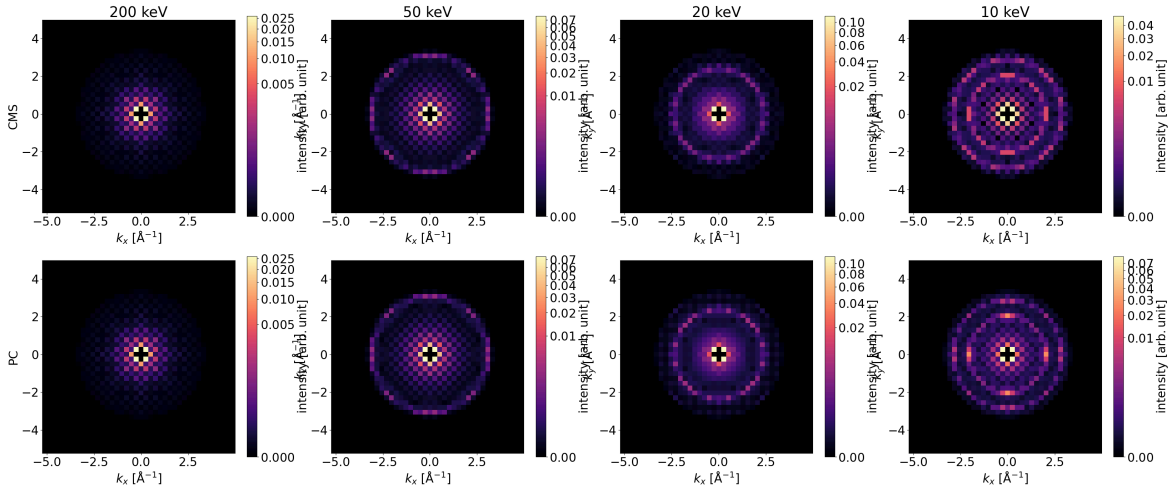


Figure 4: Diffraction patterns for $SrTiO_3$ illuminated by planewave for different energies with 24 unitcells thickness in z axis and power=0.25

48 unit cells z thickness

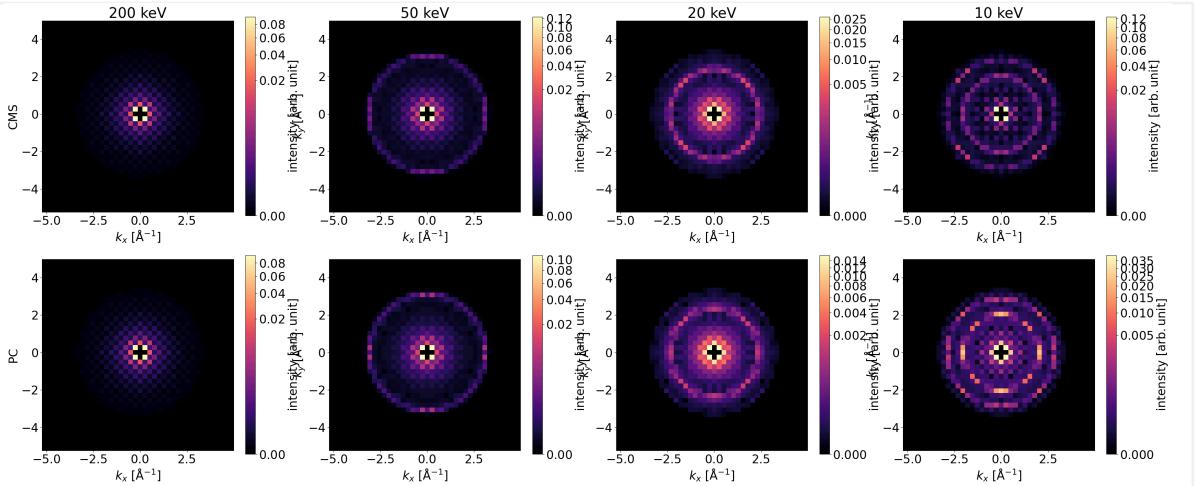


Figure 5: Diffraction patterns for $SrTiO_3$ illuminated by planewave for different energies with 48 unitcells thickness in z axis and power=0.25

In the next plots we are comparing the difference between the two methods. Instead of blocking the direct beam and taking a power we calculate the lower and upper 3% quantiles of the two upper rows combined as well as for the difference plot and clip the image using those values.
The result is the same dynamic range for all images allowing for a greater comparison. In the lowest row, the two plots are subtracted and again clipped by the 3% quantiles of all subtracted plots combined.

24 unit cells z thickness

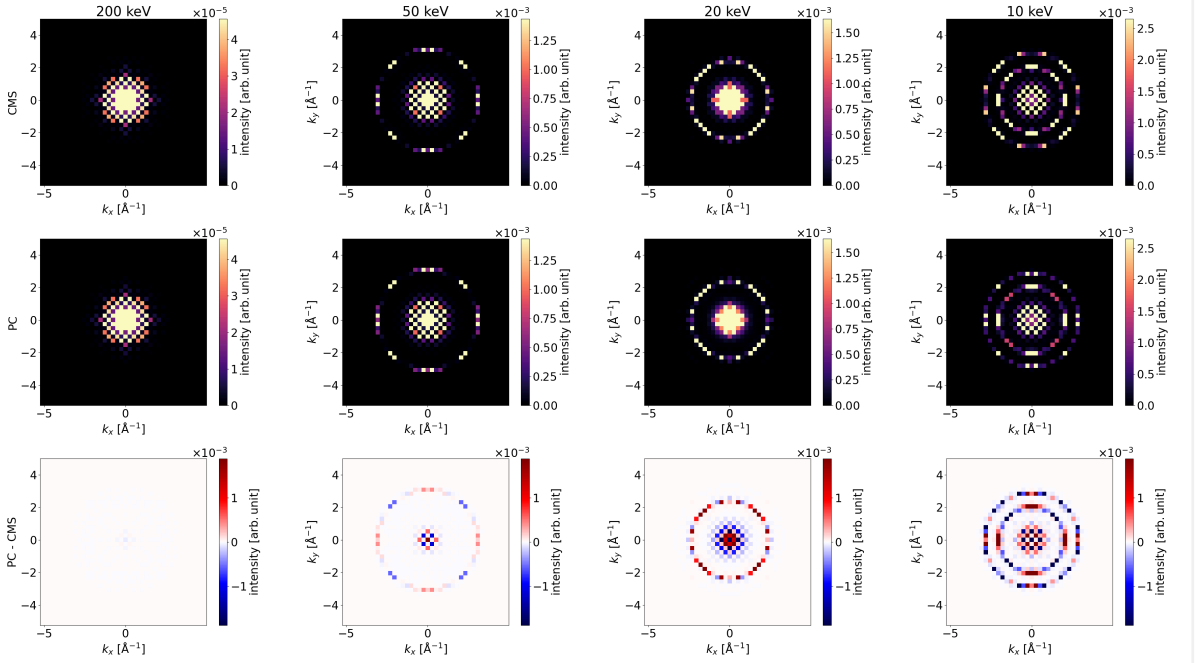


Figure 6: Comparison of the conventional multislice (CMS) and the propagator corrected multislice (PCMS) for $SrTiO_3$ with 24 unitcells thickness in z axis

48 unit cells z thickness

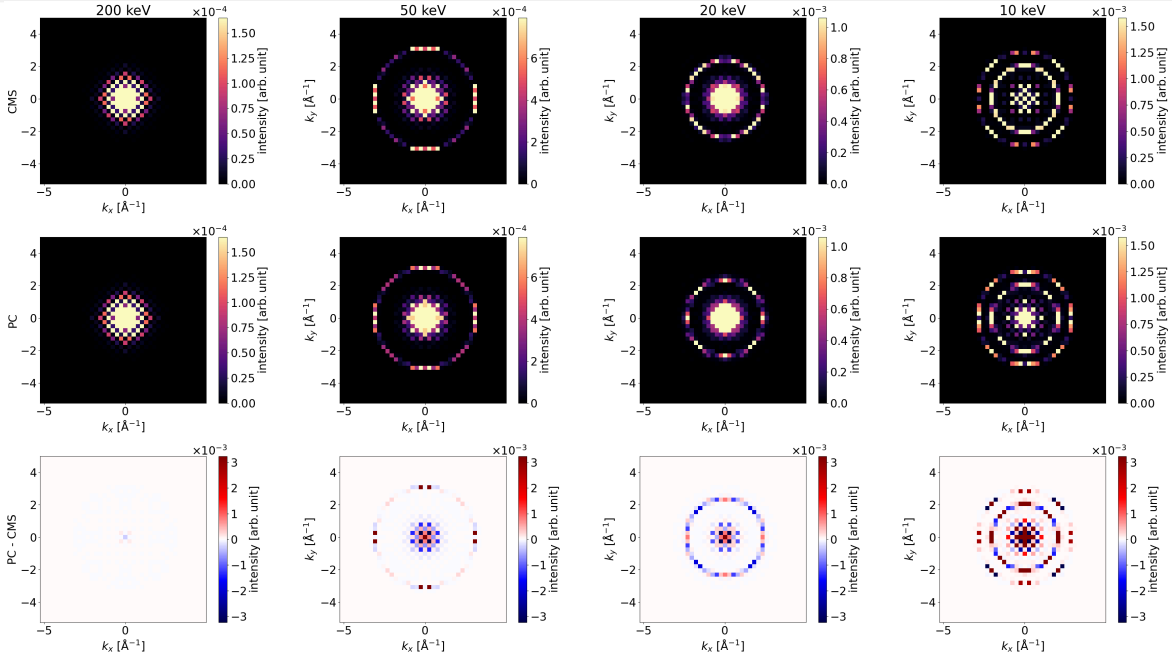


Figure 7: Comparison of the conventional multislice (CMS) and the propagator corrected multislice (PCMS) for $SrTiO_3$ with 48 unitcells thickness in z axis

Fully Corrected MS

Theoretical Framework

Improved Multislice

Starting from Eq.∼\ref{time-independent relativistically corrected Schrödinger equation}, one can come to the following version of the multislice equation, as demonstrated by∼\citet{Ming2013}

$$\phi_j(\boldsymbol{R}) = \exp\left[2\pi i\varepsilon K_0\left[\sqrt{1+\frac{\Delta}{(2\pi K_0)^2}+\frac{\sigma}{\pi K_0}U_j(\boldsymbol{R})}-1\right]\right]\phi_{j-1}(\boldsymbol{R})$$

Again we solve the inconvenience of the operator inside the square root by taylor expanding it which results in the following solution to the fully corrected multislice.

$$\phi_j(\boldsymbol{R}) = \exp\left[i\varepsilon\left\{\left[\frac{\Delta}{4\pi K_0}+\sigma U_j(\boldsymbol{R})\right]-\frac{1}{4\pi K_0}\left[\frac{\Delta}{4\pi K_0}+\sigma U_j(\boldsymbol{R})\right]^2+\frac{1}{8\pi^2K_0^2}\left[\frac{\Delta}{4\pi K_0}+\sigma U_j(\boldsymbol{R})\right]^3-\ldots\right\}\right]\phi_{j-1}(\boldsymbol{R})$$

This formula is a power series of the part inside the exponent of the original CMS and for that reason we can calculate the nth power by applying the the same function used to calculate the inside of the exponent n times.

Backscattering

Besides the improvement in accuracy achieved by Eq.(1) we can also simulate the backscattering effect where the electrons have a small chance of backscattering of the sample. We define a wave vector operator G. Chen, Y. He, W. Ming, C. Wu, D. Van Dyck, and J. Chen [15]

$$\hat{\boldsymbol{k}}_j(\boldsymbol{R}) = \pm K_0\sqrt{1+\frac{1}{(2\pi K_0)^2}\Delta+\frac{\sigma}{\pi K_0}U_j(\boldsymbol{R})}$$

Using this operator we define a backscattering operator

$$\hat{\boldsymbol{B}}_{j+1,\ \ j} = \frac{\hat{\boldsymbol{k}}_{j+1}-\hat{\boldsymbol{k}}_j}{2\hat{\boldsymbol{k}}_{j+1}} \approx \frac{\sigma}{4\pi K_0}(U_j-U_{j-1})$$

With

$$\hat{\boldsymbol{k}}(\boldsymbol{R},j\varepsilon) = K_0\left\{1+\frac{1}{2\pi K_0}\left[\frac{\Delta}{4\pi K_0}+\sigma U(\boldsymbol{R},j\varepsilon)\right]-\frac{1}{8(\pi K_0)^2}\left[\frac{\Delta}{4\pi K_0}+\sigma U(\boldsymbol{R},j\varepsilon)\right]^2+\ldots\right\}$$

And

$$\frac{1}{\hat{\boldsymbol{k}}(\boldsymbol{R},j\varepsilon)} = \frac{1}{K_0}\left\{1-\frac{1}{2\pi K_0}\left[\frac{\Delta}{4\pi K_0}+\sigma U(\boldsymbol{R},j\varepsilon)\right]+\frac{3}{8(\pi K_0)^2}\left[\frac{\Delta}{4\pi K_0}+\sigma U(\boldsymbol{R},j\varepsilon)\right]^2+\ldots\right\}$$

Applying this operator to the wavefunction for each slice gives the backscattered wave. In general, the amplitude of this backscattered wave is much smaller than the forward scattered wave. This is dependent on the atomic mass of the atoms inside the sample. By looking at the approximation of (4) we can also see a dependence on the wavelength and thus the energy of the electron ($\sigma/K_0 \propto \lambda^2$). We can further improve the forward scattering accuracy of the FCMS by subtracting the backscattered wave from the forward scattered wave.

If we additionally store the backscattered part of the wave at each slice, we can reconstruct the total backscattered wave that would be detected by an upstream detector (i.e. in reflection, without disrupting the incident beam). The total backscattered wave can be reconstructed from the final exitwave J. Chen and D. Van Dyck [9]

$$\phi_{backscattered}(\boldsymbol{R}) = -\sum_{m=1}^{n}\left(\prod_{j=m}^{1} \mathrm{e}^{2\pi \mathrm{i}\hat{k}_j \varepsilon}\right)\left(\hat{\boldsymbol{B}}_{m+1,m}\mathrm{e}^{2\pi \mathrm{i}\hat{k}_m \varepsilon}\right)\phi_N(\boldsymbol{R})$$

Because the backscattered waves are complex valued, this formula assumes perfect coherence allowing the backscattered waves to interfere with each other. In reality however, this is often not the case and instead the waves are incoherent or partially coherent.

Python Implementation

Improved Multislice

As described earlier, we can calculate the powers of the conventional multislice step inside the FCMS taylor series by applying the CMS step multiple times. Summing these powers until a chosen order n with the correct prefactor gives us Eq.(2). After which we again use the exponential series to calculate its exponent Code. 4.

Because Eq.(5) and Eq.(6) also use the fully corrected multislice power series we can reuse it, except for the prefactor in Eq.∼\ref{khat_operator_reciprocal_taylor_expansion} being slightly different. For this reason, we introduce the ability to override the prefactor.

Backscattering

To calculate the total backscattered wave, we need to calculate and store the backscattered wave from each slice which would make up the total wave and multislice each part through the rest of the sample. Calculating the backscattered wave at each slice is done by Code. 5 For storing the backscattered wave at each slice, abTEM provides a functionality we exploit called exit planes which allows the specification of indices where the wavefunction at that slice index is stored in an array. This is very useful to see the forming of the exit wave inside the sample. But, by overriding these exit planes with our backscattered wave at each slice, we can easily calculate the total backscattered wavefunction without heavy modifications to the existing codebase.

Because we store a backscattered wave at each slice and not the operator and since the multislice operator is linear, we can use Eq.(7) in a more efficient way. Starting at the last slice ($j = N$), we take the part of the wave that backscattered at that slice and apply the multislice operator once to take it to slice $N - 1$. There we add the backscattered wave coming from that slice and apply the multislice operator again on this new total wave. This process is repeated for the entire specimen until we reach the entrance surface Code. 6. In the case of fully incoherent waves, the individual backscattered waves

from each slice should be summed by their intensities. For that reason this method does not work since we sum as we go through the sample. This is identical to using Eq. (7). Finally, we compute the farfield intensities of this total wave, as it would reach a detector.

Results

The same sample and parameters were used as in Propagator Corrected MS. Therefore, the results are presented the same way as that chapter. Also like in Propagator Corrected MS the order of the FCMS operator goes up to the third power.

