Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Master Thesis

Simultaneous Algebraic Reconstruction Technique for Electron Tomography using OpenCL

submitted by
Beata Turoňová

submitted
June 9, 2011

Supervisor
Prof. Dr. Philipp Slusallek

Advisor
Mgr. Lukáš Maršálek

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement under Oath

I confirm under oath that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, _____ _____

(Datum / Date) (Unterschrift / Signature)

Abstract

Electron tomography (ET) estimates 3D structure of an object from series of 2D transmission images taken from different perspectives. Various methods can be used to obtain the 3D reconstruction. Expansion of Graphics Processing Units (GPUs) made it possible to explore use of iterative algebraic techniques in ET. To these methods belong Algebraic Reconstruction Technique (ART), Simultaneous Iterative Reconstruction Technique (SIRT), and Simultaneous Algebraic Technique (SART). While ART and SIRT are employed in various SW packages for ET, SART seems to be neglected in this area. In this thesis we present GPU-based SART implementation modified for ET, called ET SART. After short introduction to ET, we describe mathematical principles of algebraic techniques and present previous work in this area as well as we introduce the basic concepts behind OpenCL and CUDA. We describe main features of ET SART as well as some practical aspects of its implementation. Finally, we compare ET SART with other methods and present our results. Part of this work is also implementation of wrapper providing unified API for CUDA and OpenCL.

Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Dr. Philipp Slusallek, who made this thesis possible. I also would like to thank my advisor, Lukas Marsalek for the assistance he provided at all levels of the research project.

Contents

Chapter 1

Introduction

Electron Tomography (ET) plays essential role in structural studies of macrocells. It estimates 3D structure of a studied specimen from set of 2D images taken at different perspectives with Transmission Electron Microscope (TEM). Since the images usually have subnanometer resolution, we can study specimens at near-molecular resolution [LFB05]. To achieve that, precise 3D reconstruction technique is necessary. The most common method used in ET is Weighted Back Projection (WBP) [Rad92] since it is not computationaly demanding and delivers results of reasonable quality. Expansion of Graphics Processing Units (GPUs) during last years made it possible to explore itrative reconstruction techniques as an alternative to WBP. To these methods belong Algebraic Reconstruction Technique (ART) [GBH70], Simultaneous Iterative Reconstruction Technique (SIRT) [Gil72] and Simultaneous Algebraic Technique (SART) [AK84]. While ART converges fast but produces very noisy results, SIRT gives reconstructions of high quality but needs significantly more iterations to converge. SART was designed to combine fast convergence of ART with good results obtained with SIRT. Despite that, SART is practically not used in any software for ET and also research seems to be focused mainly on SIRT and ART.

1.1 Objectives

The main goal of this thesis is to implement SART for ET and explore its performance on real data sets. To obtain high quality results we employ exact weighting factors and we also take into account possible non-perpendicularity of the electron beam and the specimen [DSF06]. Neither of which is to our knowledge included in current SIRT implementations. We then compare our results to those produced by SIRT and WBP. The comparison should comprise both quality of the reconstructions and time needed to obtain them.

We implement SART using OpenCL technology [Opea] to provide application portable accross GPUs from different vendors.

As a starting point, we use framework for computed tomography by Michael Kunz [Kun10].

1.2 Overview

The thesis is structured as follows. In Chapter 2 we explain principles of electron tomography. We describe transmission electron microscope along with acqusition of projections and give overview of the reconstruction workflow. Chapter 3 deals with algebraic reconstruction techniques and their existing implementations. In Chapter 4 we describe basic concepts of OpenCL and then compare it to CUDA. We present our SART implementation called ET-SART in Chapter 5 and give more details on practical aspects of the implementaion in Chapter 6. We present reconstructions obtained with ET SART in Chapter 7. Conclusion of this work as well as possibilites of further research can be found in Chapter 8.

Chapter 2

Electron Tomography

Electron tomography (ET) is the technique which aims at estimating a 3D density distribution of an object from series of 2D transmission images taken from different perspectives. This approach is similar to medical Computed Tomography (CT) or Magnetic Resonance Imaging (MRI), but in ET we reconstruct microscopic samples from images with nanometer resolution. Conceptually, we can divide the ET into two phases: 2D image series acquisition and 3D reconstruction. In this thesis, we deal with the reconstruction part, but since the principles of the image series acquisition are essential to understand our choice of the reconstruction algorithm, we briefly describe them in this chapter as well. First, we explain the basic concept of transmission electron microscopy (TEM) and then, we focus more closely on the electron microscope and the image acquisition. We also deal with limitations of TEM. In the last part, we introduce the problem of the 3D density distribution reconstruction casted in the ET context.

2.1 Transmission Electron Microscopy

The purpose of any microscope is to magnify the object under scrunity, so we can see details that our eyes would not normally resolve. In a TEM, we create magnified image of a small specimen by illuminating it with a high voltage electron beam. The reason, why we have to use electrons in order to obtain an image with nanometer resolution, is explained in the first part of this section. Then we explain how the interactions between electrons and the specimen form the image.

2.1.1 Why Using Electrons

If one assumes that all the optical and technical elements of a microscope are optimal then the resolution of conventional microscope is ultimately limited by the wavelength of the illumination particles and the numerical apperture. More precisely, the relationship between the wavelength and the resolution of Visible Light Microscope (VLM) is given by

Rayleigh criterion:

$$\delta = \frac{0.61\lambda}{\mu \sin(\beta)} \tag{2.1}$$

where δ is the smallest distance that can be resolved, λ is the wavelength of the radiation, μ is the refractive index of the viewing medium, and β is the semi-angle of collection of the magnifying lens [DBW09]. For example, for the green light which is in the middle of visible spectrum and has wavelength around 550 nm, the resolution of a good VLM is about 300 nm. Since atom diameters are in range from 50 pm to 520 pm, this resolution is clearly not sufficient for studying materials at atomic levels.

Using electrons instead of photons at least theoretically, alleviates this problem. Similarly to VLM, the relationship between electron wavelength and the resolution of a TEM can be approximated by

$$\delta \approx \frac{1.22\lambda}{\beta} \tag{2.2}$$

where again δ is the resolution, λ is the wavelength of the electrons in the beam, and β is the semi-angle of collection of the electromagnetic lens [DBW09]. Moreover, the wavelength λ of electrons is also related to their energy E. This relationship can be approximated (ignoring relativistic effects) by following equation:

$$\lambda = \frac{1.22}{E^{1/2}} \tag{2.3}$$

where energy E is in electron volts (eV) and electron wavelength λ in nm. Thus, increasing electron energy leads to decrease of its wavelength and thereby to better resolution. For example, for 100 keV electron, λ is about 4 pm.

In fact, the resolution of TEM is limited mainly by technical issues and specimen properties as will be discussed later. Thus, current TEMs are far from reaching the theoretical resolution. The beam energies in current TEMs are in the range from 100 keV to 400 keV and the best achieved resolution to date is 0.05 nm [FEI10]. This resolution could be further improved by special microscope construction which uses multiple illuminations sources or employs multiple detectors placed in different positions [FEI10].

2.1.2 Image Formation in TEM

In TEM, the image is formed due to interaction of electrons with a specimen. Since electrons have dual character, the interaction can be understood in two different ways. Either we consider electrons to be particles and then the interaction with the matter can be described by scattering, or we think of electrons as of waves and then we describe the interaction with the matter through diffraction. In the following text we will use the scattering approach.

Before the electrons hit the specimen, they form an incident beam which has (ideally) uniform intensity over the illuminated area. While traveling through the specimen, some electrons remain unaffected while others are scattered in various directions (see Figure 2.1).

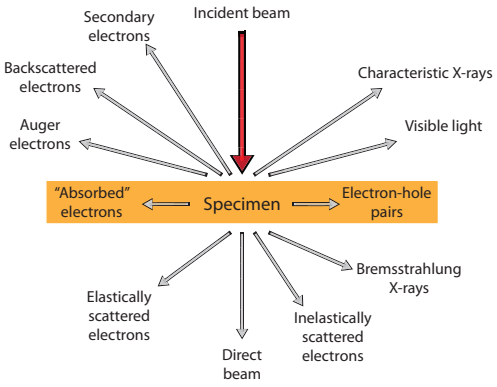


Figure 2.1: Electron Scattering, redraw from [DBW09]

The distribution of electrons emerging from the specimen is therefore no longer uniform. It is the non-uniformity and its spatial distribution what gives us the information about our specimen.

The spatial distribution of electrons can be observed as intensity in the image. Next to direct electrons, also elastically and inelastically scattered electrons contribute to this intensity which is rather undesirable. During elastic scattering, the electrons are scattered away and either do not hit the detector at all or they hit it on other places. In latter case, they do not give us any useful spatial information as we do not know where they come from. During inelastic scattering the electrons lost some of their energy and therefore it is hard to relate intensity on the detector to specimen properties as the detector response is not linear in electron energy. Thus both types of scattered electrons lower constrast in the image and make it harder to correctly interpret the intensity information. Moreover, due tu diffraction issues, they alter the achievable resolution. In current TEMs, we can reduce their influence by using proper objective aperture as will be discussed in 2.2.4. We will see later, that most current reconstruction techniques assume direct electrons are the only source of the signal.

Instead of spatial distribution, we can display angular distribution of the scattering in which case we obtain so called diffraction patterns.

Note, that all secondary signals displayed in Figure 2.1 could be used to form an image of some kind. Many of these signals are for example used in Analytical Electron Microscopy.

2.1.2.1 Ray Tracing for Electrons

We want to use ray tracing to approximate path of the electrons through the specimen, as required by the algebraic techniques. However, ray tracing has been developed for paths of the light and it relies on the fact that photons are not charged and thus do not affect each other. In contrast, electrons are negatively charged and repulse each other when they are too close. Therefore, if we want to simulate path of an electron through the specimen by a ray, we have to be sure it will be in sufficient distance from other electrons so they will not affect its path and vice versa.

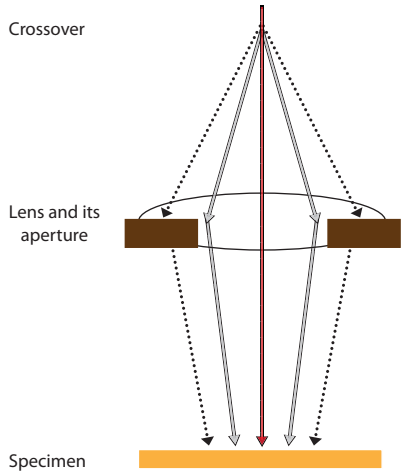


Figure 2.2: Aperture of an electromagnetic lens, redraw from [DBW09]

It has been shown that most electrons transit the specimen one at a time ([DBW09], [FEI10]). The following explanation was given in [FEI10]:

> A typical electron beam has a current of about 10 pA (1 pA = 10^{-12}A). Since 1 ampere is 1 coulomb/sec and the electron has a charge of 1.6 x 10^{-19} coulomb, approximately 60 million electrons per second impinge on the specimen. By their high speed of 200000 km/sec would be the average distance between the electrons over 3 meters.

With only one electron passing through the specimen at a time, it is possible to approximate its path using ray tracing method.

2.2 Electron Microscope

Electron microscope consists of an electron source, system of lenses for a specimen illumination, holder/stage system with the specimen, lenses for imaging and a sensor (see in Figure 2.3). In this chapter we briefly describe all these parts with focus on their influence on image acqusition. Note, all of the mentioned components are placed and operated in vacuum.

2.2.1 Electron Sources

Generally, electron sources could be characterized through their properties such as brightness, temporal coherency, energy spread, spatial coherency or stability. The brightness (the current density per unit solid angle of the source) is the most important one - it influences the resolution, contrast and signal-to-noise capabilities of the microscope. A

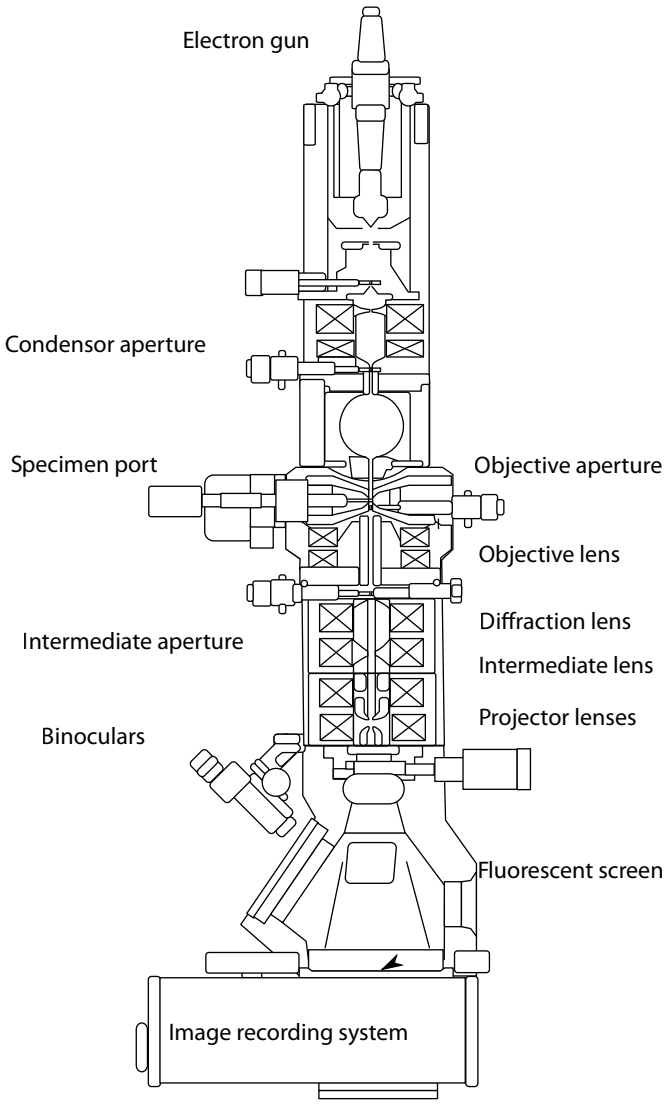


Figure 2.3: Scheme of a TEM, taken from [Tem]

source with smaller size gives higher brightness and better spatial coherency, but is also less stable.

Nowadays, two kinds of electron sources in are used in TEMs: a thermionic source and a field-emission source. The first one produces electrons by heating the source. The second one produces electrons by applying a large electric potential between an anode and the source.

A source is incorporated into a gun, which enables to control and focus the electrons coming from the source. According to the type of the source, we have either thermionic gun or field-emission gun (FEG). The size of the first one is bigger, so it can illuminate large areas on the specimen at relatively low image magnification (<50-100,000x) without losing current density [DBW09]. FEG is more suitable for high-resolution imaging, since

it has source of smaller size and thus produces beam with higher brightness and better spatial coherency.

2.2.2 Electromagnetic Lenses

Electromagnetic lenses form crucial part of TEM as they control electron path the whole way from the source to the viewing plane and are therefore responsible for focus of the beam, magnification of the image, and to a large extent for the information contained in the final image. Therefore, it is useful to introduce at least their basic properties and before we discuss their use in TEM.

First, it is important to understand that the positions of the lenses are fixed. Thus, if we want to change focus or magnification, we have to change the strength of the lenses. It holds, that strong lenses magnify less and demagnify more. Note, that this is in direct contrast to VLM, where we adapt the positions of glass lenses, since we cannot change their strength [DBW09].

We should also be aware of the role of apertures. Using them, we can control the divergence or convergence of the electron paths through the lenses. In that way we can create either more parallel or more convergent beam hitting the specimen. Smaller aperture creates more parallel beam by selecting electrons near the optic axis (see Figure 2.2) and thereby it also lowers the energy of the beam reaching the specimen.

Unlike in other microscopes, in TEM, we gain much more useful information if we are operating out of focus [DBW09]. As you can see in Figure 2.4, the defocus can be of two types. The overfocus arises by increasing the strength of the lens. As a result, the image forms above the image plane. The second defocus is called underfocus. The strength of the lens is decreased and the image is formed below the image plane. Underfocused lenses form more parallel electron beams and therefore were used in older TEMs for creating the incident beam. Note, that term *image plane* has general meaning in this context and has nothing to do with image formation in TEM.

Last but not least, lenses suffer from astigmatism, chromatic and spherical aberration and these defects can negatively influence the quality of the final image. Astigmatism arises due to imperfection in cylindrical symmetry in the systems of lenses. As a result, a circle in the specimen becomes an ellipse in the image. Fortunately, it can be easily corrected [DBW09, FEI10]. Chromatic aberration occurs because the power of the lens varies with the energy of the electrons in the beam. Since the energy spread in the beam does not vary more than 1 eV, the chromatic aberration can usually be reduced. The main problem is spherical aberration caused by the difference between the power in the center of the lens and the power at the edges. In other words, the further off the optic axis the electron is, the more it is bent back toward the axis. As a result, a point object is imaged as a disk of finite size [FEI10]. This lens defect primarily limits resolution of most TEMs [DBW09].

In a TEM the *upper* lens system is responsible for illumination of the specimen. After an electron beam is generated in the gun, the lenses have to transfer it to the specimen. Moreover, the lenses are responsible for the form in which the beam is hitting the specimen. It can form either parallel or convergent incident beam. The convergent beam is useful

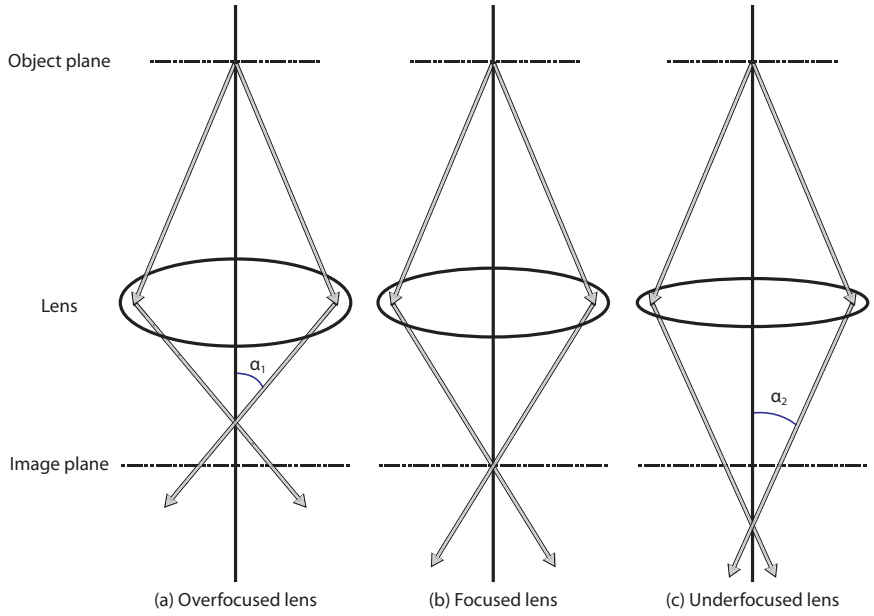


Figure 2.4: Illustration of overfocused (a), focused (b) and underfocused (b) lens, redraw from [DBW09]

when we want to increase the intensity of the beam on a specific area of the specimen. This is needed in some specific fields of electron microscopy such as scanning transmission electron microscopy (STEM) or electron energy-loss spectrometry (EELS). In our case however, we are dealing with the data obtained with parallel beam and therefore we will focus only on that. It is also important to have in mind, that although we say *parallel* beam, the beam hitting the specimen is never exactly parallel. However, the angles between the optic axis and the electron paths are less than 0.0057°, so we can consider such a beam to be a parallel [DBW09].

