Towards Real Time Characterization of Grain Growth from the Melt

Christopher James Wright

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2020

ABSTRACT

Towards Real Time Characterization of Grain Growth from the Melt

Christopher James Wright

Single crystal materials have unique properties which are endowed by their long ranging atomic order. Growing these crystalline materials can be difficult, as entropy favors disordered grains. The optical floating zone furnace provides an efficient way to make novel single crystal materials, enabling the study of crystals with complex chemical makeup which other techniques would not be able to provide. However, the growth mechanisms of these crystals is poorly understood, leaving the process of making them prone to trial and error and limiting its application in the broader research community.

This work aims to understand the microstructural dynamics of floating zone growth using x-ray scattering techniques. These techniques include x-ray diffraction tomography and two dimensional crystal mapping. One focus of this work is building the computational infrastructure to process the large stream of heterogeneous data which results from these techniques. Additionally, this work includes computational and experimental commissioning of the x-ray diffraction tomography technique, helping to understand the advantages and limitations of the current approaches. These pieces of infrastructure are then used to characterize the growth of Rutile crystals via a float zone furnace. Particular attention is paid to the competition amongst the grains, and how certain grains are selected from the plethora which are created at the beginning of the growth.

Table of Contents

List of Tables

List of Figures

Acknowledgments

This dissertation represents a rare pleasure, the culmination of many years of work in the x-ray science and software fields. As such, there are a great many people who deserve my gratitude for all their help, advice, support and mentorship.

To my parents, the last time I wrote one of these, I said that I doubted if language could express my gratitude for everything you have done. Having gotten another chance to try to put into words my feelings, I find that language is just as lackluster a medium as last time. However, I am eternally thankful that you were able to see this, and all your work supporting my endeavors, come to fruition.

To my experimental family at the NSLS-II and NSLS, you have always provided fertile ground for me to grow my ideas, experiments and software. To Dr. Sanjit Ghose, thank you for introducing me to x-ray science. To Dr. Eric Dooryhee, thank you for your work on the tomography experiments. To Dr. Dan Olds, I have always appreciated your help, our conversations and your attitude towards software. To Dr. Milinda Abeykoon, thank you for your experimental support.

To Dr. Thomas Caswell, Dr. Dan Allen and Dr. Stuart Campbell, thank you for providing mentorship and a spectacular opportunity to grow my software skills.

To Dr. Anthony Scopatz, thank you for all your support; moral, technical and otherwise.

To my collaborators, especially Dr. Peter Khalifah and Mr. Jonathan Denny, thank you for your time, samples, insight and most of all patience.

To Dr. David Sprouster, collaborating with you on the tomography projects was a true joy.

To my committee, thank you very much for your insightful comments, support and revisions

To Prof Simon Billinge, never before have I had an academic advisor so enthusiastically invested in my work and vision. Your commitment to this work is staggering, as is your support for my imagination.

To Diane D. & Donald G. Wright

And Eternity in an hour

- William Blake

Chapter 1

Float Zone Synthesis

Introduction

The unique properties of single crystals have made their synthesis and growth important technologies, impacting industries from computers to aviation. Single crystal materials often exhibit properties different from their polycrystalline counterparts. For instance, high temperature mechanical systems use superalloys, because of their durability and strength retention under high heat load. However, even superalloys can deform under thermal and mechanical stress. Polycrystalline materials derive much of their strength from grain boundaries that trap dislocations and prevent them from propagating through the material. At high temperatures creep is the predominant mechanism for deformation, and these grain boundaries become a source of weakness, as they can allow grains to slip past one another. Single crystal superalloys solve this problem by having a single lattice, allowing operation at much higher temperatures. The Lockheed SR-71 Blackbird used used these crystalline superalloys in it engine. Because of the alloy's ability to withstand temperatures up to up to 1760 °C without deforming, the SR-71 flies at Mach 3.2 and holds the record for fastest air breathing manned aircraft [Defense Technical Information Center, 1966].

Making Single Crystals

Single crystals are made via several methods:

The oldest method for making single crystals is the Verneuil method, developed to make

synthetic rubies and sapphires from a powdered Al2O3 feed. In this method pure oxygen blows the feed material through a tube. Outside this tube is another tube filled with pure hydrogen. At the opening of these tubes the two gasses are ignited, melting the Al2O3 into small droplets. The droplets fall onto a rod, crystallizing. As more droplets fall a larger crystal forms and the rod moves down to enable new droplets to fall on the crystal and add to its mass. The rod, or boule, is then be broken off to yield the gemstone [Levin, 1913].

One of the most common methods is the Czochralski (CZ) method, which is used to make the Si boules that are then cut into wafers for computer chips. In this method raw material is melted in a crucible, a small seed crystal is lowered into the melt and pulled out, growing a crystal behind it. However, this approach can lead to side reactions and impurities from the crucible, which is often made of a compatible oxide like SiO2 or an inert metal like Pt or Ir. Boules made from this process can be quite large, up to 190 mm for Gd3Ga5O12 [Wang *et al.*, 2015].

In Flux growth the precursors are dissolved in a compatible solvent called the "flux". Once the material is dissolved either the temperature is lowered or the flux evaporated to result in a supersaturated solution which then nucleates crystals. This is similar to how rock candy is made from water supersaturated with sugar. This approach can result in lower defect density crystals as the temperature used to make the crystals is lower than if the flux had not been used. However, this technique often produces small crystals [Wang *et al.*, 2015; Janssen *et al.*, 2013]. Flux growth can be combined with floating zone growth, known as "traveling solvent floating zone growth"[Dbkowska *et al.*, 2015]

In Bridgman growth, the precursors are loaded into a crucible, the crucible is then lowered into a high temperature furnace where the precursors melt. After melting the crucible is further lowered into a lower temperature region, which then causes the system to crystallize. The crystallization gradient can be controlled by the lowering rate and the two furnaces' temperatures. This method benefits from having a sealed system, so mass transfer out of the system, like evaporation of precursors, is eliminated. However, similar to flux growth,

the crucible in this technique can react with the melt, producing impurities in the crystal. A method similar to the Bridgman is used to produce the single crystal alloys for jet turbines [Bridgman, 1925].

In Float Zone (FZ) growth two rods, a seed and a feed rod, are melted at the tip and brought together to form a molten floating zone. This zone is moved up the feed rod, creating a crystal on top of the seed rod. FZ growth is the method of choice for crystals of complex oxides with centimeters long axis and millimeter diameters, especially for precursors which react with crucible materials since no crucible is used. Additionally, control of the atmosphere can allow for complex gas environments. FZ growth is limited by: difficult to control parameters, surface tension limited diameters, and works poorly for high vapor pressure or high viscosity materials [Muiznieks *et al.*, 2015]. Overall, the FZ method is amenable to growing novel or chemically complex materials.

Uses of FZ Crystals

FZ grown crystals are used in many contexts, including industrial materials, although most applications are for novel research materials. The uniformity and size of FZ crystals enable their use in optical applications, including Al2O3 for ruby lasers [Saito, 1986], Rutile TiO2 for its large refractive index and birefringence for optical communications [Higuchi and Kodaira, 1992; Higuchi *et al.*, 2000]. Research use of FZ crystals have included magnetic materials, like the Ba3Cr2O8 spin dimer system [Aczel *et al.*, 2008] and superalloy materials using RuAl and TiAl [Dbkowska *et al.*, 2015] and many more. FZ growth is particularly powerful for research as the technique enables use of a diverse range of precursor chemistries, atmospheres and growth rates. Control of the growth process is key to the production of these properties and their end uses, as growth parameters can impact the atomic and macroscopic structure of the crystal [Koohpayeh *et al.*, 2013; Prabhakaran *et al.*, 2003]. However, quantitative understanding of the growth-structure-property relationships remains elusive, as an mechanistic understanding of crystal growth

and microstructure is still lacking.

History

The FZ process was initially applied to silicon for use in the semiconductor industry in 1952 as a purification technique [Muiznieks *et al.*, 2015; Keck and Golay, 1953; Theuerer, 1962]. During the FZ process impurities preferentially move into the melt, leaving purer material behind. This is similar to the production of ice wine via freeze distillation, where the sugars in the grapes, acting like the impurities in the silicon, are preferentially left in the liquid as pure ice is frozen out. In the case of silicon, unlike ice wine, the desired material is the purified silicon. This process made higher quality crystals than the previously used CZ process, as fewer oxygen impurities were included in the crystal. This lead to silicon crystals with exceptional carrier lifetimes, which are used in high performance solar panels. However, limitations in the size of the crystals and tolerance for higher oxygen impurities prevented wide adoption in semiconductor processing. FZ production of oxide materials started in 1969 with the production of ferrite crystals [Akashi *et al.*, 1969]. FZ synthesis of oxide materials has since grown to be applied to a wide range of materials, with over 100 listed in [Dbkowska *et al.*, 2015] as of 2015 with applications to: superconductors [Tanaka *et al.*, 1975; Takeya *et al.*, 2001; Behr *et al.*, 1999; Revcolevschi and Jegoudez, 1997], magnetic frustration [Koohpayeh *et al.*, 2013; Anand *et al.*, 2018], optical electronics [Muiznieks *et al.*, 2015; Frazer *et al.*, 2015; Chang *et al.*, 2013], and superalloys [Dbkowska *et al.*, 2015].

How FZ Growth is Performed

The first step of a FZ growth is the production of the seed and feed rods. The rods are made by putting the powder precursor(s) into a balloon which is then evacuated. The balloon serves as a barrier between the powder and the hydraulic liquid which provides hydrostatic pressure across the entire rod, pressing it into shape. The seed and feed rods are then sintered

in a furnace under an atmosphere of the grower's choice. The rods are then assembled in an optical floating zone furnace, with the seed on bottom and the feed on top. The tips of the seed and feed rods are then heated and brought into contact, forming a molten zone bridge of roughly 0.4cm^{-3} in volume. Heat is supplied to this bridge by parabolic mirrors and halogen lamps which are focused onto the zone. The rods can be rotated individually or together in either the same or opposite directions with rotation rates ranging up to 50 rpm to maintain even heating and mixing. To allow the base of the melt to crystallize the molten zone is moved into the feed rod by either moving the two rods down or the heaters up. As the melt moves up the bottom of the melt cools and crystallizes, forming many grains at the beginning of the growth, eventually a single crystal out competes the others as the growth progresses, as shown in Fig. 1.1. FZ growth is a dark art and lots of iterative testing of process parameters is needed to get good results. There are experts at making particular kinds of crystals, but a general approach to making these crystals is unknown.

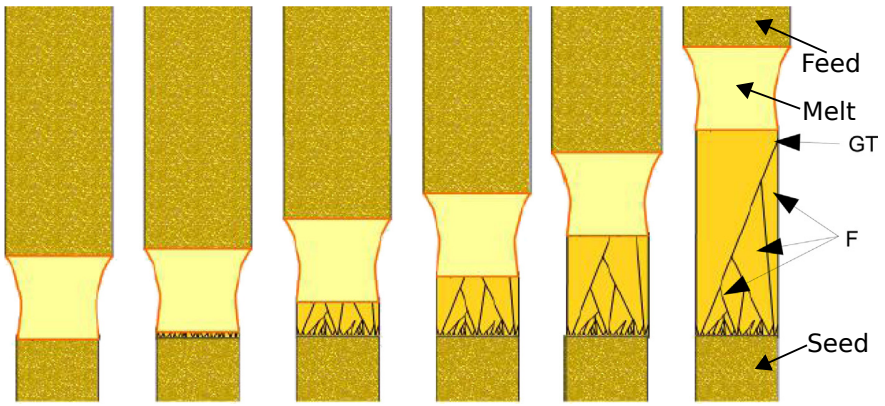


Figure 1.1: Schematic of float zone synthesis modified from [Dbkowska *et al.*, 2015] As the melt, or float zone, moves into the feed rod it leaves crystals in its wake (F). The crystals compete with one crystal eventually winning. This is signified by a grain termination (GT).

Opportunities and Challenges

FZ synthesis and growth is used produce oxide materials, some of which are quite complex, including high temperature superconductors. However, the growth of each material is

unique, requiring its own painstaking parameter optimization, with the difficulty of growing a material varying such that some may not be made at all. Even when parameters are optimized, key process variables are not well understood, like the temperature distribution in the melt and rods. The temperature can even depend on the composition of the powders, as some chemicals absorb more light than others. This tuning of parameters like the heater power, internal temperature, atmosphere composition and pressure, pull rate, rotation rate and more, stymies use of this technique more broadly. While there have been a few attempts to understand the dynamics of the oxide molten zone, a unified theory remains elusive. More in-situ experiments, which have currently been limited to optical imaging, pyrometer temperature sensing, [Behr *et al.*, 2010] and neutron imaging [Tremsin *et al.*, 2017], could provide insight. Techniques which are sensitive to the atomic structure of the materials, the orientation of the crystals, and their microstructure, like x-ray diffraction, atomic pair distribution function analysis, and their associated tomography techniques, could provide insight into how FZ crystal growth occurs, providing an approach to understanding the impact of processing parameters on the crystal quality and properties.

Chapter 2

X-Ray Diffraction and Tomography

X-ray Scattering

X-ray scattering is a phenomena that occurs when x-ray light is shone on a sample. The x-rays interact with the electron density in the sample, causing the x-rays exiting the sample to have the Fourier transform of the density encoded [Miao *et al.*, 1999]. Unlike most imaging methods, where a lens is used to invert the Fourier transform, diffraction analysis often takes place either in Fourier space, also known as inverse or Q space, or by performing the Fourier transform computationally.

Diffraction

The phenomena of diffraction occurs when the spacing between the atoms in a material's lattice is on the order of the wavelength of the incident x-rays [Misture and Snyder, 2001]. In this case the scattered x-rays are concentrated along certain directions. In powder diffraction the sample contains many of the same crystal oriented in all directions, causing the scattered x-rays to form rings. In single crystal diffraction only one orientation is present, causing the scattered x-rays to form points. The position of these points or rings is dictated by the distance between the planes of atoms, giving rise to Bragg's law $d = \frac{2\pi}{q}$ or $2d \sin\theta = n\lambda$ where d is the spacing between the lattice planes, q is the scattering vector, λ is the wavelength of light, θ is the angle between the incoming x-rays and the lattice planes. The relationship shows that for a given lattice plane distance x-rays will be concentrated at a corresponding

angle.

X-ray diffraction was discovered in 1912 by Max von Laue and collaborators, winning the 1914 Nobel Prize for Physics [Eckert, 2012]. The capacity for x-ray diffraction to determine the atomic structure of materials has been applied in many different fields from biology to condensed matter physics, yielding high impact research.

X-ray diffraction is used to quantify various aspects of the atomic structure of materials, including: strain [Noyan and Cohen, 2013], orientation [Busing and Levy, 1967], mosaicity [Sauter *et al.*, 2014], and crystallite size [Patterson, 1939; He, 2018; Yager and Majewski, 2014; Misture and Snyder, 2001]. Residual strain is measured from x-ray diffraction by comparing the unstrained reference position of the peaks to the position of strained peaks. Since the peak position is related to the spacing between the lattice planes any compression or expansion of the lattice will cause a commensurate peak movement. Single crystal and polycrystalline x-ray diffraction measurements can extract the orientation of the crystal being measured by determining the angular relationships between detected Bragg spots [Schmidt, 2014]. Furthermore diffraction can be used to understand the distribution of crystal orientations which can occur in single crystals. This phenomena is called mosaicity and describes the extent to which a small piece of the single crystal's orientation deviates from the bulk orientation. A schematic of what this does to the diffraction spots is shown in Fig. 2.1 [Sauter *et al.*, 2014]. Rocking the sample on one of its axes can help to determine the mosaicity, as the rocking motion brings different parts of the smeared out peaks into the diffracting condition. Crystallite size, and its analogue number of grains, can be determined two ways depending on the size of the crystals. The standard approach is the Scherrer equation, which uses a peak's width to determine the size. The Scherrer equation is limited to crystallites on the scale of nanometers, with an upper bound of roughly 100 nm [Patterson, 1939]. As crystallites grow to be larger than 100 nm the central assumption of powder diffraction breaks down, the sample is no longer diffracting equally in all directions. This causes azimuthal deviations in the scattered intensity. These deviations can be analyzed via

statistical or spot counting methods to determine the crystallite size [He, 2018; Yager and Majewski, 2014].

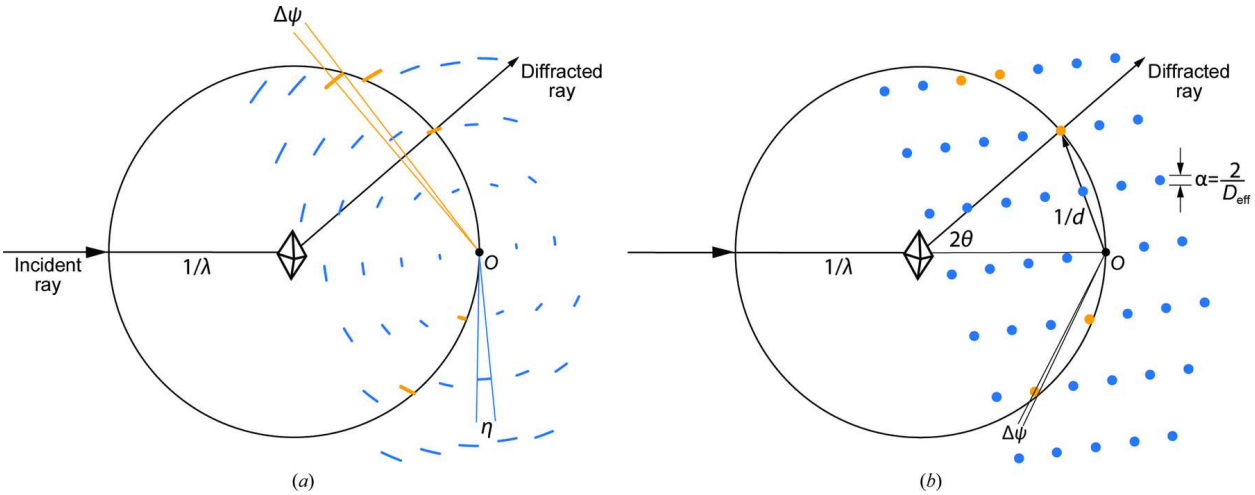


Figure 2.1: Impact of mosaicity and domain size on the reciprocal lattice and observed diffraction from [Sauter *et al.*, 2014]. The circle denotes the Ewald sphere, with the reciprocal lattice in orange and blue. Points in reciprocal space that touch the Ewald sphere, in orange, are in the diffraction condition and will show up as spots on the detector Mosaicity has the effect of smearing out delta functions into Gaussians, whose size and shape describes the distribution of orientations and grain size of the single crystal.

Synchrotrons produce x-rays by accelerating electrons through a nearly circular pipe. The electrons are turned by magnetic fields created by bending magnets, with each field causing the electrons to give off x-rays. Synchrotrons also incorporate insertion devices that produce strong oscillatory magnetic fields. These fields produce higher energy photons with greater brilliance by accelerating the electrons more. In this way a single synchrotron can support many experiments at once by providing each experiment, usually called a beamline, with a dedicated magnetic field. X-ray diffraction and scattering beamlines usually fall into one of two categories based on their experimental setup. High resolution beamlines like 11BM at the Advanced Photon Source used point detectors to measure the x-ray scattering pattern. The point detectors travel along a particular angular trajectory to capture the scattered photons. Because the motors moving the detectors can move in small increments, these beamlines provide high resolution data. The detectors are sometimes equipped with crystals designed to only accept one energy of x-ray, further enhancing the resolution of the

instrument. Point detector based approaches can be too slow for techniques which require many scattering patterns, like tomography, or in-situ techniques where time resolution is of the essence. For these experiments area detectors are used to capture the data. Since each pixel is similar to its own point detector the area detector can perform upwards of 4 million measurements at once [Chupas *et al.*, 2003].