24 unit cells z thickness

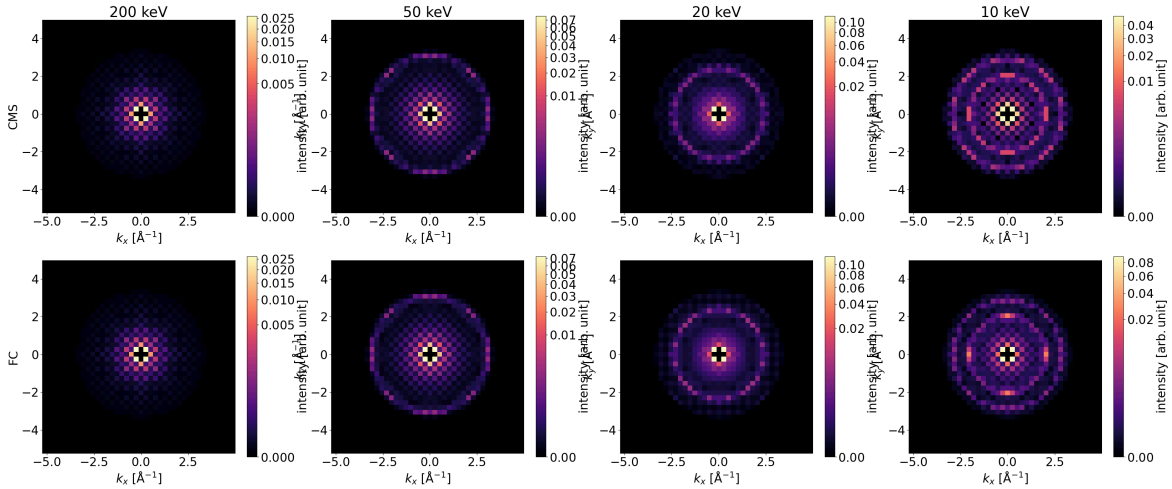


Figure 8: Diffraction patterns for $SrTiO_3$ illuminated by planewave for different energies with 24 unitcells thickness in z axis and power=0.25

48 unit cells z thickness

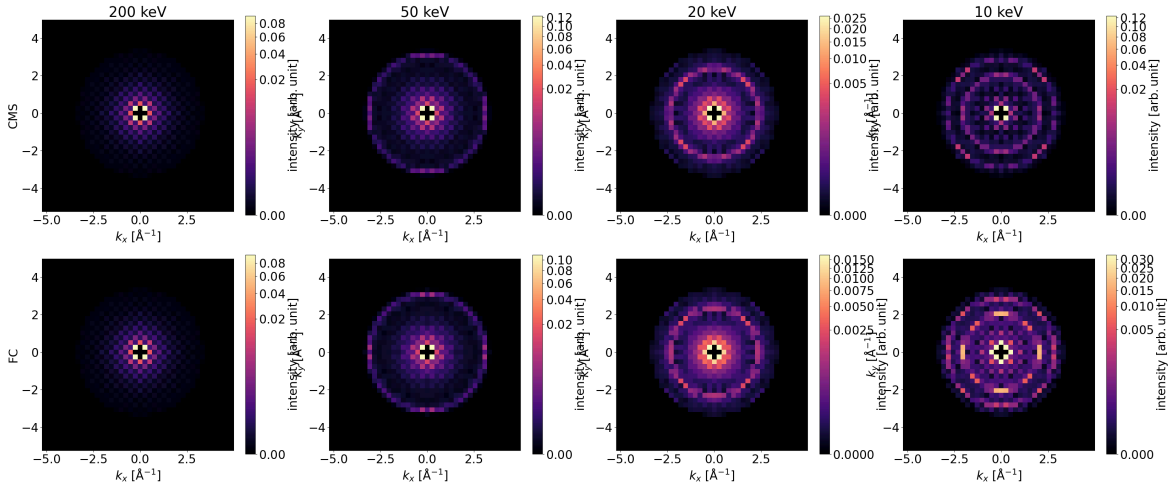


Figure 9: Diffraction patterns for $SrTiO_3$ illuminated by planewave for different energies with 48 unitcells thickness in z axis and power=0.25

24 unit cells z thickness

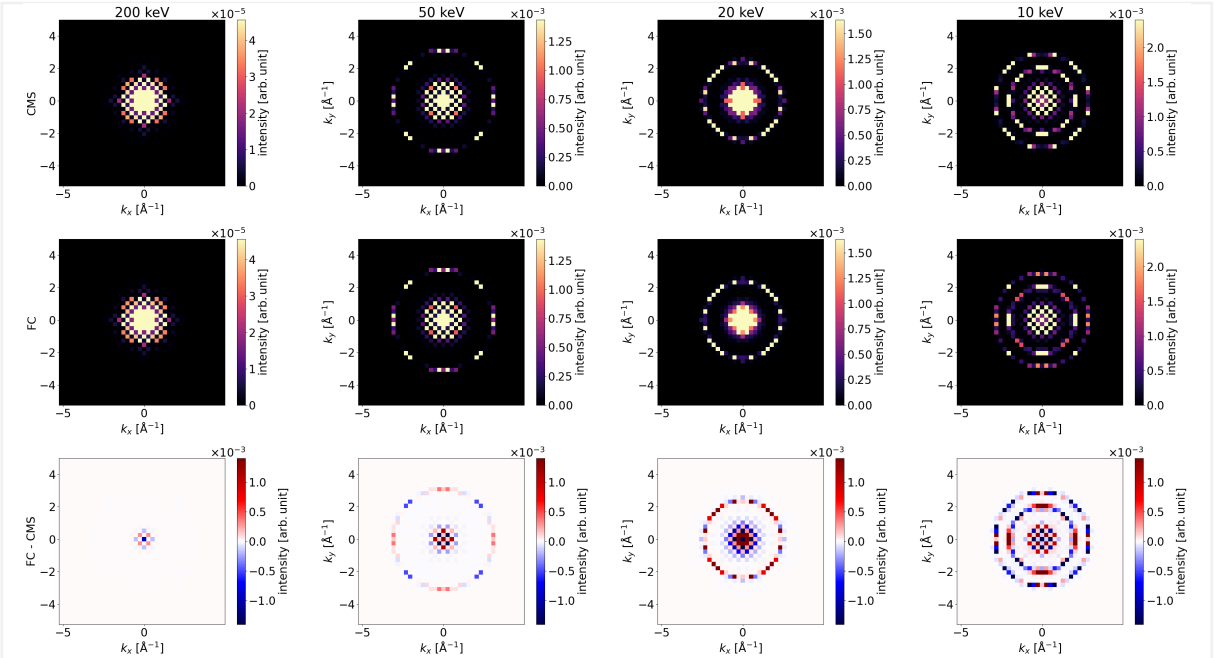


Figure 10: Comparison of the conventional multislice (CMS) and the propagator corrected multislice (FCMS) for $SrTiO_3$ with 24 unitcells thickness in z axis

48 unit cells z thickness

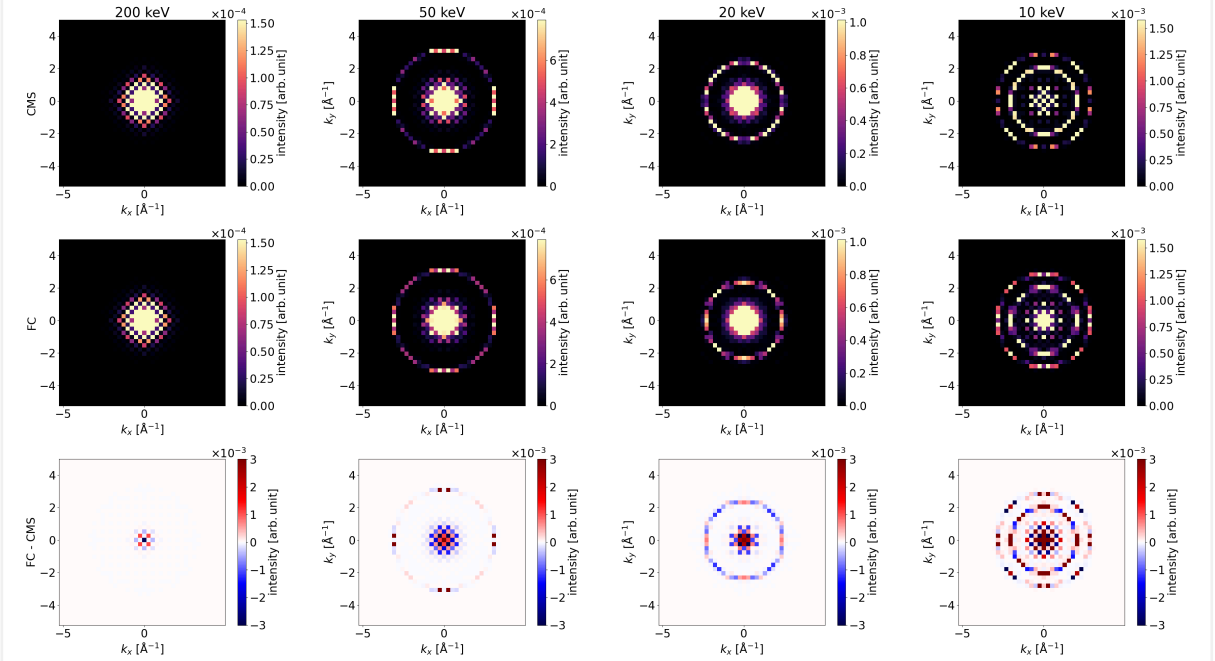


Figure 11: Comparison of the conventional multislice (CMS) and the propagator corrected multislice (FCMS) for $SrTiO_3$ with 48 unitcells thickness in z axis

Now we present the total coherent backscattered wavefunction for the same sample of SrTiO3 with 24 unit cells thickness. Note this appears qualitatively different from electron backscattered diffraction (EBSD) patterns, which are incoherent and usually modelled by leveraging the principle of reciprocity A. Winkelmann [16].

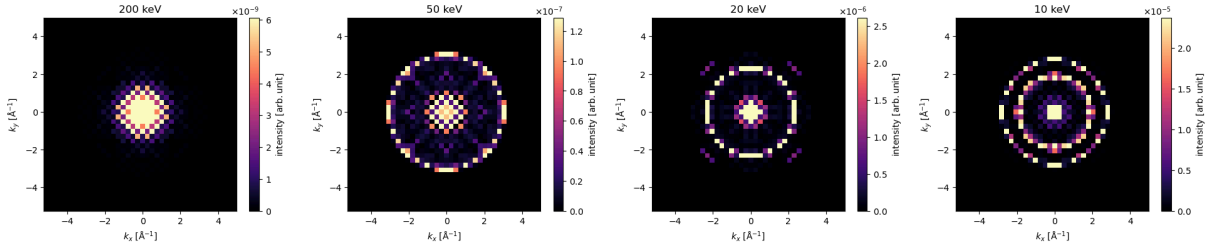


Figure 12: Reconstructed backscattered wave for $SrTiO_3$ with 24 unitcells thickness in z axis

Comparative Analysis

Planewave Diffraction Comparison

At first glance, the PCMS and FCMS seem to create the same changes compared to the CMS in realspace. There are clear differences in the higher scattering angles between both methods with conventional multislice. To show the differences and similarities between the 3 methods we employ a channel merging approach, where we assign a RGB color to each and plot the composition of all colors. This will result in a greyscale pixel if they agree and otherwise, the pixel will be a different color. We do this for the same sample as in previous chapters with 2 different sample thicknesses.

24 unit cells z thickness

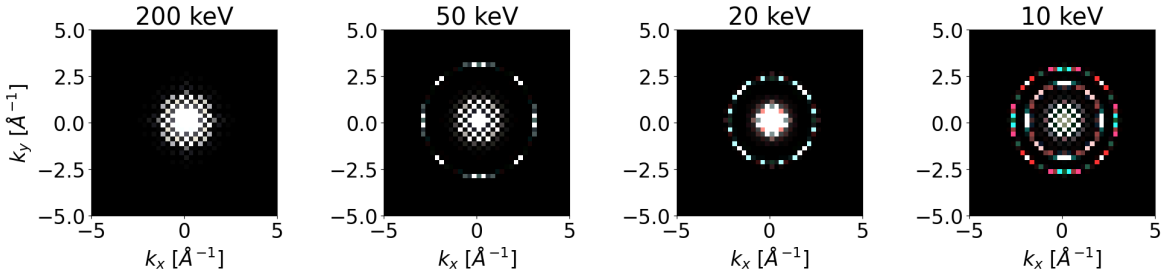


Figure 13: RGB comparison between R = CMS (RS), G = PCMS, B = FCMS for sample thickness 24c

48 unit cells z thickness

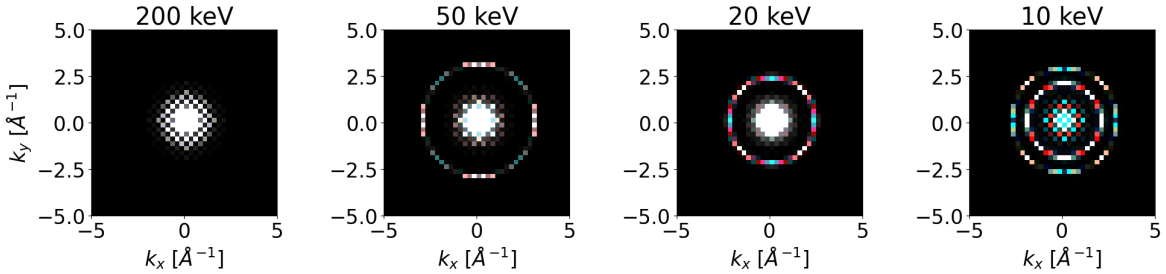


Figure 14: RGB comparison between R = CMS (RS), G = PCMS, B = FCMS for sample thickness 48c

We see that, as the voltage lowers, the CMS and the other 2 methods diverge. Still, at 10 keV, the PCMS and FCMS still mostly agree since we only see red and cyan (green and blue combined in equal proportion). To show the difference between them we can go to an even lower voltage, namely 5 kV.

24 unit cells z thickness

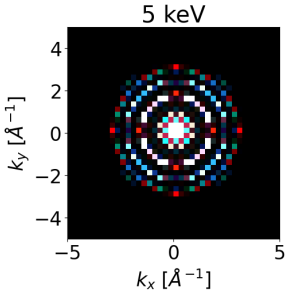


Figure 15: Thickness 24c

48 unit cells z thickness

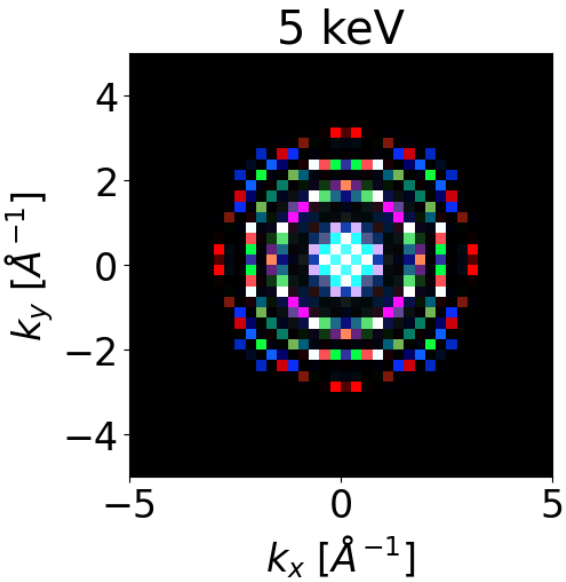


Figure 16: Thickness 48c

Here we can see clear separation of all 3 colors. From these plots we can observe that the 3 methods differ in terms of high angle scattering.

Converged Probe Comparison

Single scan position

Thus far we have only used planewave illumination. Although this is great for demonstrating the method, it is not possible to create pure planewave illumination in a real TEM. Therefore, we will now perform the same simulation but with a converged probe.