The basic concept of illumination lens system can be demonstrated using only two lenses (see Figure 2.5(a)). The first condenser lens (C1) demagnify the image of the source (called crossover) and forms an image of it in the object plane of the second condenser lens (C2). The C2 lens is responsible for the form, in which the beam hits the specimen. This is given by the fact that C1 has not only fixed position but also the strength and thus only strength of the C2 lens can be changed. To be concrete, if we want to form parallel beam, we have to underfocus the C2 lens (again look at Figure 2.5(a)). As mentioned before, we can also use the C2 aperture to make the beam even more parallel at the expense of total number of electrons.

The practical realization of this principle is of course much more complicated. In older TEMs, at least three lenses had to be used to achieve desired effect. Modern TEMs use even more complex electron optics called the condenser-objective (c/o) lens system. In such a system, the specimen is located in the objective lens. For us, the most important change when using c/o system is the defocus: in order to form parallel beam, the lenses have to be overfocused [DBW09].

2.2.3 Holders and Stage

Holders are used to insert a specimen into the TEM stage. There are two basic types of the holders. The first one is called the top-entry holder. It uses small specimens, but is detached from outside world while using the microscope. The second type is the side-entry holder. It enables to use larger specimens, but the specimens are attached to a long lever arm which connects them with the outside world. Thus, it is more unstable. Despite this, side-entry holders are used more. (In the future, the side-entry holders should leave the specimen in the stage and only the computer-controlled stage should manipulate it.)

There are also special types of holders allowing changing the specimen during the observation in the TEM. These are called in-situ holders. There is e.g. the heating holder, cooling holder, straining holder, etc. For cryo electron microscopy, the cryo-transfer holder is important. It permits to insert the specimen prepared at cryogenic temperatures into the TEM without water vapor.

The holder/stage system is responsible for translating, rotating and tilting the specimen. By rotating, we mean that the specimen rotates about the optic axis, but remains in the plane perpendicular to it. While tilting, the angle between the specimen and the optic axis is changing. The more tilted is the specimen, the longer is the path of an electron through it. Thus, the tilt angle is limited, which is a major limitation for ET where projections from all perspectives are desirable.

2.2.4 Imaging Lenses

The role of the imaging system is to collect the electrons transmitted through the specimen, create either diffraction pattern or image out of them, magnify it and project it onto the detector (see Figure 2.5(b)). Since diffraction patterns are not used in ET, we focus only on the image creation.

The most important role play the objective lens and its aperture. This lens collects the electrons transmitted through the specimen and forms the image out of them. The size of the objective aperture (placed into the back-focal plane of the objective lens) determines the collection angle. Thus, if the size is small, then only electrons near the optic axis will form the image. With wider aperture we collect also more distant electrons. In that way, we could obtain more information but the distant electrons make the spherical aberration of the objective lens more apparent and thereby lower the quality of the final image. In other words, the objective aperture directly influences the resolution of the image.

The rest of the imaging lens system is responsible for image magnification and projection onto the detector.

2.2.5 Sensors

TEM enables us to observe the specimen in real time and also record the image of it. For the real-time imaging, we need some sort of viewing screen, which can display electron intensity as light intensity, because our eyes are not sensitive to electrons. In older TEMs

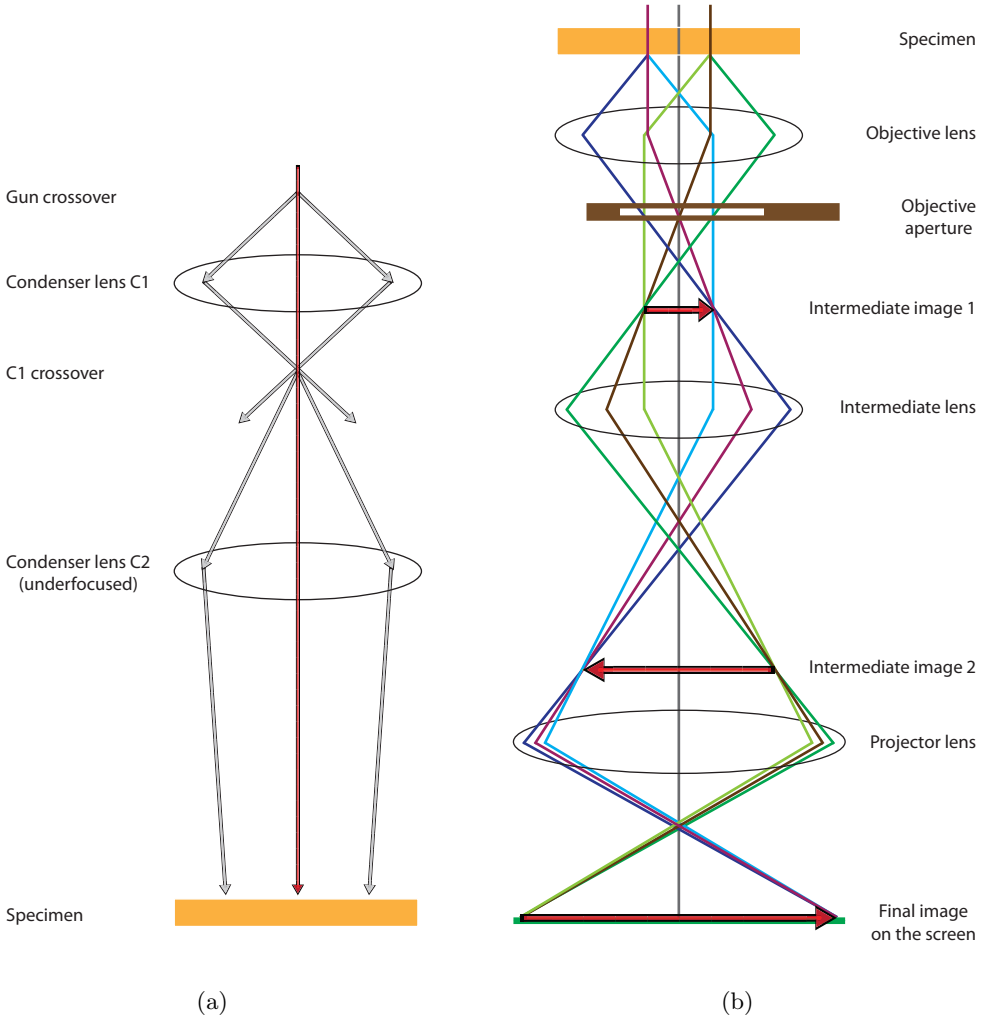


Figure 2.5: Illustration of illumination system (a) and imaging system(b) in TEM, redraw from [DBW09]

fluorescent screen is used, while modern TEMs use flat-panel computer display.

For recording the image, either film camera is used (again, mostly in older models) or CCD (charge-coupled device) camera. Generally, CCDs store charge generated by light or electron beams. Nowadays, image detector elements in CCD cameras are covered with a scintillator. It is responsible for converting incident electrons to light, which then creates charge. Unfortunately, the scintillator is also responsible for some loss of resolution and worse contrast. Recently, direct electron detectors have been introduced, that should improve significantly both image resolution and contrast [FEI10].

Digital recording of the images makes it possible to apply various post-processing methods on the data in order to suppress or enhance information. Any post-processing has to be done very carefully; otherwise some important information could be lost or interpreted in

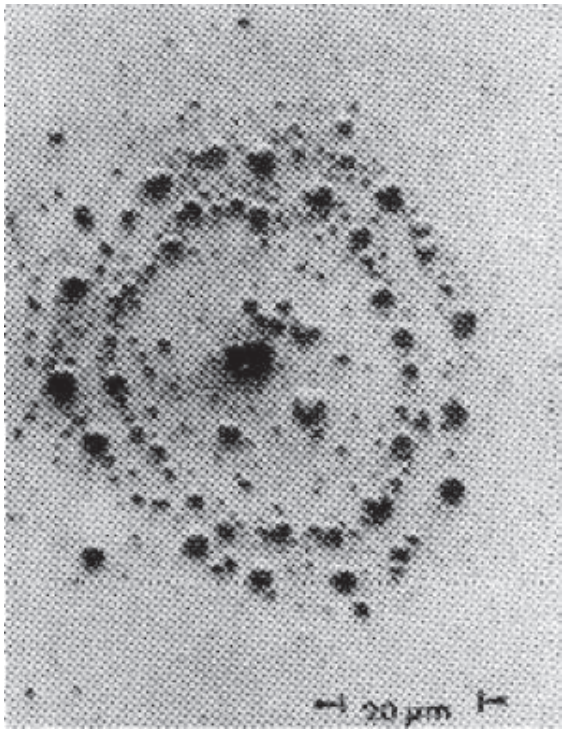


Figure 2.6: Relativistic electron beam damage patterns produced on polystyrene witness foils in the laboratory, taken from [Bos86]

a wrong way.

2.3 Limitations of TEM

2.3.1 Beam Damage

As we have already mentioned, all components of TEM are operated in vacuum including the specimen. With that in mind the specimen preparation becomes a non-trivial task since proper fixation is needed. Nowadays, two techniques are used: chemical and cryo fixation. The chemical fixation can alter the specimen (e.g. change its structure or chemistry) which is undesirable. The cryo fixation preserves internal structure by freezing the specimen to liquid nitrogen temperatures [LFB05] but it is technically more demanding. Moreover, it puts constraints on energy of the electron beam since the beam heats up the specimen and if its energy is too high the specimen unfreezes and vaporates.

The possible vaporization is not the only reason, why the energy of the beam should not be too high. Electrons are type of ionizing radiation and as such they could damage the specimen (and in some cases also the sensor). To avoid this, the total dose received by the specimen has to be minimized. The decrease of the beam energy unfortunately increases noise in the obtained images. For example in biological TEM only few hundred electrons

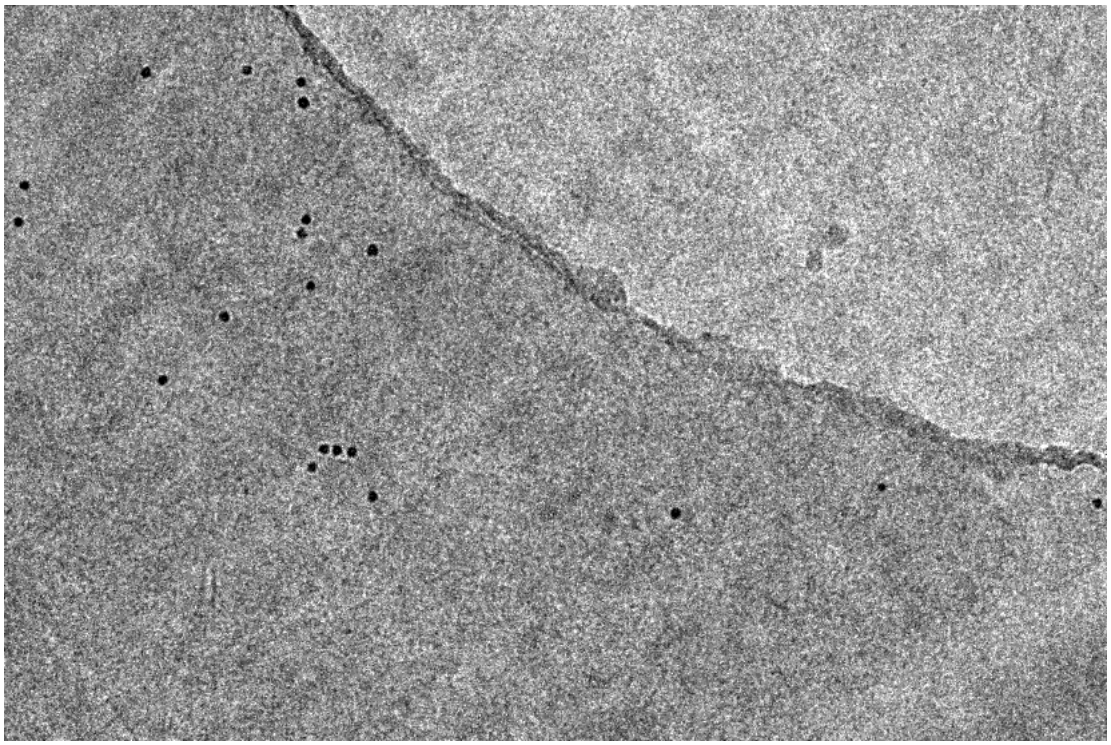


Figure 2.7: The projection of S. cerevisiae obtained by cryo-sectioning, taken from [nki]

on nm^2 are hitting the specimen [DBW09] and you can see the image acquired with such a low energy in Figure 2.7.

2.3.2 Small Sized Specimens

A major limitation of the TEM is the small size of a specimen. The diameter of a specimen is usually around 3 mm and so we can look only at a small part of the sample at one time. The bigger problem, however, is a thickness of a specimen. In a TEM, a specimen has to be thin enough to be electron transparent. In other words, sufficient amount of electrons have to come through the specimen in a reasonable time, so we can obtain any interpretable information. Therefore specimens thinner than 100 nm should be used [DBW09], [FEI10], which makes their preparation even harder. Moreover, thinner specimen have lower contrast.

2.3.3 Incomplete Projection

Another limitation of the TEM is the limited tilt angle. When tilting the specimen around the single axis, the trajectory of the transmitted electron through the specimen is getting longer. The electron could be then scattered more times and it could also lose all its energy. In other words, at high tilt angle, the specimen becomes non-transparent for electrons. Another problem is the mechanics of the holders, e.g. it is problem to keep the specimen

in place while not obscuring it. For that reasons, we cannot currently tilt the specimen through full 360 degrees. In fact, usually the specimen could be tilted only from -70° to +70° which introduces problem of the missing wedge.

The development of the dual axis microscopy could partly overcome this problem. It allows rotation of the specimen also about a second axis perpendicular to the first one and thereby reduces the problem of missing wedge to a missing pyramid. Even more promising is so called tomography holder - new design of the holder that enables tilting the specimen through 360 degrees. The problem with low-contrast images near ±90° howevere remains.

2.4 Reconstruction in Electron Tomography

Now, after the introduction to image acquisition using a TEM, we describe the second part of the electron tomography - the 3D reconstruction process. The standalone reconstruction technique forms only one part of this process and many things have to be done before it. Therefore, we first make a brief overview of the whole reconstruction pipeline and then we focus on the reconstruction methods used in ET. We also mention existing programs used in this field.

2.4.1 Reconstruction Pipeline

We can divide the pipeline into four main parts: pre-processing, alignment of the tilt-series, the reconstruction and post-processing. The realization of each part might differ depending on the used software. Since our data was processed in the free software package for electron tomography called IMOD [KMM96], we describe each part in the way it is done in this package. The complete IMOD pipeline is shown in Figure 2.8.

Pre-processing

The preprocessing step should eliminate those things in the image sequence that could complicate the alignment or the reconstruction. In IMOD, peaks with high frequency (X-Ray peaks) are removed based on mean values around them. We can also exclude *bad* projections from further processing.

Alignment of Tilt Series

While tilting, the specimen is shifted and thus the images in the obtained sequence are not aligned with the respect to each other. Therefore, the correspondences between the images have to be found first and the tilt-series has to be aligned according to them. The alignment should be done precisely since its quality directly influences the quality of the reconstruction.

In IMOD, a coarse alignment is done first. The projections are binned to obtain lower resolution and then cross-correlation is used to compute the rough alignment.

Then we choose one projection (usually with low tilt angle) and select set of fiducial markers. The markers are then tracked in the rest of projections and we obtain so called fiducial model in which selected markers are linked together through all projections in which they appear. Alternatively, we can use patch tracking to obtain the model without selecting the markers first. In both cases the tracking is far from perfect and we have to correct the model manually.

Once we have precise fiducial model, we can use it to compute fine alignment.

Based on this alignment, rough reconstruction is made. If fiducial markers in the reconstruction have oval shape, we know that the alignment is good. If the markers has rather S or C shape, the alignment is not good enough and we should go few steps back and try to improve our fiducial model to obtain better alignment.

Once we are satisfied with the rough reconstruction, we can generate final alignment in full resolution.

Reconstruction

The goal of reconstruction step is to estimate 3D volume out of the set of aligned projections. We provide detailed description of reconstruction techniques in the following chapter and therefore here we should only mention, for complete overview of the IMOD reconstruction pipeline, that by default the reconstruction step in IMOD is done by Fast Fourier Summation [SMB03], a variation of WBP algorithm.

We want to implement reconstruction method which could be integrated into this pipeline replacing current reconstruction step. Thus, we deal in the rest of this thesis only with

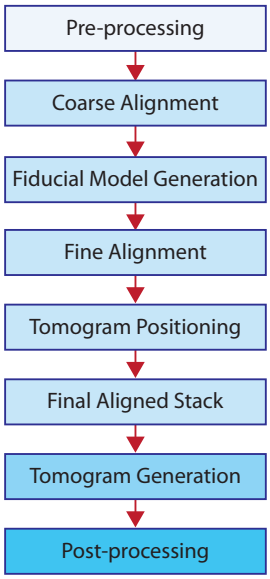


Figure 2.8: The recosntruction pipeline of IMOD

reconstruction techniques.

Post-processing

In post-processing step we can rotate, trim, flatten or squeeze the volume. We can also convert the data type in which the reconstruction is stored. By default, the data type is the same as by projections. We can also use anisotropic filtering to suppress noise.

We have now described the complete way from image set acqusition to 3D reconstruction and its post-processing. What remains is analysis of obtained 3D model. We do not deal with it in this thesis but it is essential to know, that it is very important part of ET workflow.

2.4.2 Software

There exist a lot of programs for handling data from electron microscopy. Here, we mention most commonly used packages for electron tomography (with focus on reconstruction methods).

We have already introduced IMOD and mentioned that it primarily uses fast Fourier summation for the reconstruction. The SIRT is also available but only in a form of a script without user interface and therefore use of it is very difficult. In general, IMOD offers lot of options but with cluttered and hard to orient-in user interface. On the other hand, it provides very detailed logging options that allow to precisely follow what exactly has happend in the reconstruction. Thsi is especially useful when given input does not generate satisfactory reconstractuons.

Bsoft [HCWS08] is another widely used free package for ET. It provides routines for all steps in the reconstruction pipeline. In contrast to IMOD, the methods are more automatic and require less user interaction (which is not necessarily better). They developed own reconstruction technique similar to the one used in IMOD.

SPIDER (System for Processing Image Data from Electron microscopy and Related fields) [FRP+96] was originally developed for single particle analysis but nowadays offers tools useful also for ET. The provided reconstruction techniques are WBP and SIRT.

TOM software toolbox [NFL+05] provides unified-platform for all ET phases: image set acqusition, alignment, reconstruction and analysis (see Figure 2.9). The reconstruction method used in this toolbox is WBP.

Fei [FEI10] offers also software for whole ET workflow called Xplore3D^{TM} [Xpl]. It consists of modules corresponding to image set acqusition, reconstruction and visualisation. The reconstruction module is called Inspect3D^{TM} and provides WBP, ART, and SIRT.

DigiECT [Dig] is the commercial program developed specially for electron tomography. It does the reconstruction using SIRT, SART and also ordered-subset SART (OS-SART) [WJ04]. As far as we now, DigiECT is the only application which offers SART as the reconstruction technique for ET. However since it is commercial, there are no details of the implementations or results of the reconstructions available.