Tomography

Introduction

Tomography is a mathematical method to construct cross sectional images of a sample from a series of projections of the sample. Tomography has been successful in the medical field, where absorption tomography, called a "CAT scan", provides an in depth view of the body, enabling diagnosis of diseases without the need of exploratory surgery. While any kind of radiation can be used for tomography, provided that it penetrates the sample, x-rays ability to penetrate most materials make them the most commonly used. Additionally x-ray tomography images have contrast based on the electron density of the material, with materials containing light elements, like soft tissues, showing less contrast than heavier elements, like bones. The contrast can even be modulated via specific contrast agents containing heavy atoms.

Many pieces of tomography have been around since the early to mid 20th century, with publication of the Radon transform in 1917 [Radon, 1986] and x-ray tomography for examination of the lungs in 1953 [Pollak, 1953]. However, the technique truly took off with the advent of modern computing, which enabled the reconstruction of features with unprecedented speed and ease of use [Herman, 2009].

As previously mentioned the application of tomography to medical imaging has been extensive, winning the Nobel Prize in 1979. Tomography has also been applied to engineering materials, providing insight into the structure of nuclear fuel cladding during simulated

accidents [Grosse *et al.*, 2013], understanding of the structure in concrete [du Plessis and Boshoff, 2019], and evaluation of aerospace titanium castings [du Plessis and Rossouw, 2015]. Tomographic techniques have even been applied to electrons [Frank, 2006], enabling 3D cross sections of nanoparticles to be made [Yang *et al.*, 2017].

How Tomography Works

Data Acquisition

The fundamental unit of tomography data is the sinogram. A sinogram tracks the intensity, or any other scalar, as a function of translation and rotation and is obtained by performing measurements at a series of rotations and translations. While often this is performed sequentially, some recent work has shown that interleaving the rotation angles can speed up the acquisition, as fewer projections are needed when using an optimal strategy [Vamvakeros *et al.*, 2016; Bicer *et al.*, 2017]. In most imaging applications translations are not performed, instead a imaging detector is used to sample multiple translations at once. This is possible if the beam is larger than the sample and speeds the data acquisition considerably.

Reconstruction

The crux of tomography is the reconstruction algorithm. This algorithm provides the transformation of a series of projections taken at various angles into the cross-sectional image. Of course there is no such thing as a free lunch, so the transformation does not provide additional dimensions, it takes a translation dimension and a rotation dimension and transforms them into two orthogonal translation dimensions. Reconstruction algorithms usually fall under one of three designations: Fourier or Radon based, which use mathematical transforms to perform the reconstruction, Algebraic, which take a linear algebra based approach, and iterative or model based, where a model of the expected output is generated, usually via a physics model of the scattering and refined iteratively.

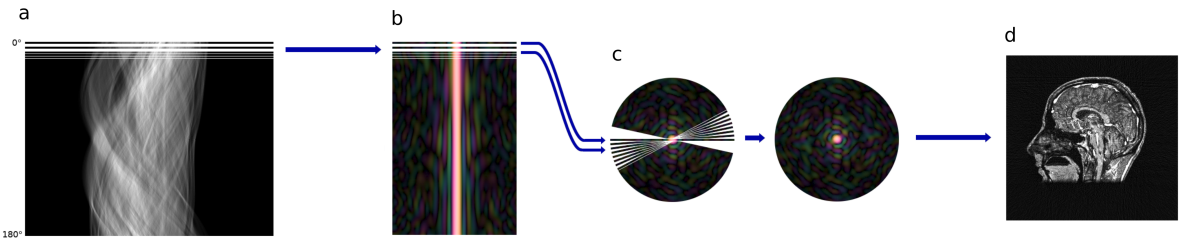


Figure 2.2: This figure shows the procedure for reconstructing a cross section from a series of measurements via a pair of Fourier transforms, adapted from [Selinger, 2018]. This is accomplished using the Fourier Slice Theorem. a) shows the series of projections taken at various angles b) shows the stack of Fourier Transformed projections c) shows the angular construction of the 2D representation d) shows the image resulting from the 2D Inverse Fourier Transformation of the representation

In transform based reconstruction the cross-sectional image is a function $f(x, y)$ with values at every point on the sample. The 2D Fourier transform of this function is

$$F(k_x, k_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-2\pi i(xk_x + yk_y)} dx dy$$

, where k_x and k_y represent the inverse coordinates. This can be rewritten as two 1D Fourier transforms

$$F(k_x, k_y) = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f(x, y) e^{-2\pi i y k_y} dx \right] e^{-2\pi i x k_x} dy$$

Thus if we choose a single slice where k_y is zero

$$F(k_x, 0) = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f(x, y) dx \right] e^{-2\pi i x k_x} dy$$

where the bracketed expression

$$\int_{-\infty}^{\infty} f(x, y) dx$$

is the definition of a single projection and the unbracketed portion is the Fourier Transform. This process can be performed for each projection angle and is called the Fourier Slice Theorem, stating that the 2D Fourier Transform of $f(x, y)$ is the same as the Fourier Transform of a series of projection slices. This can be used to reconstruct the cross-sectional

image by taking a series of projections, Fourier Transforming them, stacking them in 2D and performing the Inverse Fourier Transform. This is shown visually in Fig. 2.2.

The algebraic technique constructs a grid of pixels, based on the translation resolution. Each of these pixels is a single scalar value and each projections of through the sample is a sum of those scalar values. The reconstruction then consists of a set of linear equations to be solved.

Scattering Tomography

Introduction

X-ray scattering offers an alternative contrast mechanism to absorption. Unlike absorption, which is primarily dependent on the atomic number of the material, x-ray scattering depends on the atomic structure of the material. This enables x-ray scattering tomography to provide contrast for materials whose constituents have similar or identical atomic numbers but differing structures. Furthermore, as each pixel in the reconstructed cross section is associated with an entire scattering pattern, quantitative information about the atomic structure can be extracted for each pixel. For instance, each pixel can be associated with a nanoparticle size or defect concentration.

Experimental setup

The experimental setup for x-ray scattering tomography is similar to absorption tomography. A sample is translated and rotated in the beam to produce multiple projections which are then reconstructed. However, in the case of ctXRD and ctPDF a pencil beam is used rather than a wide beam. This requires the sample to be translated, rather than simply rotated causing the experiments to take more time. At each translation and rotation position a scattering image is taken and processed with standard x-ray scattering data processing. The resulting processed data is then reconstructed.

Examples

X-ray diffraction tomography was first developed in 1987 by Harding et.al. [Harding *et al.*, 1987] Atomic pair distribution function analysis (ctPDF) was developed in 2013 by Jacques et.al. [Jacques *et al.*, 2013] complementing ctXRD with real space analysis which is particularly effective at revealing the structure of nano scale materials. ctXRD and ctPDF have recently been applied to batteries [Jensen *et al.*, 2015; Sottmann *et al.*, 2017; Finegan *et al.*, 2019], catalysts [Jacques *et al.*, 2011; Jacques *et al.*, 2013; Ihli *et al.*, 2017], and fossils [Mrer *et al.*, 2018]. Battery studies use the localized structural information to understand how the lithium intercalated the different structures in the material [Finegan *et al.*, 2019]. ctPDF was used to understand how nanoparticle sizes evolved during catalytic operation [Jacques *et al.*, 2013].

Computed Tomography X-Ray Diffraction Setups

One of the main benefits of the ctXRD and ctPDF techniques is that the experimental setup for the sample is straight forward. All that is required is a translation and rotation stage coupled with a x-ray detector. The detector is usually an imaging detector, as opposed to a point detector, since the acquisition times will be shorter. This simplicity of setup makes ctXRD and ctPDF ideal for in-situ experiments, as shown in the work by Finegan et.al.[Finegan *et al.*, 2019] The majority of the complexity for these experiments is in the x-ray optics, which need to be precisely aligned to focus the beam into a tight pencil shape. Often this is achieved via a combination of slits, pinholes, and lenses [Somogyi *et al.*, 2005].

The Problems of Linearity and Invariance

All the computed values must be linear and translation and rotation invariant. One can think of the algebraic technique as a serial subtraction. If we are interested in the contribution of a volume (or pixel when discretized) of material to a projection we can subtract

the contributions of all the volumes in the beam's path. Doing the procedure en mass is essentially solving the linear system. However, this only works if the function which computes the value for each pixel is a linear operator. This is generally true for simple computations like the amount of x-rays absorbed by a sample, each pixel absorbs a given amount and the total amount of absorbed x-rays is the sum of each pixel's absorption, to first order. Note that this is not generally true, since each pixel leaves further downstream pixels with fewer x-rays to absorb. So long as the pixels do not absorb too many x-rays the approximation holds true. More complex pieces of information like peak positions or widths are generally not linear. Consider a uniform sample with a single x-ray diffraction peak with a constant width. In this case all the pixels have the same peak width, thus the value of the ith pixel is not the value of the projection minus all the other pixels. This can cause reconstructions to fail due to geometric issues when attempting to reconstruct non-linear operators, even in the absence of noise, which will be explored in more depth in Section 5.2.3.2.

Reconstruction algorithms rely on rotational and translation invariance of each pixel to perform a valid reconstruction. For the reconstructions to work a pixel's value must be the same regardless of it's position or orientation. The algebraic approach provides a good explanation as to why this is needed. To produce the independent measurements needed for a solvable linear system the sample must be rotated. For pixels which are not exactly on the rotation axis, this will result in a rotation and translation. If the pixel values changed due to the rotation or translation then the serial subtraction would not provide the correct results without a correction to the measured values. This can occur in absorption tomography, where strongly absorbing materials, like metal tooth fillings, can absorb a non-trivial amount of x-rays, depleting the x-rays for downstream absorption and changing the spectrum of the x-rays as certain wavelengths are preferentially absorbed. This effect is much more pronounced in x-ray scattering. While the x-ray scattering of powder samples is rotationally invariant, it is not translationally invariant, since any translation along the beam axis will change the sample to detector distance and therefore the position of the illuminated pixels on the

detector. The effects of sample to detector distance based invariance braking is discussed in more depth in Section 5.2.5.4. For samples composed of large single crystals the lack of rotation invariance causes the naive reconstruction algorithms to completely fail. Single crystals have a narrow angular window in which a plane is in the diffracting condition. Thus, when a rotation is performed to collect a projection the previously diffraction spot is no longer observed. Unlike the sample to detector distance issue, which can be mitigated and reconstruction still produces results which are representative of the sample, the vanishing of single crystal peaks causes naive reconstruction to fail outright. This does not necessarily imply that the problem is ill-posed, just that the simple linear approaches to the problem will not be sufficient to produce accurate results. Other approaches including more information about the physics of x-ray scattering may lead to successful reconstructions.

Chapter 3

Streaming Data Frameworks

Introduction

Accelerating data rates enable new classes of experiments across scientific fields. Many current and planned world class experimental facilities produce mountains of data. The Large Hadron Collider (LHC) produces 400 exabytes of data a year, the Laser Interferometer Gravitational-Wave Observatory (LIGO) produces terabytes a day, and the Linac Coherent Light Source-II (LCLS-II) expects to produce 100GB/s [Starr, ; LIGO, ; Thayer *et al.*, 2017]. These facilities require high data rates for their scientific missions, without them results which took years of experiments to obtain could have taken lifetimes.

Synchrotrons generate intense x-ray beams for studying the structure and excitations of materials, and also produce large volumes of data. New and upgraded accelerators will produce even more data as brighter beams are brought to bear on samples and detectors become faster and more sensitive [Blaiszik *et al.*, 2019]. Advances in x-ray detector technologies enable the acquisition of data at much higher rates with better quality. The new Pilatus 2M CdTe photon counting detector puts out 2.5 GB of data per second or 214 TB a day, while the currently used Perkin Elmer amorphous silicon imaging detectors put out 160 MB/s or 14 TB a day when running continuously. The photon counting nature of the Pilatus detector provides higher quality data by reducing electronic noise and while supporting a higher dynamic range [Dectris, ; Loeliger *et al.*, 2012]. These enhancements in both detector technology and beam brightness make more of the images usable per unit time,

allowing experiments to be completed faster and with greater time resolution. Advances in synchrotron brightness and flux also enable faster experiments and new experimental capabilities by putting more photons into a tighter beam.

The expected data rates for select x-ray facilities is provided in Table 3.1. The higher rate of data production will shift experiments from a flux limited regime to an analysis and sample loading limited regime. Fully exploiting the advances in experimental throughput requires novel approaches to handling these large data volumes. Data collected in the flux and detector limited regime only required offline batch processing and analyzing of data. Facility users would collect data, bring the data back to their home institutions and then perform data processing in a piecemeal fashion. This leads to lengthy delays in bringing results to publication, especially when data analysis reveals the need for additional experiments. Batch processing, combined with much higher data rates, exacerbate this delay between collection of data and gleaning insight. The batch approach does not scale to terabyte and exabyte a year data rates. Indeed, few institutions have the capacity to even store the data.

Facility	2021 (PB per year)	2028 (PB per year)
ALS	3	31
APS	7	243
LCLS/LCLS-II	30	300
NSLS-II	42	85
SSRL	15	15

Table 3.1: Data generation rates at major x-ray lightsource facilities. The majority of the facilities in this table (ALS, APS, LCLS) are undergoing upgrades which will result in brighter beams. The NSLS-II is still building out its beamlines and so has not reached it full data taking capacity. SSRL is not undergoing upgrades, thus its data rates are the same [Campbell, 2019].

Streaming data processing, where the data is processed and reduced as it is being collected, has transformed the handling of large data volumes at scientific facilities. For example, the LHC uses streaming data processing to make acquired data more manageable by selecting interesting subsets of the whole set of acquired data and down-sampling, resulting in outbound data sets almost 100 $\times$ smaller [Starr,]. These data sets are much easier to

store for subsequent use and enables the processing of data whose volume is too big to fit in one computer's memory.

ctPDF and other measurements which use x-ray scattering to map a sample's structure, need streaming data processing's downsampling capability, as they often entail large data sets, containing thousands to tens of thousands of images. Holding all this data in memory at once would require terabytes of RAM, which is often infeasible. Streaming processing of the scattering images to 1D patterns reduces the memory footprint of the data by a factor of one thousand, making the data handling viable. Streaming also enables real time visualization of reduced data, empowering users to interpret the results on the fly, helping to design or modify the experiments during operation. This accelerates experiments by enabling users to halt lackluster experiments and extend well performing experiments, which is especially important for the ctPDF technique where a single experiment can take many hours. Additionally, streaming data processing enables autonomous experiments, which need live reduced data to provide real time feedback between independent variable selection and experimental outcomes, which could further accelerate materials discovery [Bicer *et al.*, 2017; Granda *et al.*, 2018; Pablo *et al.*, 2019].

Facilities, like the LHC, often use Field Programmable Gate Arrays (FPGAs) to provide their streaming data reduction. FPGAs allow scientists to essentially program the data processing procedure into the circuitry itself. Building the software into the hardware provides faster computation than standard programming approaches. The instrument sends the data to the FPGAs which send the results to any downstream consumers, reducing large data volumes in real time. "On the wire" approaches, which use network hardware to perform processing, provide similar functionality by consuming and creating streams of data flowing through a network. FPGAs and "on the wire" based analysis approaches suit facilities that produce homogeneous data sets and only need to provide a handful of pre-set analyses at once. Synchrotrons often have many different experiments running at once, each with their own data acquisition and processing requirements. These requirements tend to change from

user to user and experiment to experiment, creating new procedures on the order of minutes to days. The rapid fluctuations in experiment type and user interests make the resulting raw and analyzed data heterogeneous. This heterogeneity makes FPGAs and "on the wire" based analysis approaches inappropriate.

X-ray scattering, especially ctPDF, experiments are an example of streaming heterogeneous datasets. Raw x-ray scattering data can be analyzed many ways, from blob detection on images to grain size extraction of 2D diffraction to strain evaluation of 1D diffraction patterns and PDFs. Each of these analyses produces a wide range of outputs further compounding the heterogeneity. These varied analysis approaches consume and produce many different kinds of data making the streams heterogeneous. This is doubly true of ctPDF where these disparate pieces of data are reconstructed into 2D and 3D data sets. Thus, a one-size-fits-all "on the wire" approach can be unwieldy. The streaming data processing approach has been useful for x-ray absorption CT, providing ways for autonomous experimentation [Bicer *et al.*, 2017].

With analyzed data being produced in such large volumes it can become difficult to find the data of interest, like searching for a needle in an ever growing haystack. Even after finding the data it can be difficult to properly compare data sets, especially if they are heterogeneous. Since the data is produced via automated pipelines it can also be difficult to understand exactly how that data was produced and how to properly reproduce it. This is less of a problem for homogeneous data streams, as the data is automatically comparable with other data produced by the facility, data processing protocols are tightly controlled and change infrequently, and there are a handful of fields to search over for finding relevant data. However for heterogeneous streams, curation, discoverability and reproducibility of data sets is a concern.

ctPDF data processing pipelines require many different pieces of configuration, controlling pre-reconstruction, reconstruction, and post-reconstruction parameters. These parameters can have an important impact on the results of the analysis. Tracking them, along with

the experimental parameters, software environment, and the data processing pipeline being used is important to understanding how the analyzed data was produced. Additionally, this metadata provides important hooks for searching through the data, enabling direct comparison of how different processing parameters changed outcomes or how different sample preparation procedures impacted material properties. In this way the problem of ctPDF data's heterogeneity is turned into a strength, enabling discoverability and reproduction of results. These configurational knobs are important metadata for discovering, understanding and reproducing the analysis.

We have addressed these issues by creating a streaming data processing framework which combines high throughput analysis with heterogeneous data handling. This approach enables users to create pipelines flexibly which process heterogeneous data and capture metadata to provide searchability and insight into the analysis procedure. Additionally this approach allows for the easy parallelization of computation, and reproduction of analyzed results from raw data. The framework we developed is partitioned into two, the first, `rapidz` handles the flow and transformation of base homogeneous data and the parallelization of the data processing. The second, `SHED` is designed to translate between between heterogeneous data and homogeneous base data, and produce and track metadata about the analysis itself these are described below.

Rapidz

Introduction

An unbounded stream of data constantly produces new material. Twitter feeds, stock exchanges, weather satellites are all examples of data streams because their systems are constantly updating with newly generated data. Streams of data create problems for batch processing, where a block of data is downloaded and analyzed directly. Streaming data processing embraces the ceaseless influx of data by handling each piece one at a time without

the need for batching. For example, in the case of a Twitter feed, a single Tweet would be analyzed when it arrived and not when the full news cycle is over.

While various software products have been created to handle streaming data processing, nearly all share a common representation of the data processing procedure, often called a pipeline, as a directed acyclic graph (DAG). The DAG model's ability to create complex workflows from otherwise simple pieces is particularly powerful. These DAGs describe how pieces of data flow between the various data processing procedures. The DAG represents data processing steps as nodes in the graph, and flow of data between these steps as edges. The DAG model allows for individual nodes to perform small tasks, like multiplying two numbers together. These small tasks are then combined to perform more complex tasks. An example of a data analysis pipeline is shown in Fig. 3.1

Most of the existing streaming technologies were written for Java and Scala with compatible Python APIs. This means that the resulting software is not particularly pythonic, making the interfaces rather clunky. While many of the existing streaming technologies use the DAG as the central model for thinking about streaming data processing, often that is where the similarities end. Many of the existing systems, even ones written with Python in mind, like pilot-streaming, combine the DAG logic with other pieces of software, like how to schedule jobs on a HPC cluster [Luckow *et al.*, 2018]. Amalgamating multiple problems like this leads to more brittle software, as it restricts application the of the code to other use cases, and tends to cause the problems that one part of the code tries to address to bleed into the other parts of the code. Additionally, these systems often create their own data types, as in the case of Spark based systems and Resilient Distributed Datasets (RDDs)[Zaharia *et al.*, 2016]. Custom data types can introduce the need to translate between the types provided by standard libraries and the new library's data types, making the streaming library incompatible with most of the software it is try to use.