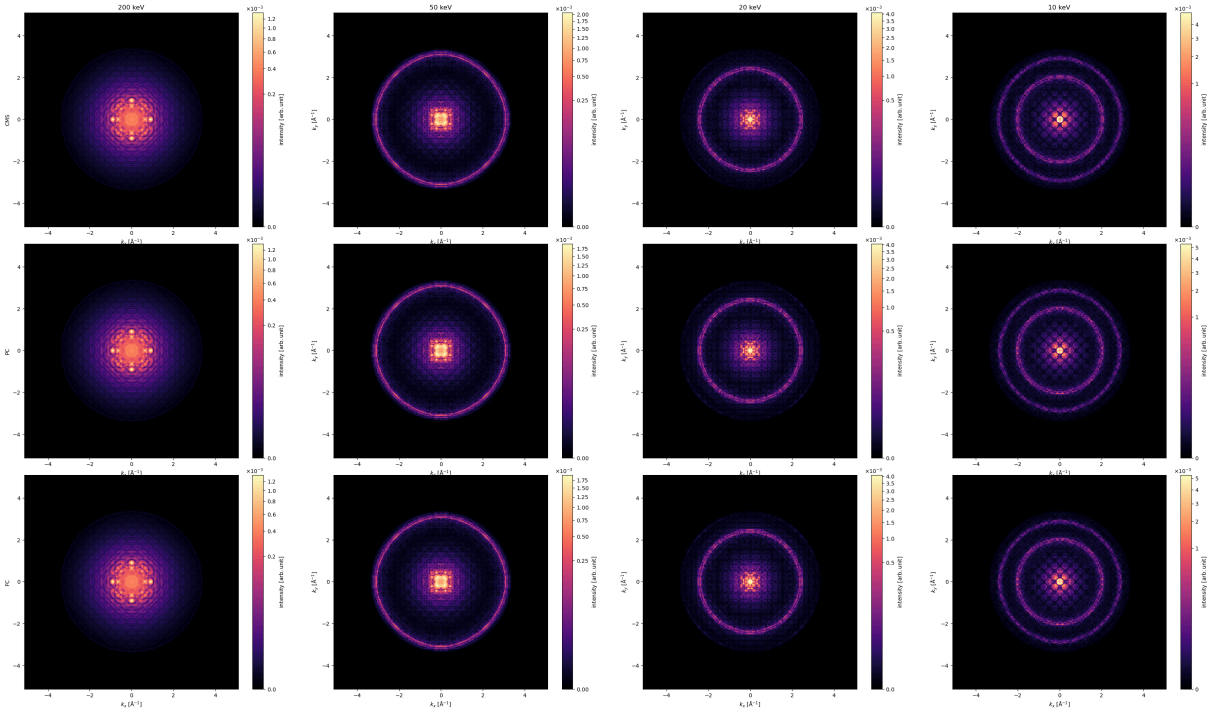


Figure 17: CBED scans for a sample of 8x8x24 unitcells of SrTiO3 for different methods at various energies

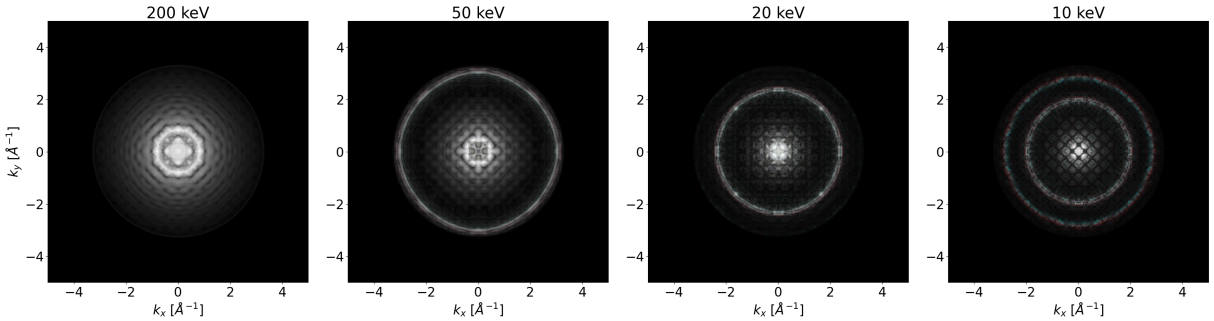


Figure 18: RGB comparison between R = CMS (RS), G = PCMS, B = FCMS for sample thickness 24c for Figure 5

Although less clear than for the planewaves, we again observe a slight separation of red and cyan at 20 keV, which gets more clear at 10 keV.

STEM

Unlike the planewave simulation, the CBED simulation is dependent on its position above the sample. To account for this, we perform a STEM measurement where we take the same converged probe but take a measurement at many different positions.

A gridscan was performed on a grid of 2x2 the unitcell size with a sampling distance of 0.2 Å on a sample of 8x8x24 unitcells of $SrTiO_3$. The sampling was based on the Nyquist sampling rate of the highest energy beam. For each scan a multislice simulation was performed with a sampling size of 0.1 Å and a slice thickness of 0.2 Å $with a probe semi-angle cutoff of 20 mrad. For each of the final exit waves for each position a annular detector was applied with an inner diameter of 25 mrad (just outside the central beam) and no outer radius. The lowest energy of these scans was 20 keV, because at 10 keV, the probe size is much larger than the distance between the atoms. This makes the probe exitwave less position dependent since at each scan location the probe sees most of the unitcell, resulting in a very blurry scan. This process can also be observed with the chosen voltages since for each lowering of the voltage the final image becomes blurrier.

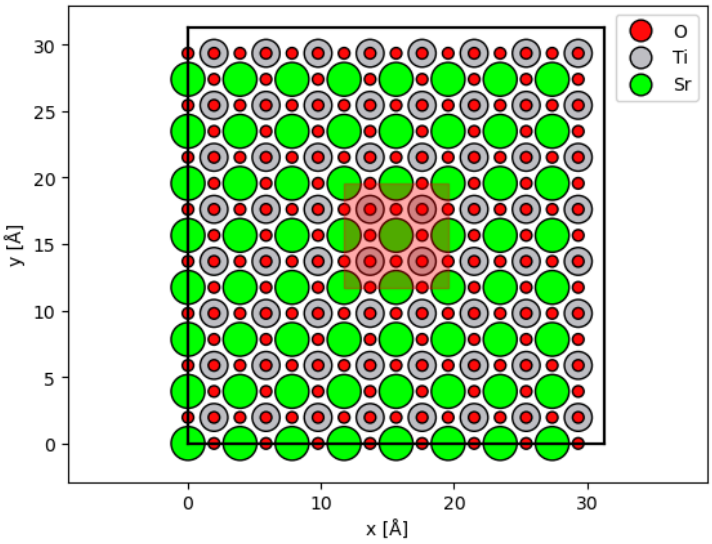


Figure 19: Position and size of the grid scan for a sample of 8x8x24 unitcells of SrTiO3

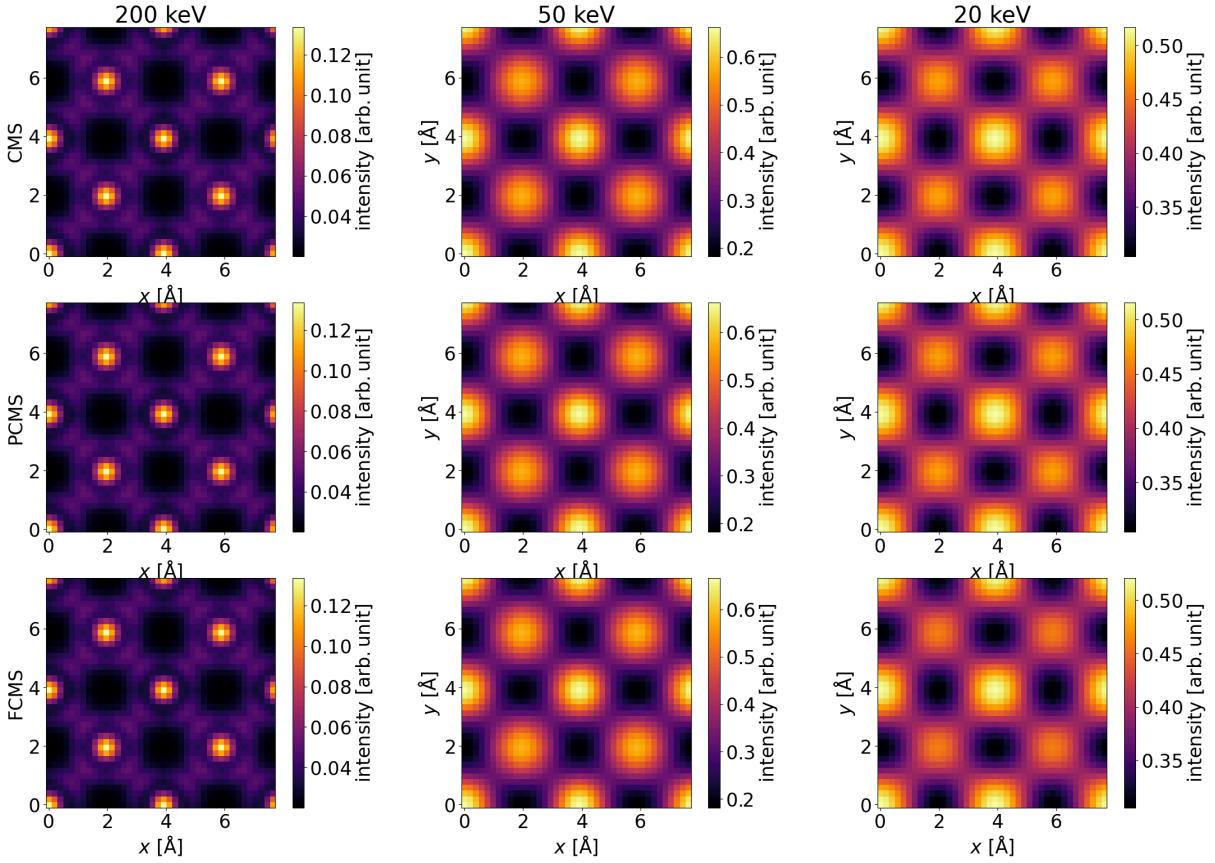


Figure 20: Grid scan with converged probe of 2x2 unitcells SrTiO3

Just looking at these images there does not seem to be any difference across the three multislice implementations. To quantify the difference we will calculate the relative difference for the PCMS and FCMS with the CMS and plot them on the same scale determined by the absolute highest value in all plots.

$$\mathrm{Diff}_{rel}(x,y) = \frac{I(x,y) - I_{\mathrm{CMS}}(x,y)}{I_{\mathrm{CMS}}(x,y)}$$

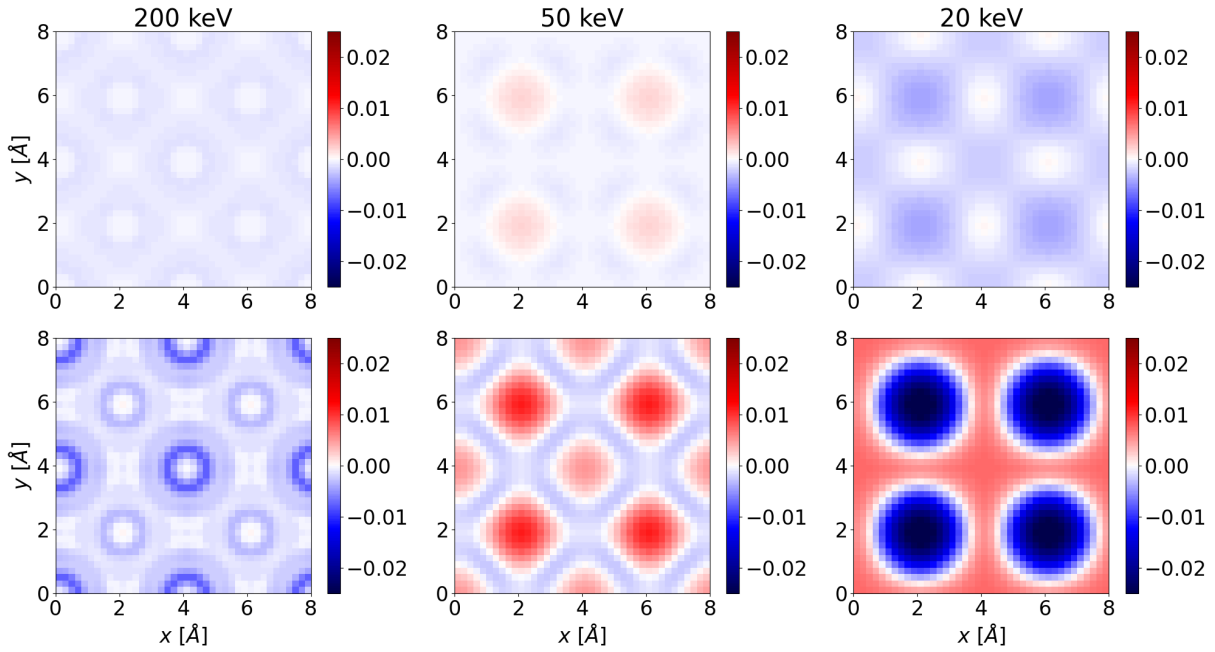


Figure 21: Relative difference plot for PCMS (top row) and FCMS (bottom row) using Figure 8

Here we observe the maximum relative difference increasing for lower energies. Note that, for the 200 kV beam the difference is entirely negative, meaning the new methods predict less electrons scattering on the annular detector. In the other two plots we see a greater dependence on probe position where the PCMS and FCMS predict more scattering on top of the titanium atoms.

Performance

To demonstrate the increased computation time we perform all three methods on a sample of one unit cell of SrTiO3 for various sampling distances. Both the PCMS and FCMS were calculated up to a power of 3 like in previous chapters.

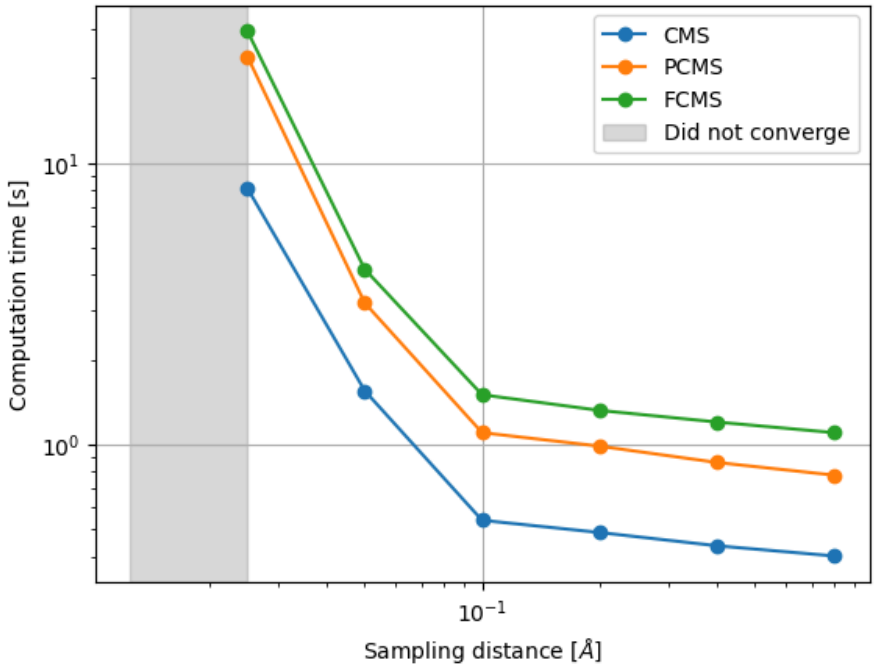


Figure 22: Computation time for one unitcell of SrTiO3 for various sampling distances on log scale

Conclusion

This thesis presented the open-source implementation of two more accurate multislice algorithms, namely the propagator corrected multislice (PCMS) and the fully corrected multislice (FCMS). Both methods were implemented into the open-source Python package abTEM, and the contribution is now under review by the package maintainers. These methods address the limitations of the conventional multislice (CMS) algorithm, specifically the parabolic approximation of the Ewald sphere and the lack of backscattering contributions, which degrades in accuracy at low electron energies.

Comparative analysis of SrTiO3 simulations confirms that for high electron energies (200 keV), all three methods yield nearly identical diffraction patterns. This validates the CMS for ordinary TEM simulations. However, significant divergence is observed as the electron energy decreases. Consistent with theoretical predictions, deviations between CMS and the corrected methods become noticeable below 50 keV.

The PCMS and FCMS methods remain in agreement down to approximately 10 keV. At extremely low energies (5 keV), the methods diverge from one another, suggesting that the higher-order potential interactions and backscattering effects accounted for in the FCMS become non-negligible at these energies. Furthermore, the implementation of the FCMS enabled the reconstruction of the backscattered exit wave. The results successfully demonstrate the expected theoretical dependence of the backscattered signal on electron wavelength, providing a new tool for simulating coherent EBSD signals in the scanning electron microscope (SEM).

In summary, the addition of PCMS and FCMS extends the validity of abTEM simulations for low voltage simulations, offering the ability for modeling radiation-sensitive materials and interpreting low-energy electron diffraction data.