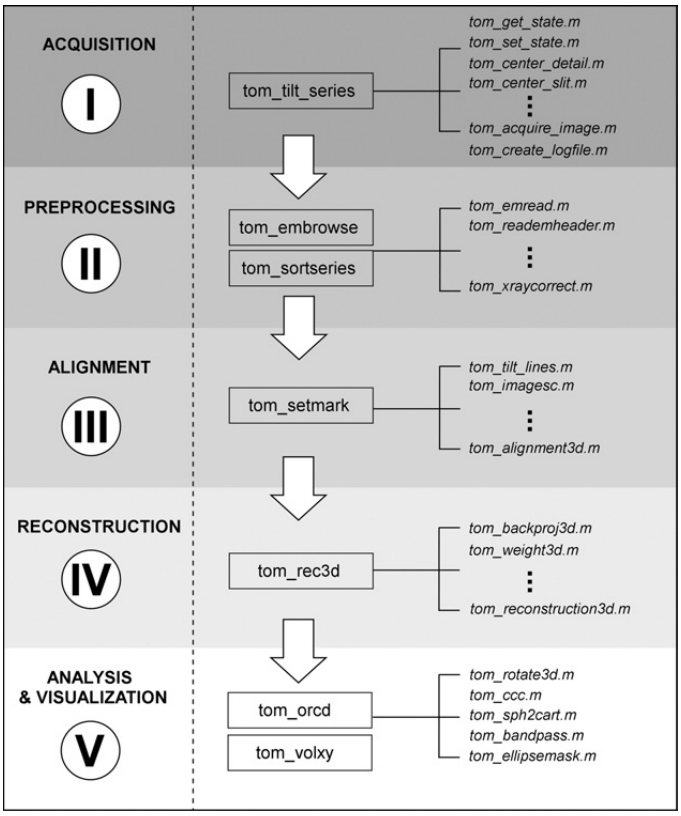


Figure 2.9: Complere electron tomography workflow in TOM, taken from [NFL+05]

All these packages provides routines for whole ET workflow. Among them, there are lot of specialized programs that perform only some of the tasks in the ET pipeline (e.g. Alignator [CDSAAF10], XMIPP [SMVM+04]). Specifically there are lot of tool for analysis (e.g. Amira [PMF08]).

Chapter 3

Algebraic Reconstruction Techniques

We can classify algorithms for reconstruction in electron tomography in two groups: the transform methods and series expansion methods. The transform methods are conceptuallz based on the Fourier central slice theorem which relates the Fourier coefficients of object projections to the Fourier coefficients of the object itself. These techniques perform well in cases where the complete set of projections uniformly distributed over 180° or 360° is available. Moreover, the transform methods are very sensitive to noise. Thus, they are not well suited for noisy incomplete projection set obtained in TEM. Despite this, the weighted backprojection methods (WBP) [Rad92], which belong to this category, present standard reconstruction technique in ET. The reason is simple - the algorithm is not computationally demanding and it can deliver reasonable results in acceptable time.

To the second category belong algebraic reconstruction techniques, like Algebraic Reconstruction Technique (ART) [GBH70], Simultaneous Iterative Reconstruction Technique (SIRT) [Gil72] and Simultaneous Algebraic Reconstruction Technique (SART) [AK84]. Unlike transform methods, algebraic methods do not require complete set of uniformly distributed projections for precise reconstruction and they are also more stable under noisy conditions [MHC98, FLR$^+$02]. Furthermore, they allow using a priori information in reconstruction process [SfBB96]. However, algebraic techniques are computationally very demanding and until recently, they were not able to compete with transform methods. The development in parallel programming and GPUs has changed that and nowadays the algebraic methods present a viable alternative to WBP. In this chapter, we explain the basic concepts behind algebraic methods and explain the difference between ART, SIRT, and SART. The rest of this chapter is devoted to previous work concernig these methods.

3.1 Algebraic Methods

First algebraic method for the reconstruction of 3D objects from a set of projections was the Algebraic Reconstruction Technique (ART) proposed by Gordon, Bender and Herman in [GBH70]. This method formulates the reconstruction problem as system of linear

equations. By solving the system using an iterative method we obtain the reconstructed 3D object. The reconstruction methods based on ART, namely the Simultaneous Iterative Reconstruction Technique (SIRT) and Simultaneous Algebraic Reconstruction Technique (SART) model the problem using the same system of equations but they propose different iterative method for its solution. Therefore, we first derive the system of linear equations and then present the solutions for this system proposed in ART, SIRT, and SART.

3.1.1 Mathematical Background

We have a projection set of an object taken from a different perspective. The rays come from a source, transit the object and impinge on the detector. Each pixel of the detector thus contains the information about the object density at the location where the object was traversed by the rays contributing to that pixel. In the continuous space we can represent this by the following equation:

$$\int_{\Re} \mu(x, y, z) ds = p \tag{3.1}$$

where p is a pixel value and μ describes the object and its density distribution. Of course, the continuous representation is not very practical and thus we rather express the same situation in discrete space. For that we approximated the object by three-dimensional grid containing N elements called voxels denoted as v. The value of the pixel p is then given as

$$w_1 v_1 + w_2 v_2 + w_3 v_3 + \cdots + w_N v_N = p \tag{3.2}$$

where w_j is weight of the contribution of voxel v_j hit by the ray to the pixel value. Earlier, the ray was considered to have some diameter [GBH70] and the weight w_j represented the fractional volume of the voxel v_j intercepted by the ray. Currently, we often consider the ray being the line and the weighting factor w_j approximates the traversal length of the ray through the voxel v_i [Mue98].

Extending the Equation 3.2 for all pixels in all projections we obtain the following system of linear equations:

$$\begin{aligned}
w_{11} v_1 + w_{12} v_2 + w_{13} v_3 + \cdots + w_{1N} v_N &= p_1 \\
w_{21} v_1 + w_{22} v_2 + w_{23} v_3 + \cdots + w_{2N} v_N &= p_2 \\
w_{31} v_1 + w_{32} v_2 + w_{33} v_3 + \cdots + w_{3N} v_N &= p_3 \\
&\vdots \\
w_{M1} v_1 + w_{M2} v_2 + w_{M3} v_3 + \cdots + w_{MN} v_N &= p_M
\end{aligned} \tag{3.3}$$

that could be also expressed using the matrix notation:

$$WV = P \tag{3.4}$$

where W is a matrix of the size $M \times N$ containing the weights, V is $N \times 1$ matrix representing the unknown voxels and finally P is the $M \times 1$ matrix storing the pixel values.

To obtain the reconstruction of $\mu(x, y, z)$ we have to solve the given system. Since the number of unknown voxels N is usually larger than the number of known pixels M, the system is underdetermined and furthermore both N and M are very large (e.g. for data from ET N is about $2048^2 * 300$). Thus, we cannot use the matrix-inversion based types of algorithm to solve it. The iterative methods, on the other hand, are suitable for the large underdetermined system of linear equations. To be concrete, ART algorithm uses the iterative method proposed by Kaczmarz in [Kac37].

3.1.2 Principle of Iterative Techniques

We have presented mathematical point of view on algebraic techniques and now we present their algorithmic description. We first provide common algorithm for ART, SIRT and SART and then we brifely describe each of this method separately to enhance the differences between them as well as their pros and cons.

The pseudo code for general algorithm is shown in Table 3.1.2. We denote the initial estimate of the volume as V^0 and until we reach given termination criterion we iterate over all pixels p_i from all projections P. The pixels are divided into m subsets S and for each subset we compute new estimate of the volume. The update of the volume can be split into three phases: forward projection, compare step and back projection. In forward projection we simulate the acqusition of real projection. For each pixel we generate one ray and trace its path through the volume gathering value and weight of each voxel hit by the ray. In that way we obtain so called virtual projection for each pixel (see Figure 3.1). In the correction step we compute for each pixel the error of its virtual projection towards its real value. Finally, in back projection we correct value of each voxel according to calculated errors of all pixels that contribute to given voxel. After processing all pixels from given subset we obtain new estimate of the volume. In other words, the volume estimate V^k, where k denotes number of updates, is obtained as a function of V^{k-1} and

```
Initialize volume
Split pixels into subsets
Until converges repeat
    For all subsets do
        For all pixels from subset do
            Forward projection
                %compute virtual projection
            Compare step
                %compute error
            Back projection
                %update volume according to the error
```

Table 3.1: General algorithm for algebraic techniques

subset S. Keep in mind, that in general one *update* does not form one iteration of the algorithm.

The way in which the are divided into subsets forms the diferrence between ART, SIRT and SART.

ART

In ART one subset S contains only one pixel. The volume is thus updated after one pixel is processed. For a voxel v_j we can describe the update step by the equation:

$$v_j^{k+1} = v_j^k + \lambda \frac{p_i - \sum_{n=1}^{N} w_{in} v_n^k}{\sum_{n=1}^{N} w_{ij}^2} w_{ij} \tag{3.5}$$

where λ is a relaxation factor from $(0, 1]$.

The ART is relatively fast but the quality of the results is not that high. Reconstructions usually suffer from salt and pepper noise and also contain some artifacts [AK84].

SIRT

To improve the quality of the reconstructions simultaneous iterative reconstruction technique (SIRT) was presented by Gilbert in [Gil72]. In the SIRT one subset S contains all

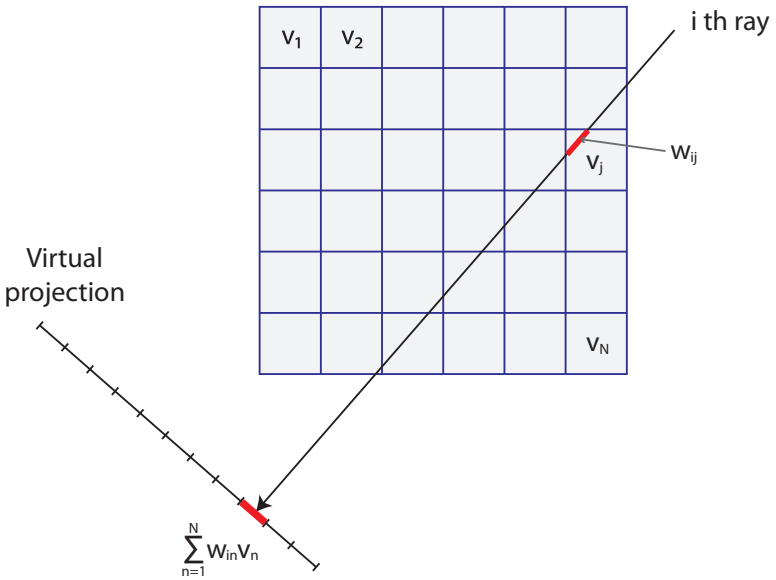


Figure 3.1: Scheme of forward projection step

pixels from all projections. Thus, the volume update corresponds to one iteration of the algorithm and for one voxel can be expressed as:

$$v_j^{k+1} = v_j^k + \lambda \frac{\sum_{p_i \in P} (\frac{p_i - \sum_{n=1}^{N} w_{in} v_n^k}{\sum_{n=1}^{N} w_{in}}) w_{ij}}{\sum_{p_i \in P} w_{ij}} \tag{3.6}$$

Compared to ART, the results of SIRT are better but the convergence is significantly slower.

SART

Andersen and Kak presented simultaneous ART (SART) in [AK84] as a compromise between ART and SIRT. In SART, one subset S contains all pixels from one projection P_ϕ. For a voxel v_j this update can be expressed as

$$v_j^{k+1} = v_j^k + \lambda \frac{\sum_{p_i \in P_\phi} (\frac{p_i - \sum_{n=1}^{N} w_{in} v_n^k}{\sum_{n=1}^{N} w_{in}}) w_{ij}}{\sum_{p_i \in P_\phi} w_{ij}} \tag{3.7}$$

where $\sum_{n=1}^{N} w_{in} v_n^k$ sums all weighted voxel values that contribute to pixel p_i^k and $\sum_{n=1}^{N} w_{in}$ sums the weights of these voxels, p_i denotes corresponding pixel from actual projection P_ϕ and $\sum_{p_i \in P_\phi} w_{ij}$ represents all weights in voxel v_j. By using this update strategy SART combines best of both previous techniques - it provides fast convergence of ART with the results comparable in quality to SIRT [KS88].

3.2 Previous Work

We focus on previous work in the area of electron tomography. Numerous literature exists in other application areas (like CT) but this is out of the scope of this thesis. For survey on foundations of algebraic techniques and their early parallelization efforts we refer reader to [Kun10].

3.2.1 Algebraic Techniques in ET

The algebraic techniques for reconstruction of cellular structure were presented in [FCG04]. This work focused mainly on ART using blobs as basis functions. Note, similar approach was already described in [MHC98] but for reconstruction in single partice analysis. In both works, the results obtained with ART were significantly better in compare to WBP results. This ART implementation has been further improved over the years, especially in terms of lowering the time necessary for reconstruction [BCMS+09].

Although sirt is used in a lot of commercial et software packages [KMM96, FRP+96, Dig, AF11], not many scientific publications regarding its efficient implementation are available.

The exception is [XXJ+10] where modification to SIRT, called OS SIRT, was proposed and its results compare to traditional SIRT as well as to SART. According to published results, OS SIRT has the best time-quality performance. However, neither of their implementations takes possible non-perpendicularity of the rays and the tilt-axis into account and we can assume, based on the implementations description, that integration of the additional tilt would slow down the reconstruction process more in the case of OS SIRT than SART since the parallelization of OS SIRT is more dependent on the perpendicularity of the beam.

As far as we know, there exists only one work fully devoted to use of SART in (cryo) ET presented in [WZL09]. The authors pointed out that arbitrary initial volume may lead to slower convergence of the algorithm and proposed Modified SART (MSART) which adds two modification to the original SART. First, instead of using zero volume as an initial guess for the recosntruction, MSART computes backprojection and uses it as the initial estimate which should lead to faster convergence of the algorithm. The second alternation is the addition of an another data-driven weighting factor to Equation 3.7 in order to improve the quality of reconstructions. However, presented results show that authors were using even 100 iterations to obtain reconstructions which is too many considering the fact, that SART was designed to produce results of good quality after only one iteration [KS88] which was also proven empirically [DMF07], [XXJ+10]. Moreover, their implementation does not consider possible declination of the beam.

3.2.2 Long Object Compenstaion

In ET, the specimen under scrunity is longer than detector which introduces vignetting artifacts into the reconstruction. To avoid this, we can reconstruct larger volume and then trimm it as proposed in [SMB03] or we can add special weighting into compare step as suggested in [XXJ+10]. We will discuss the long object problem as well as both mentioned approaches in Section 5.4.

3.2.3 Beam Declination

Reconstruction methods for ET assume the incident beam is perpendicular to the specimen but it was observed in [TF99] that the beam can be slightly tilted. The possible beam declination was briefly mentioned in [SMB03] where slice shifting in combination with interpolation was used to compensate for the beam tilt. The beam declination and its influence on reconstruction quality were studied in [DSF06]. It was shown that the beam tilt should not be neglected in order to obtain geometrically correct recosntructions. According to authors, the specimen should be, in case of beam declination, tilted around axis which was rotated about the declination angle. We deal with this problem more deeply in Section 5.3.

3.2.4 Efficient Implementation of Algebraic Techniques

High Performance Computing (HPC) was always needed to cope with high computational demands of algebraic methods but only the general-purpose computing on graphics processing units (GPGPU) made it really possible to fully explore their potentional and nowadays, even efficient implementations on multicore CPUs are being developed. We briefly present current research in implementing algebraic techiques both on GPUs and CPUs and then we focus on one particular SART application used in computer tomography on which we would like to base the SART for electron tomography.

First GPU implementations of ART, SIRT, and SART were presented in [DMF07] along with pseudocode for SIRT. All three algorithms were written in Cg [cg]. According to presented reuslts, SART yielded the best trade off between quality and speed of the reconstruction. The comparison between this Cg SART implementation and implementation written in CUDA was presented later in [CDMS+08] and it was shown that Cg implementation was faster than the version written in CUDA. However, the speed-up was not high enough to compensate for the higher effort needed to write and debug code in Cg. The implementation of SIRT and SART by Xu et al. presented in Section 3.2.1 was also GPU-based.

The possibility of using multicores CPU to compute the algebraic reconstructions was explored in [BCMS+09] where fast ART implementation for electron microcopy was presented. The algorithm presented here was using blobs instead of voxel grid to approximate the volume. SIRT on multicores was presented in [AF11]. According to provided times, this SIRT implementation even outperforms the GPU-based SIRT presented in [XXJ+10].

The MSART for electron tomography by Wan et al. (introduced in 3.2.1 was implemented on cluster system but the presented times are not comparable with any other methods or platforms since there is no information in this work about the z-dimension of the reconstructed volume.

From the software for ET presented in Section 2.4.2, SIRT in IMOD can be also performed on clusters or GPUs. SPIDER offers the possibility of parallel reconstruction by using clusters but only for WBP, not SIRT. For complete overview of HPC in electron microscopy, we refer a reader to [Fer08].

SART for Computed Tomography

Michael Kunz presented in his master thesis [Kun10] GPU-based Combined Voxel and Ray Driven SART (CVR-SART) for computer tomography. Forward projection, computation of correction step as well as back projection are performed on GPU. In forward projection one ray for each pixel from the actual projection is simulated and its intersection with the volume is computed. The DDA algorithm [dda] is used to obtain values stored in the voxels hit by the ray as well as the intersection lengths since CVR-SART uses lengths of the ray-voxel intersections as the weighting factors. The sum of weighted voxel values along the ray is stored to the corresponding pixel of virtual projection. Each pixel/ray is processed by one thread on GPU. In correction step the error of virtual projection towards the real projection is computed and stored in so called error map. In back projection four

voxels are processed by one thread. All pixels that could possibly contribute to a processed voxel are found and for each pixel at least one ray is simulated whose intersection with the voxel is then computed.

This technique forms the starting point of our work and we modify it and enhance for electron tomography.

Chapter 4

OpenCL

OpenCL^{TM} (Open Computing Language) by Khronos Group is the first open, royalty-free standard for parallel programming across heterogeneous platforms consisting of CPUs, GPUs and other processors [Khr]. Different vendors (e.g. AMD, Apple, IBM, Intel, Imagination Technologies, NVIDIA) decided to support this standard and so applications written in OpenCL should also be portable accross platforms from these vendors.

In this chapter, we first introduce basic concepts of OpenCL. Since CVR-SART is implemented in CUDA, we also compare these two technologies to see, if one of them is significantly better than the other. Finally, we explain why it is convenient to have our implementation both in CUDA and OpenCL and how we can achieve that without using two separate applications.

Note, that information in following text concerns OpenCL 1.1 and CUDA 3.2.

4.1 Basic Concepts of OpenCL

Any technology for parallel programming can be well described by its execution, programming, and memory model. Execution model represents the way how the instructions are mapped onto the hardware and programming model describes the execution model from the programmer"'s point of view. The way data are stored and accessed is described by memory model. In this section we present realization of these models in OpenCL. After that, we briefly mention the domain specific functions. The last part deals with drawbacks of OpenCL. Note, we provide only core concepts behind OpenCL. For complete overview of its features we refer reader to the OpenCL specification [Ope10].

It is essential to know that an OpenCL application is divided in two parts: host program and OpenCL program. The host program uses the OpenCL API to control the devices. The OpenCL program comprises the kernels - functions that run on devices. It is written in OpenCL C programming language which is based on C99 specification with specific restrictions and extensions. In the following text, we refer to the OpenCL program as to a kernel program.

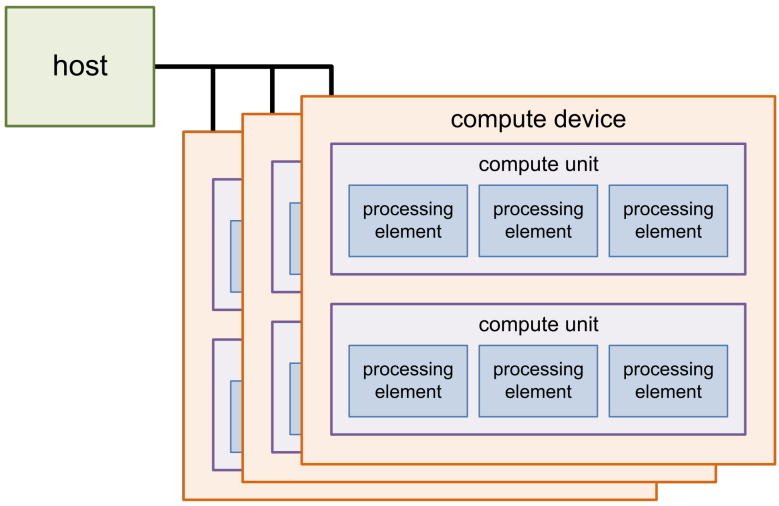


Figure 4.1: OpenCL platform model, taken from [Opeb]

4.1.1 Execution Model

As written in the previous section, the application is split into the host program and the kernel program and thus two corresponding execution models exist. We start with the model for the host program which is executed first.