We attempt to avoid these issues by making certain that our streaming library does one thing, and only one thing, well [Robinson *et al.*, 2019]. This idea is also known as "separation

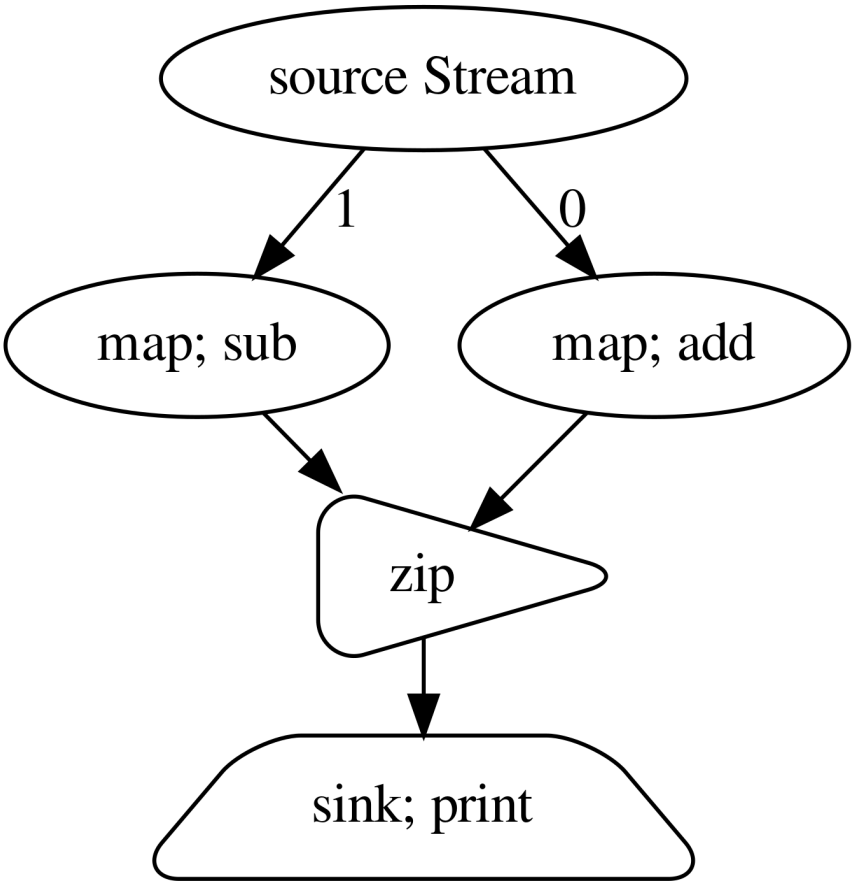


Figure 3.1: Directed Acyclic Graph representing a data processing workflow. This DAG represents the combination of two numbers which have been produced from an input number by subtracting one and adding one to the input.

of concerns". In previous work streaming data processing systems overlapped with database interfaces, resource management tools, and other pieces of functionality. Issues like these are not generally fixable once the library has already been implemented and adopted, hence we designed a new library `rapidz`. The main design consideration was to make certain that the system handles streaming data well and leaves openings for other systems, which may handle databases or resource management, to communicate with the streaming data processing system without explicitly tailoring the system to those applications. This way users can swap out pieces of the software for better ones seamlessly, so long as the new pieces communicate with the pipeline in the same way.

`rapidz` Library Design

Architecture

`rapidz` is a streaming data processing library written in Python. The focus of the design of rapidz is simplicity of the node classes and separation of concerns. The main entities of `rapidz` are the `Stream` class and its subclasses. These classes are used to form nodes of a data processing DAG, and all have three main methods.

1. The `init` method specifies which other nodes, if any, the node being created will listen to for incoming data.

2. The `update` method specifies what to do when new data is presented to the node

3. The `emit` method describes how to propagate data downstream to nodes listening to this node.

Built on this three part structure, various subclasses of the `Stream` class can be created, providing capabilities like applying a function on the data to transform it (`map`), removing data from a stream based on a criterion (`filter`), and combining data from different streams (`zip`, `combine_latest`).

For example, the structure of the `map` class, shown in Listing 3.2, which applies a transformation, `func`, to each piece of data passing through the node. The transformation acts like, and in general is, a python function.

```
class map(Stream):
    def __init__(self, upstream, func, *args, **kwargs):
        self.func = func
        self.kwargs = kwargs
        self.args = args
        # listen to the upstream nodes
        Stream.__init__(self, upstream)

    def update(self, x, who=None):
        # apply function to data
        result = self.func(x, *self.args, **self.kwargs)
        # send data downstream
        return self._emit(result)
```

Figure 3.2: Example `map` source code showing the new `init` and `update` methods

The transformed data is then sent downstream to other nodes which will use it for their own processing. Similarly the `combine_latest` class, shown in Listing 3.3, combines the most recent data item from one stream with the most recent data item from other streams:

`combine_latest` is commonly used to upsample or downsample data. Upsampling takes two streams of data and creates a stream which has the same rate as the quickest updating stream. For instance, consider a x-ray detector and the calibration of that detector. The calibration changes slowly, or not at all, while the detector's value can change at a rate of 10 Hz or more. In the upsampling case the new data is a combination of the newest value from the detector and the most recent value of the calibration, enabling users to take advantage of the latest calibration without forcing them to take a new one every shot.

`emit_on` is a feature that enables `combine_latest` to be used for downsampling. When `emit_on` is provided it allows data to propagate from the `combine_latest` node only when a specified incoming stream presents a new data item. For instance, consider the previous x-ray detector with a temperature sensor, which has a high measurement rate. A `combine_latest`

```
class combine_latest(Stream):
    def __init__(self, *upstreams, **kwargs):
        emit_on = kwargs.pop("emit_on", None)
        self.last = [None for _ in upstreams]
        self.missing = set(upstreams)
        if emit_on is not None:
            if not isinstance(emit_on, Iterable):
                emit_on = (emit_on,)
            emit_on = tuple(
                upstreams[x] if isinstance(x, int)
                else x for x in emit_on
            )
            self.emit_on = emit_on
        else:
            self.emit_on = upstreams
        Stream.__init__(self, upstreams=upstreams, **kwargs)

    def update(self, x, who=None):
        if self.missing and who in self.missing:
            self.missing.remove(who)
        self.last[self.upstreams.index(who)] = x
        if not self.missing and who in self.emit_on:
            tup = tuple(self.last)
            return self._emit(tup)
```

Figure 3.3: Example `combine_latest` source code showing its `init` and `update` methods.

node with an `emit_on` from the detector would downsample the data, only allowing the combination of the latest detector value with the latest temperature value, even if the temperature updated 100 times in the interval between detector shots. This looses the intermediate temperatures with only the relevant one, that is the most recent one, retained. Out of order access is prevented by not accepting new values until the current value's computation has been finished. This guarantees that the most recent temperature will be the closest to the detector image when it arrives. This idea, called "backpressure", is explored in more depth in Section 3.2.2.2.

Impacts of Architecture

Good software produces properties which were not explicitly written into the source code, but naturally arise from the interaction of various pieces of the code. The presence of these properties indicate that one or more abstract concepts that the code seeks to express work well together. These properties are critical for understanding how a piece of software works and understanding if the correct abstractions have been made. Rapidz focus on simplicity and separation of concerns leads to some important emergent properties.

For example, take the pipeline shown in Fig. 3.1 the numbers 1, and 0 represent the order in which data will propagate. If the number 1 was pushed into the source node, first it would pass to the add node via the path labeled 0, which would add 1 and pass the resulting 2 to the zip node. Since the zip node combines data in a one-to-one fashion the node caches the data and returns execution to the source. With execution returned to the source the data then flows along the 1 path where the number 1 is subtracted from the input, resulting in 0. The 0 is passed to zip which, having two values, passes the tuple (2, 0) to the print statement.

A stable order of execution is one of the emergent properties of rapidz. For any given rapidz pipeline and known order of inputs the data which comes out of the pipeline will come out in a well defined order. The above example shows how the execution order is stable and

deterministic.

Another property is that the pipeline does not accept new data until all of the execution is finished on the existing data, called "backpressure". This prevents the data from being computed and returned out of order and is especially important for scientific applications where data order is tightly correlated with other information, like temperature and motor positions. Backpressure works by taking control of the python process running the pipeline, computing the results of each node's computation. Since the node computations occupy the entire process no additional data can be ingested by the pipeline, since that requires the python process to be free to receive the new data. Similarly, data is computed in order, since there is an order to the stack of computations that are performed, with the first piece of data computed first.

Rapidz does introduce some computational overhead associated with the DAG management of the code execution, as opposed to a script written without a pipeline. The library's simplicity helps to limit this overhead. A rapidz pipeline adds a few microseconds of overhead per element [Rocklin, 2017]. For many computations, like those applied to x-ray scattering, the computation of the mathematical transformation takes more time than the rapidz overhead by orders of magnitude.

The DAG itself, rather than the node classes, holds the complexity of combining, transforming and controlling the data. By pushing complexity to the DAG users can conceptualize and visualize their data processing as a graph rather than worrying about the specific internals of the classes. Most importantly, rapidz does not care what your data is so long as it is a python object. Rapidz provides a way to describe the order in which transformations are applied to pieces of data for an arbitrary number of inputs.

Parallel Streaming

Parallel processing provides an important route to scaling data processing and analysis. In parallel processing, many computations are performed at once, speeding up the processing

compared to if the computations were done in serial. Many modern processors are built for parallel processing, with general purpose computation on graphics processing units (GPGPUs) being an especially economical source of parallel computing. Many large facilities also support parallel computing via cluster and super computing. Parallel computing can even use cloud resources to provide scalable computing.

While parallel processing can speed up analysis, it often significantly complicates the software. These complications stem from a coupling of the analysis code, which handles the ordering of operations, and the parallel code, which manages the processors and memory making certain that maximum efficiency is maintained and that the computer doesn't run out of memory. The coupling makes parallel analysis code quite brittle, as details of the parallel processing implementation, how many cores to run on, the available memory, etc. are hard coded into the analysis, making it difficult to run in any other computing environment. Ironically, this can make the code more difficult to parallelize further as one must remove the hard coded information when moving to a larger parallel computing platform.

`rapidz` is well placed to fix this issue. Since `rapidz` acts as a way to specify which transformations to apply in what order, the actual computations can occur wherever is convenient. Python expresses this concept of performing computations elsewhere via the `Executor` class.

For instance, a user gives an `Executor` a python callable, which is usually a function, and any additional arguments and keyword arguments to provide to the function. The `Executor` then returns a `future`, a computational IOU, representing the result of the requested computation. The `Executor` then schedules the computation on whatever resources it can access. The user can then query if the computation the `future` represents is finished or request the result, preventing any further action until the computation is finished and the user can access the result. This means that the user can submit multiple jobs to the `Executor`, creating multiple `futures`. Depending on the infrastructure the user has access to, the computations for these `futures` could occur in parallel. The `Executor` model provides a separation be-

tween the desire for a computed result and the actual execution of the computation making it a powerful abstraction. Thus, the user does not need to know the specifics of the memory allocation, network protocols, or CPU usage to request the results. This frees up the computer to run multiple futures in parallel, potentially on hardware not even local to the user. `Executors` can even make decisions based on information that the user does not have while specifying the data processing, like requesting additional computational resources. Various packages behave like `Executor` classes enabling various parallel execution models, including projects like `dask` which enables automatically scaling computing on clusters, or `MPI` which is used extensively on supercomputers without changing more than a handful of lines of code.

Rapidz, combined with executors, provides a good platform for parallel computing. Since Rapidz is all about specifying what is to be done to each piece of data in which order, the problem that executors solve is orthogonal. Thus, rapidz can use executors to perform each computational step in parallel elsewhere, on the local computer, a dask cluster, or even a supercomputer. To make this work two additional nodes are introduced, scatter and gather. Scatter nodes pass the data into an executor, producing futures. These futures are then passed down the pipeline. When a future is passed into a computational node, like map, the same executor produces a new future. This new future represents the result of the mapped function applied to the result of the prior future. Futures can be chained this way, creating a stack of computations which may or may not have completed yet. When these futures are passed to a gather node, the node calls their result method, halting the pipeline until the results have come back. Since the scatter and gather nodes respect backpressure only one piece of data is allowed in the pipeline at a time. However, a buffer node breaks the backpressure and allows multiple computations to happen at once. The buffer node caches the futures up to a user defined limit. The limit, once reached, will cause the pipeline to stop accepting new entries, producing the same effect as the single process backpressure mentioned in Section 3.2.2.2. The buffer then waits for a future to have finished its computation and passes the future downstream to a gather node where it is turned back into base types and

the buffer now allows the pipeline to accept more data. This approach avoids deadlocks by having the Rapidz graph structure be acyclic, thus no computation can rely on a second computation that in turn relies on the first computation.

A analogy for this is a tag team computation, where a group of people sit at desks. Each person does exactly one computation, e.g. adding two numbers together, taking the mean of a set of numbers, etc. In the serial setup the first person in the pipeline receives a number on an index card, they apply their transformation to the data, write the result on a new index card and pass it to the next person. Similar to how the pipeline operates, if any computation is slow then the entire pipeline slows down as that person crunches their numbers. In the parallel version of this analogy, the person does not perform the computation but simply writes the computation to be performed on the outside of an envelope which contains the index card and is passed to the next person, who puts that envelope in another envelope with their computation on the outside. This continues until the data is passed to the buffer desk. At the buffer desk the envelopes are passed out to many people who can perform the computations by opening all the envelopes and doing each of the transformations in order. The buffer node then waits for the results from the first set of envelopes to be returned at which point it passes it further downstream. In this analogy the parallel computation is performed when the envelopes are opened, the actual method of how the computation happens is separate from our declaration of what we'd like done. This means that the pipeline does not need to get bogged down in the details of running the computation, it just needs to state what it would like done to which data in what order and allow other, potentially more computationally adept, computers to do the heavy lifting.

Impacts of Parallel Streaming Architecture

`rapidz`'s `Executor` driven parallel processing architecture enables easy parallelization, which can scale without any changes to the data processing topology. All that is required is three extra nodes to send the data to the computational cluster, buffer the futures while awaiting

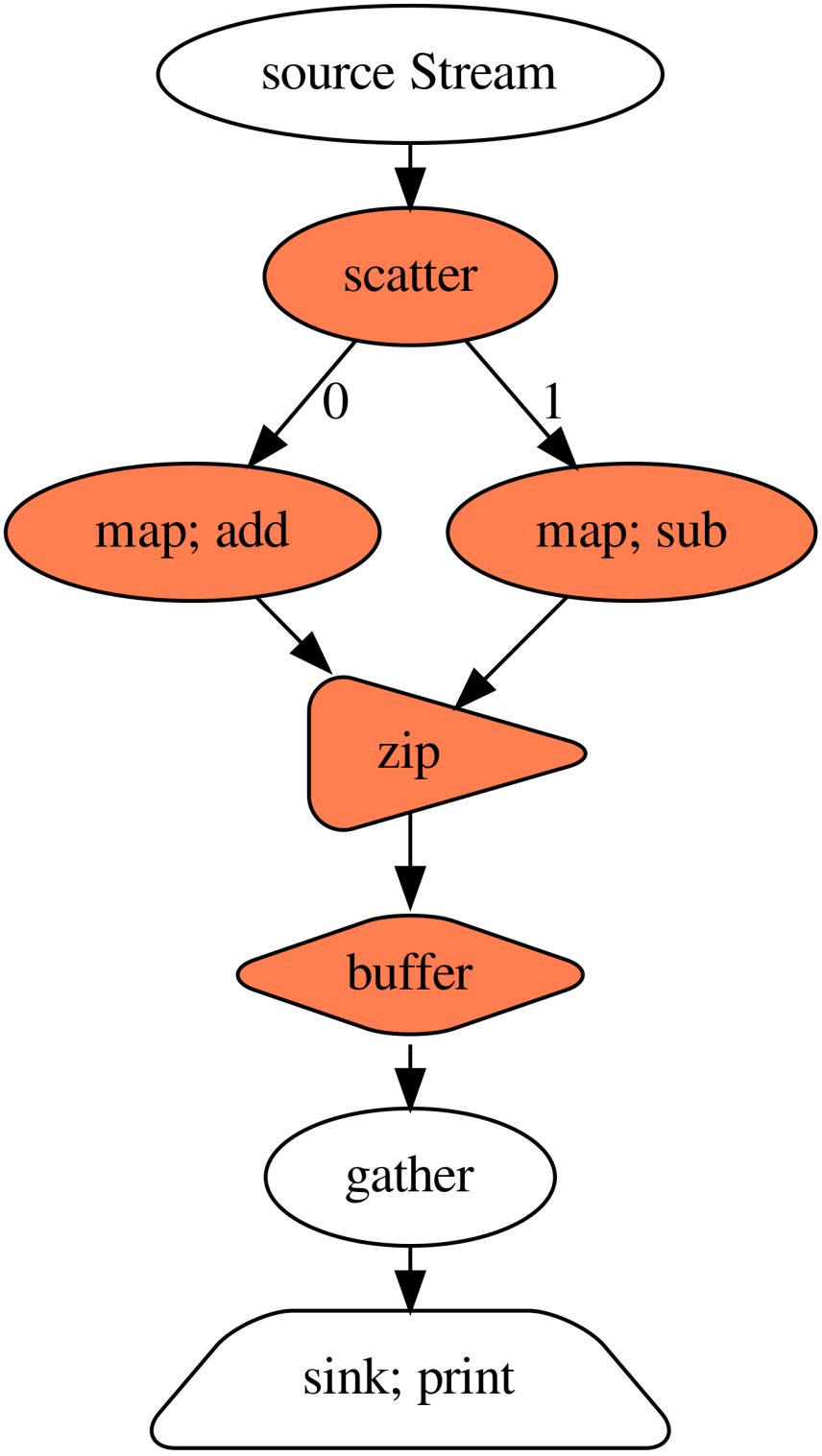


Figure 3.4: The same graph as 3.1 but parallelized. Note the additional `scatter` `buffer` and `gather` nodes. Parallel computation occurs with the coral colored nodes.

results, and gather the data back to the local computer when the computation is finished. Fig. 3.4 shows a parallel pipeline version of Fig. 3.1. The parallel pipeline shown in Listing 3.6 is much faster than the serial pipeline shown in Listing 3.5 for the same amount of work, since it can run all the jobs at once. If the resources are limited, which can be done artificially by reducing the buffer size below the total amount of the data, then the for loop will take longer. For instance if the buffer in Listing 3.6 was set to five, then the for loop would take about one second, as the loop must wait for the data to clear from the buffer at least once, which takes one second due to the sleep. If the parallel pipeline contained five maps the submission of the data would take about 0.007 seconds, almost the same as the one map since submission to the executor is fast. One can see from the code in Listing 3.6 and Listing 3.5 that the difference is exactly three lines, making implementing parallel pipelines quite easy. This approach is useful for streaming data processing where data points can be treated independently.