Recommendations

This thesis implemented two new algorithms and coherent backscattering functionality to abTEM. While develop- ing these methods, the focus lay in the theoretical implementation of these algorithms. Therefore, performance and efficiency were not a priority . Although the laplace finite difference was adapted to run on the GPU, performance compared to the CMS using the FFT method is much slower. Even for relatively low orders of the PCMS and FCMS they are slower than the realspace CMS. Of course this is to be expected because of the extra terms. Yet, optimization of the codebase may drastically improve performance.

Beyond simulation, it is important to bridge the gap between theoretical and experimental. It would be highly valuable to compare the results from this thesis against real low voltage S/TEM experiments.

Having implemented the coherent backscattered contributions, this paper only shows its possibility for STEM-in- SEM applications. Testing the implementation against a broader range of materials and for various optical parameters would be very useful.

Finally, when reconstructing the backscattered signal the current code assumes perfect coherence where the complex backscattered signals are allowed to interfere. A great addition would be to also have fully incoherent backscattered signals where instead of the complex wavefunction we sum the wave intensities.

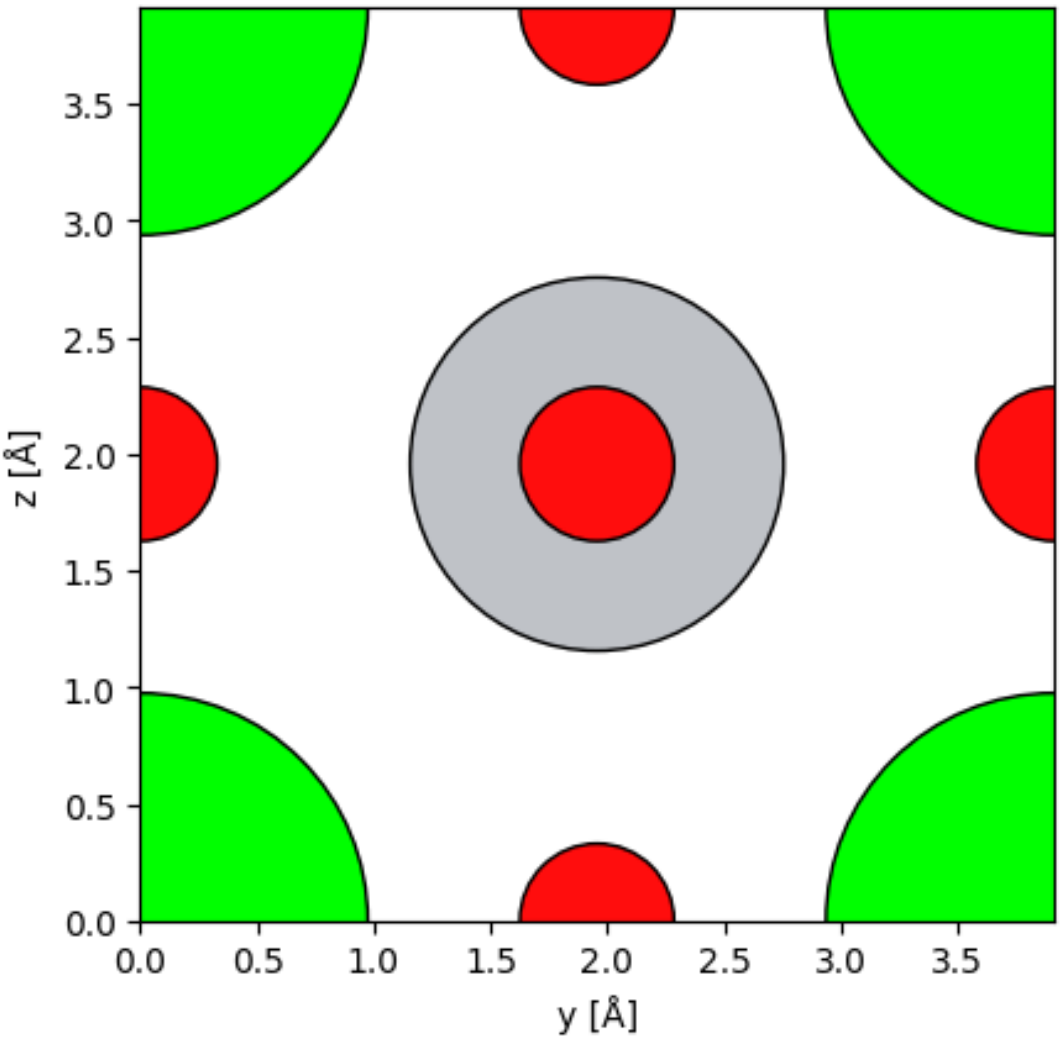
Notebooks

Multislice (Fast)

This notebook will give a general tutorial on multislice simulation in abTEM. In comparison to all simulation discussed in the thesis, this notebooks will use the Fourier method (9) to compute the multislice much faster. To compute a multislice simulation we will outline the following steps:

1. Define sample (using ASE)
2. Convert sample into potential
3. Define incident beam (planewave, converged probe)
4. Run multislice simulation

```
import abtem
import ase
import matplotlib.pyplot as plt
import numpy as np
STO_crystal = ase.Atoms(
    "SrTiO3",
    scaled_positions=[
        (0.0, 0.0, 0.0),

        (0.5, 0.5, 0.5),

        (0.5, 0.0, 0.5),
        (0.5, 0.5, 0.0),
        (0.0, 0.5, 0.5),
    ],
    cell=[3.91270131, 3.91270131, 3.91270131, 90, 90, 90],
    pbc=True
)
# abTEM multislice requires crystals to be orthogonal
STO_orthorhombic = abtem.orthogonalize_cell(STO_crystal)
fig,ax = plt.subplots()
abtem.show_atoms(STO_orthorhombic,plane='yz',scale=0.5,tight_limits=True,show_periodi
(<Figure size 640x480 with 1 Axes>, <Axes: xlabel='y [Å]', ylabel='z [Å]'>)
```



Crystal potential

```
sampling = 0.1 #Angstrom

slice_thickness = 0.5 #Angstrom

thickness = 48 #unit cells

sample_size = (1,1,thickness)

energy = 20e3 #eV
unit_cell_potential = abtem.Potential(
            STO_orthorhombic,
            sampling=sampling,
            parametrization="lobato",
            slice_thickness=slice_thickness,
            projection="finite",
        )
```

```
potential = abtem.CrystalPotential(
    unit_cell_potential,
    repetitions=sample_size,
)
```

Planewave

> **Note**
>
> Here it is important that the planewave grid is matched to the potential grid

```
planewave = abtem.PlaneWave(energy=energy).match_grid(potential)


detector = abtem.PixelatedDetector(max_angle=None)
```

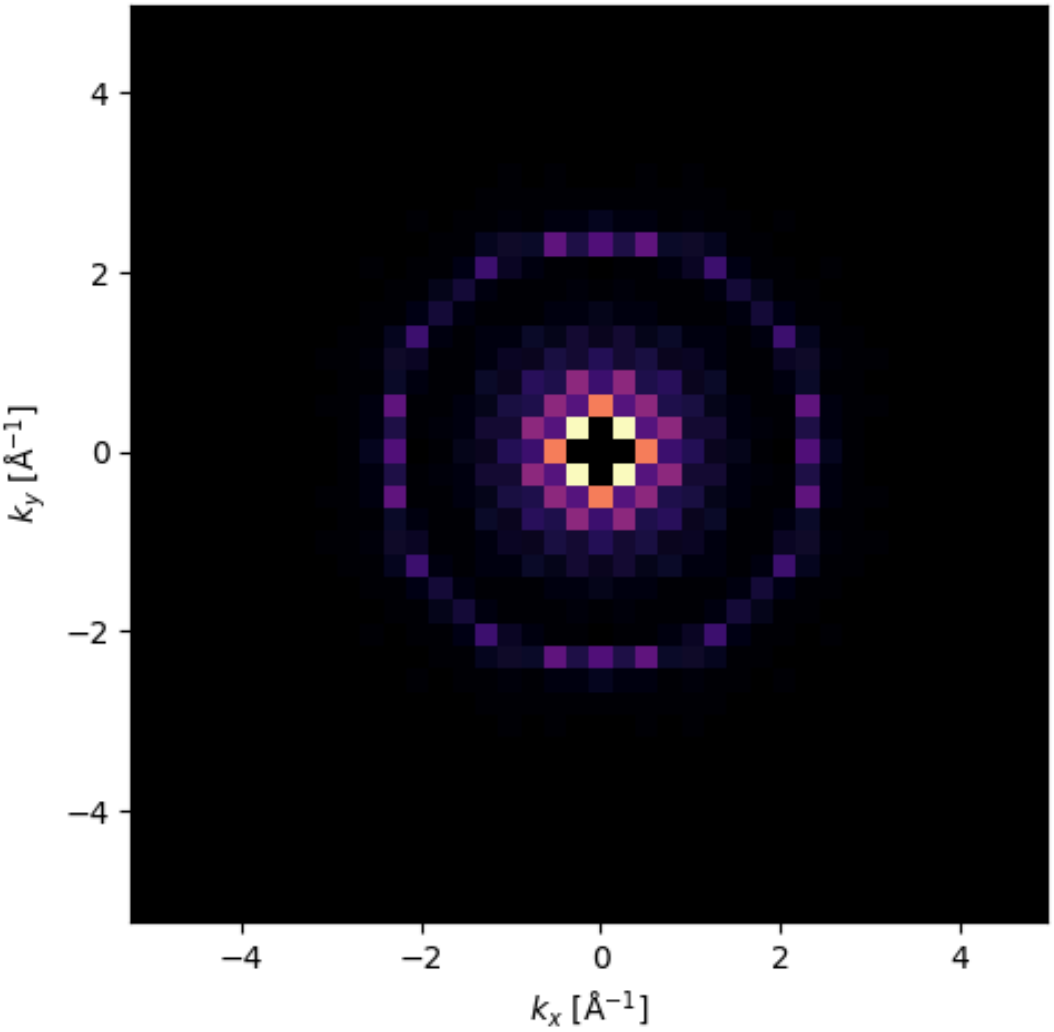
Running the simulation

```
exitwave_pw = planewave.multislice(
    potential=potential,
    detectors=detector,
    pbar=True,
    lazy=False
    )
```

```
multislice:   0%|          | 0/384 [00:00<?, ?it/s]
```

```
exitwave_pw.block_direct().show(
    cmap='magma',
    power=0.5,
    vmin=0
)
```

```
<abtem.visualize.visualizations.Visualization at 0x1325ec6e0>
```



Note

In the plot above the incident beam is blocked out by simply setting the central pixel to 0. This is done to highlight the detail of the scattered beams since most electrons go unscattered creating a bright spot.

CBED (Converged Beam Electron Diffraction)

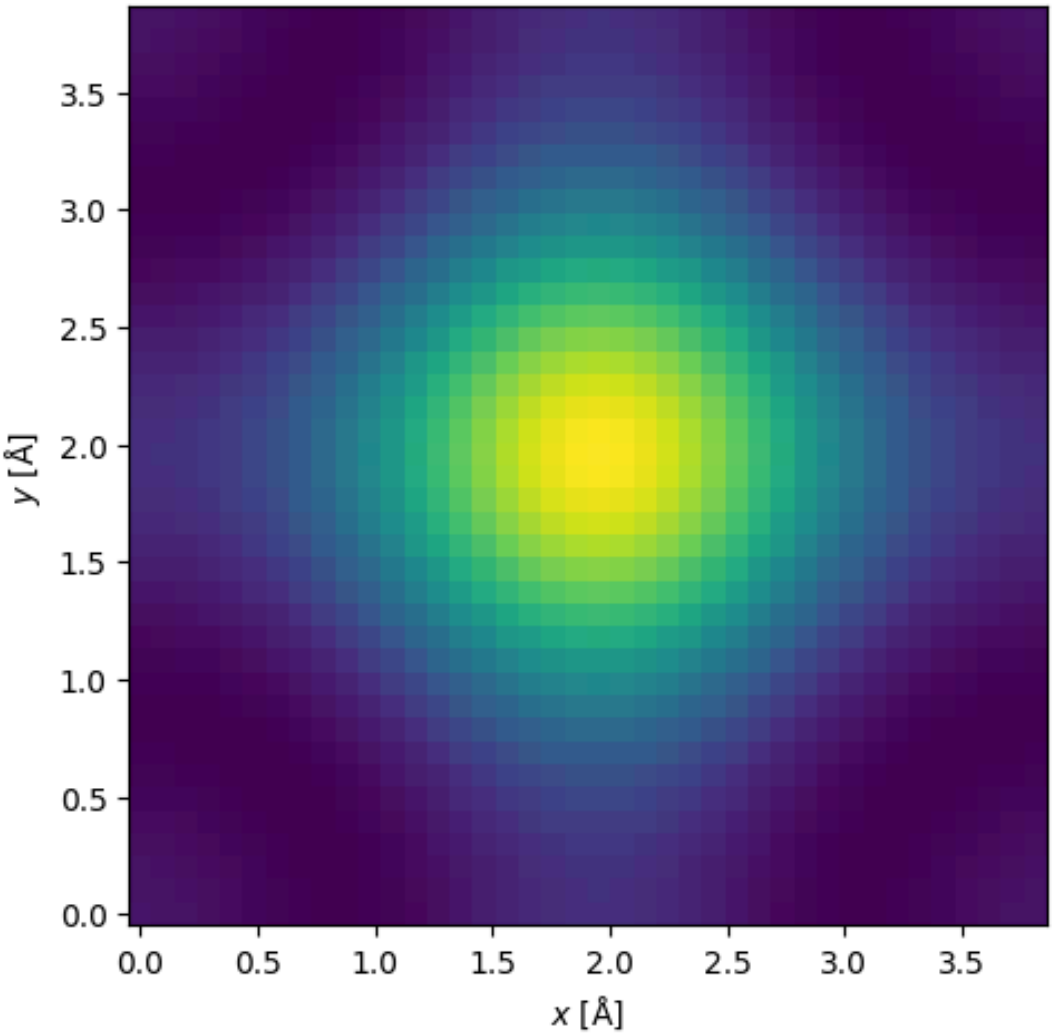
For a CBED simulation, instead of using a planewave we use a abTEM probe object which has an aperture to allow different beam angles up to a certain maximum. We set that maximum angle using the semiangle_cutoff parameter.

```
semiangle = 20
```

```
probe = abtem.Probe(energy=energy,
semiangle_cutoff=semiangle).match_grid(potential)
```

```
probe.show()
```

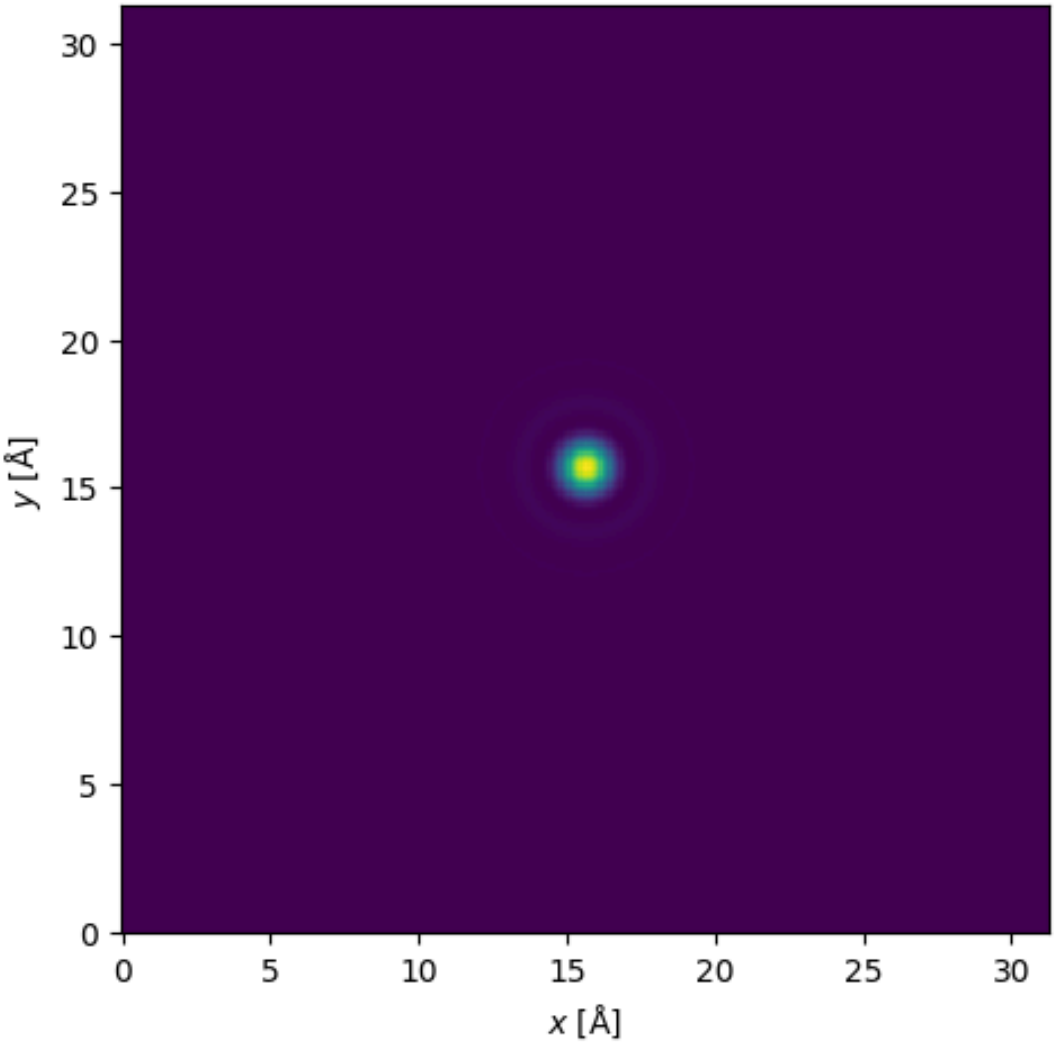
```
tasks:   0%|          | 0/2 [00:00<?, ?it/s]
<abtem.visualize.visualizations.Visualization at 0x1325ecf20>
```