First, a platform has to be initialized and then, based on the platform properties, so called context is created. The context could be understood as an environment in which any communication with a device(s) take place. The context therefore includes one or more devices on which the kernels are executed. To synchronize commands for these devices, at least one command-queue has to be created for each of them. Furthermore, within the context, all memory objects are also created. Finally, before launching the kernel, the program object has to be created. Usually it takes the source with the kernel implementation and compiles it into the executable but it is also possible to create the program directly from already existing binaries. After setting arguments for a kernel function, everything is ready for the kernel execution.

For execution on device is important to know that each device is organized into compute units (CUs) containing one or more processing elements (PEs) (see Figure 4.1). The execution of each kernel requires setting of an index space. In OpenCL, the index space is called an NDRange, where N refers to the number of dimensions and could be one, two, or three. The size of each dimension is given by number of kernel instances that should be executed. These instances are called work-items and are further organized into the work-groups (see Figure 4.2). The work-items within one work-group are executed concurrently on processing elements of one compute unit. We can identify each work-item within a work-group by its local ID. Analogously, a work-group is identified within an NDRange by its group ID. The position of a work-item within an NDRange is given by its global ID, but it can be also computed from its local ID, corresponding group size and group ID.

Last but not least, the execution between the host and the device is asynchronous.

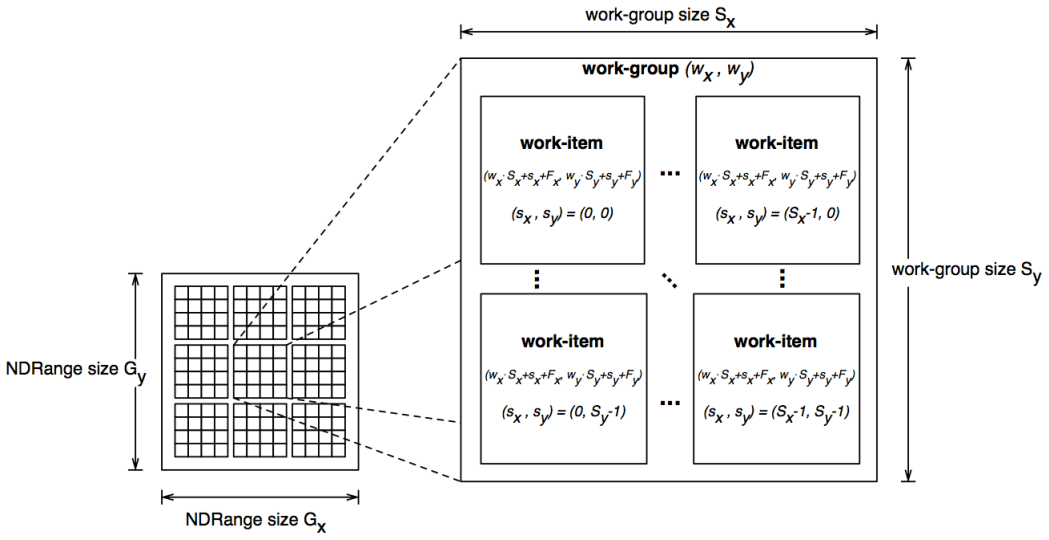


Figure 4.2: OpenCL device execution model, taken from [Ope10]

4.1.2 Programming Model

OpenCL explicitly supports two programming models: data parallel and task parallel. In the data parallel programming same set of instructions runs in parallel on different data points. This is realized in OpenCL by executing each work-item on some element(s) of a memory object. User can specify number of work-items and also dimensions of work-groups into which the work-items are going to be divided. There is also possibility to specify only number of work-items and let OpenCL do the organization into the work-groups.

In the task parallel model different instructions are executed in parallel on the same or different data. In OpenCL we enqueue commands for different devices within one context. On each device a single work-item, independent of any index space, is then executed. Task parallelism does not have so strong support in OpenCL, since the development of this technology was driven by the data parallel programming model [Ope10].

4.1.2.1 Synchronization

Both of the programming models strongly depend on the ability of OpenCL to synchronize the tasks and the memory access both on the host and the device. The synchronization on the host is done by command-queues and events. The command-queues organize the order of all tasks (including memory operations) to be executed on the host. These tasks can be executed, relative to each other, either in-order or out-of-order. By out-of-order execution the events are used to ensure proper synchronization. They can also be used to synchronize commands from different command-queues.

On the device we have to ensure correct access both to global and local memory. For that we can either use proper barriers or atomic functions. OpenCL provides (since the version 1.1) built-in atomic functions for 32-bit signed and unsigned integers and only one

for single-precision float.

4.1.3 Memory Model

The memory model comprises four different types of memory: global, constant, local, and private memory (see Figure 4.3). Not all types can be accessed by both host and kernel program and the same holds for allocation. Therefore, in the following text, we describe these properties for each memory type. After that we also introduce memory objects used in OpenCL.

Global memory is accessible by all work-items from all work-groups. Each work-item can both read from this memory and write to it but cannot allocate it. The dynamic allocation is possible by the host application which also has read/write access to this memory region. Depending on device properties, global memory may be cached [Ope10].

Constant memory is part of the global memory and remains constant during the kernel execution. Again, host program can do the dynamic allocation and has both read and write access to this memory. Kernels can only read from constant memory and any allocation done by them is static. In OpenCL, any global variable in a kernel program has to be in constant memory and has to be initialized by its declaration. In other words, we cannot declare global variable on a device and initialize it from host program.

Local memory is shared by all work-items in one work-group. Objects in this memory can be allocated dynamically from the host, which then has no access to them. Kernels can both read from the local memory or write to it. Moreover, the static allocation by kernels is also possible.

Private memory is used only within one work-item and thus all variables statically allocated by one work-item cannot be used by other work-items. Clearly enough, the host program has neither access to this memory nor possibility of allocation.

For work with the memory two objects are provided in OpenCL: buffer and image. The difference between these two is given by the way how the data in them are stored and accessed. In a buffer, we can store both scalar/vector data type and also a user-defined structure. The format of the data is not changed in any way and in a kernel we can access the data by using a pointer. Within one kernel we have read/write access to the same buffer.

Images have predefined formats according to which the data is stored. In kernels, image elements are converted from the original format to a 4-component vector and can be accessed only using special built-in functions. Direct access using pointer is not possible. Within one kernel, we can either read from the image or write to it, both operations are not allowed and the type of access has to be specified in a kernel argument list. It is worth mentioning that images are either 2D or 3D and while the read/write operations on 2D image as well as read operation on 3D image are standard in OpenCL, the write operation on 3D image requires proper extension.

In the host application the commands concerning memory objects are organized by command-queues. Moreover, each memory operation on the host can be blocking or non-blocking to

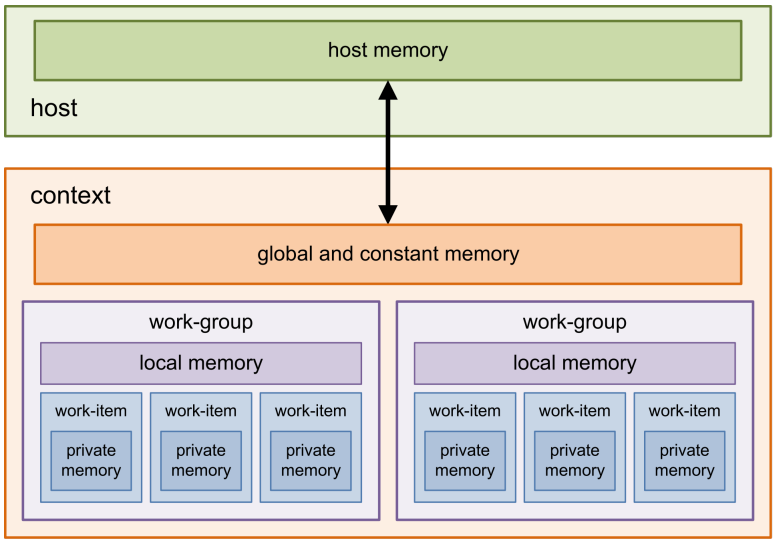


Figure 4.3: OpenCL memory model, taken from [Opeb]

enhance the synchronization possibilities.

4.1.4 Domain Specific Functions

Though the OpenCL is not domain specific it still provides data types and built-in functions specific for graphics. The reasons for that are mostly historical. The purpose of GPUs was to accelerate graphics rendering and so GPU programming languages were graphics specific. They provided vector data types and optimized functions for operations on them since in a computer graphics, many things can be expressed as two, three, or four component structure (e.g. point coordinates or colors). Despite the fact, that OpenCL is general-purpose programming language and it is not intended only for GPU, it offers data types and functions on them useful for graphics algorithms. Namely vector data types, images, and samplers.

4.1.5 Drawbacks

OpenCL was introduced in 2008 so it is rather new technology and as such it has some unresolved issues (for more details see [Ope10], section 9.10.8). These are most likely to be overcome with further development and therefore are not considered to be real drawbacks of this technology. We are thus more interested in shortcomings that arise from the OpenCL design and will probably not be removed in the future.

The most obvious drawback is the user interface. OpenCL is a low-level API written in C and as such it is really not user-friendly. The syntax for basic routines is too complicated, as you can see in following example in which a context and a command-queue are created.

```
1 cl_context                    context;
2 cl_device_id                  *devices;
```

```
3 cl_command_queue          cmd_queue;
4 size_t                    cb;
5
6 // create the context on a GPU device
7 context = clCreateContextFromType(0, CL_DEVICE_TYPE_GPU, NULL, NULL, NULL);
8
9 // get the list of GPU devices associated with context
10 clGetContextInfo(context, CL_CONTEXT_DEVICES, 0, NULL, &cb);
11 devices = malloc(cb);
12 clGetContextInfo(context, CL_CONTEXT_DEVICES, cb, devices, NULL);
13
14 // create a command-queue
15 cmd_queue = clCreateCommandQueue(context, devices[0], 0, NULL);
```

There is C++ Wrapper API for OpenCL in which the basic functions are implemented more conveniently for user. However, the wrapper does not provide all OpenCL calls and contains some bugs (e.g. in reference counting by retaining/releasing objects).

Another weakness of OpenCL is rather poor set of built-in functions for the kernel programming. This is of course given by the fact that OpenCL is as much platform and vendor independent as possible. Therefore, the OpenCL standard contains only basic built-in calls and more complex functions are possible only with some extension. Since extensions are platform and vendor specific by using them we risk losing the portability of the code.

To portability is also related the last (and from our point of view also the biggest) drawback of OpenCL. This technology claims to be portable across different platforms. Thus, when using basic OpenCL C programming language without any vendor or platform specific extensions, it should be possible to run the application on any platform without necessary changes to the code. However, this is not true. Image is a basic memory object in OpenCL and therefore using it should not cause any portability problems. Platforms, however, do not have to support the image structure. If we use the image data type in a kernel on a platform without the support, the kernel will not compile. In that case, we have to rewrite the kernel so it does not use the image structure (we can use buffer instead). In other words, images are not portable across different platforms and therefore also OpenCL is not.

4.2 Comparison of OpenCL and CUDA

Before confronting both technologies, we very briefly introduce CUDA itself. Then, we compare execution, programming, and memory model, focusing mainly on differences between both technologies. Comparison of performance follows. Finally, a summary is presented. We cover main aspects of CUDA and OpenCL. For more detailed description of the differences, see [AMD] or [NVIDIA09].

4.2.1 CUDA

CUDA (Compute Unified Device Architecture) is a whole architecture for parallel programming first introduced in 2006. It is being developed by NVIDIA and is intended for

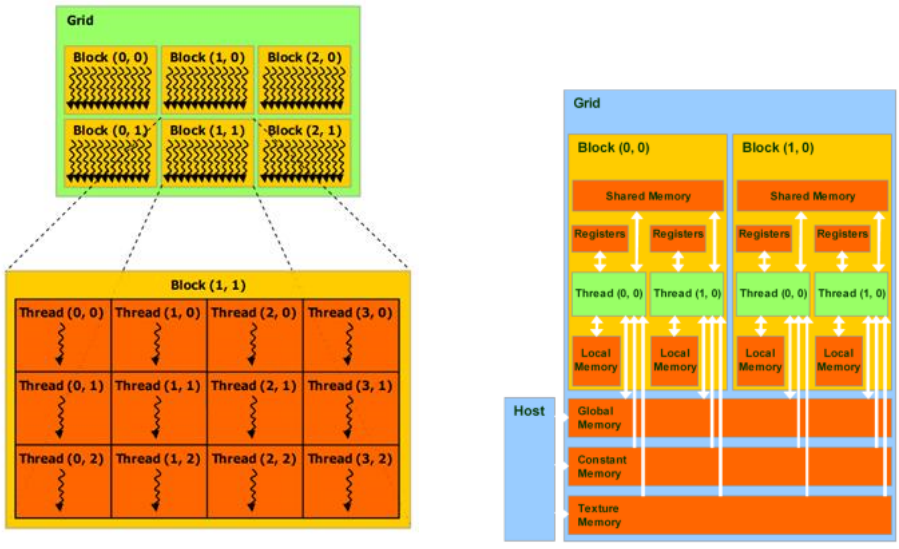
NVIDIA hardware and GPUs only.

Similarly to OpenCL, CUDA application consists of a host program, which uses CUDA Driver API or CUDA Runtime API to access a device, and CUDA device program. The device program is written in C for CUDA which is based on C (again with some extensions and restrictions). As in previous section, we refer to the device program as to the kernel program.

4.2.2 Execution Model

The CUDA execution model for the host program is almost the same as in OpenCL. Creation of context is also required and analogously to platform in OpenCL in CUDA driver needs to be initialized. However, there is a significant difference in kernel compilation. That is, in OpenCL the compilation of a kernel file into the program is done directly by a host program at runtime while CUDA provides standalone NVCC compiler for the kernel code. Thus, it is first compiled either to CUBIN or to PTX file, which is then loaded by a host program at runtime.

The kernel execution differs in terminology, as you can see in Table 4.2.2, but the concept is again practically the same (see Figure 4.4(a)): the space index is called a grid and it is divided into blocks consisting of threads. On each thread, one instance of kernel is executed. In CUDA however, the grid can be only two-dimensional and size of each dimension is given by a number of blocks. There is also nothing like global ID, so we have to compute this identifier manually.



(a) (b)

Figure 4.4: ÇUDA execution model (a) and memory model (b), taken from [NVIDIA09]

OpenCL	CUDA
Work-item	Thread
Work-group	Thread block
NDRange	Grid
Local ID	Thread index
Group ID	Block index
Global ID	Thread ID in a grid

Table 4.1: Different terminology in execution models of OpenCL and CUDA

4.2.3 Programming Model

CUDA programming model differs from OpenCL in only one major thing: it supports only data parallel programming. The realization of the data parallelism in CUDA is then similar to that in OpenCL. The same holds for synchronization. On the host, streams are responsible for organizing the operations. The synchronization in kernel program is done by barriers and atomic functions.

4.2.4 Memory Model

The CUDA memory model is illustrated in Figure 4.4(b). You can see that the main concept is similar to OpenCL but CUDA uses slightly different terminology (summarized in Table 4.2.4) and also provides additional type of memory called texture memory. This memory is part of a global memory, but it is cached and allows direct read-only access. It should be used, when threads in a thread block read locations that are close together (from a spatial locality point of view). In CUDA structure called texture is responsible for allocation of the texture memory. It contains description of how a memory object in CUDA should be treated. It is bind to that object before the kernel execution. The memory object is then not passed as a parameter to the kernel and reading from it is done using the texture. We can bind one texture with one or more memory objects.

OpenCL provides similar concept using samplers and images but there is no special cached memory dedicated to these data types (they are placed in the global memory). We have to copy or map the data into the image and create sampler containing a description of how the image should be treated in a kernel. Both a sampler and an image have to be passed as kernel arguments. We can then read from images or write into them using kernel built-in functions that have sampler as one of their parameters. Similarly to textures, we can use one sampler for more images. Unfortunately, both images and samplers have their limitations. Neither can be part of a user-defined structure and passing them as arguments to other functions in kernel program is restricted in many ways ([Ope10], section 6.8). Furthermore, it was shown in [DWL+10], that copying data to the image is expensive and it pays off only for large data sets.

When it comes to global memory, CUDA has one more advantage over OpenCL. It enables

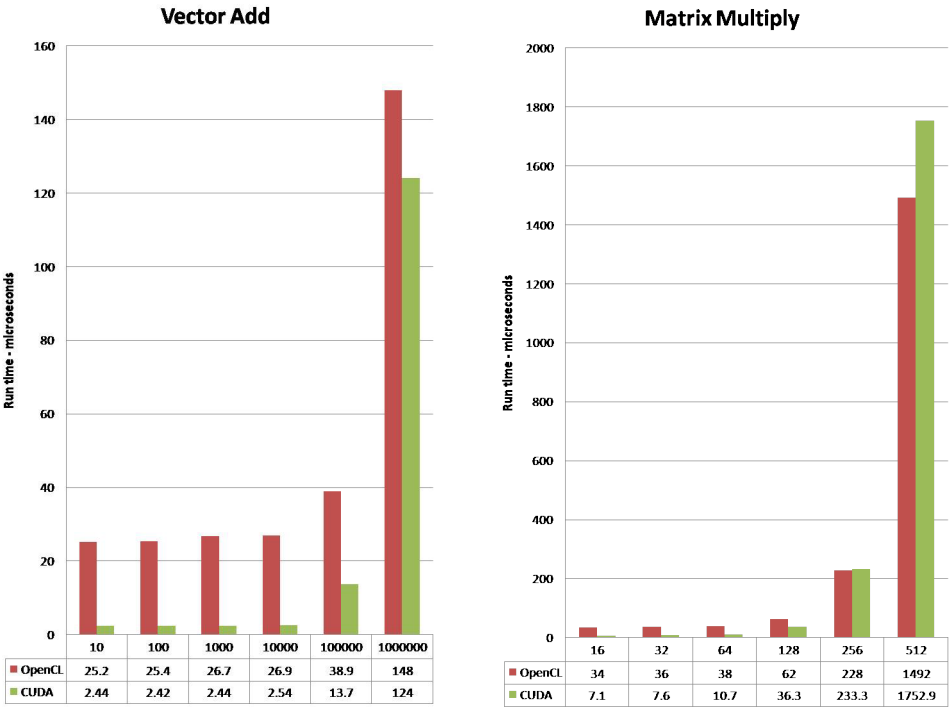
OpenCL	CUDA
Private memory	Local memory
Local memory	Shared memory
Constant memory	Constant memory
Global memory	Global memory
-	Texture memory

Table 4.2: Terminology in memory models of OpenCL and CUDA

declaring global variables in a kernel program and initializing them from the host.

4.2.5 Performance

To make our comparison complete, we have to cover also the differences in performance of both technologies. For that, we summarize results from tests comparing performance of CUDA and OpenCL published here [Coma] and [Comb]. One can see from the results, that there are cases in which CUDA performs better (Figure 4.5(a)) than OpenCL and vice



(a) (b)

Figure 4.5: Performance Comparision of CUDA and OpenCL, taken from [Coma]

versa (Figure 4.5(b)). In all (most?) tests, the performance of OpenCL on smaller data sets was worse. What exactly causes the overhead, if it is kernel compilation or something in OpenCL API, was not further explored in the articles. However, kernel compilation and its influence on OpenCL performance are studied in [DWL+10]. Here it was shown that using other, more optimized compiler leads to better performance results.