Filter nodes are an important part of the rapidz framework, providing the ability to remove data from the stream. This can be useful for thresholding data analysis to only operate when enough counts are on the detector or because the scan is of a certain type. These nodes are particularly difficult to implement in parallel streaming. In normal operation the filter nodes use a function, called a predicate, to determine if a piece of data is allowed to propagate downstream, if the predicate is true then the data propagates, if not then the node does nothing effectively removing the data from the stream. This is not possible when operating on futures, since the results are unknown until the end of the pipeline. The way this is handled is via two python decorators. Decorators are functions which take in other functions and return a modified version of those functions. A simple decorator might take in a function and return a second function, whenever that second function is run it calls the first function and then prints the result of that first function in addition to returning the results. In the case of filter the first decorator takes in a predicate function and based on the result of that function returns either the value of the stream or a `NULL_COMPUTE` sentinel.

```
from rapidz import Stream
import time


def sleepy_increment(x):
    time.sleep(1)
    return x + 1


source = Stream()
L = source.map(sleepy_increment).sink_to_list()
t0 = time.time()
for i in range(10):
    source.emit(i)
print(time.time() - t0)
while len(L) < 10:
    time.sleep(1e-4)
print(time.time() - t0)
```

Figure 3.5: Serial pipeline with an increment taking at least one second. The for loop takes ~ 10.01 seconds and the while loop takes no time at all. This is because all the sleeps are run in series, with 10 sleeps of 1 second a piece but the results are finished once all the of the data is submitted to the pipeline since there is no backpressure release via buffer.

```
from rapidz import Stream
import time
def sleepy_increment(x):
    time.sleep(1)
    return x + 1
source = Stream()
L = (
    source.scatter(backend="thread")
    .map(sleepy_increment)
    .buffer(50)
    .gather()
    .sink_to_list()
)
t0 = time.time()
for i in range(10):
    source.emit(i)
print(time.time() - t0)
while len(L) < 10:
    time.sleep(1e-4)
print(time.time() - t0)
```

Figure 3.6: Parallel pipeline with an increment taking at least one second. The for loop takes ~ 0.006 seconds and the while loop takes 1.01 seconds. This is because all the sleeps are run in parallel, thus the amount of time to compute the whole data set is the time it takes to compute one piece of data, assuming that one has enough resources to service all the computations at once. The for loop is especially fast because the pipeline is only submitting the jobs to the executor, which is fast.

The sentinel is a special python string, which signals that any additional computations should not be performed. The string is chosen so that the probability of the string being generated by user code is small. The second decorator is used with nodes which compute values, map, accumulate, etc. This decorator determines if any of the function arguments or keyword arguments are `NULL_COMPUTE`, if so the internal function is not triggered and a `NULL_COMPUTE` is returned. If a `NULL_COMPUTE` reaches a gather node then the result is not reported, making it as if the data was filtered out. In the analogy of the envelopes, one of the envelopes has the command to either continue computing the stack or throw out the envelope based on the value of the current computation.

Some nodes currently implemented in serial have no equivalents with parallel processing. Unique, which only reports novel data, is one of these nodes. In serial pipelines unique uses an internal cache to check if a given value is novel, and based on that assessment passes the data downstream or not. However, in distributed systems this would require a distributed cache which can be quite difficult, or the direct local evaluation of the computation which would defeat the purpose. In the analogy of envelopes, the serial execution has one of the desks contain a running list of all the seen results to be compared against. In the parallel analogy this table would need to be located in such a way that it could be seen by all the people actually computing the results, which is not feasible if those people were in different rooms. Additionally this can produce issues with out of order execution. While the results are gathered in order, the futures themselves can be computed in whichever order is most convenient for the scheduler. This means that the cache can be corrupted with entries which are to be entered after the current data would be processed, causing data to be missed. These issues may be solvable with fast key-value stores like RocksDB and careful tracking of the order of computation. However, at time of writing rapidz does not implement these solutions.

While the implementation of parallel pipeline nodes can be quite complex, the implementation of parallel pipelines themselves are not. Currently rapidz supports two main

backends, dask and threads, with the easy addition of any interface which uses the Executor class. This ease of use enables pipelines written and prototyped for single process execution to scale to almost arbitrary parallel execution schemes. The separation of data, computation topology, and execution enables flexible streaming data processing where the user experience focuses on the development of the computation topology and not on the provisioning and management of the parallel processing resources.

Simplifying the Construction of Complex Pipelines

Complex pipelines can become quite large, including many nodes and edges, making them somewhat unwieldy and difficult to adapt for reuse. Pipelines that operate under many conditions become particularly complex, as each condition can add a multitude of nodes and edges. Pipelines can become more modular via a process called "chunking". Chunking is the process in which pipelines are broken up into smaller pieces. These smaller pieces could be put into their own modules, so when the pipeline is to be built it can be imported in pieces and reassembled. However, creating a pipeline via imports makes it brittle as a module can only be imported once. This means that if the pipeline needed to be regenerated for whatever reason the entire interpreter would need to be restarted. A better approach is to put the pipeline piece inside of a function, which creates that part of the pipeline when the function is run. This design pattern is called a "factory function", since this is a function which builds another piece of software.

There are three main parts of a chunk factory: requirements, the pipeline chunk itself, and the resulting namespace. Pipeline chunks, regardless of if they are created via factories or imports, usually have requirements. Requirements are upstream nodes which a pipeline chunk is expecting to exist when adding the chunk to the pipeline. In chunk factories these requirements are part of the arguments of the function, making them required to run the factory. Users can inspect which arguments are required by either reading the documentation or using the python `inspect` library to inspect the function signature. The `inspect` library

can be used to spoof requirements, providing empty nodes attached to nothing in the place of required nodes. Keyword arguments can provide additional information, like configuration parameters. The pipeline chunk itself creates new nodes by either instantiating them outright with no upstream nodes, or by connecting to upstream nodes which were listed as arguments to the pipeline. Finally, the chunk returns a namespace representing all of the nodes and parameters of the pipeline as it existed at the end of the function. These chunks can then be run, creating pipelines from chunks. An example chunk factory is shown in Listing 3.7

```
def image_process(
    raw_foreground,
    raw_foreground_dark,
    raw_background,
    raw_background_dark,
    bg_scale=1.0,
    **kwargs
):
    dark_corrected_foreground = raw_foreground.combine_latest(
        raw_foreground_dark, emit_on=0
    ).starmap(op.sub)
    dark_corrected_background = (
        raw_background.combine_latest(raw_background_dark, emit_on=0)
        .starmap(op.sub)
        .map(op.mul, bg_scale)
    )
    bg_corrected_img = dark_corrected_foreground.combine_latest(
        dark_corrected_background, emit_on=0
    ).starmap(op.sub, stream_name="background_corrected_img")
    return locals()
```

Figure 3.7: Pipeline chunk for performing background and dark correction to x-ray scattering images. This chunk has four requirements and one optional parameter.

Chunk factories could be used on their own, providing the arguments by hand to produce larger and larger pipelines. However, `rapidz` provides an automated `link` function that makes this process simpler. The `link` function enables users to provide a list of chunks and keyword arguments from which to create a pipeline. The code for the `link` function is provided in Listing 3.8. The `link` function calls each chunk function in order, with the

arguments to the next chunk function being pulled from the currently available namespace. The chunk function then adds its own nodes into the namespace, making more nodes available for connections. This requires the names of the arguments for each chunk function to the be the name as the node from a prior chunk. Additionally, this requires all chunks to take `**kwargs` so any unneeded nodes and parameters can be passed through without an unexpected keyword argument error.

```
def link(*args, **kwargs):
    namespace = kwargs
    for pipe in args:
        new_namespace = pipe(**namespace)
        if new_namespace:
            # flatten out the kwargs so we
            # don't have kwargs all the way down
            namespace.update(new_namespace.pop("kwargs", {}))
            namespace.update(new_namespace)
    return namespace
```

Figure 3.8: The link function, which iterates through a list of chunk factory functions and builds the next pipeline chunk with the available namespace.

Advantages and Disadvantages

Rapidz consists of a series of design decisions. These design decisions have advantages and disadvantages discussed below.

Disadvantages

One of the most glaring disadvantages is that writing pipelines is more complex than writing plain python code. This is especially true when the python code is inside a Jupyter notebook [Perez and Granger, 2007]. Interactive python sessions, including Jupyter, make intuitive sense and provide easy real time feedback between the code and the user. Pipelines can be more complex to write, requiring careful weighing of the order of execution, the exact state of the graph at any time, and handling of numerous contingencies. Network effects, where

one part of the graph influences other parts of the graph can make this particularly tricky. Tools like the linking and chunking systems are designed to help users to build pipelines, but future work will most likely continue to focus on these pain points. Some of this complexity may never go away, handling an unbounded flow of data may inherently be more complex than the static data sets handled by interactive python sessions.

Advantages

Rapidz provides some major advantages over standard data analysis tools. A pipeline provides structure to the analysis procedure that might not otherwise exist in interactive sessions. This is especially true of Jupyter sessions, where top to bottom execution order can be broken, leading to analysis which does not run properly in a single pass. Additionally the graph structure produced by rapidz allows the creation and use of tools which manipulate the graph itself. These tools could check if a pipeline will fail based on the number of incoming edges and number of arguments in map nodes. They could automatically produce GUIs which enable users to modify pipeline parameters on the fly by walking the graph and exposing all the additional arguments and keyword arguments provided by the graph nodes. Live visualization could be performed on the graph, showing its exact status at every point, making debugging easier. These kinds of tools are hard to produce, or impossible with standard interactive python systems, since they lack the needed graph structure to represent the computation's relationships. As discussed above, the ease of parallelization is an advantage for the rapidz system. By separating the data and the processing order from the execution of the processing, a handful of lines of code can be used to provide significant speedups. This could be much more difficult with interactive python sessions or other pipeline tools, which rely on their own infrastructure to provide parallel computing.

SHED

Introduction

Unlike most sources of streaming data, experimental data is quite heterogeneous. Experimental data could be associated with a single reading from a detector or motor, a series of readings, a single sample, or an entire campaign of measurements with multiple samples, techniques, and investigators. Consider a campaign of experiments aimed at understanding how a series of samples structures are impacted by radioactive bombardment. There is information about the sample: where did it come from, how much irradiation did it experience, when was it made, what is the composition,etc. There is information about the experiment: who ran the experiment, what is the beamline wavelength, what kind of scan is being run, was the experiment successful etc. There is information about the individual measurements: what is the value of the detector, when was the measurement made, etc. These heterogeneous layers all provide different types of information, all of which are valuable for processing and analyzing the results. Similarly processed and analyzed data has many layers, including the kind of analysis being performed, the units of the analyzed results, and the results themselves. No existing software systems are able to capture and interface with this level of heterogeneity in a streaming context. Streaming Heterogeneous Event Model (SHED) is designed to handle these issues by providing a translation between these heterogeneous data sources and rapidz data processing pipelines, while automatically capturing information about the data processing.

Provenance

Unlike raw data, which is produced once, analyzed data can be produced multiple times using different paths and the same base data to produce results. These results can be comparable, using the same units and attempting to describe the same phenomenon, or

not, using the same raw data for completely different analyses. The results can also be similar or wildly different, even if they were using the same analysis procedure with different parameters. These differences create issues around reproducibility, where two similar analyses can produce different results. For example there are a large number of tomography reconstruction algorithms, each of which produces the same type of result but don't always produce the same result, as previously discussed in Section 5.2.4. If these kinds of analysis parameters are not tracked then it is difficult to compare results between different parameters or researchers. This lack of this kind of reproducibility can lead to controversies, especially when software is involved. For seven years there was a bitter dispute over the nature of water between David Chandler and Pablo Debendetti, where a slightly different procedure for producing atomic configurations [Palmer *et al.*, 2018; Smart, 2018]. It is even possible for the same analysis with the same parameters to produce different results if the software versions are different, making it difficult to determine why the outcomes are different. This compounds the discoverability problems discussed above, where despite sharing the same base data the results could be different and even describe different effects.

Provenance provides a method to address these discoverability and reproducibility problems. The concept comes from art, where the tracking of art ownership enables potential buyers to know if they are purchasing a fake or the original. In the computational context provenance describes the tracking of every transformation applied to the data, along with metadata about that transformation. For instance the provenance of a number being multiplied by a constant on a computer would include: the number itself, the constant and the version of the software being used to multiply them. This combination of data and metadata helps address the problems of discoverability and reproducibility. To reproduce the results of the multiplication computation we need the information stored by provenance, allowing us to use the same version of software and constant to reproduce the result. Similarly this information helps discoverability by keeping track of the parameters one might use to query

their analyzed results.

Here we discuss two examples where the tracking of provenance facilitates better scientific outcomes. Tomography, as discussed in Section 2.2.2, relies on the use of a reconstruction algorithm to transform the sinogram, a series of data points taken at various rotations and translations, into a full 3D tomogram. Different reconstruction algorithms can be used to produce the tomogram. As will be discussed in more depth in Section 5.2, these various algorithms can produce different results. Keeping track of which algorithm was used to produce a given tomogram is important as the differences in results from one algorithm to the next could impact the scientific conclusions. Additionally, tracking of the data's provenance allows for direct comparison of the algorithms, enabling parameter and algorithm exploration. When provenance is not tracked, reproducibility can break down.

Although provenance provides a path to reproducibility and discoverability, having been cited by two Department of Energy reports [Bethel *et al.*, 2016; Windus *et al.*, 2017], capturing data provenance can be difficult. One of the main stumbling blocks for this is user interaction. Users often are not interested in entering large amounts of metadata into a database, especially metadata about their computational work [Davison, 2012]. Even when metadata is supplied by users that metadata could be inaccurate or incomplete. This means that any provenance capture system needs to provide automated capture. While VisTrails and Sumatra provide some automated provenance capture when working with batch data processing, neither provide provenance capture for streaming data processing discussed in Section 3.2 [Silva *et al.*, 2007; Davison *et al.*, 2014]. Conversely, few of the existing streaming libraries have a provenance system to capture the needed metadata.

Heterogeneity and the Event-Model

The NSLS-II Data Acquisition, Management and Analysis (DAMA) group document/event model is specifically designed to handle the heterogeneous raw data described above. The Event-Model partitions the various types of data into four main document types: start,

descriptor, event and stop. These documents also handle asynchronous data sources, allowing multiple detectors running at different acquisition rates to operate independently but as part of the same experiment. Start documents signal the beginning of the experiment and capture beginning metadata Descriptors provide information on the data which will be collected in a given data stream, a single start document can have multiple descriptors. Multiple descriptors is common when partitioning data which has different acquisition rates, or where the type of data is fundamentally different. For example light images and dark images are often put into separate descriptors, since the images are not comparable and their data processing is quite different. Events contain the data itself, detector readings, images, motor positions, and others. Events also contain timestamps for all the measured quantities. Stop documents provide metadata about the end of the experiment, if the experiment was a success, when the experiment finished, and how many events were captured per descriptor. Fig. 3.9 shows the chronological ordering of these documents for three different experiments.

The Event-Model specification enables data curation and discovery by providing handles for querying and comparing data. This is accomplished via the Databroker project, which is a database system to store data which is in the event model. In addition to storing the data, the Databroker provides a way to query data by the metadata in start documents and replay any stored experiment, providing a stream of data as if it was coming from the past experiment. The spec also allows additional systems to be written on top, including a richer query system using ElasticSearch. While not initially designed to capture processed and analyzed results, SHED uses and extends the Event-Model to capture the data and metadata from data processing and analysis.

SHED design

The first issue SHED addresses is the extraction of data from the Event-Model. Since the Event-Model has its own data structure and most data processing software operates on base types, integers, floating point numbers, and others, SHED provides a translation from

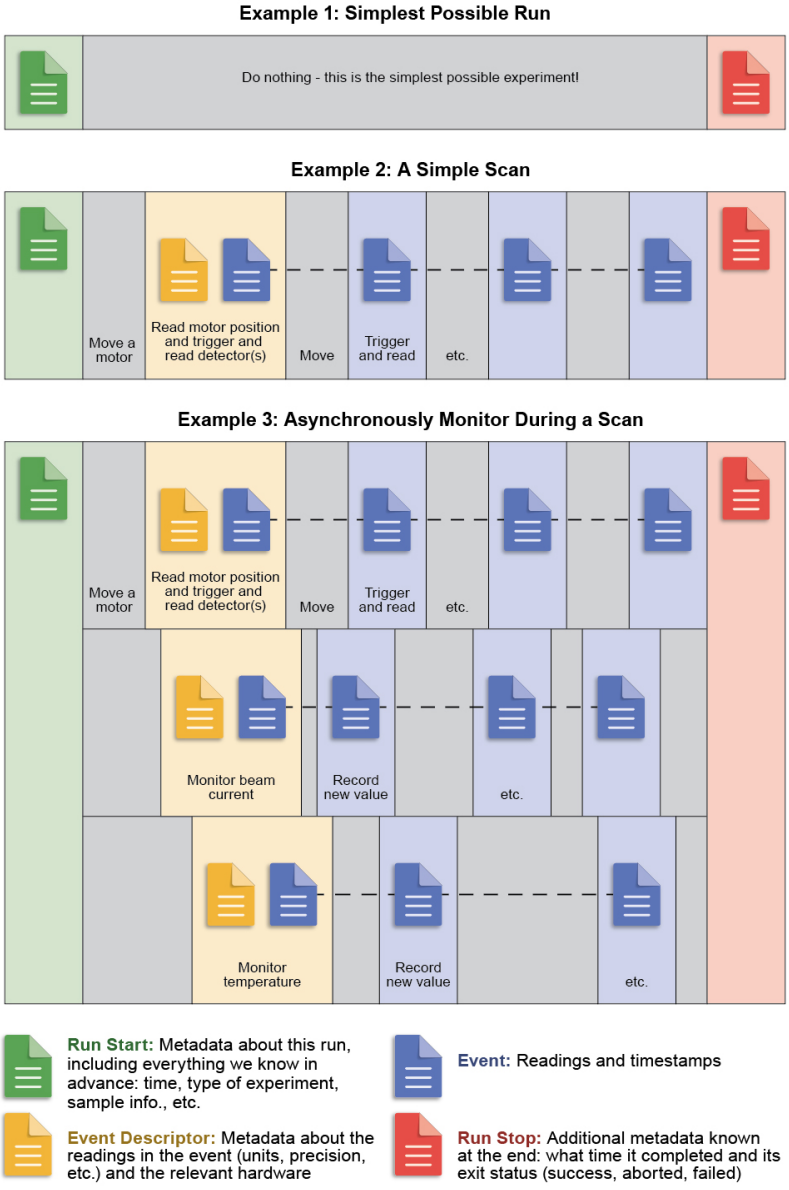


Figure 3.9: The Event-Model documents and how they are created by experimental scans. Example 1 shows a simple experiment where no readings are taken and only stop and start documents are issued. Example 2 shows an experiment where one stream of readings, from a motor and detector being read at the same time, is taken. Example 3 shows a fully asynchronous experiment, where data is taken from multiple sources at different rates. From [group, 2019].

the Event-Model into these base types. SHED has three main arguments which control data extraction: document name, which specifies which document the data will come from, data address, which specifies which keys inside the Event-Model dictionary to access, and event stream name which specifies which asynchronous stream to listen to, if not all. The combination of these three pieces of information pinpoints a single value of data or metadata within the Event-Model and plumbs that data into a rapidz node for further processing and analysis. This functionality is provided by the `FromEventStream` nodes, which translate from the event model to base types.