The plot above shows the probe extending all the way to the edges of the simulation grid. This will cause incorrect results since the periodic boundary conditions abTEM uses will cause values to wrap over. To fix that we extend the simulation grid by repeating the crystal potential in the x and y directions. This was not necessary for the planewave simulation since a repeated crystal in the x and y directions would simply repeat the planewave exitwave.

```
potential_cbed = abtem.CrystalPotential(
    unit_cell_potential,
    repetitions=(8,8,thickness),
)
semiangle = 20

probe = abtem.Probe(energy=50e3,
semiangle_cutoff=semiangle).match_grid(potential_cbed)
```

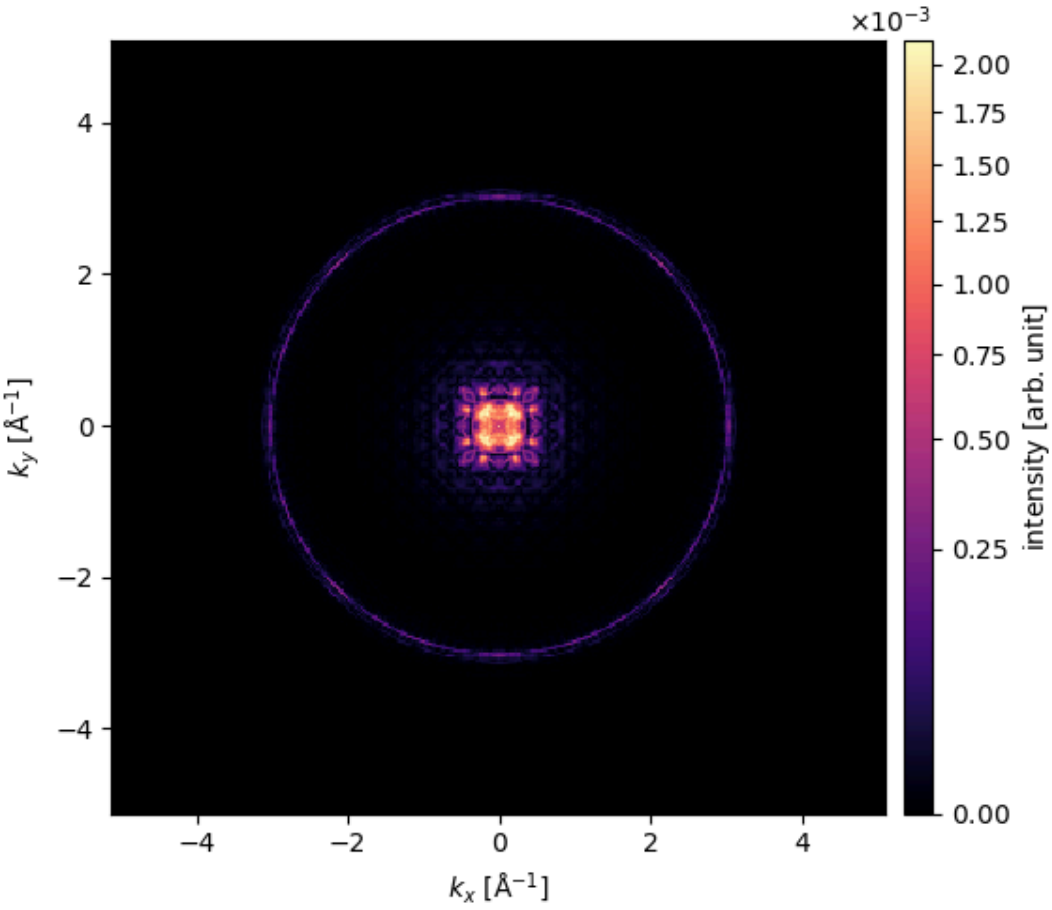
```
probe.show()
tasks:   0%|          | 0/2 [00:00<?, ?it/s]
<abtem.visualize.visualizations.Visualization at 0x1325c9fa0>
```



Now the probe is appropriately sized and we can proceed with the simulation

```
exitwave_probe = probe.multislice(
    potential=potential_cbed,
    detectors=detector,
    pbar=True,
    lazy=False
    )
multislice:   0%|          | 0/384 [00:00<?, ?it/s]
exitwave_probe.show(
    cmap='magma',
    power=0.5,
    vmin=0,
```

```
    cbar=True
)
<abtem.visualize.visualizations.Visualization at 0x132eef5c0>
```

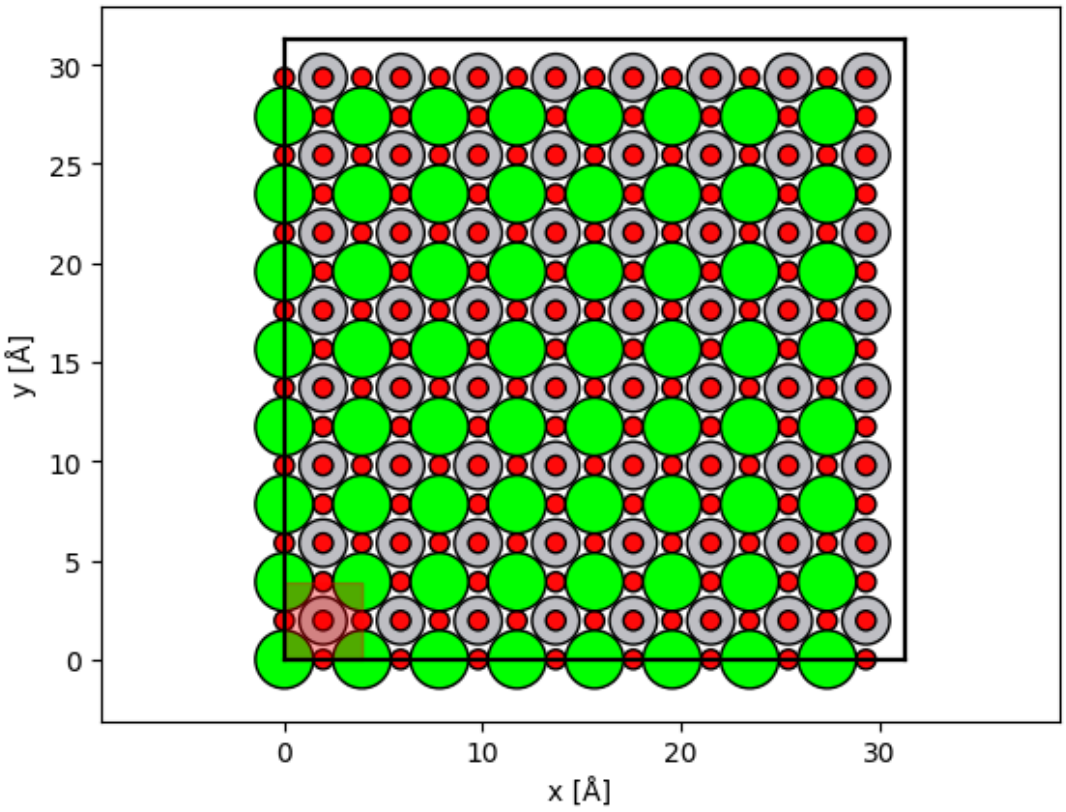


PACBED (Position Averaged CBED)

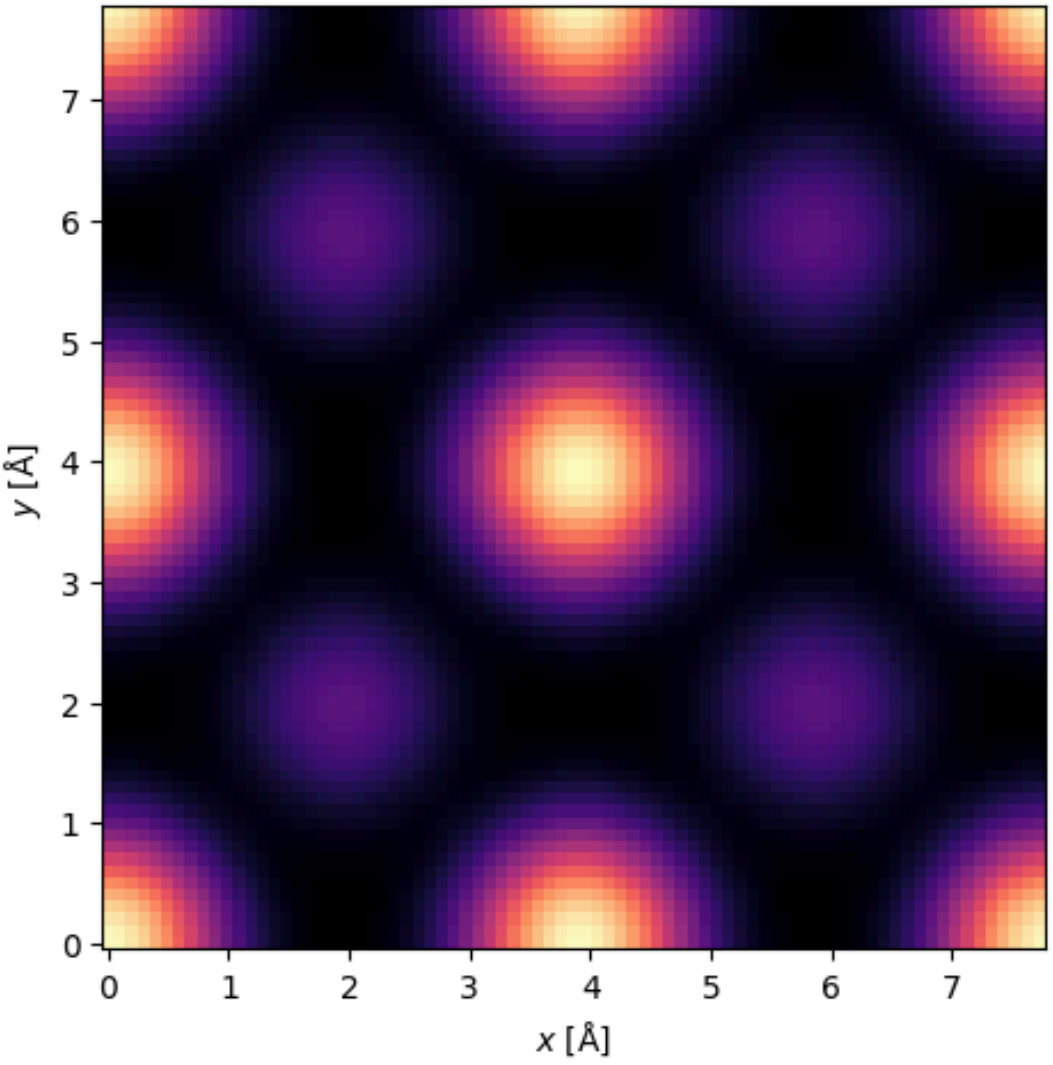
Lastly we will cover PACBED scans. Since at higher energies the probe size decreases even to a size smaller than the unit cells, there is a position dependence on the final exit wave. Positioning the probe directly above a heavy atom will create a different exit wave than if it were positioned somewhere between the atoms (viewed along the optical axis). Therefore it will be interesting to scan over a range of probe positions and average the exitwaves at each position which creates a final image. There is one small problem with this approach which is easily resolved: because our simulation is fully elastic without backscattering effects taken into account (See Fully Corrected MS), the average of every scan position will be the same. To deal with this we can average over a specific region of the detector. For this we use a annular detector which integrates over a region between two angles. This gives us a measure of how much scattering is happening. This in turn tells us a lot about the crystal. Directly above a heavy atom you expect more scattering than above mostly empty space. As explained in Section. STEM, the PACBED scans gets blurrier at lower energies since the probe size gets physically larger. So for this

demonstration we will increase the probe energy even though this thesis is mainly focussed on lower energy simulations.

```
haadf = abtem.AnnularDetector(inner=60, outer=None)


grid_scan = abtem.GridScan(
    start=[0, 0],
    end=(STO_orthorhombic.cell[0,0]*1, STO_orthorhombic.cell[1,1]*1),
    sampling=probe.aperture.nyquist_sampling,
    potential=potential_cbed,
)


fig, ax = abtem.show_atoms(STO_orthorhombic*(8,8,1))
grid_scan.add_to_plot(ax)
```




```
measurements = probe.scan(potential_cbed, scan=grid_scan, detectors=haadf,
lazy=False, pbar=True)
multislice:   0%|          | 0/13824 [00:00<?, ?it/s]
measurements.tile((2,2)).interpolate(sampling=0.1).show(cmap='magma')
<abtem.visualize.visualizations.Visualization at 0x131fbc680>
```


0
1
2
3
4
5
6
7
y [Å]
0
1
2
3
4
5
6
7
x [Å]

Planewave simulation

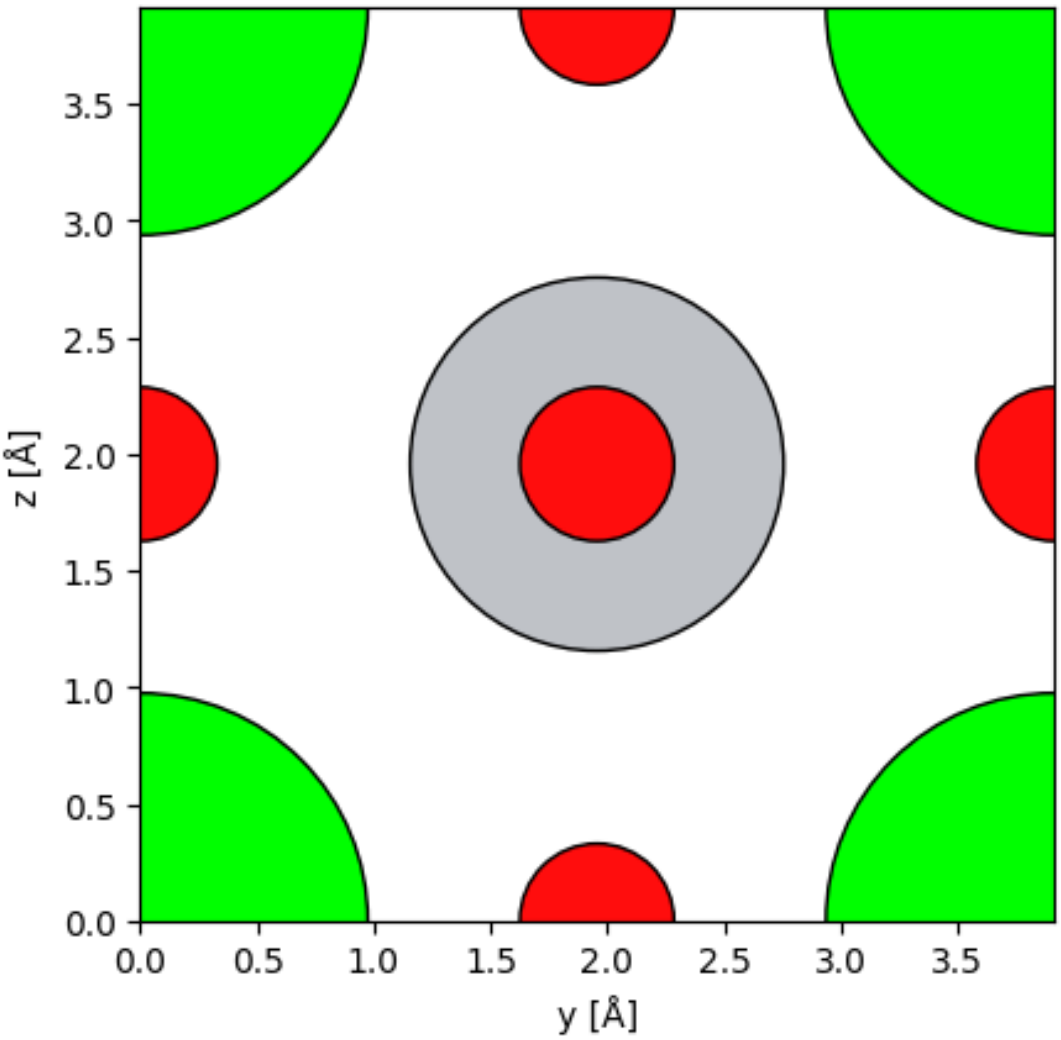
This notebook takes you through a simple planewave simulation for the same crystal used in the thesis and creates a RGB overlay to compare the differences. Feel free to play around with the simulation parameters like the electron energy.