4.2.6 Pros and Cons

CUDA is two years older and most advantages over OpenCL arise from that. CUDA is more mature and contains less unresolved issues. It offers more developed interoperability with e.g. OpenGL or DirectX. It also provides better support for developers (e.g. mature libraries, forums, tutorials, code samples). Generally, we can say that both CUDA Driver and Runtime API are more user-friendly. Moreover, CUDA 4.0 release is being prepared nowadays and it will offer C++ support for kernel programming.
The biggest disadvantage of CUDA is its vendor and platform dependency. We can use it only on NVIDIA GPUs. On CPUs CUDA can be run only in debug mode, which is really not optimized for performance. However, recently there are efforts to develope optimized CUDA CPU compilers [Por].

OpenCL is nowadays the only platform and vendor independent framework for parallel programming. It also provides, with certain exceptions, code portable across device from different vendors. These are the most significant advantages of this technology. Most of OpenCL drawbacks arise from the fact that it is a quite new technology and we can assume them to be overcome in the future.

To summarize it, nowadays we cannot say that one technology is generally better than the other since both have their benefits and shortcomings.

4.3 Wrapper

As already mentioned, we have the SART algorithm implemented in CUDA. Since the comparison has shown that CUDA and OpenCL differs, from programmer's point of view, mostly in terminology and the core ideas behind both are very similar, we have decided to use the existing code and rewrite it into OpenCL in order to make the implementation platform and vendor independent.

However, the CUDA implementation performs very well on NVIDIA GPUs and, based on the performance tests results from the previous section, we could not guarantee better or even the same performance of OpenCL. Therefore, it would be convenient to keep both implementations and use always the one more suited for a given situation. However, to have two completely separate applications is not very practical. Especially, since the host program is from the most part the same and differs only in places where functions specific for OpenCL or CUDA are used. This led us to the idea of some wrapper that would contain all technology dependent routines and thereby would allow writing only one host program for both CUDA and OpenCL.

4.3.1 Previous Work

We were looking for this kind of wrapper only to find out that nothing like that exists. There are various guidelines [AMD, NVIDIA09] how to rewrite code from CUDA into OpenCL and vice versa. They usually focus on basic terminology and syntax differences showing direct correspondences between some basic calls. They do not deal with tasks that require more than just renaming few functions.

There also exists a tool for converting CUDA to OpenCL called Swan [Swa]. The original purpose was to translate source-code of kernel programs from CUDA to OpenCL. User responsibility was to rewrite the host code using Swan API and the whole converted application was then executed using OpenCL. Nowadays, Swan enables to run the final application also by using CUDA. However, the CUDA kernel translation is still present. The output of the conversion creates a special file which is then passed to the host code instead of the CUDA kernel (no additional compilation with nvcc is necessary). Of course, host program has to be also rewritten and for that Swan provides special host API. The API wraps both OpenCL and CUDA calls but it is limited only to the basic routines. More demanding differences between both technologies (e.g. setting size of local memory, initializing global variables) are not covered. There is also lack of support for OpenCL images and samplers. Furthermore, kernel compilation for CPUs is not provided and the necessary kernel translation excludes the possible use of C++ templates in CUDA kernels. Last but not least, Swan is implemented only for Linux OS.

4.3.2 Concept of the Wrapper

After not finding any wrapper enabling to write only one independent host code for both CUDA and OpenCL, we decided to implement it by ourselves. The wrapper should provide unified user-friendly API for both OpenCL and CUDA. Furthermore, it should remove the image portability drawback of OpenCL. The realization of the wrapper is described in Section 6.2.

Chapter 5

ET SART

In this chapter we present new efficient implementation of SART for electron tomography called ET SART. We describe its geometric model and also the basic idea behind the parallelization scheme. We then introduce problem of non-perpendicular beam in ET and describe the way, we can solve it. After that, so called long object compensation problem is introduced and solution is proposed. We end this chapter with description of weighting factors used in our implementation.

5.1 Geometric Model

In order to reconstruct 3D volume out of an set of projections we have to simulate the path of the electrons from the source, through the specimen, and onto the detector. In TEM this path is rather complicated and therefore simplified model (see Figure 5.1) is usually used for the reconstruction. This model considers the electron rays to be parallel and perpendicular to the specimen which is then tilted around y-axis. The assumption of the rays being parallel is reasonable considering the very low angle (less than 0.0057°) of deviation from the optic axis. On the other hand, the assumption of perpendicularity is not correct as discussed later in this chapter.

Therefore, in ET-SART we use a more robust model in which we only assume the rays are parallel. This generalization allows us to easily accommodate to arbitrary rotation scheme, as we can expect that even more inacurrances of the simplified model concerning rotation and tilting will be discovered in the future.

5.2 Parallelization

A common way how to paralellize reconstruction in electron tomography is to split the 3D volume into set of 2D slices perpendicular to the tilt-axis and reconstruct these slices from corresponding set of 1D projections [AF11], [Fer08]. In other words, instead of computing one 3D reconstruction, we compute set of 2D reconstructions which are independent and can be therefore performed in parallel. Moreover, in SIRT one can also process all 1D

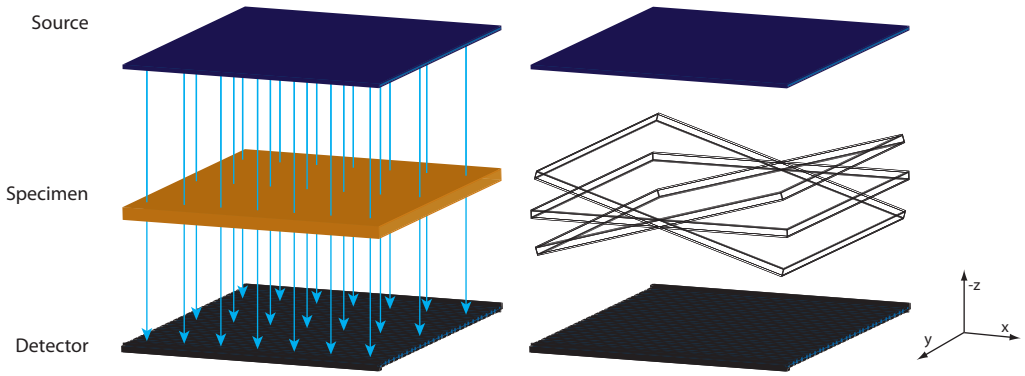


Figure 5.1: Simplified model for tomographic reconstruction assuming parallel rays and beam perpendicular to the tilt-axis

projections corresponding to given 2D slice in parallel since they do not depend on each other. Most of current SIRT imlementations are based on this basic concept [XXJ+10], [AF11].

In our SART implementation we use different approach. Instead of splitting the reconstruction in 2D subsets, we allways process whole projection towards the entire volume and the parallelization is done in forward and back projection. In forward projection we process each pixel of a projection in parallel while in back projection we divide the volume into groups of four voxels and then procces these groups in parallel. This approach is more convenient in the case of imperfect geometry setting as we will see in Section 5.3.

5.3 Tilt of The Incident Beam

In ideal geometric setting the incident beam is perpendicular to the tilt-axis which is axis around which the specimen is rotated. However, it was shown in [DSF06] that the beam and the tilt-axis are not always perpendicular to each other which, when ignored, leads to incorrect results. More precisely, if we do not account the declination of the beam into the aligning process we obtain geometrically incorrect alignment and by ignoring the non-perpendicularity in the reconstruction process the rays will have wrong direction and thus the reconstruction will be inaccurate as you can observe in Figure **??**. The distortions become more apparent with higher angle of the beam declination but high resolution could make them visible also at lower declination angle.

The tilt of the beam, called x-tilt, is not known and has to be computed first. This is usually done during coarse alignment [DSF06, HCWS08] since the final alignment should be done already with incorporated x-tilt information.

To account for x-tilt also in reconstruction, two different approaches were proposed. The first one presented in [SMB03] was designed for FFS used in IMOD. It shifts slices in z direction to cover the reconstructed area, reconstructs these slices, and interpolates

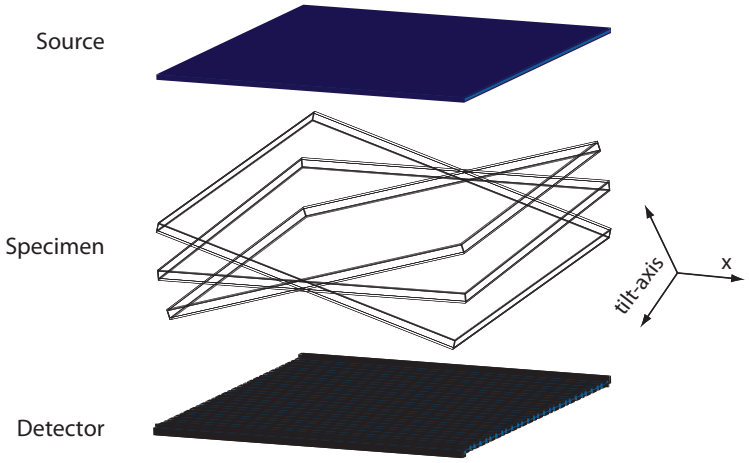


Figure 5.2: Tilting around general axis

between them to obtain an output slices tilted around the x-axis. This method demands a lot of additional computations and provides only approximative solution. According to the second approach presented in [DSF06], the original tilt-axis has to be rotated about the declination angle and the projections should be then rotated around this new axis, as you can see in Figure 5.2. This method gives exact solution and we can easily intagrate it to our implementation.

While the x-tilt is considered in WBP reconstructions (e.g. in IMOD or Bsoft), the SIRT implementations as far as we know do not include it. The reason for it might be the SIRT parallelization described in Section 5.2. Once the beam is not perpendicular to the specimen, we can no longer reconstruct one 2D slice with corresponding set of 1D projections since these projections now contain also iformation from other slices. In other words, 2D sub-reconstructions are no longer prossible. The incorporation of x-tilt would therefore require significant changes to parallelization in current SIRT implementations which would probably slow down the reconstruction. Alternatively, solution similar to the shift approach described above could be used to avoid the change of parallelization scheme but the necessary changes in implementation would remain indispensable. By contrast, all we have to do to include x-tilt in our ET-SART implementation is to rotate the tilt-axis.

5.4 Long Object Compensation

In electron tomography the specimen is significantly larger than the electron beam and the detector and therefore we usually reconstruct only a part of it (see Figure 5.3). This, if neglected, introduces vignetting artifacts in final reconstructionss. These effects are caused by the fact that some projections integrate information also from the part of the specimen outside the reconstructed area. This situation is illustrated in Figure 5.3(a) where pixel p_i contains information corresponding to the ray-volume intersection W_L.

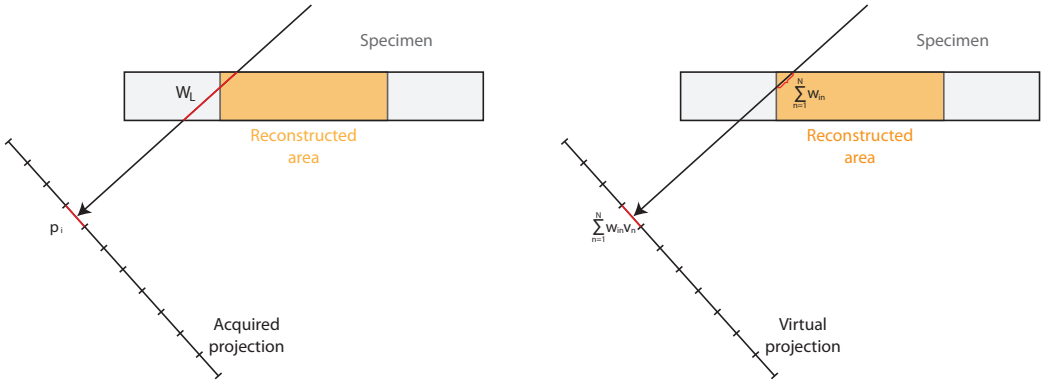


Figure 5.3: Scheme of long object problem

However, in forward projection we sum for each pixel weighted contribution only of those voxels from the reconstructed area as shown in Figure 5.3(b). The error is then computed and normalized as follows:

$$\frac{p_i - \sum_{n=1}^{N} w_{in} v_n^k}{\sum_{n=1}^{N} w_{in}} \tag{5.1}$$

Since pixel p_i contains more information than can be obtained in forward projection, normalizing it with $\sum_{n=1}^{N} w_{in}$ which represents only part of the ray leads to inaccurate correction that introduces vignetting distortions to the final recosntruction.

In order to avoid this we can use larger reconstruction area and then trimm the final volume to the region of our interest as described in [SMB03]. Since the extended area is around twice as long as the original one, this method requires significantly more computations and thus slows the process.

Therefore, we have decided to employ more efficient method proposed in [XXJ+10]. It uses different scheme for correction step according to which pixel p_i should be weighted by W_l:

$$\frac{p_i}{W_l} - \frac{\sum_{n=1}^{N} w_{in} v_n^k}{\sum_{n=1}^{N} w_{in}} \tag{5.2}$$

The compensation suppress the vignetting artifacts without increasing computational time since it only requires computing the length of the ray in the extended volume W_L which in the case of parallel rays is even constant for all pixels from one projection.

5.5 Weighting Factors

As described in Section 3.1.1, the weighting factors corresponds to ray traversal length in voxels. Usually, in forward projection ray marching method [Lev88] is used to traverse the volume in which case the ray length in a voxel is approximated by sample distance. In back projection step, mostly binary weighting is used.

We believe, precise weighting factors can improve the quality of reconstructions and therefore should be computed as accuratly as possible and therefore we use the approach from CVR-SART. In forward projection we use DDA algorithm to traverse the volume and for each ray we compute exact length of its intersections with all voxels hit by this ray. In back projection, we generate for each pixel that could contribute to the processed voxel a ray and and again compute exact ray-voxel intersection.

Chapter 6

Practical Implementation

This chapter deals with practical aspects of our ET SART implementation and is divided in two parts. The first one deals with the changes we had to make in order to extend CVR-SART framework also for data from ET as well as with general improvements of the framework. The second part describes the implementation of the wrapper for OpenCL and CUDA and its incorporation to our application.

6.1 SART

6.1.1 Improvements

The data type in which the volume was stored was unsigned short but all computations on GPU were performed in floats. This lead to unnecessary conversions between these two types everytime one projection was processed and thus to possible lost of data. The volume is now stored in floats and during the whole process only one volume conversion occurs - before writing the reconstructed volume we convert it to the same data type in which the corresponding projections were stored.

The division of GPU index space for back projection was designed only for cubic volumes which is not convenient especially, if we want to split the volume into the subvolumes which are almost never cubic. New division of the index space depends on volume dimensions in voxels and user can optionally set x dimension of group size for NDRange (in CUDA terminology size of the block in the grid).

The DDA algorithm used in forward projection to compute a length of ray-voxel intersection was implemented rather imprecisely which in some cases lead to artifacts in the reconstructions. In order to avoid this, we implemented this part from scratch.

Since the volume is typically larger than memory on graphics cards, it is useful to split it into as many subvolumes as needed to compute forward and back projection on given GPU. Though a framework for subvolumes was implemented in [Kun10] it was not working correctly due to implementation flaws in forward projection. We corrected these errors and also add the possibility for a user to set number of subvolumes, which effectively allow

the user to set maximum amount of used GPU memory.

The order of projections is no longer set randomly but is chosen so that the actual projection has the largest possible angular distance from the previous projection.

Last but not least, some special cases such as division by zero were not treated well (or at all), so we correct that.

In general, we can sum up that the original code of Kunz was overarchitected at its specific purpose and its generalization was less than straightforward process.

6.1.2 Geometry Setup

The geometry setup used in our implementation is illustrated in Figure 5.2. The new coordinate system with origin in the volume center better suits the model for electron tomography described in Section 5.1. However, instead of tilting the volume around the tilt-axis, we rotate the source and the detector around it. The tilt-axis corresponds to y-axis if the x-tilt angle θ is zero. Otherwise, we first rotate y-axis around the x-axis about θ and then use it as the tilt-axis.

6.1.3 New Kernels for ET-SART

In computer tomography we have divergent rays coming out from point source and thus we have to compute direction for each simulated ray within one projection. With parallel rays in electron tomography we can easily precompute the direction for all rays in one projection as well as its inverse and pass both values to GPU as constants. Within the kernels, we have to then compute source position for each ray. Because of these changes we implemented new kernels for forward and back projection called forwardProjectionET.cl and backProjectionET.cl.

In these kernels all ray-voxel intersection tests are done in 3D because of possible x-axis tilt. However, if θ is zero, the rays have constant y-coordinate while transiting through the volume and thus 2D intersection tests are sufficient. Therefore, we also provide kernels containing only 2D tests called forwardProjection2D.cl and backProjection2D.cl. Since we pass the kernels we want to use as parameters to our application, we can easily switch between these two kernel pairs without rebuilding the code.

To compansate for long specimen, we compute the correction factor as described in Section 5.4. On CPU we compute W_L as the intersection of an arbitrary ray corresponding to actual projection with the slab in which the volume is placed. The value is then passed to correctionET.cl kernel in which we compute and normalize the correction factor.

All mentioned kernels are also implemented for CUDA in corresponding *.cu files.

6.1.4 Input Data

As input data we use stack of aligned projections generated by IMOD and stored in MRC format [CHS96, MRC]. The data stack consists of header followed by the projection set. The header contains basic information about the data set such as number of projections, their resolution, the data type they are stored in etc. but it does not include the information about projections tilt angles that are necessary for the reconstruction. Tilt angles can be passed in two ways: either they are included directly in the data set as a part of so called extended header or there is an extra file generated by IMOD during the alignment process which contains list of tilt angles corresponding to aligned projections. If this extra file is available, it is better to use this rather than extended header since the angles contained in the header correspond to the projections before alignment and thus are not that accurate. To further improve the reconstruction, x-tilt (if known) should be passed as a paramater, otherwise it is considered to be zero.

6.1.5 Output Data

Before writing the volume into MRC file (without extended header), we convert it into the same data type in which the projections were stored. To avoid loss of data, we use the entire range of given data type. This scaling is possible since grey values in electron tomography do not have a fixed meaning as in CT. This however makes it impossible to compare absolute values in different reconstructions.

The grayscale of the output corresponds to the density in the specimen - dark areas represents low density while bright shades corresponds to high density. Thus structures in specimen are bright. However, in IMOD the meaning of grey level is reversed and therefore we also provide the possibility to generate output with reversed grayscale.

6.2 Wrapper

The main goal of the wrapper is to provide unified host API for both OpenCL and CUDA. Thus, it has to cover the differences between host programs of both technologies. We can divide these differences into three categories.

The first one includes all equivalent calls which differ only in terminology. Therefore, we had to provide unified calls for them.

The second category consists of tasks that can be implemented in both CUDA and OpenCL but the correspondence is not so straightforward. For example, using textures in CUDA requires setting their properties and binding them to some data. Analogously, if we want to use an image in OpenCL, we have to set its properties by creating a proper sampler and copy the data to the image. In other words, the concept is very similar but the implementation is quite different. To cover this category, the wrapper provides calls corresponding to logical steps of a given task and thereby hiding the implementation differences in both APIs.

The last category includes routines in CUDA that do not have an equivalent in OpenCL and vice versa. However, there are cases, in which we are able to achieve the same results by simulating the missing routines. For example, CUDA provides calls for initializing global variables declared in a kernel program from a host program. We cannot do that in OpenCL since, as already mentioned, OpenCL requires all kernel global variables to be constant and initialized by declaration. Therefore, to achieve the same effect, we have to pass all the variables as constant arguments to kernel function. Obviously, we cannot completely remove this difference but we provide a function for setting global variables. In CUDA this function initializes the global variables using proper CUDA calls. In OpenCL it passes a pointer to a structure containing all the variables to the kernel.