While translating from the Event-Model is simple, translating back into the Event-Model, which enables reuse of the rich data management, data visualization, and data searching capabilities created for raw data, is much more complex. The main difficulty of this is because experiments are not unbounded streams of data. Experiments have starts and stops, which are represented by documents in the Event-Model. A rapidz pipeline however, has no internal conception of the beginning or end of an experiment, all the data is perceived to be from an unending stream, with maybe an irregular time gap between data points. The needed break from the unbounded streaming model requires extra information to be passed between the top and bottom of the graph, capturing if a piece of data is associated with one experiment or another. This information is passed by an approach we call "side band signalling". Side band signalling works by establishing a separate channel, or band, between `FromEventStream` nodes to all their counterpart `ToEventStream` nodes. Since there may be more than one `FromEventStream` node in a pipeline, usually when more than one piece of information is needed for the analysis, a principal node is declared to act as the controlling node. Only the principal node issues the signals to the `ToEventStream` nodes to issue a start or stop document.

The approach is implemented by walking up the DAG to find the `FromEventStream` nodes when a `ToEventStream` node is instantiated. It is important to note that the previously discussed backpressure property of rapidz pipelines, Section 3.2.2.2, prevents stop or start

documents from being issued before all the data is finished being processed. This makes certain that no pieces of analyzed data are without start and stop metadata, since incoming start or stop metadata cannot be ingested until previous documents have been processed.

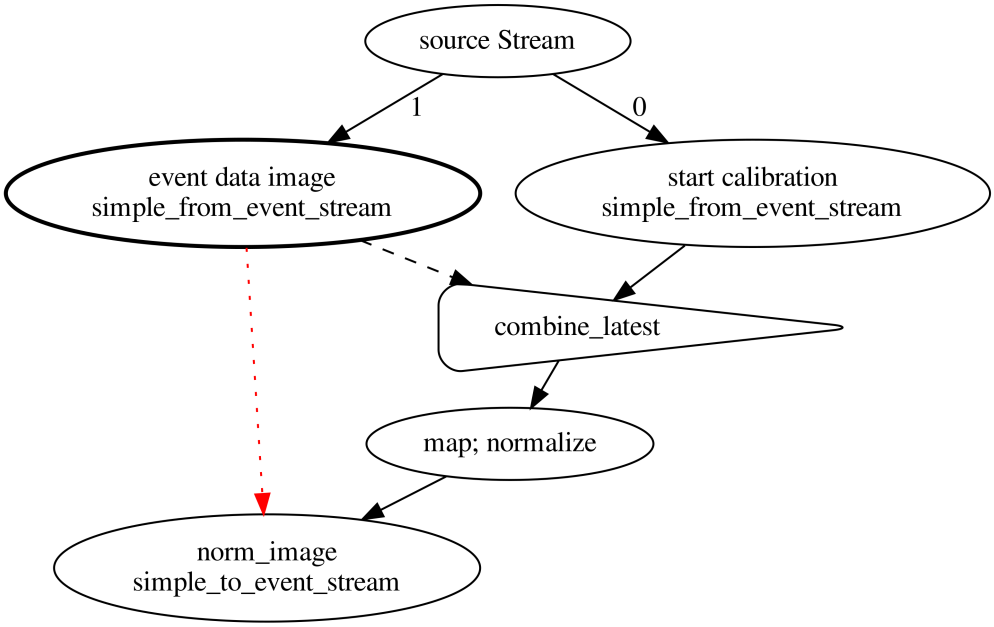


Figure 3.10: Example data processing pipeline with SHED nodes. Note that the node with the bold rim is the principal node, which is used as the reference for creating start and stop documents. The first name for the FromEventStream nodes is the document the node takes data from. The dotted red edge denotes the path of the side band signalling, providing stop and start metadata from the principle node (`event data image`) to the output event stream node (`norm image`).

Consider an experiment which takes a single image, producing four documents: start, descriptor, event, and stop. We'll track these documents as they pass through the pipeline shown in Fig. 3.10, that normalizes images by a calibration which is known at the start of the experiment.

The top of the pipeline is assigned to a variable `raw_source`. The `event data image` node is initialized with the `principal=True` keyword argument, declaring that it is the principal node of the pipeline The start document is passed into the pipeline by `raw_source.emit(('start', start_doc))` the `source Stream` node. This node has two exiting edges labeled 0 and 1 which indicates the order in which they are traversed. The

`source Stream` node therefore passes the start document to the `start calibration` node. This node needs information about the experiment to carry out the calibration which it extracts from the start document metadata. The extracted calibration metadata is passed down to the `combine_latest` node, which simply caches the data since no other data is in the node. A characteristic of a `combine_latest` node is that it will wait until it has data from its two incoming nodes to combine before doing anything, so it hands control of the process back to the top of the stream so the next document can be processed by the pipeline. The same start document is then passed down the second edge emerging from the `source Stream`, labeled 1, which sends it to the `event data image` node. Since this node is a principal node, the node has the property that when it receives a start document it signals all the `to_event_stream` nodes, in this case `norm_image` to emit their own start document. The `event data image` node then does not send data to the `combine_latest` node, since it is looking for data from events not starts. In this case it is the last node in the pipeline so after signalling for the start document it will pass control of the process back to the source which can start processing the next document in the event stream.

The next document in the stream is the descriptor document, which is passed into the source that passes it first to the `start Calibration` node. This a `from_event_stream` node which has the property that it does nothing with descriptor documents and returns control of the process back upstream, so the same descriptor is emitted down the second edge to the `event data image node` which is also a `from_event_stream` nodes, which does nothing and allows the next document in the stream to be emitted into the pipeline.

The next document is the first real event in the stream. The event document is passed into the source, which passes it to the `start calibration` node. When this node receives an event it does nothing and hands control back to the source to emit the same event down the second edge to the `event data image` node. When this node receives an event document with an image in it, it extracts the image data and passes it to the `combine_latest` node. The `combine_latest` node now has a document from both its input edges, so it combines

them and sends the combined calibration metadata and image data on down the pipeline. The next node is a `map` node which executes a transformation on the data, in this case, computing the normalized data using the calibration parameters. The normalized data is then passed on to the next node which is the `norm_image` node. The `norm_image` node reads the data type of the normalized image, issues a descriptor which includes the shape of the data and its type, and then issues an event document with the normalized image. After emitting these documents control is passed back to the source which emits its next document. In general these will be more image event data documents, but in this example there was only one such event, so the next and last document in the stream is the stop document.

The stop document is passed into the source, which passes it to the `start calibration` node, which does nothing and then to the `event data image` node which as the principal `from_event_stream` node issues a signal to the next `to_event_stream` node, the `norm_image` node, which then emits a stop document. The last `to_event_stream` node has by then emitted a complete new event stream, with a start document, descriptor, event and stop documents that could be saved into an analysis databroker database, or sent on to another stream of analysis steps.

SHED Provenance Tracking

Capturing the provenance of the data output from a SHED graph requires three pieces of metadata: the graph itself, the unique identifiers for all the incoming data sets, and the order in which documents went into which `FromEventStream` node. The graph is stored by serializing the node class and any arguments and keyword arguments. Serialization of classes and functions is performed using the python standard library `importlib` module. This results in both machine and human readable metadata that can be de-serialized into operating code. The data's IDs are provided by the Event-Model itself, which requires a unique ID field. The `FromEventModel` nodes also track exactly when pieces of data flow through the nodes,

tabulating the documents ID and the time that the document went into the node. Finally the `ToEventModel` nodes track the current software environment, tabulating which software is being used and their versions, enabling the comparison of analyzed data across software versions. These comparisons can be used to track bugs in software and see how they impact results. All of these pieces of data are put into the start or stop documents of the outbound data, where it is usually stored by a Databroker.

The human readable serialization enables searching of the provenance metadata, allowing users to search their output data sets by the exact parameters of the data processing which were used. The serialization also enables the hashing of a pipeline. Hashing, in computer science, is where an entity has a unique, or almost unique, ID associated with it via a cryptographic algorithm. The hash IDs allow two pipelines to be checked against one another, if the IDs are the same then the pipelines used are exactly the same, down to the pipeline parameter and functions used. The approach used by SHED is a Merkle tree, where the hash of the previous nodes are used to create the next node's hash, which helps guarantee uniqueness. Merkle trees are used in making block chain driven registers, like Bitcoin, operate.

There are limits on what SHED can accomplish via provenance tracking. Some code does not lend itself to provenance. For instance python `lambda` functions are not able to be serialized using `importlib` since there is no module containing the source code. Additionally the lambda function is stored in python as bytecode, so there is no effective way to obtain the source code and bytecodes may not be reusable for future sessions. This means that if the analysis pipeline contains any lambda functions there will be gaps in the data's provenance. A similar restriction applies to callable classes used as arguments or keyword arguments to nodes. Since these classes can change state during the operation of the pipeline their provenance is not well defined. The limitation on classes is a minor one, since most of the stateful information should be tracked by the DAG itself and that one can have factory functions which make classes to be used by downstream nodes, since the inputs to these

factories and therefore the output classes are controlled solely by the pipeline itself.

All of this metadata can be used for more than just database searching and data comparison, it can be used for replay of data analysis. SHED's replay capability uses stored data and metadata to recreate the pipeline as it was run. The replay then loads the incoming data and sorts it by the time stamp it was inserted into the `FromEventModel` nodes. Replay then passes the sorted data to the node that the data was passed into. Once the data is passed into the pipeline processes it using the same procedure it was initially processed with. The replay functionality can be used to recreate just the data processing DAG, allowing new data to be processed with the same graph as the previous data.

Summary

SHED provides data handling between the Event-Model and base python types and tracks the provenance of analyzed data. The provenance can be used for discovering how analyzed data is produced, allowing for direct comparisons of data and parameter exploration. Furthermore, the provenance of the data can be used to directly reproduce the analyzed data from scratch.

Summary

This chapter detailed the decisions and design behind the creation of rapidz and SHED. Rapidz was designed to provide a pythonic system for performing streaming data processing, including distributed streaming data processing. SHED was designed to handle the inherently heterogeneous data and metadata created by experiments, providing a translation between heterogeneous data and the rapidz system. SHED also provided provenance of the analyzed data by tracking the data processing pipeline itself and the data which passed through.

Chapter 4

Streaming Data Reduction and Reconstruction

Introduction

Accelerating data rates at synchrotrons offer unique opportunities for scientists to probe physical and chemical phenomena. X-ray scattering techniques provide important windows into these phenomena, illuminating the atomic structure of materials. Higher data rates confer many benefits on x-ray scattering, as more images can be taken, providing more detailed maps of samples behavior and higher throughput experiments. Matching this experimental capability with automated data analysis will help ensure that users and facilities are not overwhelmed by the data volume. Additionally automated analysis opens the technique to users which are not skilled in the data processing required for x-ray scattering measurements [Toby *et al.*, 2009].

Various tools for XRD and PDF data processing exist, fit2D, pyFAI, pdfgetx3 however, none provide streaming capabilities [Kieffer and Wright, 2013; Hammersley *et al.*, 1996; Juhs *et al.*, 2013]. Streaming capabilities provide live data processing and analysis, which enable live evaluation of experimental data which in turn can guide the experiment to the most interesting results. Extensions of streaming data processing will enable autonomous experimentation, where streaming analyzed data allows the computer to perform the steering. Tomography in particular stands to benefit from streaming data processing, especially techniques like ctXRD and ctPDF that take large volumes of data. To fully take advantage of these next generation x-ray sources, a system which enables high throughput data acqui-

sition and analysis, while providing enough flexibility to empower users to add capabilities, needs to be implemented. xpdtools, xpdAcq, and xpdAn provide such a system for acquiring data and providing real time automated analysis of x-ray scattering data. These systems are used at the 28-ID beamlines of the NSLS-II, providing users with rich metadata capture, live analysis and visualization.

XPDtools

Introduction

Analyzing data from modern x-ray diffraction and total scattering experiments requires multiple corrections, statistical cleaning and other steps, making it quite complex. Automating these steps will help scientists keep up with the push towards high-throughput experimentation as currently the rate of experimental data acquisition outstrips that of manual data processing. High volume experiments, like ctXRD and ctPDF experiments [Jensen *et al.*, 2015; Jacques *et al.*, 2011], combinatorial materials discovery [Roncallo *et al.*, 2010; Ren *et al.*, 2017], and autonomous experimentation [Tabor *et al.*, 2018; Nikolaev *et al.*, 2016] need this kind of high performance processing. Software for x-ray scattering data reduction exists, but can rely too much on human interaction, handles data too slowly and not in a streaming modality [Toby *et al.*, 2009]. To address this need, we developed data analysis protocols focused on pipelines that can handle streaming data. Our approach provides reproducibility, the ability to share and adapt complete analyses, and ease of use. This library, xpdtools, acts as "glue" combining multiple other libraries into a complete pipeline for end to end data analysis.

Design

The main design drivers for this software are:

1. Ease of Use
2. The ability to handle streaming data
3. Reuse of analysis software, allowing for the adaptation and rerunning of analyses either in whole or piecemeal

The xpdtools software architecture has a ternary structure consisting of data transformation functions, data processing pipelines, and a command line interface (CLI). This structure makes xpdtools modular, enabling users to engage at whichever level they feel comfortable. Expert users, who have their own data processing infrastructure, can use the simple function based tooling to perform transformations to their data. XPDtools implement pipelines using the rapidz library, making streaming data handling a flagship use case. Expert users can modify and extend these pipelines to provide additional functionality or parallelize the processing in a streaming context. This enables users to reuse and retool existing pipelines, allowing for reanalysis of data. Finally users can also use existing pipelines via the CLI, which performs data processing for many files at once.

Implementation

X-ray diffraction and total scattering experiments are usually performed by taking light, dark, and calibration data on area detectors [Chupas *et al.*, 2003]. This data must be combined and transformed to properly correct the data and perform the reduction from 2D image to 1D pattern.

Function Tools

The foundation of xpdtools are functions which operate on numpy arrays and basetypes [Walt *et al.*, 2011]. These functions perform the core x-ray scattering computations, including masking images, integrating images to 1D scattering patterns, and others [Wright and

Zhou, 2017]. Most of the functionally in this layer uses software from other libraries, including scikit-beam, pyFAI, and pdfgetx3, casting many object based operations into functions [Kieffer and Wright, 2013; scikit-beam team, 2019; Juhs *et al.*, 2013]. This collection of mathematical tools, gives us powerful interoperability and extensibility with all the scientific python stack.

Pipelines

The data processing pipelines in the middle layer of xpdtools provide the fundamental logic of how the data processing steps fit together. Each piece of data processing, detector corrections, pixel masking, integration, etc. is associated with a "chunk" factory function, as discussed in Section 3.2.4. This allows users to mix and match individual data processing pieces as needed. This is particularly helpful as different detectors produce data at different levels of processing with some providing dark subtraction internally. Furthermore the chunking approach enables users to add more chunks, each of which can perform additional data processing steps, or modify existing steps.

CLI

While, the combination of the tooling and pipeline layers formally has all the pieces needed to perform x-ray scattering data processing, their do it yourself approach may not be appropriate for all users. Many times users would like to have their data processed without fiddling with the exact execution order or which piece of masking code is used. xpdtools provides this in the form of a CLI. The CLI accepts entire folders of image data, performs best effort data processing and outputs the resulting integrated data. While most users may find this best effort approach appropriate for their data, the CLI accepts additional parameters, like scaling of the background signal, so users can tune their processing without making the pipeline themselves. The flexibility of this approach allows users to explore how different parameters impact their analysis results.

The complete software project is tested with coverage in excess of 92%, documented, licensed under a BSD-3-Clause license and released via Conda-Forge.

Summary

xpdtools provides data processing tools for x-ray scattering data. These tools aim to help beginners who can use the CLI without opening a python session, and experts who can use, modify and extend the pipelines provided, and use the function based tools in their own data processing systems. The design of xpdtools focuses on modularity and user friendliness and is implemented in python, using the scientific stack and `rapidz`. The use of streaming as an integral part of the software enable live visualization steering and automation of experiments.

xpdAcq and xpdAn

Introduction

Writing a streaming data processing system for x-ray scattering is an important step in providing automated data reduction and analysis. However, a complete system needs to know which processing to apply in different circumstances in addition to processing the data itself. This requires high quality metadata associated with the raw and processed data. This metadata can be inspected by the pipelines so they can perform the appropriate processing. Metadata also provides a platform for searching, comparing, and visualizing data. This combination of metadata and data enables users to be more efficient on the beamline and off. On the beamline, live data processing and visualization allows users to understand their experiment during its execution. Off the beamline, captured metadata provides hooks for querying the available raw and processed data. Scientists can use these queries to compare data sets and understand the steps taken to produce each one. Furthermore, the software can

inspect the metadata to select the appropriate set of data visualizations to provide to the user. This will enable collaboration between scientists, and reduce reliance on institutional memory as databases can store all the relevant information needed to understand the experiment and analysis.

xpdAcq

Motivation

Many pieces of important metadata, like sample composition, calibration conditions, and others, are needed at the beginning of an experiment. Often this kind of metadata is not provided in a meaningful way, either it is stored in lab notebooks or not available at all. Other times users are too busy running and troubleshooting their experiment to provide additional metadata.

Implementation

xpdAcq aims to solve these issues by combining three differently scoped pieces of metadata and making each easy to write and access. Beamtime level metadata is information supplied at the beginning of a beamtime, including the experimenters taking the data, the beamline's x-ray energy, configuration of the beamline, etc. This data is critical for searching for data, as it provides hooks for queries like "find me the data I took three beamtimes ago", and for performing the data processing. Sample metadata is information about the sample, its composition, name, the person who made it. Scan metadata is information about the scan itself and is provided by bluesky. Bluesky is a data acquisition system from Brookhaven National Laboratory that combines metadata capture and experiment planning into one software interface. This combination allows metadata about the scan to be captured automatically as the experiment is running. xpdAcq wraps the bluesky data acquisition system so the scan metadata is provided by bluesky itself [Allan *et al.*, 2019].

The sample metadata is made easy to access as an enumerated list of available samples. This not only makes the composition of an experiment, a combination of scan and sample, easy it provides important incentives for users to provide that sample information, since without it running scans is much more difficult. To further encourage user buy in, the sample information is entered by an excel spreadsheet that not only maps well onto beamtimes, which usually feature multiple samples, but also maps well onto how scientists providing the samples track their synthetic products and data. The interplay of xpdAcq and xpdAn provides a final incentive for users to provide detailed and accurate metadata. xpdAn uses the metadata provided during the acquisition of the data to perform the live data processing and visualization.

xpdAcq is tested, documented, licensed under BSD-3-Clause license, and released on Conda-Forge.

xpdAn

xpdAn provides the final piece of the integrated system, bringing the raw data and metadata collected by xpdAcq, the data processing from xpdtools, and live data visualization. xpdAn uses a combination of bluesky, SHED, and a message passing system to make these connections.

Motivation

xpdAcq and xpdtools provide important but separate data acquisition and analysis functionalities. The different data types used by each keeps them separated, as xpdAcq uses the event model and xpdtools is based in python objects. Bridging this separation requires the implementation of SHED nodes to translate between the data produced from xpdAcq and the pipelines from xpdtools. In this way xpdAn is to xpdtools as SHED is to rapidz. In addition to translating raw event model data into python objects for xpdtools, xpdAn provides translation of analyzed python objects into the event model. xpdAn can then use

the analyzed data in the event model to provide live visualization, database insertion, and saving of files into legacy data formats.