> **Warning**
>
> **Caution!** This simulation requires some processing power to work and is therefore not suitable to run here in the browser. If you want to test out this notebook it is recommended to download the notebook and run it locally instead. There is also the multislice_fast notebook which runs perfectly fine in the browser.

```
import abtem
import ase
import matplotlib.pyplot as plt
import numpy as np
```

Creating the crystal

```
STO_crystal = ase.Atoms(
    "SrTiO3",
    scaled_positions=[
        (0.0, 0.0, 0.0),

        (0.5, 0.5, 0.5),

        (0.5, 0.0, 0.5),
        (0.5, 0.5, 0.0),
        (0.0, 0.5, 0.5),
    ],
    cell=[3.91270131, 3.91270131, 3.91270131, 90, 90, 90],
    pbc=True
)
STO_orthorhombic = abtem.orthogonalize_cell(STO_crystal)
fig,ax = plt.subplots()
abtem.show_atoms(STO_orthorhombic,plane='yz',scale=0.5,tight_limits=True,show_periodi
(<Figure size 640x480 with 1 Axes>, <Axes: xlabel='y [Å]', ylabel='z [Å]'>)
```



Creating the crystal potential

```
sampling = 0.1 #Angstrom

slice_thickness = 0.5 #Angstrom

thickness = 48 #unit cells

sample_size = (1,1,thickness)

energy = 20e3 #eV
unit_cell_potential = abtem.Potential(
            STO_orthorhombic,
            sampling=sampling,
            parametrization="lobato",
            slice_thickness=slice_thickness,
            projection="finite",
        )
```

```
potential = abtem.CrystalPotential(
    unit_cell_potential,
    repetitions=sample_size,
)
```

Creating the planewave

```
planewave = abtem.PlaneWave(energy=energy).match_grid(potential)
```

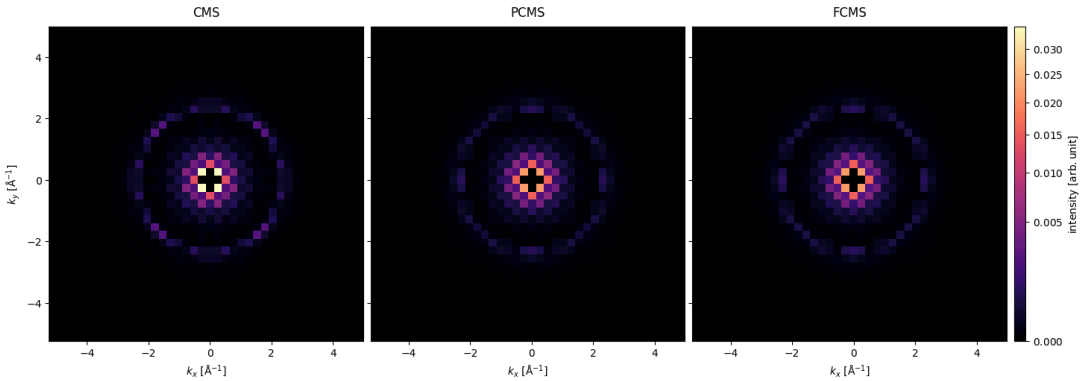
Running the simulation

```
from abtem.multislice import RealSpaceMultislice


detector = abtem.PixelatedDetector(max_angle=None)


CMS = RealSpaceMultislice()
PCMS = RealSpaceMultislice(
    order = 3,
)
FCMS = RealSpaceMultislice(
    order = 3,
    expansion_scope='full'
)
```

CMS

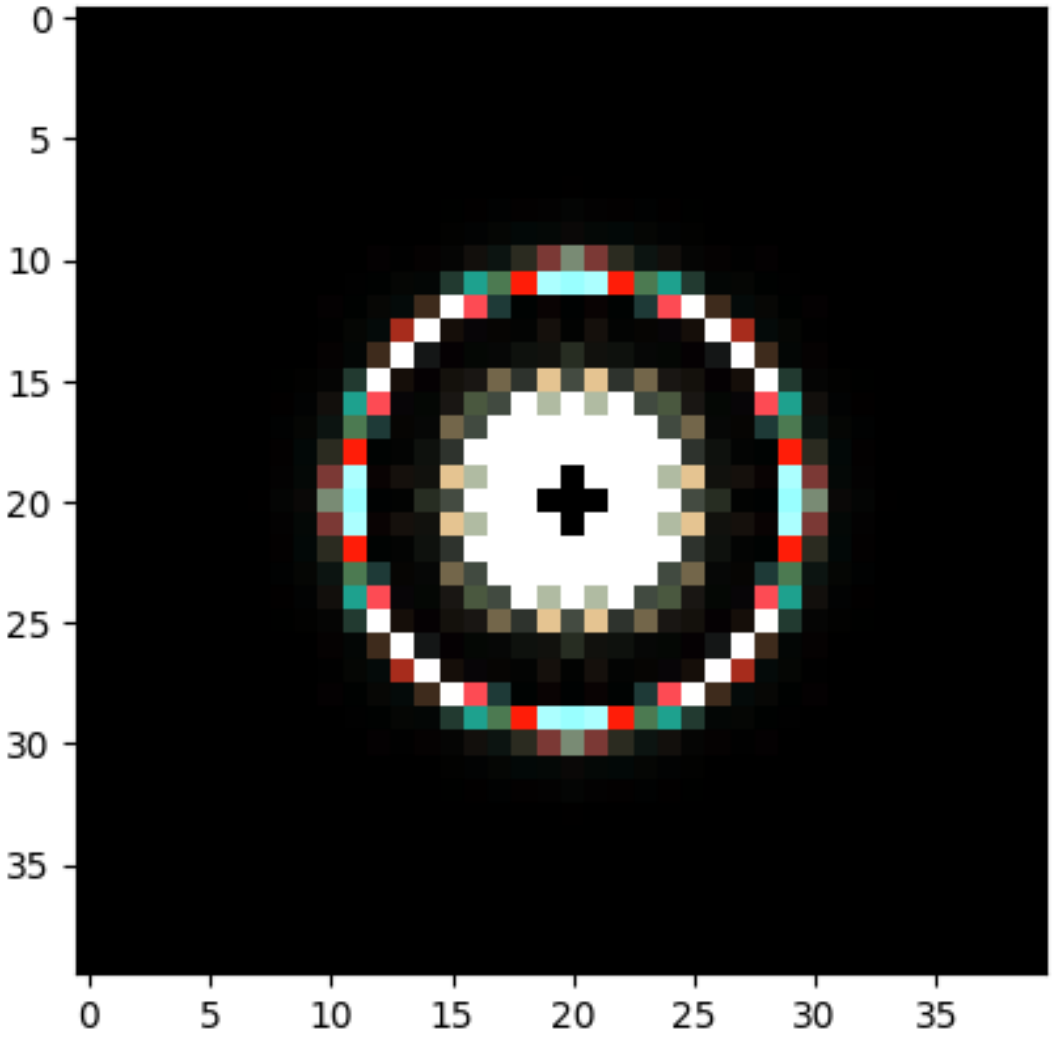
```
exitwave_cms = planewave.multislice(
    potential=potential,
    detectors=detector,
    algorithm = CMS,
    pbar=True,
    lazy=False
    )
multislice:   0%|          | 0/384 [00:00<?, ?it/s]
exitwave_pcms = planewave.multislice(
    potential=potential,
    detectors=detector,
    algorithm = PCMS,
    pbar=True,
```

```
    lazy=False
    )
multislice:   0%|          | 0/384 [00:00<?, ?it/s]
exitwave_fcms = planewave.multislice(
    potential=potential,
    detectors=detector,
    algorithm = FCMS,
    pbar=True,
    lazy=False
    )
multislice:   0%|          | 0/384 [00:00<?, ?it/s]
all_exitwaves = abtem.stack(
    (
        exitwave_cms.block_direct(),
        exitwave_pcms.block_direct(),
        exitwave_fcms.block_direct()
    ),
    (
        'CMS',
        'PCMS',
        'FCMS'
    )
)


all_exitwaves.show(
    explode = True,
    cbar=True,
    cmap='magma',
    figsize=(16, 9),
    vmin=0,
    common_color_scale=True,
    power=0.5
)
<abtem.visualize.visualizations.Visualization at 0x142ad47d0>
```



RGB overlay

```
threshold = 0.95

cms_array = exitwave_cms.block_direct().array
R_quantile = np.quantile(cms_array, threshold)
cms_array = np.clip(cms_array, 0, R_quantile)
R = cms_array / cms_array.max()

pcms_array = exitwave_pcms.block_direct().array
G_quantile = np.quantile(pcms_array, threshold)
pcms_array = np.clip(pcms_array, 0, G_quantile)
G = pcms_array / pcms_array.max()

fcms_array = exitwave_fcms.block_direct().array
B_quantile = np.quantile(fcms_array, threshold)
fcms_array = np.clip(fcms_array, 0, B_quantile)
B = fcms_array / fcms_array.max()

rgb_image = np.stack([R, G, B], axis=-1)
plt.imshow(rgb_image, vmin=0)
<matplotlib.image.AxesImage at 0x142c13d40>
```

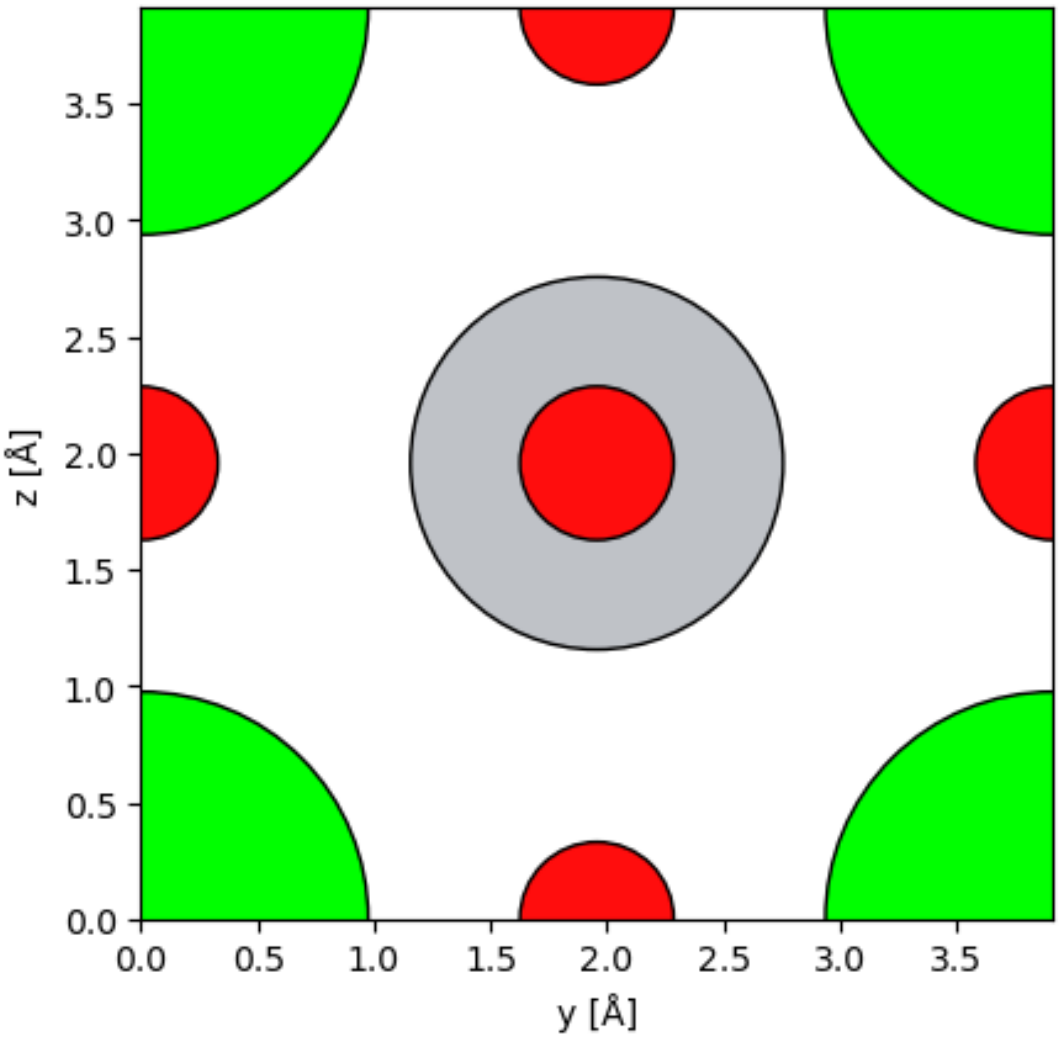

0
5
10
15
20
25
30
35
0
5
10
15
20
25
30
35

(PA)CBED simulation

This notebook takes you through a simple CBED simulation for the same crystal used in the thesis and creates a RGB overlay to compare the differences. Feel free to play around with the simulation parameters like the electron energy. This notebook also does a PACBED scan and compares the different models for that as well.

> **Warning**
>
> **Caution!** This simulation requires some processing power to work and is therefore not suitable to run here in the browser. If you want to test out this notebook it is recommended to download the notebook and run it locally instead. There is also the multislice_fast notebook which runs perfectly fine in the browser.