In addition to providing unified API, the wrapper also removes some of OpenCL drawbacks mentioned in Section 4.1.5. Firstly, the wrapper interface is more user-friendly than OpenCL API. Secondly, the portability issue concerning the image structure is solved in the wrapper as we describe in 6.2.2.

6.2.1 Structure

The class diagram of the wrapper is shown in Figure 6.1. We use abstract factory class GPUAbstractionLayer to separate GPU specific calls from the rest of the code. The interface of this class is implemented in OpenCLAbstractionLayer for OpenCL and in CudaAbstractionLayer for CUDA. These two classes are also responsible for context and device initialization.

In addition to GPUAbstractionLayer, client uses its abstract products Buffer, Kernel and MemoryStructure. Buffer contains methods for handling buffer objects in OpenCL and device pointer objects in CUDA. Analogously, Kernel comprises all methods necessary to handle operations on kernels. It is responsible for kernel loading, setting kernel arguments as well as dimensions of index space, and for launching the kernel. The MemoryStructure provides operations on textures in CUDA and on images/buffers in OpenCL which helps solving the image portabily issue in OpenCL as we explain in the next part.

6.2.2 Image Portability

As we have written in Section 4.1.5, the images in OpenCL are not fully portable. We resolve this issue in the wrapper both in the host application and in kernels. In the host application, the image and sampler are part of Image2D (Image3D) class. Object of this class is created only if the device supports the image format. Otherwise, object of Buffer2D (Buffer3D) class is created in which OpenCL buffer is used to store the data in the form of an array.

In kernels we use macros defined in ImageHandler.cl to ensure the protability. In the kernel argument list we replace the image2d_t (image3d_t) and sampler_t with the macro MEMORY2D (MEMORY3D) and within the kernel we use READ_MEMORY2D (READ_MEMORY3D) instead of read_image. In case of image support, the macros are replaced by proper image types and functions. Otherwise, the buffer structure is used instead. The IMAGE_SUPPORT flag is not contained in the kernel but is inserted to it

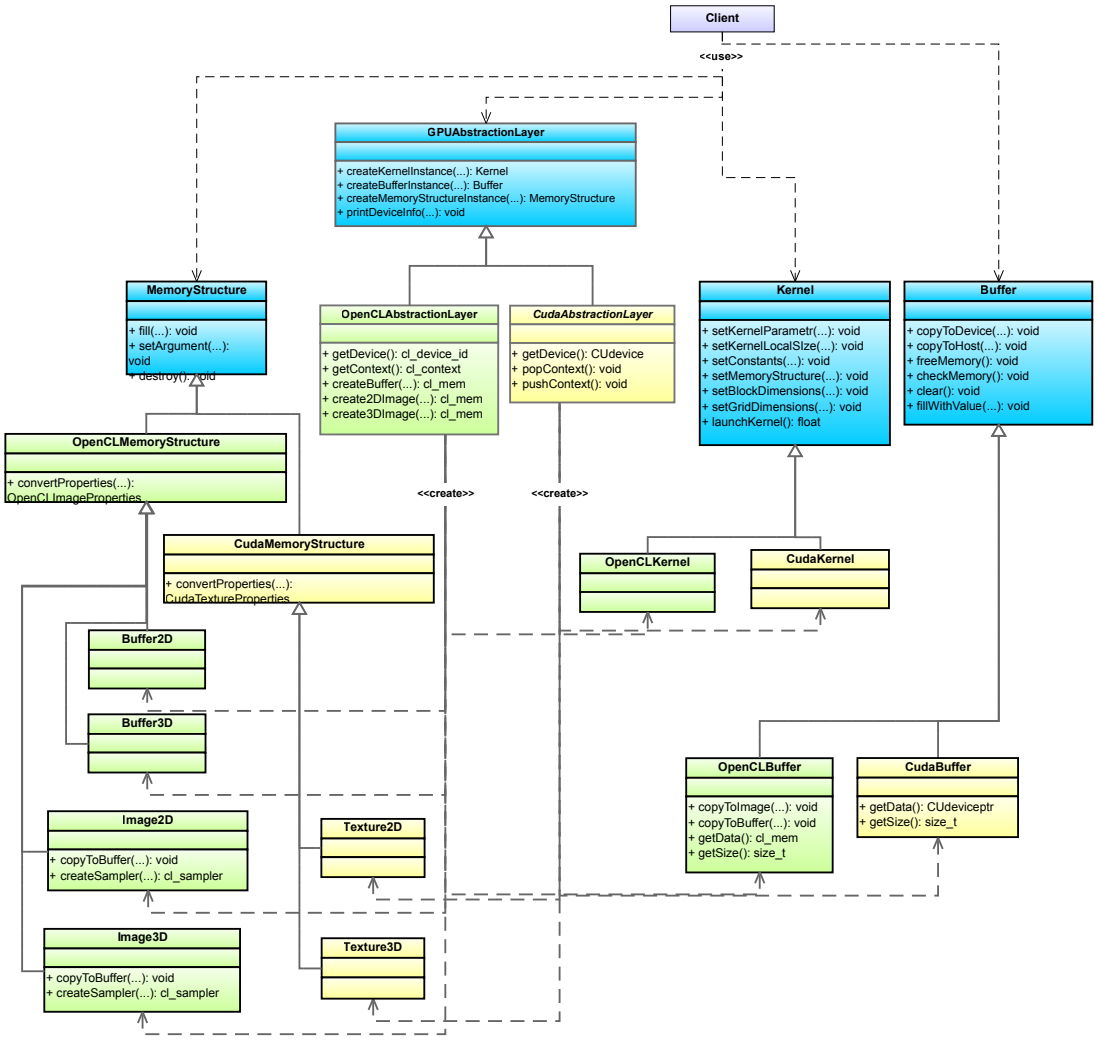


Figure 6.1: Class diagram of the wrapper, for space issue are arguments of function replaced by $\cdots$

during the kernel compilation on the host if the image format is supported by the device.

Chapter 7

Results and Discussion

In this chapter we present results of our work. First, we introduce applied quality measurements and then give brief overview of analyzed datasets. We present reconstructions of these datasets obtained with ET SART and compare them to results produced with SIRT and WBP. Finally, we discuss our findings.

7.1 Quality Measurements

Since we do not have any reference volumes to analyzed datasets, we measure the quality of reconstructions towards input aligned stacks. Using a reconstructed volume as an initial guess, we compute forward projection and compare step for all projections from the stack. In that way, we obtain an error map errorMap for each projection. These error maps are then used to compute Root Mean Square (RMS) [rms] and R-factor [rfa].

If N denotes number of all pixels from all projections we compute RMS as follows:

$$RMS = \sqrt{\frac{errorMap[1]^2 + errorMap[2]^2 + \cdots + errorMap[N]^2}{N}} \tag{7.1}$$

and R-factor R as:

$$R = \frac{\sum_{i \in N} |errorMap[i]|}{\sum_{i \in N} |realProjection[i]|} \tag{7.2}$$

7.2 Reconstructions

The basic properties of datasets we used are summarized in Table 7.1. For each dataset we computed the reconstruction using ET SART and measured both RMS and R-factor of the volume. We used IMOD to reconstruct the datasets with WBP and SIRT. RMS and R-factor of these volumes were calculated in the same way as for ET SART in order to ensure fair comparison. We first present for each dataset all three reconstructions and compare both their visual and measured quality. We also point out to some interesting

phenomenons appearing in the reconstructions. Finally, we summarize our observations in the discussion.

Name	Projection Resolution	Number of projections	Volume size	X-axis tilt	Source
Shigella T3SS	2048 x 2048	61	2048 x 2048 x 512	0.18	http://www.nki.nl/Research/
S. cerevisiae	2048 x 2048	90	2048 x 2048 x 201	0.09	http://www.nki.nl/Research/
Astrosomag3s7	2000 x 2000	61	2000 x 2000 x 300	-0.02	http://ccdb.ucsd.edu/
cb012	2000 x 2000	60	2000 x 2000 x 184	-3.13	http://ccdb.ucsd.edu/
HPFCere2a	1120 x 1632	61	1120 x 1632 x 500	-0.41	http://ccdb.ucsd.edu/

Table 7.1: Analyzed datasets

For all presented reconstruction we used 30 iterations for SIRT and 1 iteration for ET SART. Our choice of λ parameter is based on input projections. We choose smaller λ for noisy datasets and vice versa.

Note, the reconstructions are presented by selected slices (which are often trimmed because of space issues). To see actual reconstructions, we refer reader to [CG] where all reconstructions can be downloaded.

Astrosoma-g3s7

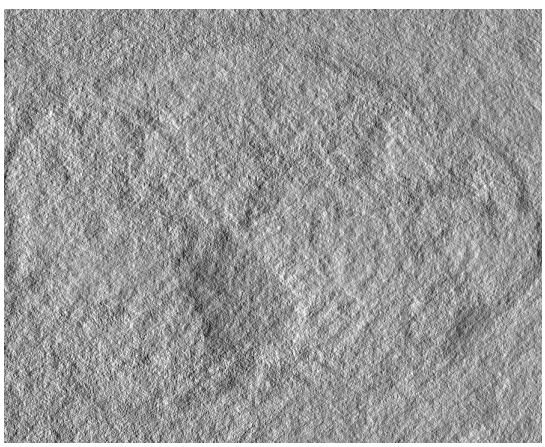
The reconstructions of astrosoma-g3s7 are shown in Figure 7.1. Both ET SART with $\lambda = 1.0$ and SIRT produced almost noiseless reconstructions of high quality while the volume obtained with WBP is very noisy and we can hardly see the astrosoma in it. It is worth noting, that while the difference in visual quality between WBP and algebraic methods is abysmal, the measured RMS and R-factor differ only slightly (see Table 7.2), but both agree with the visual impression.

	WBP	SIRT	ET SART
RMS	0.174119519	0.166333725	0.164387554
R-factor	0.292506192	0.27483353	0.269800767

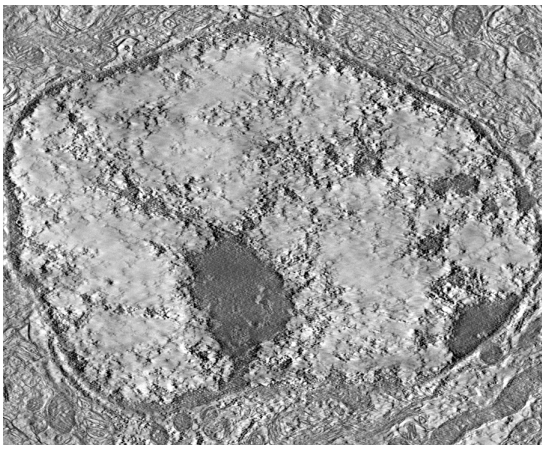
Table 7.2: RMS and R-factor of astrosoma-g3s7 volume obtained with WBP, SIRT (30 iterations) and ET SART (1 iteration, $\lambda = 1.0$)

Shigella T3SS

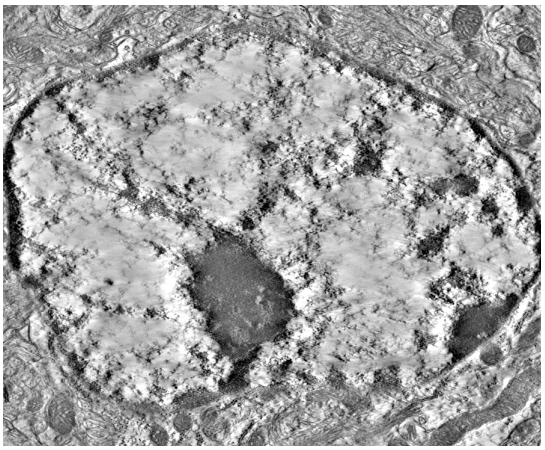
The Shigella T3SS dataset was very noisy and therefore we have expected WBP to produce results of poor quality which, as you can see in Figure 7.2(a), turned out to be true. The comparison of SIRT and ET SART reconstructions is on the other hand very surprising. Since SIRT was designed to effectively supress noise one would expect SART reconstruction



(a) WBP



(b) SIRT after 30 iterations



(c) ET SART after 1 iteration, $\lambda = 1.0$

Figure 7.1: Reconstruction of astrosoma-g3s7, full volume size: 2000 x 2000 x 300, slice 195

be more noisy. However, as shown in Figure 7.2, ET SART with produced much smoother reconstruction with relaxation parameter $\lambda = 0.15$. Moreover, it also contains more useful information. The Shigella T3SS is intended for needle complex studies [HHS+09] and in ET SART volume we can observe three different needles (see Figure 7.2) while in SIRT reconstruction only two needles can be identified due to noise.

The visual quality is however in direct contrast to RMS and R-factor as you can see in Table 7.3. WBP and SIRT differ only on the third decimal place and could be considered a measurement error, the value of the ET SART however is unlogically different.

	WBP	SIRT	ET SART
RMS	0.508854278	0.50623911	0.531041007
R-factor	0.244204095	0.242560428	0.251721701

Table 7.3: RMS and R-factor of Shigella T3SS volume obtained with WBP, SIRT (30 iterations) and ET SART (1 iteration, $\lambda = 0.15$)

S. cerevisiae

Similar results to Shigella T3SS were obtained while reconstructing S. cerevisiae (see Figure 7.4). The input aligned stack was very noisy and as a consequence we can hardly see the cells of the yeast in the reconstruction. The SIRT provides slightly better results but in compare to ET SART with $\lambda = 0.3$ it has both lower contrast and suffers more from noise. We reconstructed the datasets also with $\lambda = 0.02$ to see if we obtain similary good result. The volume is smooth and we can observe new structures in it but the contrast is low. We can very well see low-frequency information popping out. Note that $\lambda = 0.02$ is extremly low value, considering we have only one iteration.

According to quality measurment, SIRT has higher both RMS and R-factor (see Table 7.4) in constrast to visual results. However, the difference again occurs on third decimal place. Moreover, we can see that WBP, SIRT, and ET SART essentially differ only by contrast, which as such does not influence the RMS and R-factor significantly.

	WBP	SIRT	ET SART, $\lambda = 0.3$	ET SART, $\lambda = 0.02$
RMS	0.461368181	0.462263333	0.44409082	0.44294671
R-factor	0.31786278	0.318675622	0.303183731	0.301861884

Table 7.4: RMS and R-factor of S. cerevisiae volume obtained with WBP, SIRT (30 iterations) and ET SART (1 iteration)

cb012

The cb012 dataset is interesting for its high x-tilt (see Table **??**). Since x-tilt is incorporated both into ET SART and WBP in IMOD but not into SIRT, we can observe its influence on the reconstructions. It can be best seen in Figure 7.5, where a black structure is present in lower right corner both in WBP and ET SART reconstruction but is missing in SIRT volume. The structure appears in the volume more deeply. Moreover, the SIRT reconstruction contains a lot of artifacts and its quality is worse than by WBP reconstruction. This however, does not correspond to quality measurement according to which SIRT has lower both RMS and R-factor (see Table 7.5).

	WBP	SIRT	ET SART
RMS	0.110233786	0.105043626	0.096647816
R-factor	0.290356556	0.274560962	0.250968913

Table 7.5: RMS and R-factor of cb012 volume obtained with WBP, SIRT (30 iterations) and ET SART (1 iteration, $\lambda = 0.7$)

HPFcere2a

Last analyzed dataset is HPFcere2a. It was in used in [XXJ+10] to compare reconstruction quality between SIRT, SART and OS-SIRT. According to presented results SART gives slightly blurred reconstructions in compare to SIRT and OS-SIRT. However, it may be caused by lower λ which was chosen to be 0.3. We reconstructed the dataset using $\lambda = 1.0$ and compare it to SIRT reconstruction. Note, we can choose λ so high since we have low noise input data. The quality of both reconstructions is comparable and ET SART volume is not blurred (see Figure 7.6).

RMS and R-factor results are again almost identical, differing only on third decimal place.

	WBP	SIRT	ET SART
RMS	0.263612105	0.264023772	0.266139467
R-factor	0.244616354	0.244476885	0.243450811

Table 7.6: RMS and R-factor of HPFcere2a volume obtained with WBP, SIRT (30 iterations) and ET SART (1 iteration, $\lambda = 1.0$))

7.2.1 Fiducial Markers

While working with reconstructed volumes we notice two interesting phenomenons appearing in the reconstructions. If we study the volume from different point of view (literally), we can observe the x-like shape around the fiducial markers, as shown in Figure 7.7. We do not know how it arises but it is present in our volumes as well as in reconstructions obtained with WBP and SIRT. Furthermore, we are not sure if the x-shape distortions concern only fiducial markers or if the markers are only structures on which we can observe this phenomenon due to their high contrast.

Second interesting thing was observed in our reconstructions with low λ parameter. The white streaks around fiducial markers were significantly reduced everytime the markers were in focus (see Figure 7.8). Moreover, the shape of the markers in focus is more rounded than in other reconstructions.

7.2.2 Discussion

We have shown that ET SART outperforms WBP in all presented cases and it also provides at least as good results as SIRT. In case of noisy datasets (Shigella T3SS and S. cerevisiae) ET SART provides the best reconstructions.

We have also pointed out that RMS and R-factor calculated towards input projection set are not very reliable methods for reconstruction quality measurement since for volumes reconstructed from the same projection set they always give very similar values even in cases where the difference in visual quality is large (e.g. astrosoma-g3s7). There are two anomalies in the RMS and R-factor measurement (ET SART in Shigella T3SS and SIRT in cb012) where the reconstructions have visual artifacts, despite noticebly better RMS and R-factor. The Shigella T3SS could possibly be explained by the anomalous high-density white areas in ET SART. The results of cb012 remain puzzle for us. To verify our R-factor results, we have compared with [XXJ+10] and we have reached reasonably similar values, which should rule out problems with our computation.

We have proved the importance of integrating x-tilt into the reconstruction in order to obtain geometrically correct results (cb012).

We have also shown how much can λ parameter affect the reconstruction. For now we are setting λ parameter *ad hoc* and deeper study would be required to obtain λ automatically, for example according to noise levels in input projections.

Last but not least, we have observed unknown phenomenons in ET reconstructions that should be further explored.

7.3 Performance

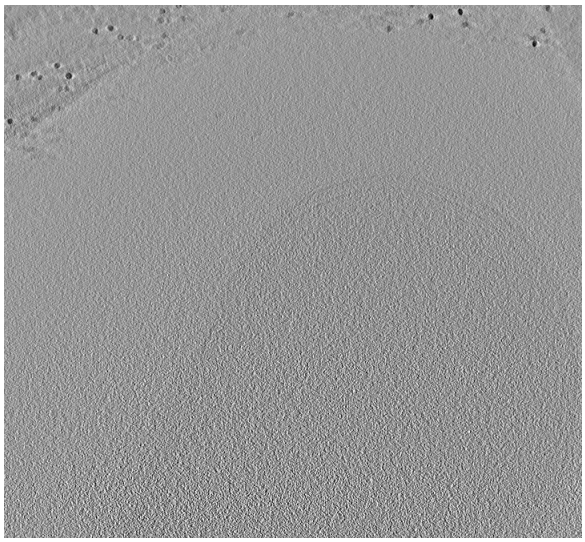
We have already proved that ET SART can compete to both WBP and SIRT when it comes to reconstruction quality. Now we would like to show, it is also fast enough. Therefore, we measured reconstruction times for ET SART (1 iteration) and 30 SIRT (30 iterations). For complete overview, we measured WBP times as well. All reconstructions were computed on an Intel(R) CORE(TM)2 Quad CPU Q6600 @ 2.40 computer with 8 GB RAM and 7200 RMP hard disk, equipped with NVIDIA GeForce GTX 480.