Implementation

The main unit for xpdAn code is the server. Each server consumes, processes, and may produce data via a message bus. xpdAn currently uses ØMQ as the messaging system, although it is designed to take any provided message bus. Each stream of data passing through the ØMQ system is labeled with a string. The xpdAn servers use these strings to determine if a piece of data is one which that server needs to consume. When a server consumes a piece of data it then performs its specialized computations on the data, for instance creating or updating a plot, calculating the peak positions, etc. Once the server finishes processing the data, the server may send data back to the message bus under its own label, which then can be consumed by other servers. While ØMQ is able to shuttle messages between data acquisition and the various data consumers it does have some limitations. The largest limitation is that it does not guarantee message delivery, unlike tools like Apache Kafka. This means that messages could be dropped in the event that the ØMQ system is overloaded. While this is a rare occurrence with the xpdAn system, it can happen causing issues with downstream processing. One remedy for this, aside from using a guaranteed message delivery system, is to reprocess the data after it has been collected making certain to not overload the ØMQ system.

xpdAn's main server is the `analysis_server`, which consumes raw data set to the message bus from the acquisition system,performs the correction and reduction of images to 1D scattering patterns and atomic pair distribution functions, and publishes data under the an label. The `intensity_server` calculates the position, width, and height of peaks within a selected region of the $I(Q)$ or PDF data. The `db_server` consumes data from all labels except the raw label and saves it to a database built for analyzed data. The `viz_server` provides data visualization, and consumes data from all the labels. Internally the visualiza-

tion server checks the metadata associated with each data stream to check if it can visualize that type of data. If there is a valid visualization then the server creates or updates plots associated with the data. Fig. 4.1 shows the system currently implemented out at 28-ID, with nodes representing servers and edges representing data flow.

The server system allows users to add, remove, and create new servers at the beamline, providing a flexible analysis environment. If these servers produce and publish data then the existing visualization and data saving servers will provide a best effort attempt to operate on the data, eliminating the need for users to create new data visualization tools for each analysis they create. All of this infrastructure can be used offline as data at any stage of analysis can be sent into the message bus from the database. The server architecture also enables fault tolerance, as individual servers can error and fall over without causing the data acquisition, or any other servers, to stop. The main disadvantage of this system is that it can be more complex to write data processing code, as there is additional cognitive load associated with writing the message consuming and producing code. Additionally, the use of a message bus can introduce a bottleneck for the data processing since the data must travel over a network. Choosing a different message bus can reduce the impact of this issue, by matching the bus and network constraints.

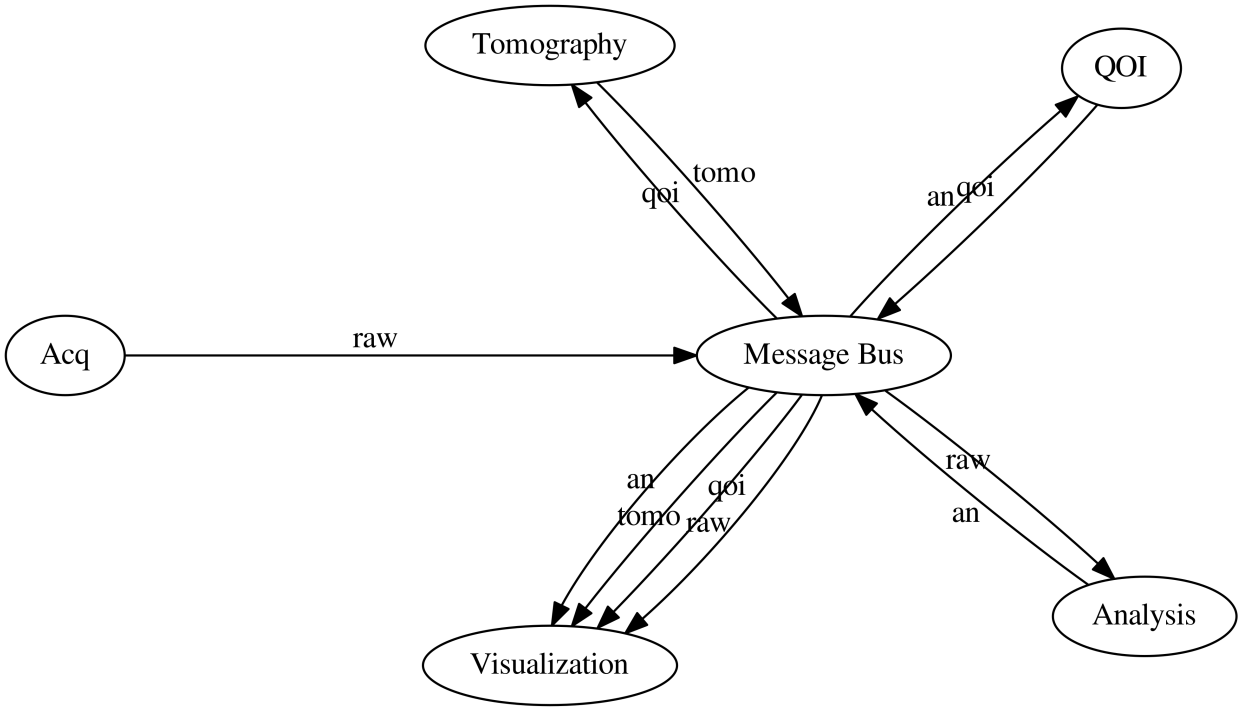


Figure 4.1: Schematic of the servers and message bus at the XPD beamline.

Tomographic Reconstruction

Introduction

The infrastructure used to build xpdtools, xpdAcq, and xpdAn can be extended to perform tomography experiments and process the resulting data. This enables live construction of sinograms and reconstruction of data, in addition to providing the metadata needed to understand the acquired data.

xpdtools

Xpdtools provides functional tools and pipelines using tomopy [Grsoy *et al.*, 2014] to perform reconstructions for both absorption tomography and scattering tomography. Fig. 4.2 shows the DAG implemented for x-ray scattering tomography. Currently xpdtools supports reconstruction of 1D patterns and scalar values extracted from x-ray scattering.

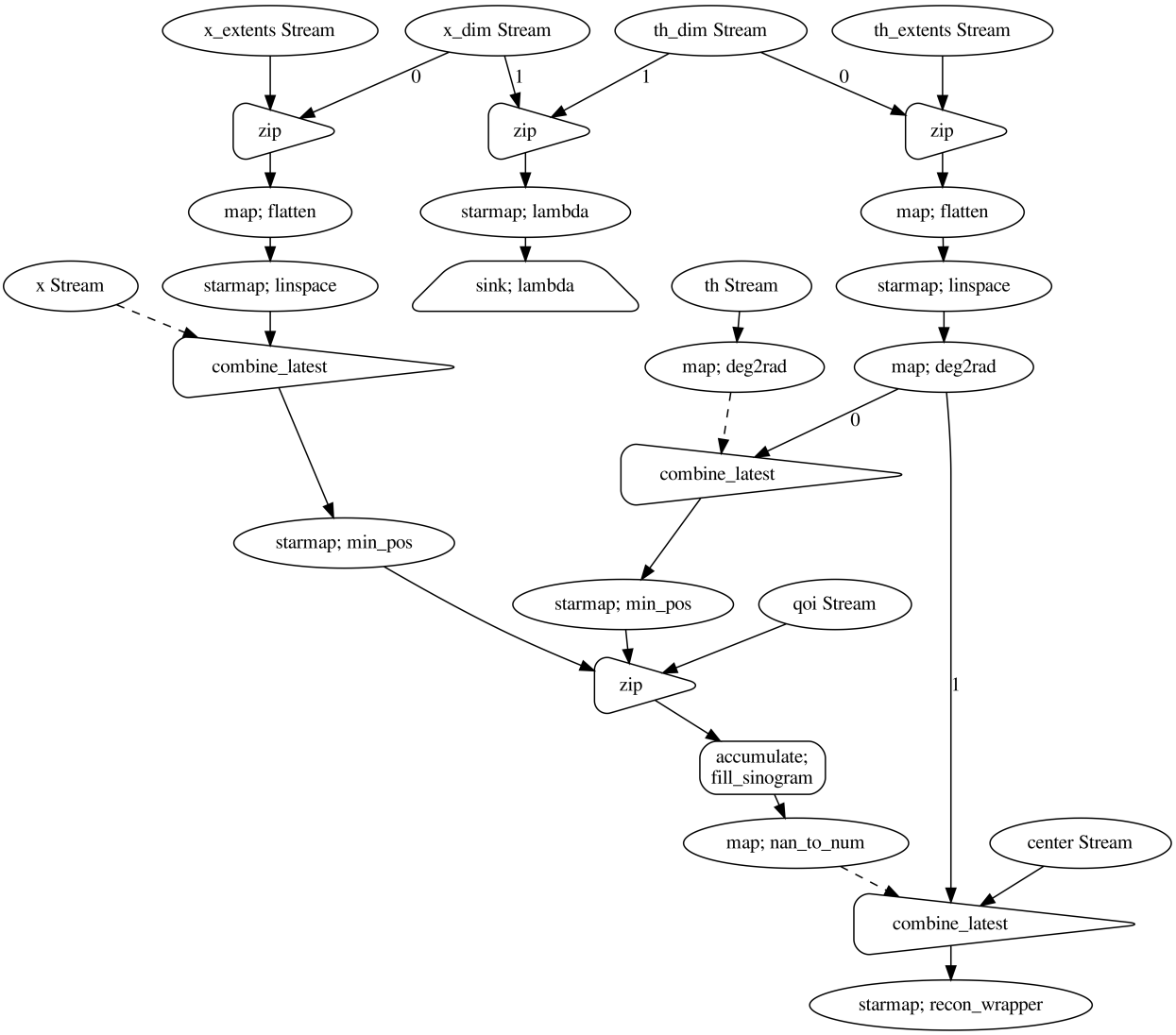


Figure 4.2: The rapidz DAG for constructing sinograms and tomograms.

xpdAcq

The proper tomographic reconstruction requires the metadata provided by xpdAcq. Tomopy's reconstruction algorithms need the data to be in a certain format, with the rotation axis as the first axis of the array and the translation axis as the last. The translation axis must be the axis orthogonal to the rotation axis. When taking tomography data users can enter the needed metadata into xpdAcq including, if the experiment was in a pencil beam or full field mode, the name of the rotation and translation motors, and if there was a second translation axis.

xpdAn

xpdAn reprises its role as bridging the acquisition and data processing for tomography. The xpdAn `tomo_server` provides tomographic reconstruction for all scalar values which pass through the message bus. The data is then reshaped into a sinogram and reconstructed. The sinogram and reconstructed data are sent back to the message bus, allowing for both data sets to be visualized. Sinogram visualization provides users with important diagnostics on their experiment, as any shifts in the sample position will require restarting the measurement. xpdAn ships an optional addition to the visualization server for 3D visualization of tomograms.

Chapter 5

Tomographic Commissioning

The goals of these commissioning experiments and simulations is to check the viability of the data reduction and tomographic reconstruction software and experimental hardware. We performed commissioning experiments on multiple beamlines and multiple phantom samples. The phantom samples were chosen to exhibit a wide range of contrasts.

Experimental Setups at the NSLS-II

Beamline Optics

This section discusses the experimental setup at XPD-D. Experiments were carried out at XPD (28-ID-2)[Shi *et al.*, 2013] and PDF (28-ID-1) in addition to XPD-D. Deviations from the XPD-D hutch setup will be noted. The major features of the XPD optics are the double Laue monochromator and the beam defining slits.

The monochromator filters the incident white beam, admitting only monochromatic light by selecting Bragg reflections of silicon crystals. The monochromator has two Si (100) crystals which focus the beam and select the wavelength. The double Laue geometry was chosen to maximize the flux while providing a wide range of operating wavelengths [Shi *et al.*, 2013]. The PDF beamline uses a side bounce monochromator.

Beam defining slits control the beam size, producing the small pencil beam required for ctXRD operation. The slits are made of 5 mm thick tungsten and can go "past closed" without clashing. The slits have 2 μm accuracy.

Sample Motors

The sample motor stack consists of two main parts, alignment motors and scanning motors. The scanning motors provide the x, y and ϕ scans needed to perform the tomography experiments. The alignment motors help to move the sample into position on the scanning motors. Importantly the alignment motors, which is on top of the scanning motors align the sample's rotation axis with the ϕ rotation axis. This alignment help to speed up the scans by removing extra translations needed to cover a precessing sample. The XPD and PDF beamlines do not have as elaborate motor stacks for performing the tomography, they are missing the alignment motors, leaving alignment to goniometer heads, or using a larger translational scan to compensate.

Figure 5.1: The experimental stage hardware and optics for XPD-D.

Detectors and Calibration

Two x-ray area detectors were used for collecting the scattering data. Area detectors have become common in x-ray scattering experiments because of their ability to probe many scattering vectors at once [Chupas *et al.*, 2003]. The Dexela 2923 detector was used for the measurements at PDF and XPD-D. A Varex imaging XRD 1611 xP amorphous Silicon flat panel detector was used at the XPD beamline. The detector locations were calibrated

using pyFAI on a Nickel sample [Kieffer and Wright, 2013]. LaB_6 was not used because of the beam's small spot size, which made the inherently grainy LaB_6 peaks more difficult to select.

Simulation of Tomography

Introduction

Systematic studies of how different reconstruction algorithms, sample geometries, and data processing procedures impact results are lacking in the ctXRD and ctPDF literature. Simulations provide an excellent approach to understanding these effects, as the makeup, expected scattering, and geometries are set and known during the simulation process. This approach provides high quality baselines to compare results against without the need to exclude other variables. In this section we discuss the simulation of tomography and the impacts of the reconstruction on the data quality.

Software design

The simulation software presented here aims to provide a framework for mocking tomography experiments. Thus, we designed the software to mimic the detector and motors which would be used in a tomography experiment. This provides high fidelity of simulation and make the simulated hardware interchangeable with the physical hardware allowing reuse of existing data acquisition and processing software. The software could be extended to simulate noise from the detectors themselves.

The mock detector produced by the software records the x-ray scattering calculated from each voxel in the simulated sample. The simulated sample is composed of a series of phases. Each phase denotes the x-ray scattering that would be observed from that pure component and how much of the component each voxel contains. The software calculates

the x-ray scattering by computing the scattering vectors for each pixel on the detector and then computing the intensity at each detector pixel.

The algorithm for calculating the simulated tomographic data is:

1. move the motors to the next translation and rotation values and rotate the phase array(s) by the rotation value
2. calculate the beam's position on the sample by subtracting the translation motor's position from the position of the center of the sample, the beam's position causes it to select a column of voxels which will scatter x-rays
3. if all phases have null values for that translation report zero to speed up computation
4. calculate voxel to detector distances for each voxel in the beam path
5. calculate the scattering pattern for each voxel in the beam path and sum across voxels and phases
6. proceed to next translation, rotation point in scan.

This procedure makes some critical assumptions and simplifications to the calculation of the x-ray scattering. This method does not calculate the beam attenuation or multiple scattering for the sake of simplicity The most critical of the assumptions is that each voxel acts as a single discrete scatterer. In a true sample the scattering would be continuously generated across the beam path. This effect can be seen when the scattering voxels are particularly large, causing the Debye-Scherrer rings on the detector to separate from one another as if there were a handful of discrete samples rather than a continuous mass. The separation of the rings can be reduced by using small voxels, where the distance from the center of one voxel to the next is small enough that the resulting diffraction rings overlap with one another. Modifications to the above algorithm could also be made to simulate multiple points across a given phase voxel which would improve the data quality.

Reconstruction Impacts on ctXRD Results

Simulation Procedure

A ctXRD experiment was simulated on a 1 mm square capillary of nickel. The nickel phase voxel size was 200 μm and the center of the capillary was offset from the axis of rotation by 4.5 mm. The tomography experiment consisted of 51 translation steps of 200 μm and 181 rotations of 1°. The sinogram of the integrated intensity, representative reconstruction, and representative integrated pattern are shown in Fig. 5.2, Fig. 5.3 and Fig. 5.4, respectively.

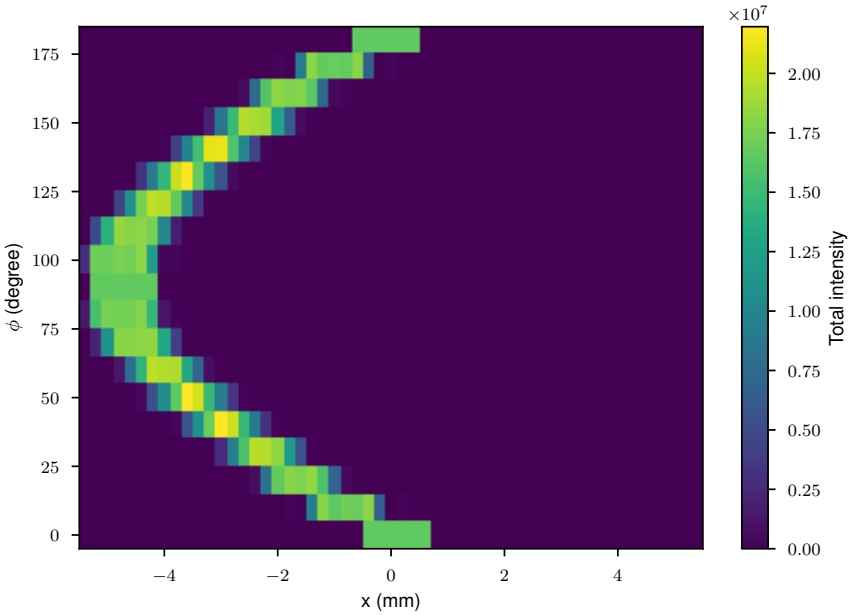


Figure 5.2: Sinogram of the summed intensity

Comparison of Reconstruction Order

The existing literature on ctXRD and ctPDF data processing has two procedures, in some cases the data processing occurs first, producing quantities of interest (QOIs) that are then reconstructed [Palancher *et al.*, 2011], in others the reconstruction is performed on the integrated data then reduced to QOIs [Jacques *et al.*, 2013]. The resulting scattering data described in Section 5.2.3.1 was processed to 1D patterns and then either processed further

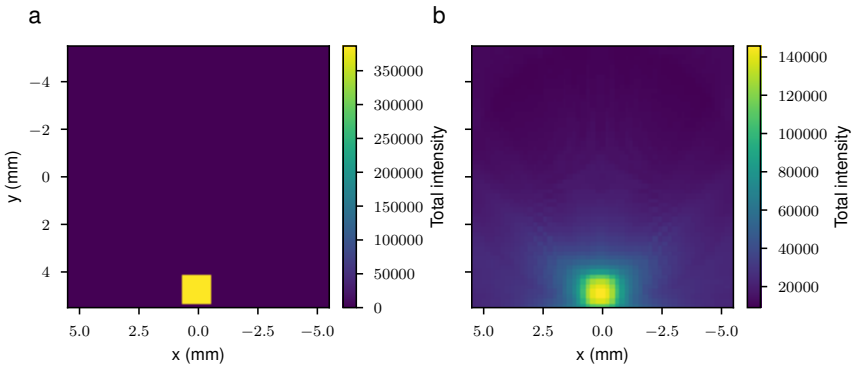


Figure 5.3: Reconstruction of the summed intensity. a) the ideal intensity distribution b) the reconstructed intensity distribution

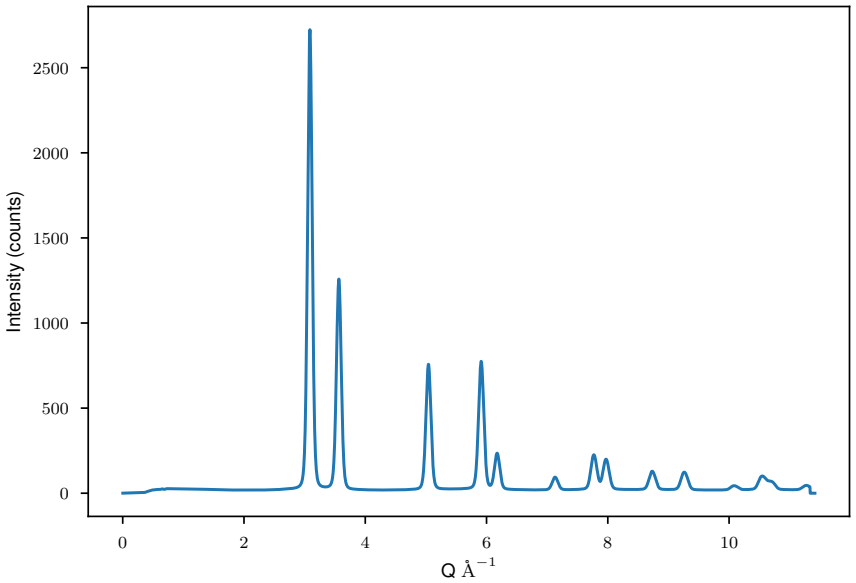


Figure 5.4: Representative $I(Q)$ pattern for the reconstruction

to a single value for the position of the first peak, Fig. 5.5 shows the sinogram associated with this order, or reconstructed first. The results in Fig. 5.6 show that better results are obtained when the reconstruction is done before QOIs are extracted. This is due to the non-linearity of the QOIs which breaks one of the key assumptions of the reconstruction algorithm.