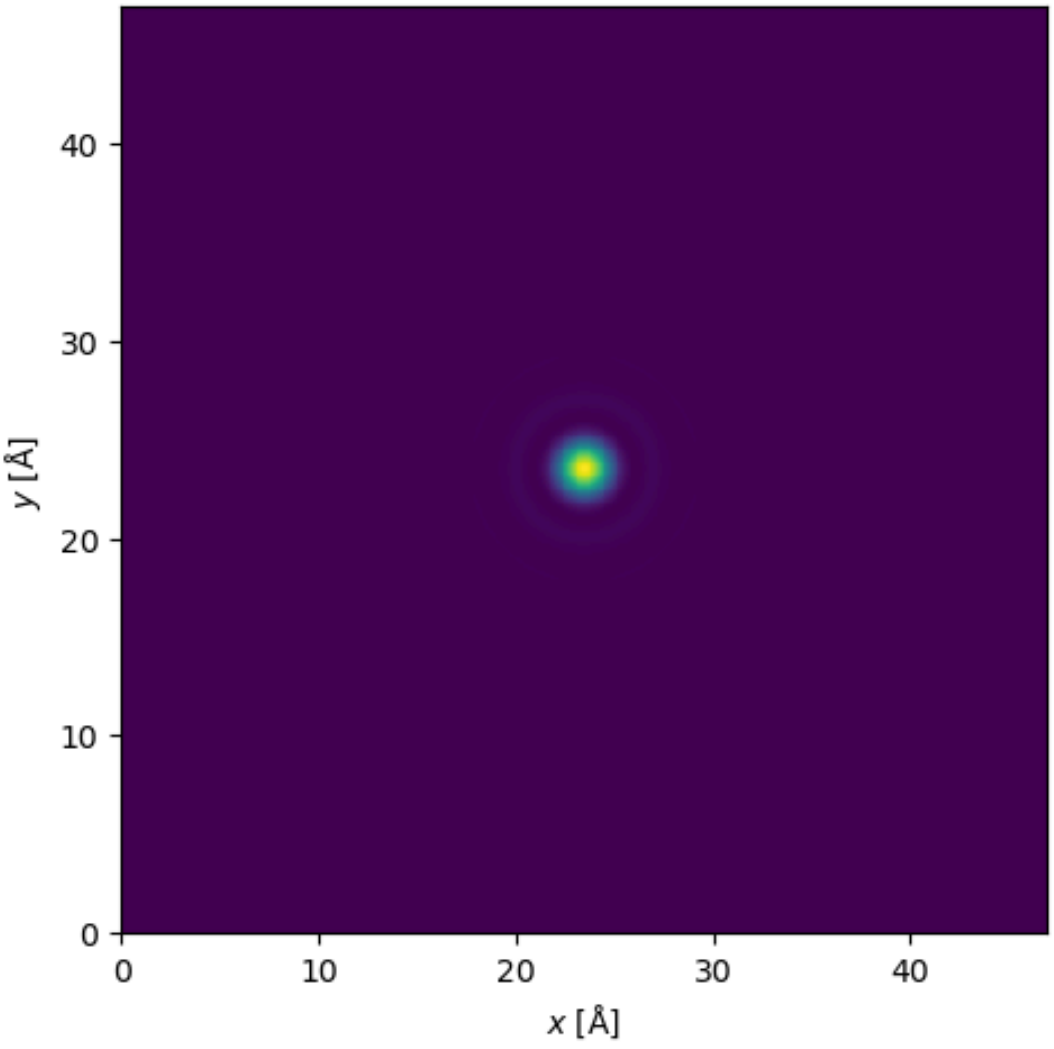
```
import abtem
import ase
import matplotlib.pyplot as plt
import numpy as np
STO_crystal = ase.Atoms(
    "SrTiO3",
    scaled_positions=[
        (0.0, 0.0, 0.0),

        (0.5, 0.5, 0.5),

        (0.5, 0.0, 0.5),
        (0.5, 0.5, 0.0),
        (0.0, 0.5, 0.5),
    ],
    cell=[3.91270131, 3.91270131, 3.91270131, 90, 90, 90],
    pbc=True
)
STO_orthorhombic = abtem.orthogonalize_cell(STO_crystal)
abtem.show_atoms(STO_orthorhombic,plane='yz',scale=0.5,tight_limits=True,show_periodi
(<Figure size 640x480 with 1 Axes>, <Axes: xlabel='y [Å]', ylabel='z [Å]'>)
```




```
sampling = 0.1 #Angstrom
slice_thickness = 0.5 #Angstrom
thickness = 48 #unit cells
sample_size = (12,12,thickness)
energy = 20e3 #eV
semiangle=20
unit_cell_potential = abtem.Potential(
            STO_orthorhombic,
            sampling=sampling,
            parametrization="lobato",
            slice_thickness=slice_thickness,
            projection="finite",
            # device='gpu'
        )
```

```
potential = abtem.CrystalPotential(
    unit_cell_potential,
    repetitions=sample_size,
)
probe = abtem.Probe(
    energy=energy,
    semiangle_cutoff=semiangle,
    # device='gpu'
).match_grid(potential)
```

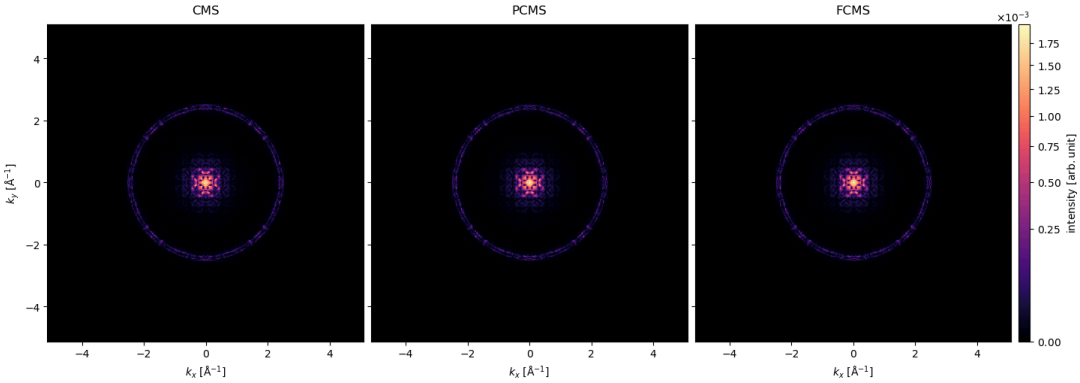
```
probe.show()
tasks:   0%|          | 0/2 [00:00<?, ?it/s]
<abtem.visualize.visualizations.Visualization at 0x7614f8340a10>
```



```
from abtem.multislice import RealSpaceMultislice

detector = abtem.PixelatedDetector(max_angle=None)

CMS = RealSpaceMultislice()
PCMS = RealSpaceMultislice(
    order = 3,
)
FCMS = RealSpaceMultislice(
    order = 3,
    expansion_scope='full'
)
exitwave_cms = probe.multislice(potential=potential,detectors=detector,
algorithm = CMS,pbar=True,lazy=False)
multislice:   0%|          | 0/384 [00:00<?, ?it/s]
exitwave_pcms = probe.multislice(potential=potential,detectors=detector,
algorithm = PCMS,pbar=True,lazy=False)
multislice:   0%|          | 0/384 [00:00<?, ?it/s]
exitwave_fcms = probe.multislice(potential=potential,detectors=detector,
algorithm = FCMS,pbar=True,lazy=False)
multislice:   0%|          | 0/384 [00:00<?, ?it/s]
all_exitwaves = abtem.stack(
    (
        exitwave_cms,
        exitwave_pcms,
        exitwave_fcms
    ),
    (
        'CMS',
        'PCMS',
        'FCMS'
    )
)

all_exitwaves.show(
```

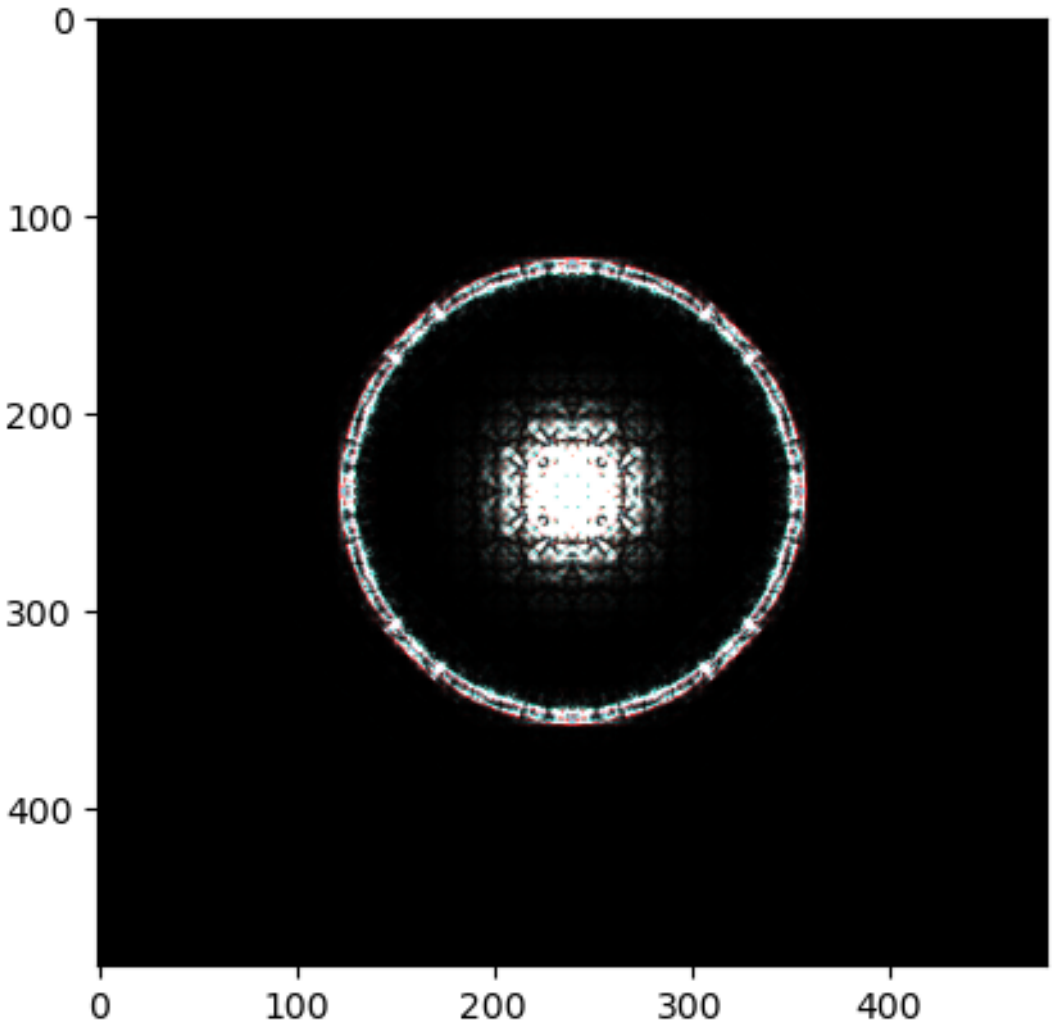
```
    explode = True,
    cbar=True,
    cmap='magma',
    figsize=(16, 9),
    vmin=0,
    common_color_scale=True,
    power=0.5
)
```

```
<abtem.visualize.visualizations.Visualization at 0x7614b8cffb90>
```




```
threshold = 0.98

cms_array = exitwave_cms.array
R_quantile = np.quantile(cms_array, threshold)
cms_array = np.clip(cms_array, 0, R_quantile)
R = cms_array / cms_array.max()

pcms_array = exitwave_pcms.array
G_quantile = np.quantile(pcms_array, threshold)
pcms_array = np.clip(pcms_array, 0, G_quantile)
G = pcms_array / pcms_array.max()

fcms_array = exitwave_fcms.array
B_quantile = np.quantile(fcms_array, threshold)
fcms_array = np.clip(fcms_array, 0, B_quantile)
B = fcms_array / fcms_array.max()
```

```
rgb_image = np.stack([R, G, B], axis=-1)

plt.imshow(rgb_image, vmin=0)
<matplotlib.image.AxesImage at 0x7614b8a158d0>
```



PACBED

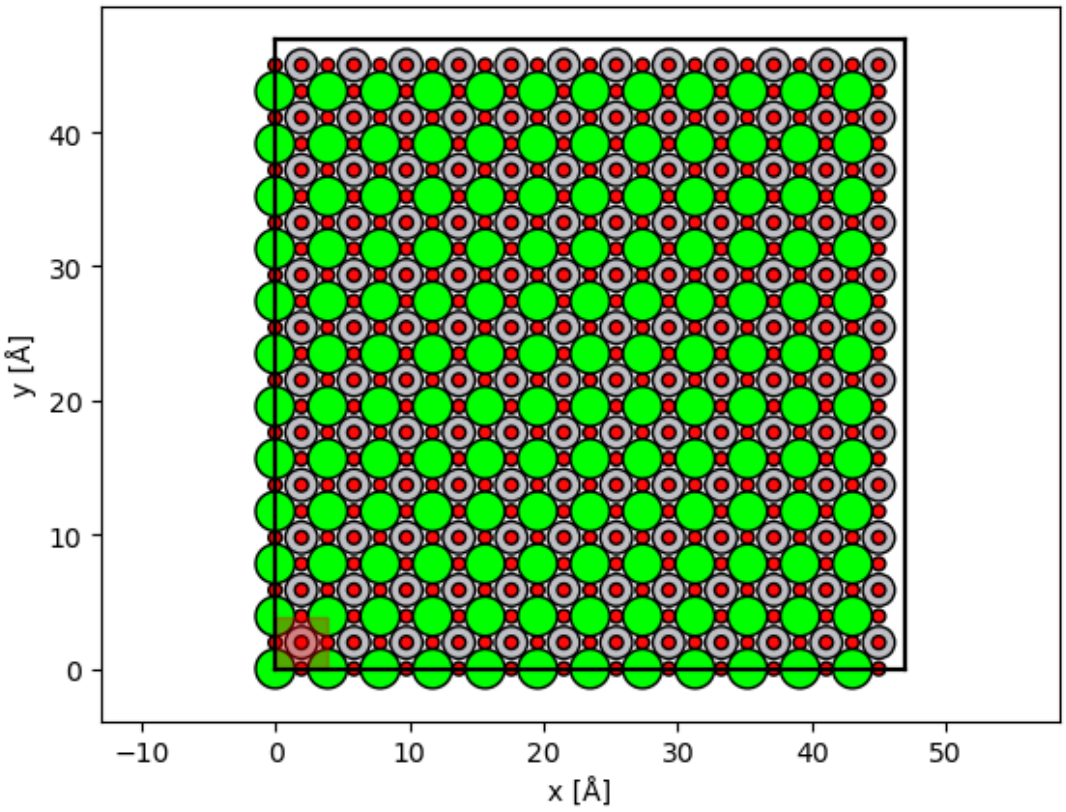
> **Note**
>
> This simulation takes quite long dependend on the machine used. Calculating on a GPU is highly recommended

Here we perform a PACBED scan for a higher electron energy and plot the relative difference between the PCMS/FCMS and the CMS as explained in Section. STEM.

```
haadf = abtem.AnnularDetector(inner=25, outer=None)


probe = abtem.Probe(energy=200e3, semiangle_cutoff=semiangle,
device='gpu').match_grid(potential)
```

```
grid_scan = abtem.GridScan(
    start=[0, 0],
    end=(STO_orthorhombic.cell[0,0]*1, STO_orthorhombic.cell[1,1]*1),
    sampling=probe.aperture.nyquist_sampling,
    potential=potential,
)
```

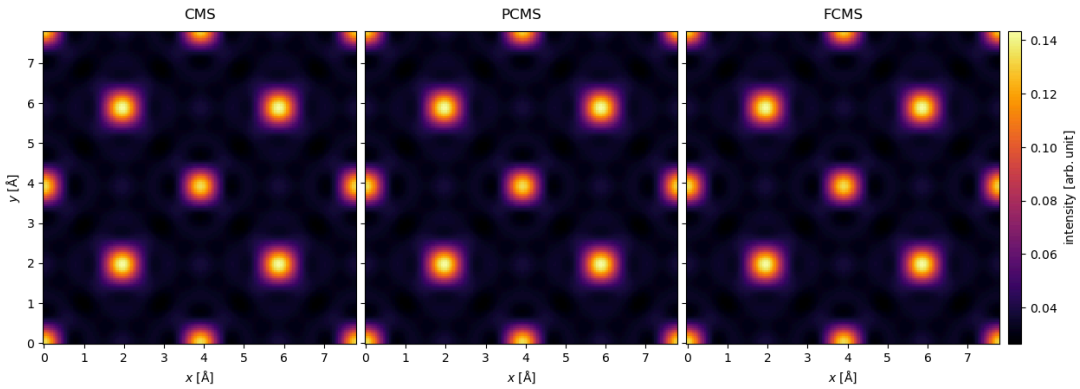
```
fig, ax = abtem.show_atoms(STO_orthorhombic*(12,12,1))
grid_scan.add_to_plot(ax)
```




```
measurements_cms = probe.scan(potential, scan=grid_scan, detectors=haadf,
lazy=False, algorithm=CMS, pbar=True)
multislice:   0%|          | 0/64896 [00:00<?, ?it/s]
measurements_pcms = probe.scan(potential, scan=grid_scan, detectors=haadf,
lazy=False, algorithm=PCMS, pbar=True)
multislice:   0%|          | 0/64896 [00:00<?, ?it/s]
measurements_fcms = probe.scan(potential, scan=grid_scan, detectors=haadf,
lazy=False, algorithm=FCMS, pbar=True)
multislice:   0%|          | 0/64896 [00:00<?, ?it/s]
interpolation_value = 0.05
```

```
all_exitwaves = abtem.stack(
    (
measurements_cms.tile((2,2)).interpolate(sampling=interpolation_value),
measurements_pcms.tile((2,2)).interpolate(sampling=interpolation_value),
measurements_fcms.tile((2,2)).interpolate(sampling=interpolation_value)
    ),
    (
        'CMS',
        'PCMS',
        'FCMS'
    )
)

all_exitwaves.show(
    explode = True,
    cbar=True,
    cmap='inferno',
    figsize=(14, 8),
    common_color_scale=True,
)
```

```
<abtem.visualize.visualizations.Visualization at 0x7614b033f4d0>
```