Reconstruction times are shown in Figure 7.9. According to them, SIRT was slower for all computed volumes except the last one. Overall, the time differences on measured reconstructions are relatively small, so we can say that both methods need the same time to compute a reconstruction.

We also measured time needed only for reconstruction, without projection loading or volume writing. The results are presented in Table 7.10.

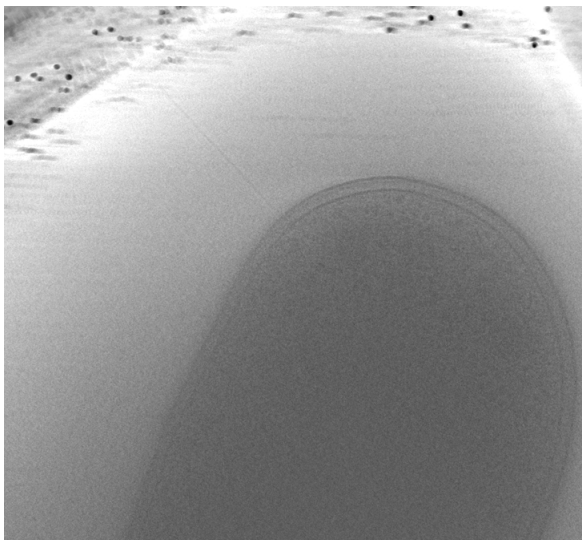
7.3.1 CUDA

Using our wrapper, we computed all reconstructions also with CUDA. The measured times show that CUDA is faster than OpenCL for all volumes (see Figure 7.11. This also holds if we consider only reconstruction time as we can sii in Figure 7.12. However, with respect to overall time, the differnces are small.



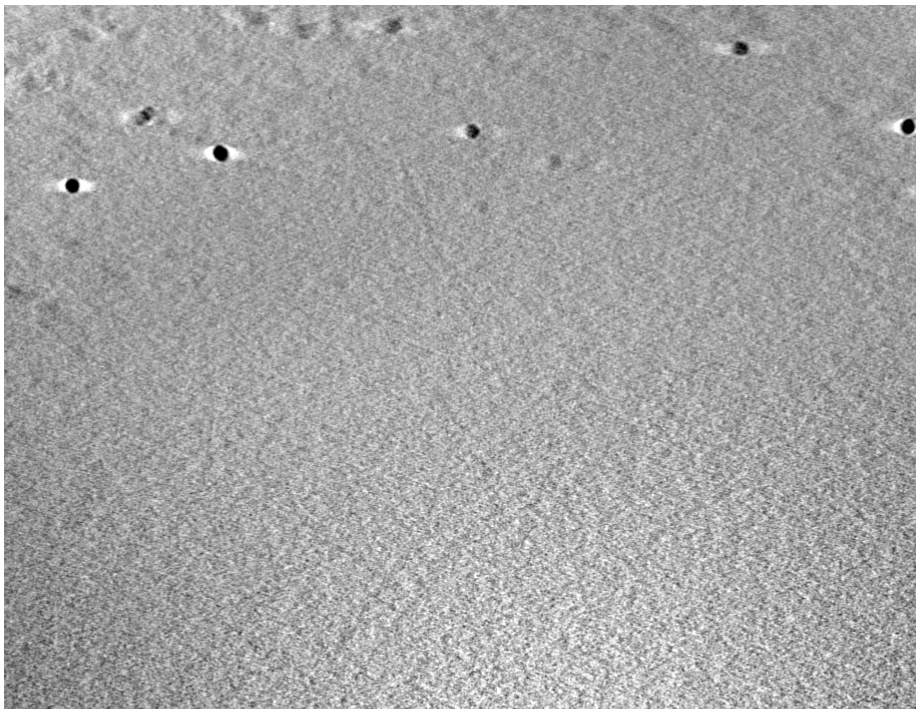
(a) WBP

(b) SIRT after 30 iterations

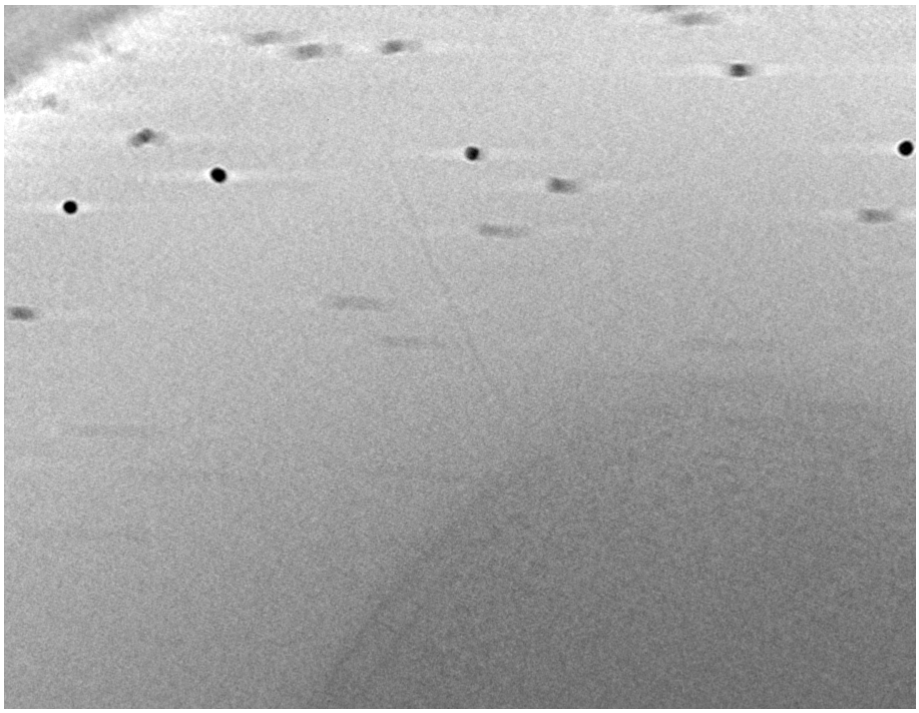


(c) ET SART after 1 iteration, $\lambda = 0.15$

Figure 7.2: Reconstruction of Shigella T3SS, full volume size: 2048 x 2048 x 512, slice 497

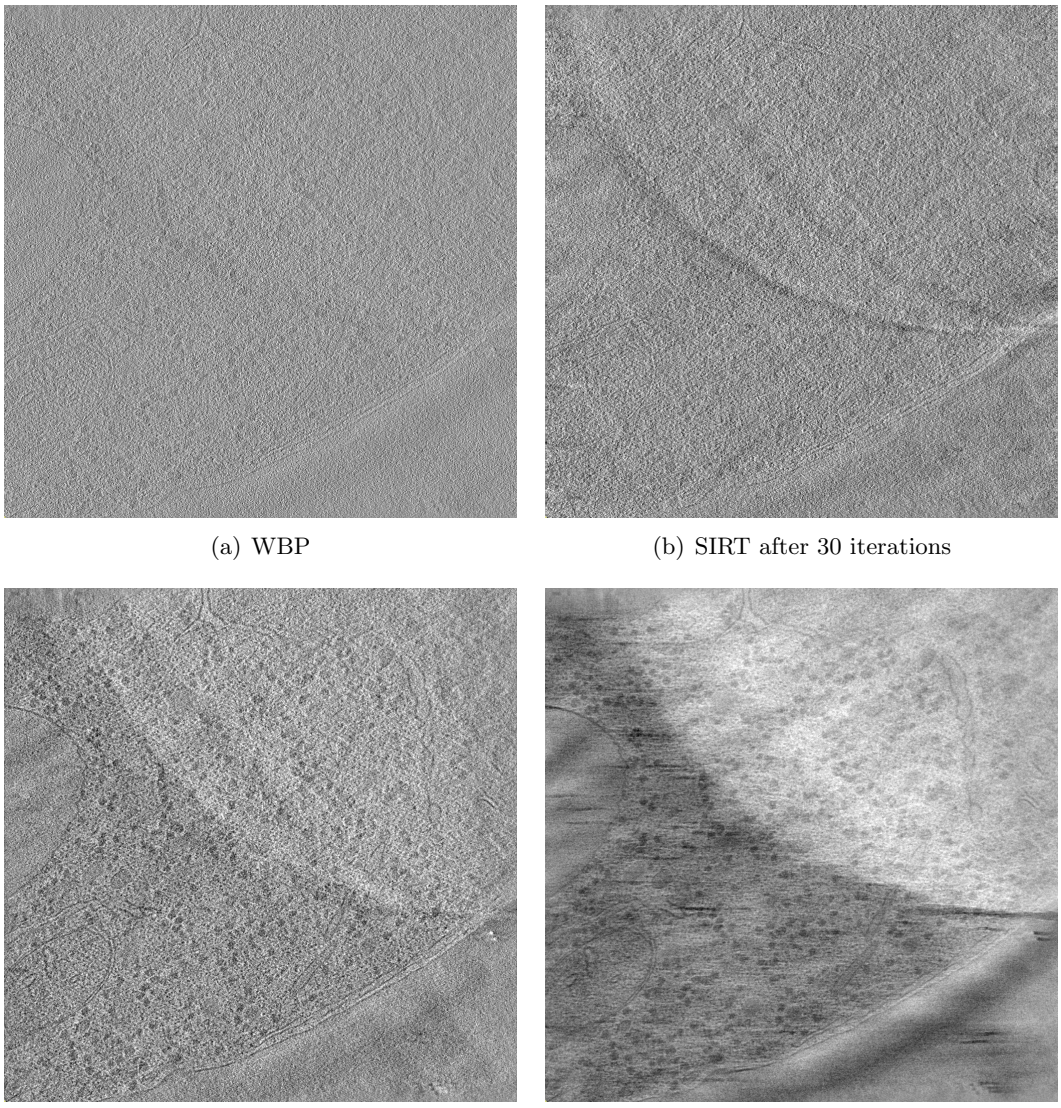


(a) SIRT after 30 iterations



(b) ET SART after 1 iteration, $\lambda = 0.15$

Figure 7.3: Reconstruction of Shigella T3SS, full volume size: 2048 x 2048 x 512, slice 265



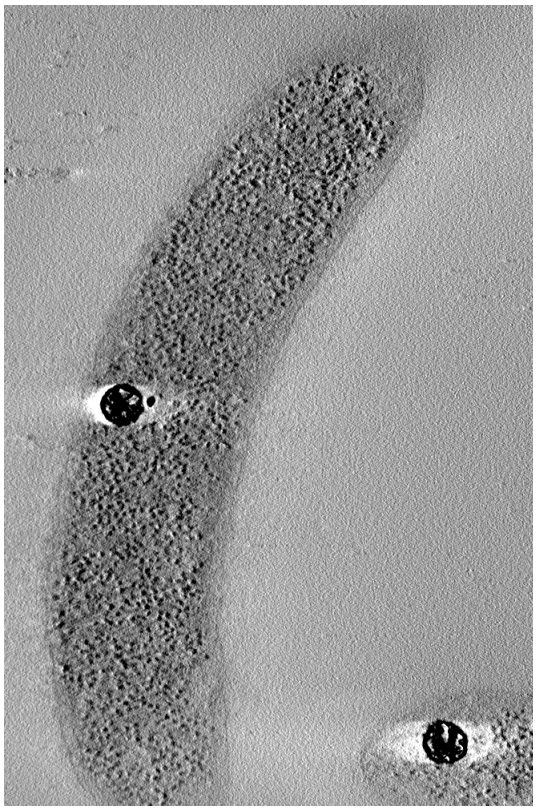
(a) WBP

(b) SIRT after 30 iterations

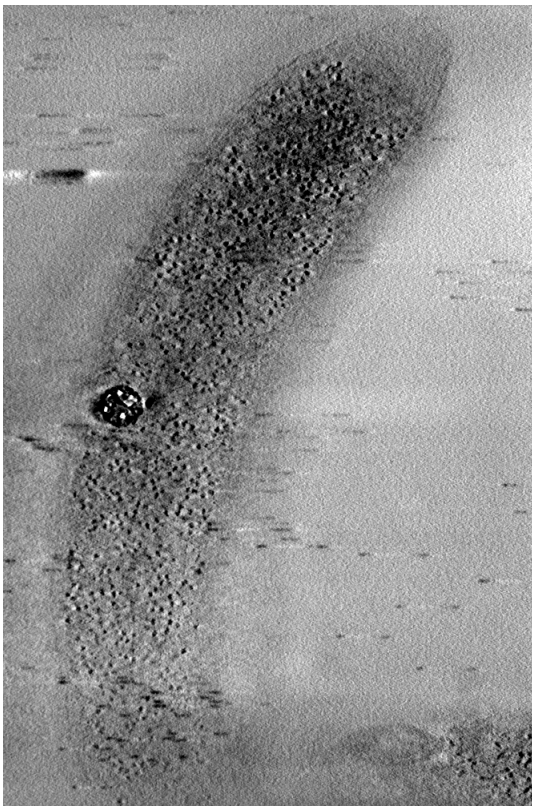
(c) ET SART after 1 iteration, $\lambda = 0.3$

(d) ET SART after 1 iteration, $\lambda = 0.02$

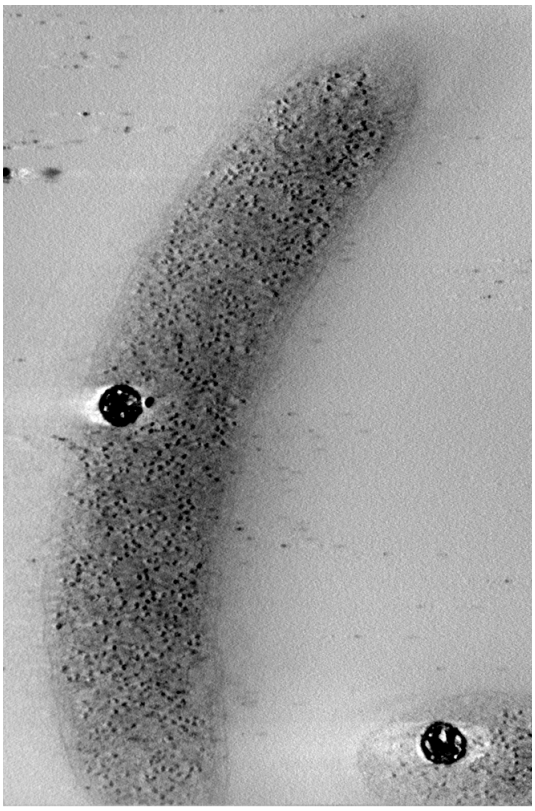
Figure 7.4: Reconstruction of S. cerevisiae, full volume size: 2048 x 2048 x 201, slice 94



(a) WBP

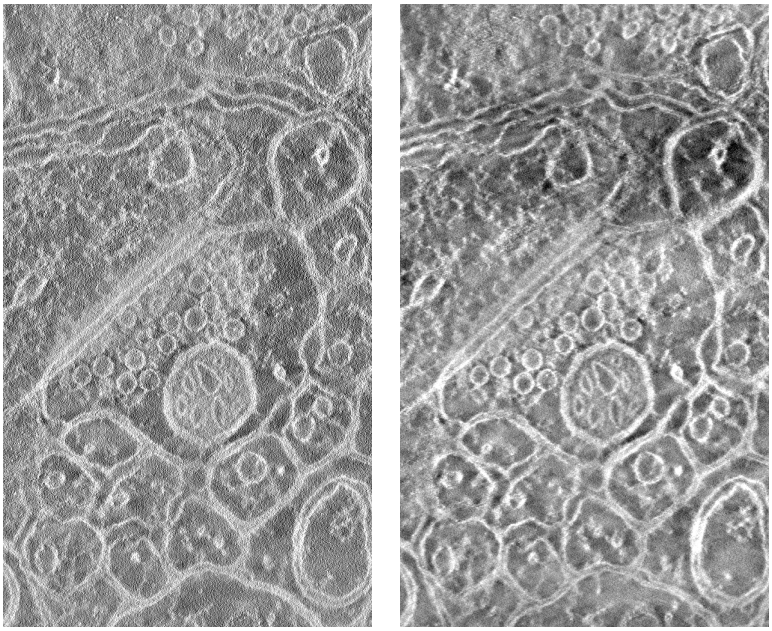


(b) SIRT after 30 iterations



(c) ET SART after 1 iteration, $\lambda = 0.7$

Figure 7.5: Reconstruction of cb012, full volume size: 2000 x 2000 x 184, slice 36

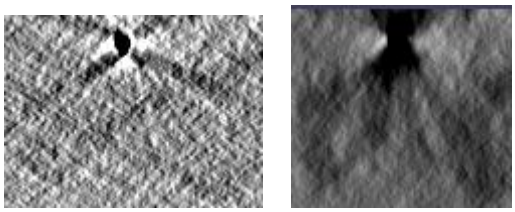


(a) WBP

(b) SIRT after 30 iterations

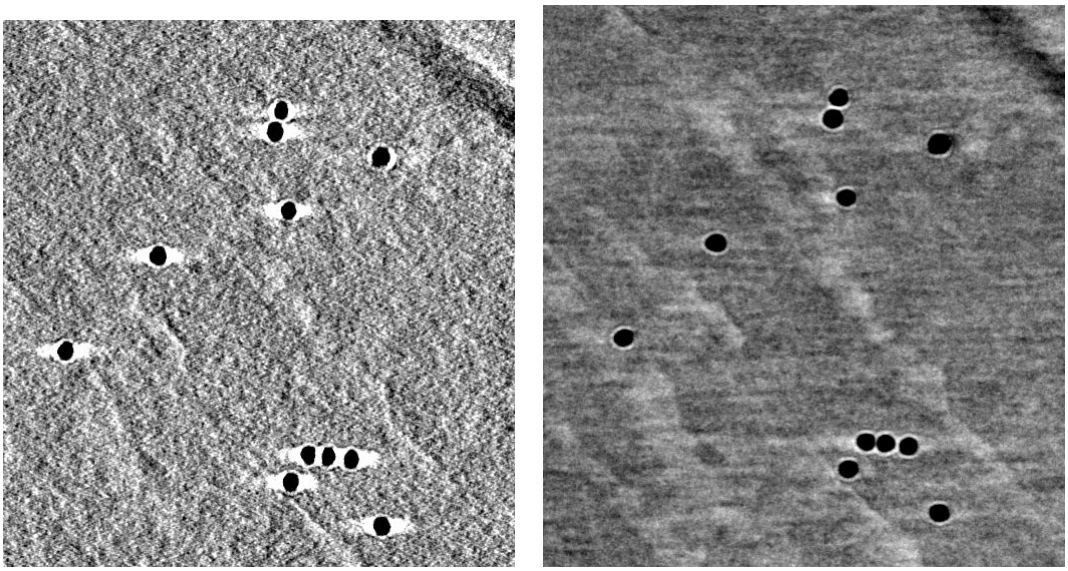
(c) ET SART after 1 iteration, $\lambda = 1.0$

Figure 7.6: Reconstruction of HPFcere2a, full volume size: 1120 x 1632 x 500, slice 251