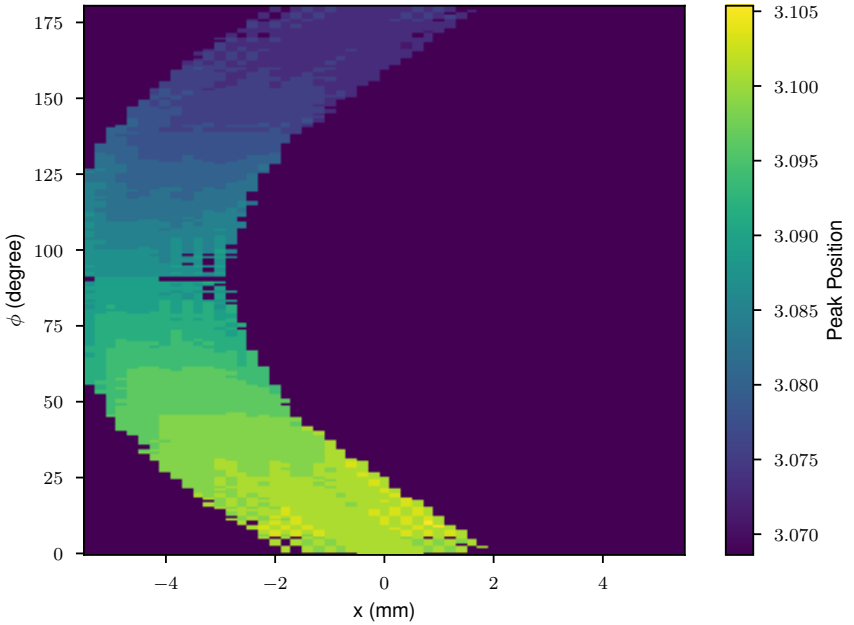


Figure 5.5: Sinogram of the peak position, note that the dark pixels are where the x-ray beam missed the sample, causing there to be no peaks.

Comparison of Algorithms

Introduction

Various algorithms can be used for reconstruction, the tomopy project [Grsoy *et al.*, 2014] has 13 algorithms for reconstruction including both algebraic and filtered back projection techniques. While the impact of these algorithms have been explored for absorption tomography, they have not for scattering tomography, with multiple algorithms being used in the literature [Palancher *et al.*, 2011; Jensen *et al.*, 2015]. To explore the impact of reconstruction algorithm on the results of a ctXRD experiment a 1 mm square capillary of powdered

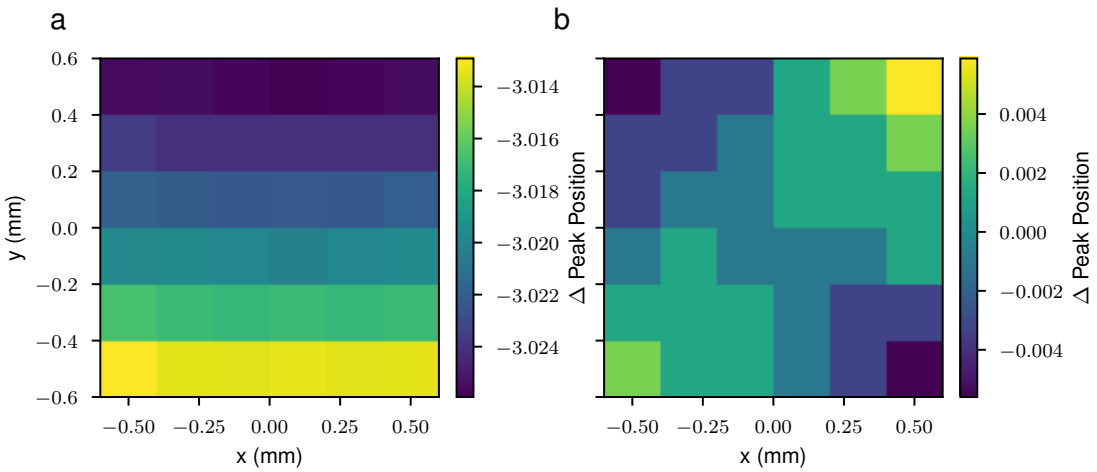


Figure 5.6: Difference between the reconstructed and ideal peak positions for simulated Ni ctXRD around 3.088 Å for peak extraction first (a) and reconstruction first (b). Note that (a) is no where near close to the correct peak position. In the ideal case figure (b) would be an uniform color.

nickel was simulated.

Procedure

The integrated data obtained from Section 5.2.3.1 was reconstructed with each of the tomopy algorithms using the default arguments for each algorithm. The position of the first peak was extracted from the resulting reconstructed integrated patterns. The value of this position was then plotted for each pixel in the nickel phase, shown in Fig. 5.7 and tabulated in Table 5.1.

Results

As Fig. 5.7 shows, most of the algorithms perform similarly. Only the `gridrec` and `art` algorithms perform poorly with the default arguments, as shown by their anomalously large spread in peak positions. More in depth statistical analysis shown in Table 5.1, indicates that the `ospml_hybrid` algorithm performs the best for x-ray scattering peak positions. Future work may include expanding the analysis to include a set of non-default parameters, like number of iterations for iterative techniques and filters for filtered back projection based

techniques, which may produce better results.

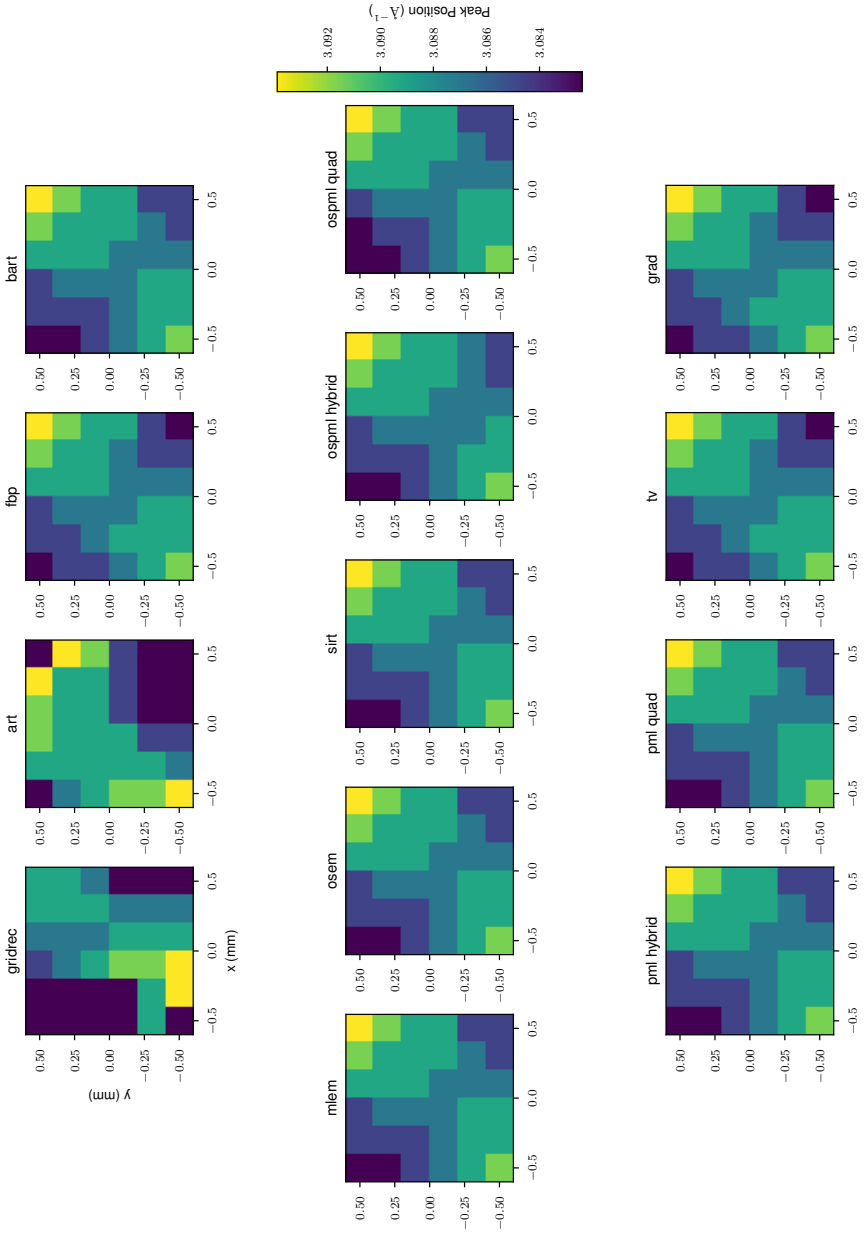


Figure 5.7: Comparison of the impact of reconstruction algorithm on peak positions of simulated Ni scattering. While most of the reconstruction techniques, including the commonly used filtered back projection, report similar results gridrec and art report a larger spread in values. The color map was clipped so the gridrec and art results did not wash out the other reconstructions.

algorithm	mean deviation ($m\AA^{-1}$)	standard deviation ($m\AA^{-1}$)	total rms deviation ($m\AA^{-1}$)
gridrec	-7.328	31.955	330.052
art	-3.881	19.874	244.404
fbp	-0.37	2.631	80.134
bart	-0.37	2.631	80.134
mlem	-0.37	2.631	80.134
osem	-0.37	2.631	80.134
sirt	-0.37	2.631	80.134
ospml_hybrid	-0.37	2.575	77.555
ospml_quad	-0.434	2.728	82.432
pml_hybrid	-0.37	2.631	80.134
pml_quad	-0.37	2.631	80.134
tv	-0.37	2.631	80.134
grad	-0.37	2.631	80.134

Table 5.1: The deviation of reconstructed nickel patterns first peak position from the expected value as a function of reconstruction algorithm. The table shows the deviation of the average value from the expected, the standard deviation across the nickel mass, and the total root mean squared deviation across the mass. While many algorithms produce similar numerical results they vary in speed. Most of the algorithms with lower error are based on maximum likely hood.

Reconstructions were also performed on simulated sinograms with added noise. The noise was created via the Poisson distribution, which is similar to the ideal noise from a x-ray detector. Fig. 5.8 shows the reconstructed peak position cross-sections for each algorithm. Comparing with Fig. 5.7 the results are quite similar for most of the algorithms implying that all the algorithms except for gridrec and art are noise tolerant. Table 5.2 shows the associated deviations from the expected output for each of the reconstructions of the noisy data. Interestingly it seems that most of the algorithms perform better, with lower root mean square deviation. This implies that the total rms deviation may be insensitive at the $m\AA^{-1}$magnitude.

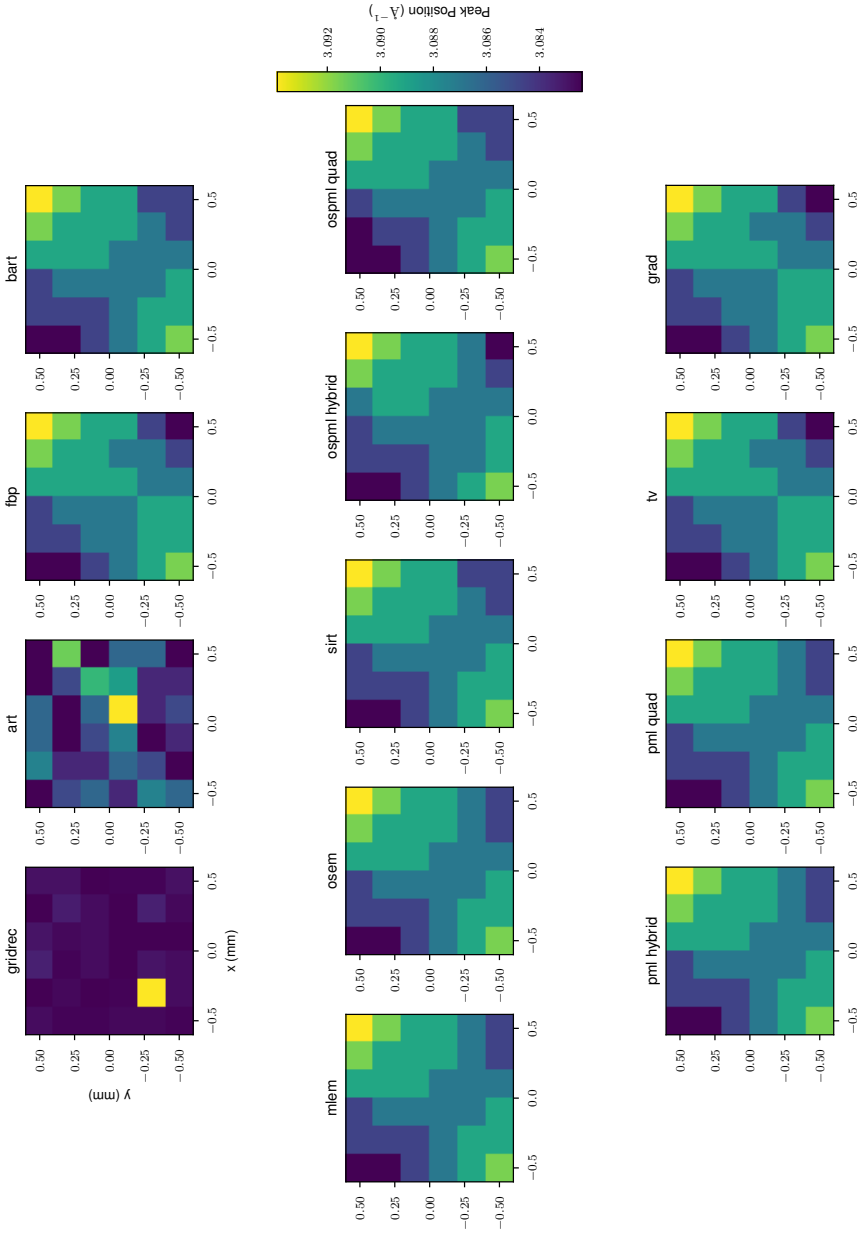


Figure 5.8: Comparison of the impact of reconstruction algorithm on peak positions of simulated noisy Ni scattering. Similar to Fig. 5.7

algorithm	mean deviation ($mÅ^{-1}$)	standard deviation ($mÅ^{-1}$)	total rms deviation ($mÅ^{-1}$)
gridrec	-199.158	34.255	7172.274
art	-205.031	4.849	7381.124
fbp	-0.37	2.687	80.134
bart	-0.434	2.618	79.853
mlem	-0.37	2.575	77.555
osem	-0.37	2.575	77.555
sirt	-0.434	2.618	79.853
ospml_hybrid	-0.561	2.643	79.29
ospml_quad	-0.498	2.714	82.15
pml_hybrid	-0.37	2.575	77.555
pml_quad	-0.37	2.575	77.555
tv	-0.37	2.687	80.134
grad	-0.37	2.687	80.134

Table 5.2: The deviation of reconstructed noisy nickel patterns first peak position from the expected value as a function of reconstruction algorithm. Similar to Table 5.1

The Parallax Problem

Introduction

While the choice of reconstruction can be somewhat to blame for errors in the results from a ctXRD experiment, as discussed in Section 5.2.4 even the best reconstruction algorithms do not eliminate all of the error. In this case the physics of the experiment itself may be the cause for the spread in the extracted QOIs. For most non-tomographic scattering experiments the sample volume is quite small, meaning that the change in sample to detector across the sample is minimal. This in turn causes the spread in the values for the peak positions across the sample to be small as well, resulting in consistent results. However, for tomographic experiments the sample has a macroscopic depth, which can cause a disparity in the sample to detector distance, and the associated lattice spacing, across the sample. This is called the parallax problem, as it is the change in results depending on the perspective of the sample, closer or farther from the detector. One of the important impacts of this is a widening of

peaks.

Procedure

Similar to Section 5.2.4 two peak positions were extracted from data reconstructed with the `ospml_hybrid` algorithm, the first peak and a peak at high scattering vectors. The goal of this extraction was to understand how position of the peak impacted its spread. Peak position is an important metric for understanding the lattice spacing of a material, which may in turn be used for determining the internal temperature of a sample. Understanding the spread of the peak position from a uniform sample would provide a lower bound on the resolution of a peak position gleaned from ctXRD experiments. Additionally, peak width, which is used to understand strain and crystallite size, is also extracted at low and high scattering vectors.

Results

Fig. 5.9 shows the peak position for the peaks expected at 3.088 $Å^{-1}$and 9.27$Å^{-1}$. Both peaks show a characteristic diagonal spread in the peak position, with the spread in the peak position at low scattering vectors around ± 5 m$Å^{-1}$and ± 15 m$Å^{-1}$ at higher vectors. The low scattering vector spread is roughly equivalent to a 100 °C spread over the sample, providing a convenient lower bound for temperature resolution using ctXRD.

Fig. 5.10 shows the spread in the peak widths. While the peak widths are more stable than the peak positions, with Fig. 5.10 (a) showing a consistent value across the sample, higher scattering vectors show some spread.

The Parallax Problem

The errors in the peak positions and widths beyond the reconstruction errors is most likely due to the parallax problem. The parallax problem is caused by the non-trivial changes in the sample to detector distance from the tomography experiment itself. As the sample

is rotated around the axis of rotation each voxel's sample to detector distance changes. This causes a change in the outcome of the x-ray scattering, peak positions move as the sample to detector distance changes, breaking the translation invariance required by the tomography reconstruction algorithms. This effect is also seen, albeit to a lesser extent, when working with large samples in standard XRD and PDF measurements, where the scattering at the front and back of the sample are different due to the change in sample to detector distance across the sample. The parallax problem causes certain values of the scattering to be unstable, like individual peak positions and their peak heights. This effect may be mitigated to some extent by using integration over the spread, for instance instead of using a single point to compute the intensity of a peak use the integrated intensity over that peak. Mitigation may also be possible via forward modeling of the scattering, explicitly including the sample to detector distance shift into the expected scattering calculations.