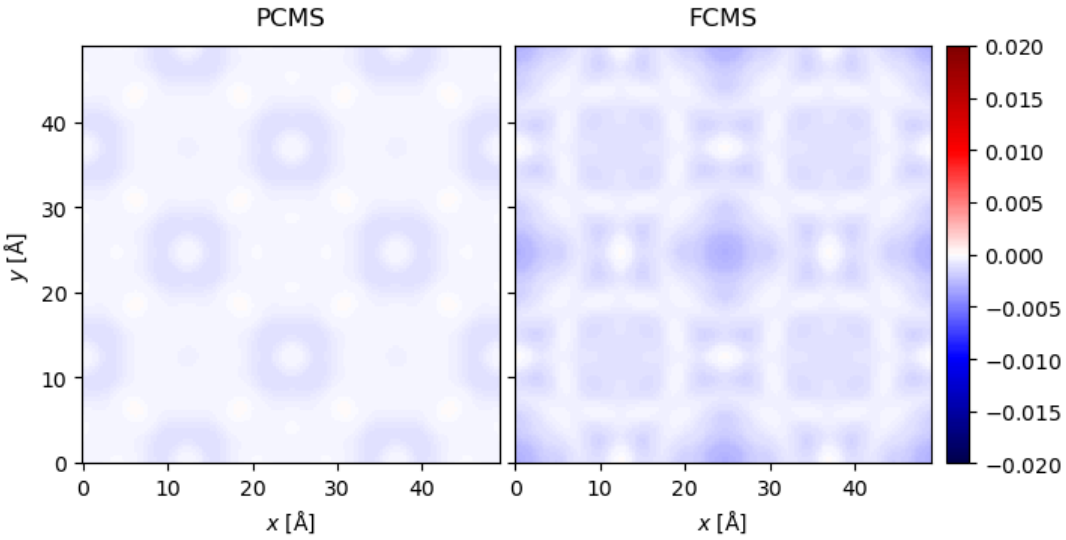



```
from abtem.measurements import Images

diff_rel_pcms =
(measurements_pcms.tile((2,2)).interpolate(sampling=interpolation_value).array
```

```
-
measurements_cms.tile((2,2)).interpolate(sampling=interpolation_value).array) /
(1e-6 +
measurements_cms.tile((2,2)).interpolate(sampling=interpolation_value).array)
diff_rel_fcms =
(measurements_fcms.tile((2,2)).interpolate(sampling=interpolation_value).array
-
measurements_cms.tile((2,2)).interpolate(sampling=interpolation_value).array) /
(1e-6 +
measurements_cms.tile((2,2)).interpolate(sampling=interpolation_value).array)

diff_rel_pcms_img = Images(array = diff_rel_pcms,
sampling=probe.aperture.nyquist_sampling)
diff_rel_fcms_img = Images(array = diff_rel_fcms,
sampling=probe.aperture.nyquist_sampling)

all_measurements = abtem.stack(
    (
        diff_rel_pcms_img,
        diff_rel_fcms_img
    ),
    (
        'PCMS',
        'FCMS'
    )
)

all_measurements.show(
    cmap='seismic',
    vmin=-0.02,
    vmax=0.02,
    explode=True,
    figsize=(8, 5),
```

```
    common_color_scale=True,
    cbar=True
)
```

```
<abtem.visualize.visualizations.Visualization at 0x7614991ee250>
```



Appendix

Additional plots

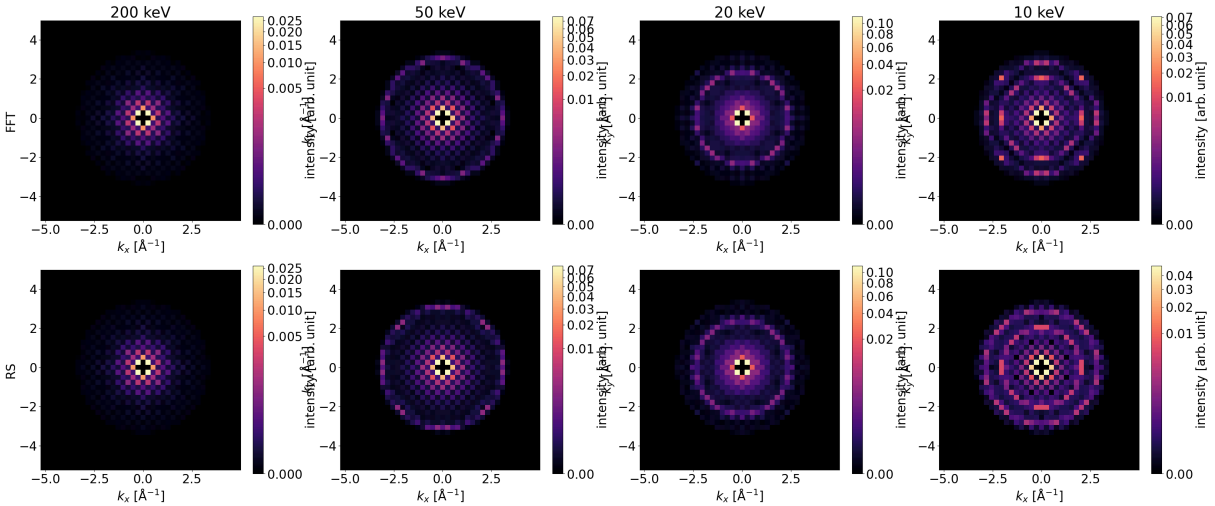


Figure 23: Comparison of the conventional multislice (CMS) calculated with the Fourier methods vs realspace method. A difference can be observed for 10 keV.

Source code

The functions used to compute the multislice algorithms described in this thesis. This notebooks is not meant to be ran. For more information on the code please visit the abTEM GitHub.

> **Note**
>
> The code has changed somewhat compared to the original paper to meet the requirements of abTEM. The code as it is shown here is the code that is currently in abTEM.

```
def conventional_operator(
    waves: np.ndarray | da.core.Array,
    laplace: Callable,
    transmission_function: np.ndarray,
    wavelength: float,
):
    """
    Split-step real-space multislice operator used in all higher-order
expansions.

    Parameters
    ----------
    waves: Waves
        Waves object to apply multislice operator on
    laplace: Callable
        Fast laplace operator stencil function
    transmission_function: np.ndarray,
        Scaled potential slice to multiply incoming waves with
    wavelength: float
        Waves wavelength
    """
    K0 = 1 / wavelength
    return laplace(waves) / (4 * np.pi * K0) + transmission_function *
waves
```

Figure 24: The realspace conventional multislice step argument including the 2D Laplacian (propagator) and transmission operator.

```
def _multislice_exponential_series(
    waves: np.ndarray | da.core.Array,
```

```
    transmission_function: np.ndarray,
    laplace: Callable,
    wavelength: float,
    thickness: float,
    tolerance: float = 1e-16,
    max_terms: int = 300,
    order: int = 1,
    fully_corrected: bool = False,
):
    xp = get_array_module(waves)
    initial_amplitude = xp.abs(waves).sum()

    if fully_corrected:
        temp = full_series(
            waves, laplace, transmission_function, order, wavelength,
thickness
        )
    else:
        temp = propagator_taylor_series(
            waves,
            order=order,
            laplace=laplace,
            transmission_function=transmission_function,
            wavelength=wavelength,
            thickness=thickness,
        )

    waves += temp

    for i in range(2, max_terms + 1):
        if fully_corrected:
            temp = (
                full_series(
```

```
                    temp, laplace, transmission_function, order,
wavelength, thickness
                )
                / i
            )
        else:
            temp = (
                propagator_taylor_series(
                    temp,
                    order=order,
                    laplace=laplace,
                    transmission_function=transmission_function,
                    wavelength=wavelength,
                    thickness=thickness,
                )
            ) / i

        waves += temp
        temp_amplitude = xp.abs(temp).sum()
        if temp_amplitude / initial_amplitude <= tolerance:
            break

        if temp_amplitude > initial_amplitude:
            raise DivergedError()
    else:
        raise NotConvergedError(
            f"series did not converge to a tolerance of {tolerance} in
{max_terms}terms"
        )
    return waves
```

Figure 25: The multislice exponential series used to calculate the exponent of the 2D Laplacian and transmission operator (6). Might not converge for certain simulation parameters.

```
def propagator_taylor_series(
    waves: np.ndarray | da.core.Array,
```

```
    order: int,
    laplace: Callable,
    transmission_function: np.ndarray,
    wavelength: float,
    thickness: float,
):
    """
    Taylor series expansion of the propagator term in the MS equation.
    Eq.(8) in Ultramicroscopy 134 (2013) 135-143.
    """
    if order < 1:
        raise ValueError("order must be a positive integer and at least 1")

    if order == 1:
        return (
            conventional_operator(waves, laplace, transmission_function,
wavelength)
            * 1.0j
            * thickness
        )

    K0 = 1 / wavelength
    laplace_waves = laplace(waves) / (4 * np.pi * K0)
    series = laplace_waves.copy()
    temp = laplace_waves.copy()

    for i in range(2, order + 1):
        prefactor = (wavelength / (-2.0 * np.pi)) ** (i - 1) * 0.5
        temp = laplace(temp) / (4 * np.pi * K0)
        series += temp * prefactor

    return (series + waves * transmission_function) * 1.0j * thickness
```

Figure 26: Taylor series of the Propagator Corrected Multislice (PCMS) (2).

```
def full_series(
    waves: np.ndarray | da.core.Array,
    laplace: Callable,
    transmission_function: np.ndarray,
    order: int,
    wavelength: float,
    thickness: float,
    override_prefactor: list[float] = [],
):
    """
    Full Taylor series expansion of the MS Eq.(14) in Ultramicrscopy 134
(2013) 135-143.
    override_prefactor used in backscatter call, Eq. (13) in Micron 190
(2025) 103778.
    """
    series = conventional_operator(waves, laplace, transmission_function,
wavelength)
    temp = series.copy()
    for i in range(2, order + 1):
        if override_prefactor:
            prefactor = override_prefactor[
                i - 1
            ]  # Note that the first prefactor always gets skipped and is
always 1
        else:
            prefactor = (wavelength / (-2.0 * np.pi)) ** (i - 1) * 0.5
        temp = conventional_operator(temp, laplace, transmission_function,
wavelength)
        series += temp * prefactor
    return series * 1.0j * thickness
```

Figure 27: Taylor series of the Fully Corrected Multislice (FCMS) (2). Also used for the backscatter operator.

```
# constants and prefactors
K0 = 1 / wavelength
```

```
# Eq. 7 in Micron 190 (2025) 103778.
backscatter = (
    1
    / (2 * np.pi * 1.0j * thickness)
    * (
        full_series(
            waves._array,
            laplace_stencil,
            transmission_function_array_next_slice,
            order,
            wavelength,
            thickness,
        )
        - full_series(
            waves._array,
            laplace_stencil,
            transmission_function_array,
            order,
            wavelength,
            thickness,
        )
    )
)

# 1/k series with custom prefactors
prefactors = [1]
for i in range(1, order + 1):
    prefactors.append(prefactors[-1] * (1 - 2 * i) / (2 * i))
for i in range(len(prefactors)):
    prefactors[i] = prefactors[i] / (1.0j * thickness) / (np.pi * K0) ** i

backscatter *= (
```

```
        1
        / (2 * K0)
        * (
            1
            + full_series(
                waves._array,
                laplace_stencil,
                transmission_function_array_next_slice,
                order,
                wavelength,
                thickness,
                override_prefactor=prefactors,
            )
        )
    )

    # Eq.10 in Micron 190 (2025) 103778.
    waves._array = waves._array - backscatter
```

Figure 28: Part of the full multislice step that calculates the backscattered wave per slice (4).

```
def _back_propagate_backscattered_waves(
    backscattered_waves: Waves,
    potential: BasePotential,
    multislice_step: Callable,
) -> Waves:
    """
    For each slice in the multislice step, a small part of the wave get backscattered.
    This function runs the multislice in reverse for each backscattered wave summing
    them for a final backscattered wave result.
    """

    xp = get_array_module(backscattered_waves.device)
    potential_slices = [
```

```
        slice
        for _, config in _generate_potential_configurations(potential)
        for slice in config.generate_slices()
    ]

    effective_slices = _aggregate_slices_by_exit_planes(
        potential_slices, potential.exit_planes
    )

    num_slices = len(effective_slices)
    if len(backscattered_waves) != num_slices + 1:
        raise ValueError("Wrong shapes")

    # zero intensity in incoming wave
    backscattered_waves[0]._array[:] = 0

    # Go through potential in reverse
    for i in range(num_slices - 2, -1, -1):
        contribution_at_slice = backscattered_waves[i + 1].copy()
        contribution_at_slice.array = xp.conj(contribution_at_slice.array)
        contribution_at_slice, _ = multislice_step(
            contribution_at_slice, effective_slices[i + 1], next_slice=None
        )
        backscattered_waves[i].array +=
xp.conj(contribution_at_slice.array)

    return backscattered_waves
```

Figure 29: Function for reconstructing the full backscatter signal by multislicing all backscattered wave back through the sample and coherently summing all parts (7).

References

[1] D. B. Williams and C. B. Carter, *Transmission Electron Microscopy: A Textbook for Materials Science*. Springer US, 2009. doi: 10.1007/978-0-387-76501-3.

[2] J. M. Cowley and A. F. Moodie, "The scattering of electrons by atoms and crystals. I. A new theoretical approach," *Acta Crystallographica*, vol. 10, no. 10, pp. 609–619, 1957, doi: 10.1107/s0365110x57002194.

[3] U. Kaiser *et al.*, "Transmission electron microscopy at 20kV for imaging and spectroscopy," *Ultramicroscopy*, vol. 111, no. 8, pp. 1239–1246, 2011, doi: https://doi.org/10.1016/j.ultramic.2011.03.012.

[4] R. Egerton, P. Li, and M. Malac, "Radiation damage in the TEM and SEM," *Micron*, vol. 35, no. 6, pp. 399–409, 2004, doi: https://doi.org/10.1016/j.micron.2004.02.003.

[5] C. Sun, E. Müller, M. Meffert, and D. Gerthsen, "On the Progress of Scanning Transmission Electron Microscopy (STEM) Imaging in a Scanning Electron Microscope," *Microscopy and Microanalysis*, vol. 24, pp. 1–8, 2018, doi: 10.1017/S1431927618000181.

[6] W. Q. Ming and J. H. Chen, "Validities of three multislice algorithms for quantitative low-energy transmission electron microscopy," *Ultramicroscopy*, vol. 134, pp. 135–143, 2013, doi: 10.1016/j.ultramic.2013.04.013.

[7] P. Schweizer, P. Denninger, C. Dolle, and E. Spiecker, "Low energy nano diffraction (LEND) – A versatile diffraction technique in SEM," *Ultramicroscopy*, vol. 213, p. 112956, 2020, doi: https://doi.org/10.1016/j.ultramic.2020.112956.

[8] J. M. Zuo and J. C. H. Spence, *Advanced Transmission Electron Microscopy: Imaging and Diffraction in Nanoscience*. New York, NY: Springer, 2017. doi: 10.1007/978-1-4939-6607-3.

[9] J. Chen and D. Van Dyck, "Accurate multislice theory for elastic electron scattering in transmission electron microscopy," *Ultramicroscopy*, vol. 70, no. 1–2, pp. 29–44, 1997, doi: 10.1016/S0304-3991(97)00057-7.

[10] D. A. Muller, E. J. Kirkland, M. G. Thomas, J. L. Grazul, L. Fitting, and M. Weyland, "Room design for high-performance electron microscopy," *Ultramicroscopy*, vol. 106, no. 11–12, pp. 1033–1040, 2006.

[11] J. Madsen and T. Susi, "abTEM: Transmission Electron Microscopy from First Principles," *Open Research Europe*, vol. 1, no. 24, p. 13015, 2021, doi: 10.12688/openreseurope.13015.1.

[12] R. Zekendorf, "Comparing Real-Space and Conventional Multislice in abTEM." Vienna, 2024.

[13] "Materials Data on SrTiO3 by Materials Project," 2020, doi: 10.17188/1263154.

[14] Department of Imaging Physics, "HPC Hardware - QIWEB." [Online]. Available: https://qiweb.tudelft.nl/hpc-imphys/hardware/

[15] G. Chen, Y. He, W. Ming, C. Wu, D. Van Dyck, and J. Chen, "Fast STEM image simulation in low-energy transmission electron microscopy by the accurate Chen-van-Dyck multislice method," *Micron*, vol. 190, p. 103778, 2025, doi: https://doi.org/10.1016/j.micron.2024.103778.

[16] A. Winkelmann, “Dynamical Simulation of Electron Backscatter Diffraction Patterns,” in *Electron Backscatter Diffraction in Materials Science*, Springer US, 2009, pp. 21–33. doi: 10.1007/978-0-387-88136-2_2.