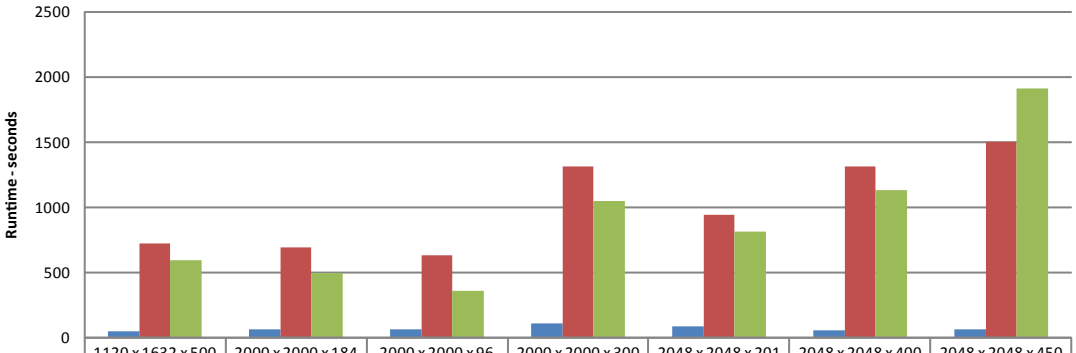
(a) SIRT (b) ET SART

Figure 7.7: X-shape artifacts observed in reconstructions of S. cerevisiae



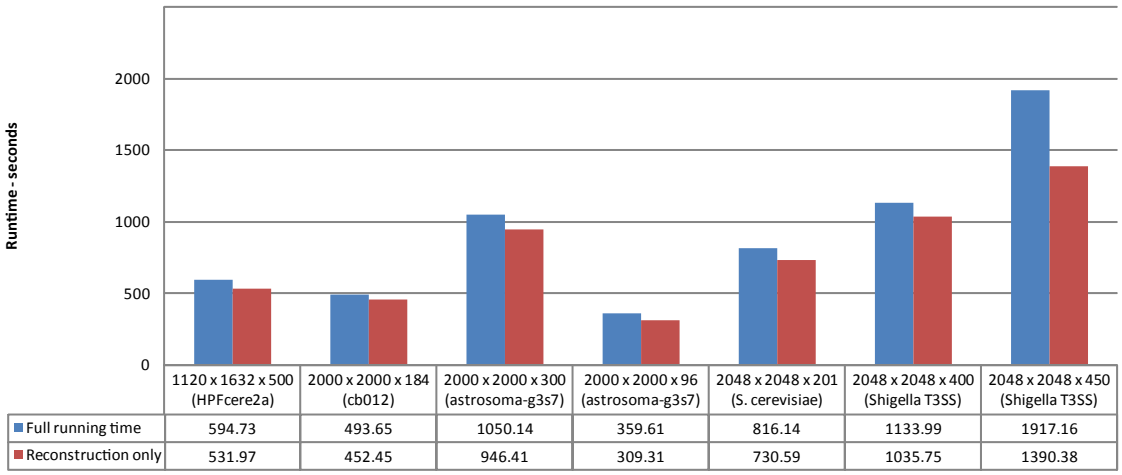
(a) SIRT (b) ET SART

Figure 7.8: Set of markers in reconstructions of S. cerevisiae



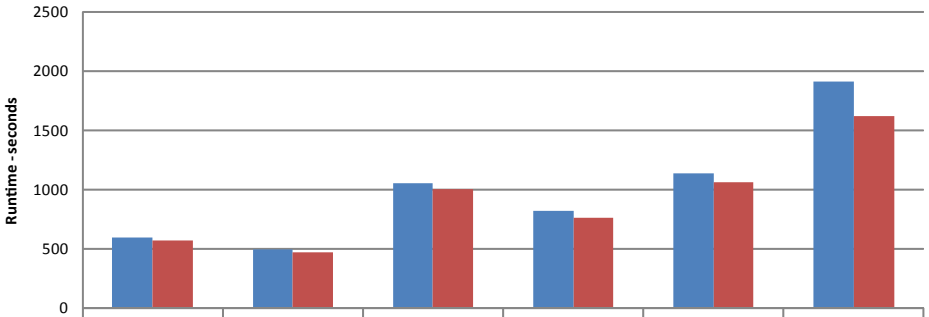
	1120 x 1632 x 500 (HPFcere2a)	2000 x 2000 x 184 (cb012)	2000 x 2000 x 96 (astrosoma-g3s7)	2000 x 2000 x 300 (astrosoma-g3s7)	2048 x 2048 x 201 (S. cerevisiae)	2048 x 2048 x 400 (Shigella T3SS)	2048 x 2048 x 450 (Shigella T3SS)
WBP	42.31	59.57	63.53	104.49	80.55	49.88	62.85
SIRT	723.17	692.77	631.63	1310.97	944.21	1315.32	1505.81
ET SART	594.73	493.65	359.61	1050.14	816.14	1133.99	1917.16

Figure 7.9: Full running times of WBP, SIRT and ET SART



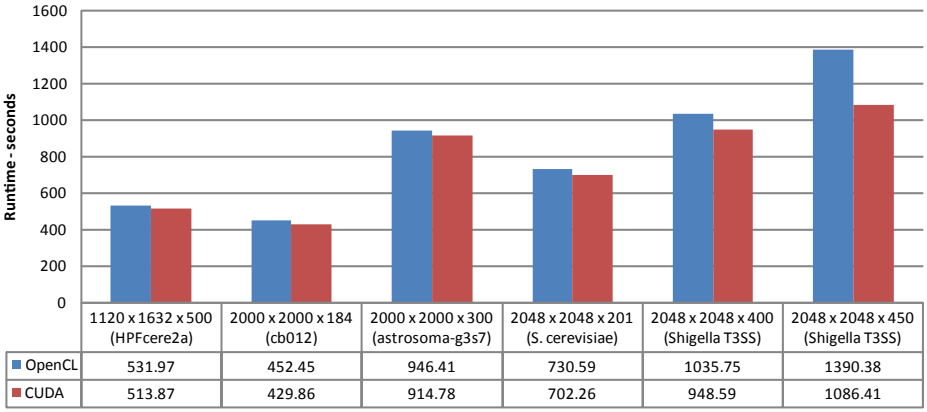
	1120 x 1632 x 500 (HPFcere2a)	2000 x 2000 x 184 (cb012)	2000 x 2000 x 300 (astrosoma-g3s7)	2000 x 2000 x 96 (astrosoma-g3s7)	2048 x 2048 x 201 (S. cerevisiae)	2048 x 2048 x 400 (Shigella T3SS)	2048 x 2048 x 450 (Shigella T3SS)
Full running time	594.73	493.65	1050.14	359.61	816.14	1133.99	1917.16
Reconstruction only	531.97	452.45	946.41	309.31	730.59	1035.75	1390.38

Figure 7.10: ET SART - Comparison of full and reconstruction time



	1120 x 1632 x 500 (HPFcere2a)	2000 x 2000 x 184 (cb012)	2000 x 2000 x 300 (astrosoma-g3s7)	2048 x 2048 x 201 (S. cerevisiae)	2048 x 2048 x 400 (Shigella T3SS)	2048 x 2048 x 450 (Shigella T3SS)
OpenCL	594.73	493.65	1050.14	816.14	1133.99	1917.16
CUDA	568.94	470.48	1001.15	764.45	1063.12	1623.18

Figure 7.11: OpenCL and CUDA comparison - full running time



	1120 x 1632 x 500 (HPFcere2a)	2000 x 2000 x 184 (cb012)	2000 x 2000 x 300 (astrosoma-g3s7)	2048 x 2048 x 201 (S. cerevisiae)	2048 x 2048 x 400 (Shigella T3SS)	2048 x 2048 x 450 (Shigella T3SS)
OpenCL	531.97	452.45	946.41	730.59	1035.75	1390.38
CUDA	513.87	429.86	914.78	702.26	948.59	1086.41

Figure 7.12: OpenCL and CUDA comparison - reconstruction time

Chapter 8

Conclusion

We have presented an implementation of SART for electron tomography called ET SART. As a starting point, we have used framework for computed tomography by Michael Kunz [Kun10]. We have made it more robust, precise and simpler. Furthermore, methods specific for electron tomography have been incorporated to the improved framework giving rise to ET SART.

We believe, our implementation outperforms other reconstruction techniques used in electron tomography in few key aspects. First, we use more robust geometry setup that allows us to easily accomodate to possible imperfect geometry setting in electron microscope. Secondly, we use different, more flexible scheme of parallelization. Thirdly, we use exact weighting factors both in forward and back projection.

To prove that, we have compared ET SART to SIRT and WBP. We have shown that ET SART reconstructions are superior to those obtained with WBP as well as they are at least as good as SIRT reconstructions. Most importantly, on very noisy datasets ET SART outperforms both WBP and SIRT.

In addition to our ET SART implementation, we have also presented wrapper providing unified API for CUDA and OpenCL. It is worth mentioning, that within the wrapper we have enhanced the portability of OpenCL programs since by solving the image portability issue.

8.1 Future Work

As we have pointed out, currently does not exist any reliable method for measurements of reconstruction quality. It could be part of further research to design a tool for quality measurement. The influence of λ on reconstruction quality could be also further explored. Since we have shown, how important it is to precisely approximate the geomtery setup in electron microscope, one could try to simulate the non-parallel rays in the reconstruction.

Bibliography

[AF11] J. I. Agulleiro and J. J. Fernandez. Fast tomographic reconstruction on multicore computers. *Bioinformatics*, 27(4):582–583, 2011.

[AK84] A.H. Andersen and A.C. Kak. Simultaneous Algebraic Reconstruction Technique (SART): A superior implementation of the ART algorithm. *Ultrasonic Imaging*, 6(1):81 – 94, 1984.

[AMD] Porting CUDA Applications to OpenCL. http://developer.amd.com/zones/OpenCLZone/programming/pages/portingcudatoopencl.aspx.

[BCMS+09] J.R. Bilbao-Castro, R. Marabini, C.O.S. Sorzano, I. García, J.M. Carazo, and J.J. Fernández. Exploiting desktop supercomputing for three-dimensional electron microscopy reconstructions using ART with blobs. *Journal of Structural Biology*, 165(1):19 – 26, 2009.

[Bos86] W. H. Bostick. What laboratory-produced plasma structures can contribute to the understanding of cosmic structures both large and small. *IEEE Transactions on Plasma Science*, 14:703–717, December 1986.

[CDMS+08] Daniel Castaño-Díez, Dominik Moser, Andreas Schoenegger, Sabine Pruggnaller, and Achilleas S. Frangakis. Performance evaluation of image processing algorithms on the GPU. *Journal of Structural Biology*, 164(1):153 – 160, 2008.

[CDSAAF10] Daniel Castaño-Díez, Margot Scheffer, Ashraf Al-Amoudi, and Achilleas S. Frangakis. Alignator: A gpu powered software package for robust fiducial-less alignment of cryo tilt-series. *Journal of Structural Biology*, 170(1):117 – 126, 2010.

[CG] CG Publications Saarland University. http://graphics.cg.uni-saarland.de/publications/.

[cg] NVIDIA Cg. http://http.developer.nvidia.com/Cg/index.html.

[CHS96] R. A. Crowther, R. Henderson, and J. M. Smith. Mrc image processing programs. *Journal of Structural Biology*, 116(1):9 – 16, 1996.

[Coma] NVIDIA Fermi with CUDA and OpenCL. http://blog.accelereyes.com/blog/2010/05/10/nvidia-fermi-cuda-and-opencl/.

[Comb] Cross-Platform GPGPU with OpenCL. http://parse.ele.tue.nl/downloads/handouts4.pdf.

[DBW09] C. Barry Carter David B. Williams. *Transmission Electron Microscopy*. Springer, second edition edition, 2009.

[dda] Digital Differential Analyzer. http://en.wikipedia.org/wiki/Digital_Differential_Analyzer_%28graphics_algorithm%29.

[Dig] DigiECT. http://www.digisens.fr/en/soft/2-DigiECT.html.

[DMF07] Daniel Castano Diez, Hannes Mueller, and Achilleas S. Frangakis. Implementation and performance evaluation of reconstruction algorithms on graphics processors. *Journal of Structural Biology*, 157(1):288 – 295, 2007. Software tools for macromolecular microscopy.

[DSF06] Daniel Castaño Díez, Anja Seybert, and Achilleas S. Frangakis. Tilt-series and electron microscope alignment for the correction of the non-perpendicularity of beam and tilt-axis. *Journal of Structural Biology*, 154(2):195 – 205, 2006.

[DWL+10] Peng Du, Rick Weber, Piotr Luszczek, Stanimire Tomov, Gregory Peterson, and Jack Dongarra. From CUDA to OpenCL: Towards a Performance-portable Solution for Multiplatform GPU Programming. *Parallel Computing*, pages 1 – 12, 2010.

[FCG04] José-Jesús Fernández, José-María Carazo, and Inmaculada García. Three-dimensional reconstruction of cellular structures by electron microscope tomography and parallel computing. *Journal of Parallel and Distributed Computing*, 64(2):285 – 300, 2004.

[FEI10] FEI. An Introduction to Electron Microscopy, July 2010.

[Fer08] J.J. Fernández. High performance computing in structural determination by electron cryomicroscopy. *Journal of Structural Biology*, 164(1):1 – 6, 2008.

[FLR+02] José-Jesús Fernández, Albert F. Lawrence, Javier Roca, Inmaculada García, Mark H. Ellisman, and José-María Carazo. High-performance electron tomography of complex biological specimens. *Journal of Structural Biology*, 138(1-2):6 – 20, 2002.

[FRP+96] Joachim Frank, Michael Radermacher, Pawel Penczek, Jun Zhu, Yanhong Li, Mahieddine Ladjadj, and Ardean Leith. SPIDER and WEB: Processing and Visualization of Images in 3D Electron Microscopy and Related Fields. *Journal of Structural Biology*, 116(1):190 – 199, 1996.

[GBH70] Richard Gordon, Robert Bender, and Gabor T. Herman. Algebraic Reconstruction Techniques (ART) for three-dimensional electron microscopy and X-ray photography. *Journal of Theoretical Biology*, 29(3):471 – 481, 1970.

[Gil72] Peter Gilbert. Iterative methods for the three-dimensional reconstruction of an object from projections. *Journal of Theoretical Biology*, 36(1):105 – 117, 1972.

[HCWS08] J. Bernard Heymann, Giovanni Cardone, Dennis C. Winkler, and Alasdair C. Steven. Computational resources for cryo-electron tomography in Bsoft. *Journal of Structural Biology*, 161(3):232 – 242, 2008. The 4th International Conference on Electron Tomography, The 4th International Conference on Electron Tomography.

[HHS+09] Julie L. Hodgkinson, Ashley Horsley, David Stabat, Martha Simon, Steven Johnson, Paula C. A. da Fonseca, Edward P. Morris, Joseph S. Wall, Susan M. Lea, and Ariel J. Blocker. Three-dimensional reconstruction of the shigella t3ss transmembrane regions reveals 12-fold symmetry and novel features throughout. *Nat Struct Mol Biol.*, 16(5):477 – 485, 2009.

[Kac37] Stefan Kaczmarz. Angenäherte auflösung von systemen linearer gleichungen. *Bulletin International de l'Académie Polonaise des Sciences et des Lettres.*, A35:355 – 357, 1937.

[Khr] Khronos Group. http://www.khronos.org/.

[KMM96] James R. Kremer, David N. Mastronarde, and J. Richard McIntosh. Computer Visualization of Three-Dimensional Image Data Using IMOD. *Journal of Structural Biology*, 116(1):71 – 76, 1996.

[KS88] A. C. Kak and Malcolm Slaney. Principles of Computerized Tomographic Imaging. IEEE Press, 1988.

[Kun10] Michael Kunz. On Improving the Accuracy and Speed of GPU-based Simultaneous Algebraic Reconstruction Techniques. Master's thesis, Saarland University, Faculty of Natural Sciences and Technology I, Department of Computer Science, July 2010.

[Lev88] M. Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. and Appl.*, 8(3):29 – 37, 1988.

[LFB05] Vladan Lucic, Friederich Foerster, and Wolfgang Baumeister. Structure studies by electron tomography: from Cells to Molecules. *Annual Review of Biochemistry*, 74:833–865, July 2005.

[MHC98] Roberto Marabini, Gabor T Herman, and José M Carazo. 3D reconstruction in electron microscopy using ART with smooth spherically symmetric volume elements (blobs). *Ultramicroscopy*, 72(1-2):53 – 65, 1998.

[MRC] MRC Specification. http://bio3d.colorado.edu/imod/doc/mrc_format.txt.

[Mue98] Klaus Mueller. Fast and accurate three-dimensional reconstrucion from conebeam projection data using algebraic methods. Master's thesis, School of The Ohio State University, 1998.

[NFL+05] Stephan Nickell, Friedrich Förster, Alexandros Linaroudis, William Del Net, Florian Beck, Reiner Hegerl, Wolfgang Baumeister, and Jürgen M. Plitzko. TOM software toolbox: acquisition and analysis for electron tomography. *Journal of Structural Biology*, 149(3):227 – 234, 2005.

[nki] The Netherlands Cancer Institute. http://www.nki.nl/Research/.

[NVIDIA09] NVIDIA. NVIDIA OpenCL JumpStart Guide, April 2009.

[Opea] OpenCL. http://www.khronos.org/opencl/.

[Opeb] OpenCL on Wikipedia. http://de.wikipedia.org/wiki/OpenCL.

[Ope10] The OpenCL Specification, version 1.1. http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf, 2010.

[PMF08] Sabine Pruggnaller, Matthias Mayr, and Achilleas S. Frangakis. A visualization and segmentation toolbox for electron microscopy. *Journal of Structural Biology*, 164(1):161 – 165, 2008.

[Por] Portland Group. http://www.pgroup.com/.

[Rad92] Michael Radermacher. *Electron Tomography by Joachim Frank*, chapter Weighted Back-projection Methods, pages 91–115. Plenum Press, New York, 1992.

[rfa] R-factor. http://en.wikipedia.org/wiki/R-factor_%28crystallography%29.

[rms] Root Mean Square. http://en.wikipedia.org/wiki/Root_mean_square.

[SfBB96] Ulf Skoglund, Lars-Göran Öfverstedt, Roger M. Burnett, and Gérard Bricogne. Maximum-entropy three-dimensional reconstruction with deconvolution of the contrast transfer function: A test application with adenovirus. *Journal of Structural Biology*, 117(3):173 – 188, 1996.

[SMB03] Kristian Sandberg, David N. Mastronarde, and Gregory Beylkin. A fast reconstruction algorithm for electron microscope tomography. *Journal of Structural Biology*, 144(1-2):61 – 72, 2003. Analytical Methods and Software Tools for Macromolecular Microscopy.

[SMVM+04] C.O.S. Sorzano, R. Marabini, J. Velázquez-Muriel, J.R. Bilbao-Castro, S.H.W. Scheres, J.M. Carazo, and A. Pascual-Montano. Xmipp: a new generation of an open-source image processing package for electron microscopy. *Journal of Structural Biology*, 148(2):194 – 204, 2004.

[Swa] Swan: A simple tool for porting CUDA to OpenCL. http://www.multiscalelab.org/swan.

[Tem] TEM on Wikipedia. http://en.wikipedia.org/wiki/Transmission_electron_microscopy.

[TF99] Hegerl R. Nickell S. Boehm J. Typke, D. and A.S. Frangakis. Measurements of the perpendicularity if the beam-direction and tilt-axis in the cm300. 1999.

[WJ04] Ge Wang and Ming Jiang. Ordered-subset simultaneous algebraic reconstruction techniques (OS-SART). *Journal of X-Ray Science and Technology*, 12:169–177, October 2004.

[WZL09] Xiaohua Wan, Fa Zhang, and Zhiyong Liu. Modified Simultaneous Algebraic Reconstruction Technique and its Parallelization in Cryo-electron Tomography. *Parallel and Distributed Systems, International Conference on*, 0:384–390, 2009.

[Xpl] Xplore3D. http://www.fei.com/uploadedFiles/DocumentsPrivate/Content/Xplore_3D_web_ds.pdf.

[XXJ+10] Wei Xu, Fang Xu, Mel Jones, Bettina Keszthelyi, John Sedat, David Agard, and Klaus Mueller. High-performance iterative electron tomography reconstruction with long-object compensation using graphics processing units (gpus). *Journal of Structural Biology*, 171(2):142 – 153, 2010.