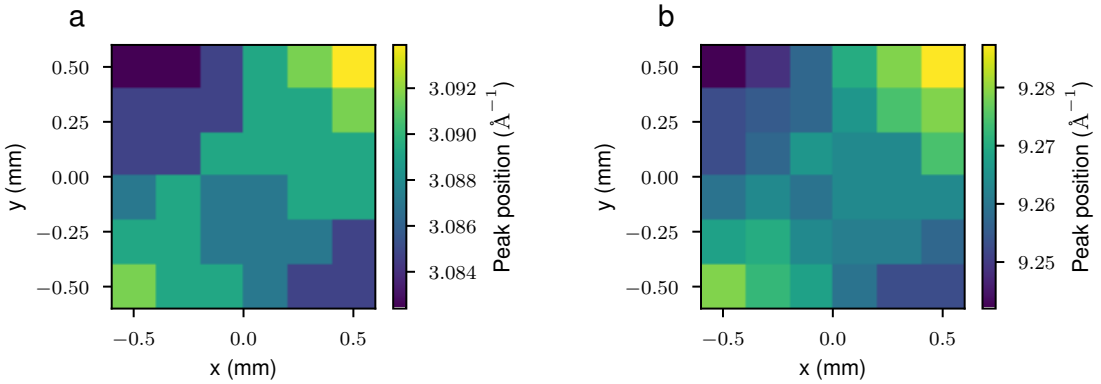


Figure 5.9: Peak positions for simulated Ni ctXRD around 3.088 Å and 9.265 Å.

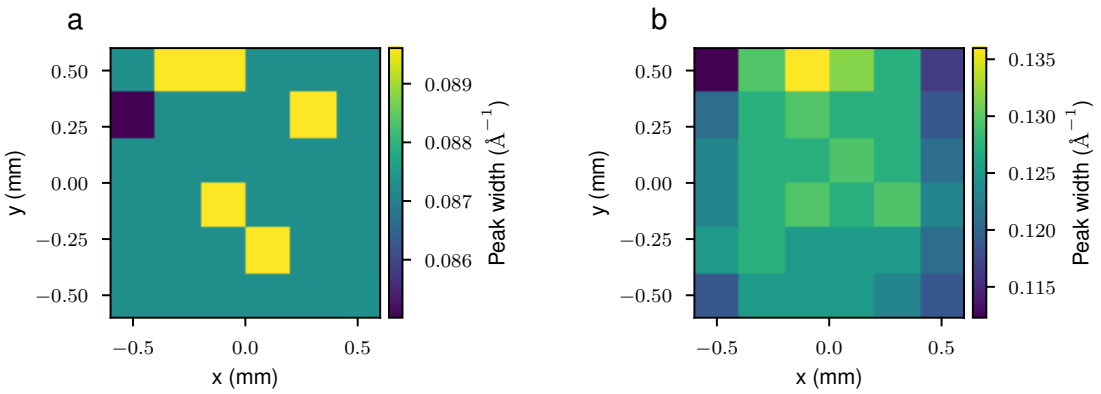


Figure 5.10: Peak widths for simulated Ni ctXRD around 3.088 Å and 9.265 Å.

Phantom

Introduction

The first sample to be run during the tomography commissioning was a graphite phantom. A phantom in tomography is a sample with known properties. These are valuable for testing both the experimental and computational capabilities as the reconstructed data can be compared against the known expected output. For ctXRD and ctPDF it is important to choose a phantom which has both Z contrast, which would show up in a standard absorption tomogram, and atomic structural contrast which can only be ascertained by x-ray scattering. Ideally a phantom is chosen so that some of the components have only structural contrast. Previously phantoms have consisted of capillaries filled with Kapton (a radiation resistant polyimide), basalt, silica glass, polystyrene and poly(methylmethacrylate)[Jacques *et al.*, 2013].

Experimental Setup

The phantom used in this study was created by drilling three holes in a graphite rod, shown in Fig. 5.11. Two of the holes were offset from the axis of rotation in the direction of the axis of rotation, a third hole was drilled diagonally across the rod. The diagonal hole was

filled with a copper wire, while the other two were filled with a second copper wire and a Kapton capillary filled with wax. The copper provided Z and structural contrast, while the wax provided only structural contrast. The phantom was attached to a goniometer head, which was then attached to a Huber 2 circle diffractometer.

The x-ray beam was reduced to 0.4 mm in each direction by the beam defining slits. The sample was rotated in steps of 2° and translated by 0.4 mm. Calibrations with a Nickel standard determined that the flat plate amorphous silicon detector was positioned at a sample detector distance of 1.56 m with a wavelength of 0.2405 Å.

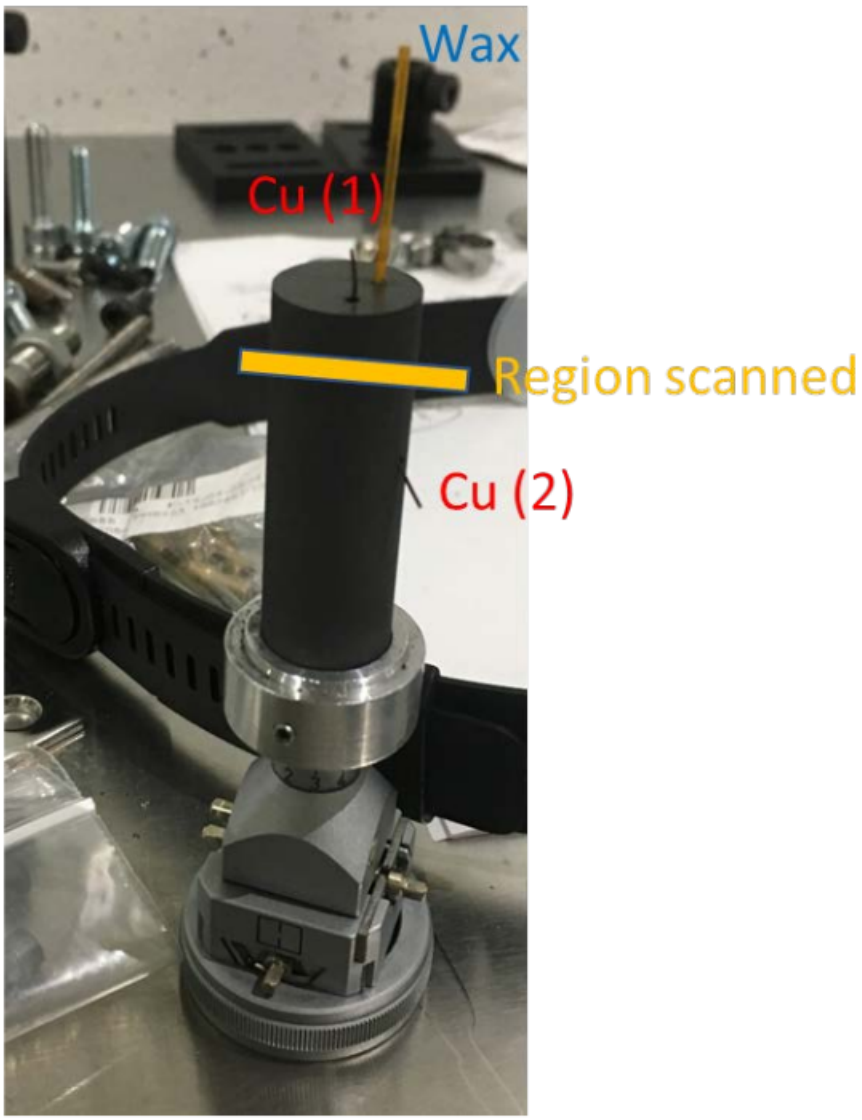


Figure 5.11: The graphite phantom, with the copper, wax and region scanned highlighted

Data Processing

The data was reduced and integrated using xpdAn. The images were dark and polarization corrected, masked to remove outliers, and integrated to 1D patterns. After integration the data was corrected by the background extracted from the edge of the sinogram. The center of the sinogram was determined by tomopy. Tomopy also performed the reconstruction using the `ospml_hybrid` algorithm.

Results

As expected the wax, graphite and copper are represented in the reconstructed images, Fig. 5.12. The wax, which has little Z contrast with the graphite, has unique peaks which provide contrast to the reconstructed images as shown in Fig. 5.13. Fig. 5.12 a) shows three holes associated with the copper and wax inserts. The copper holes have a much lower intensity than the wax hole. This effect is ascribed to the additional x-ray absorption from the copper which attenuates the non-copper scattering in those regions producing a lower intensity. The wax region has little x-ray stopping power and provides only minor attenuation in addition to the absence of graphite scattering. It is possible that the comparatively large beam size causes the inclusion of some graphite scattering into the holes.

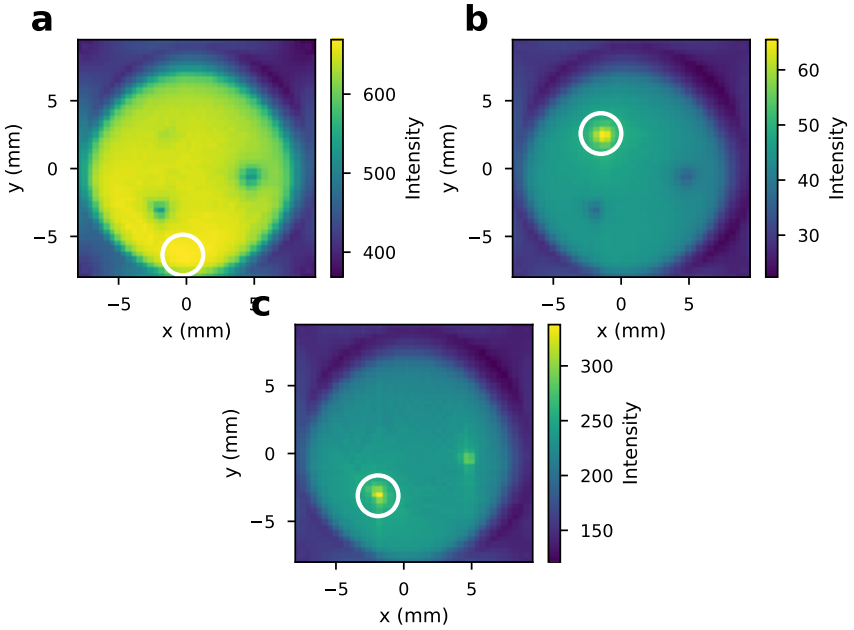


Figure 5.12: Reconstructed cross sections of the phantom. Each cross section shows a different region of the x-ray scattering pattern, highlighting each component, (a) Graphite (1.77-1.97 Å^{-1}) (b) Copper (2.92-3.12 Å^{-1}) (c) Wax (1.42-1.62 Å^{-1}) The circled region is shown in Fig. 5.13

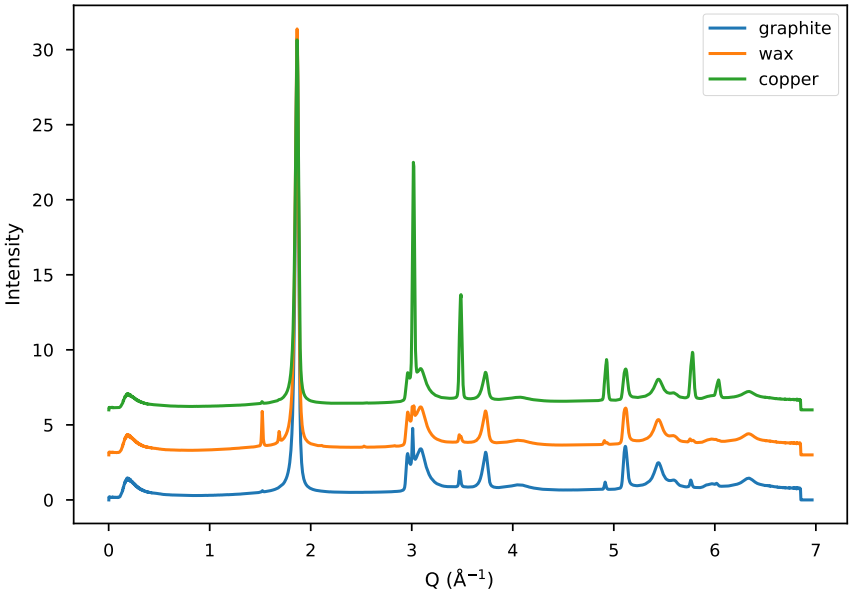


Figure 5.13: I(Q) for selected regions of the phantom. The selected regions are circled in Fig. 5.12

Silicon Carbide (SiC)

Introduction

The interplay between atomic structure and microstructure is important to the mechanical stability of materials. This relationship is especially important to nuclear materials, where a potent combination of conditions including thermal, chemical, mechanical and radiation flux places unusual demands on the material. One component of particular interest is the cladding material for the nuclear fuel, which keeps the fuel in place during the reaction and provides mechanical stability for nuclear waste disposal. SiC is of particular interest for next generation fusion and fission reactors as a structural material and fuel cladding for light water reactors. The combination of low neutron absorption cross section and SiC's strength and chemical durability make it an ideal candidate. Prior work on SiC plates explored the relationship between irradiation and microstructural features. The work by Sprouster et.al. showed that the shoulder peak associated with stacking faults and/or Frank-loops on 111 planes increased in intensity due to irradiation.

In this work we examined the stacking fault/Frank-loop number density on unirradiated SiC tubes using ctXRD to understand the macroscopic distribution of these defects in the SiC lattice. Understanding this distribution provides the first step to understanding how radiation impacts these engineering materials on operating scales, helping to elucidate the localization of faults and providing a baseline for future studies examining the evolution of these faults after being subjected to reactor operating conditions.

Experimental Setup

ctXRD experiments were carried out at the XPDD beamline of the NSLS-II. The SiC sample was centered on the Huber stage discussed in Section 5.1.2. The Dexela detector was used to capture the scattering images at a sample to detector distance of 762 mm with a wavelength

of .1899 Å. Detector calibration was performed using a Nickel standard and pyFAI. The tomography scan consisted of 90 2° steps and 40 250 μm steps with a 200 μm by 200 μm square beam.

Data Processing

The data was reduced and integrated using xpdAn. The images were dark and polarization corrected, masked to remove outliers, and integrated to 1D patterns using the default pipeline parameters. The directed acyclic graph (DAG) for the data processing is available from the implemented rapidz pipeline in xpdtools and xpdAn. After integration the data was corrected by the background extracted from the edge of the sinogram. Outliers in the sinogram were identified by summing the integrated data and calculating the Z-score of the resulting sinogram. The Z-score is calculated by $\frac{x-\langle x\rangle}{\sigma}$ where x is a pixel value $\langle x\rangle$ is the average value across the whole sinogram and σ is the standard deviation of the whole sinogram. Pixels which had Z-scores above or below 3 were identified as outliers. Upon inspection of the outlier patterns, it was determined that these were shifted by a constant value from their proper values. The outlier values were shifted by the difference between the average of the rest of the sinogram and their value, lifting the pattern to more appropriate values. This removal of outliers is important as the reconstruction can represent these outliers as streaks across the reconstructed image The center of the sinogram was then determined by tomopy. Tomopy also performed the reconstruction using the `ospml_hybrid`.

Results

Fig. 5.14 shows the distribution of intensity associated with the stacking fault, first, and second peaks. The distribution shows higher intensities on the left side of the tube for each of the selected peaks. Fig. 5.15 shows the I(Q) patterns for the tube, an unirradiated plate and plate irradiated at 0.1 displacements per atom (dpa). The patterns were normalized by their peak maxima, showing that the tube SiC at the position probed has a much higher

shoulder peak than either the unirradiated or irradiated plate. Fig. 5.16 follows the work of Sprouster et.al. showing the ratio between the stacking fault shoulder and (200) peak. Due to the instability of peak positions caused by the tomography experiment and reconstruction an alternative method was used to produce the ratio. In this case the ratio was produced by dividing the integrated area between, 2.3-2.46 $Å^{-1}$and 2.78-2.99 $Å^{-1}$for the shoulder and (200) peaks respectively. The same operation was performed on the data from [Sprouster *et al.*, 2017] allowing for direct comparison of the data. The comparison shows that the stacking fault defects not only have an azimuthal distribution across the tube, but that the density of these faults is higher than in the 0.1 dpa irradiated sample. This implies that there could be significantly different microstructural behavior over a complete tube compared to a solid plate. Additionally, these high stacking fault density sites could be potential sources of failure during operation, as the stacking fault density has a tendency to grow as does increase.

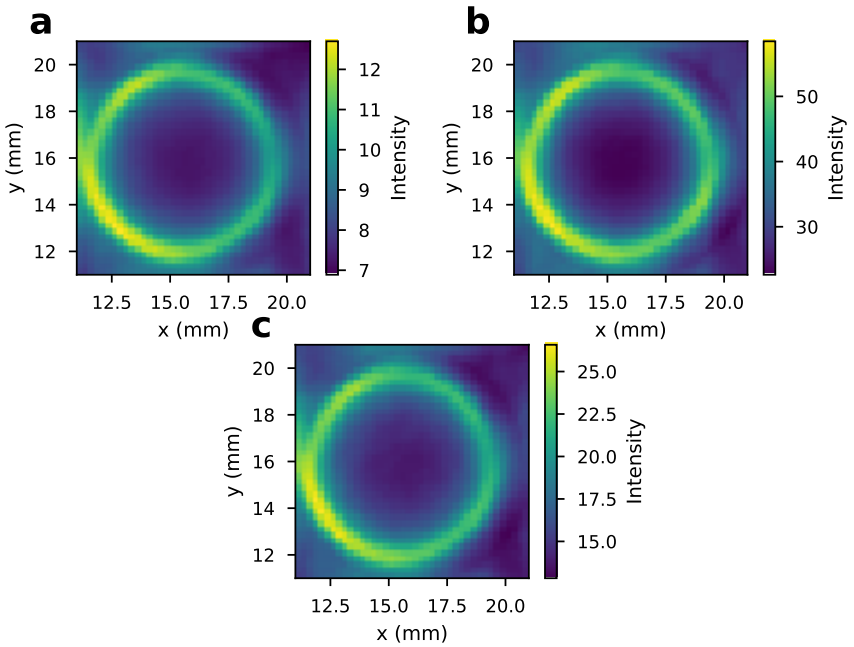


Figure 5.14: Integrated SiC selected peaks (a) Stacking fault sholder (2.3-2.46 $Å^{-1}$) (b) First Peak (2.48-2.51 $Å^{-1}$) (c) Stacking fault sholder (2.78-2.99 $Å^{-1}$)

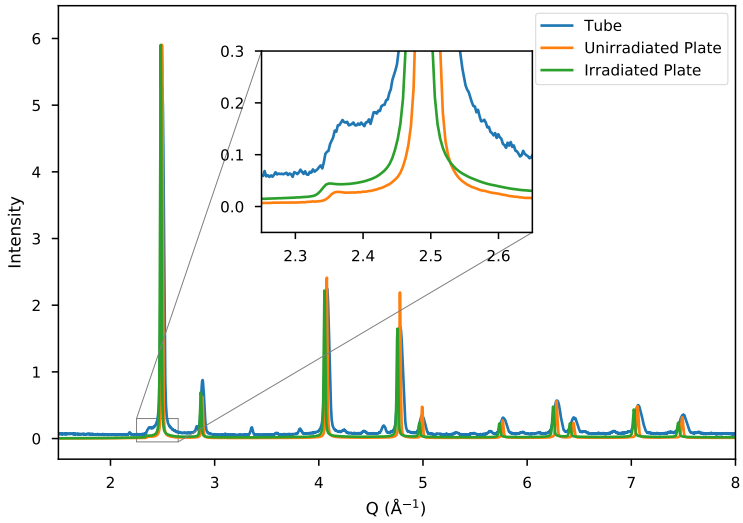


Figure 5.15: 1D scattering patterns from the measured SiC tube and two plates from [Sprouster *et al.*, 2017]. The irradiated plate was subjected to 0.1 displacements per atom (dpa) dose.

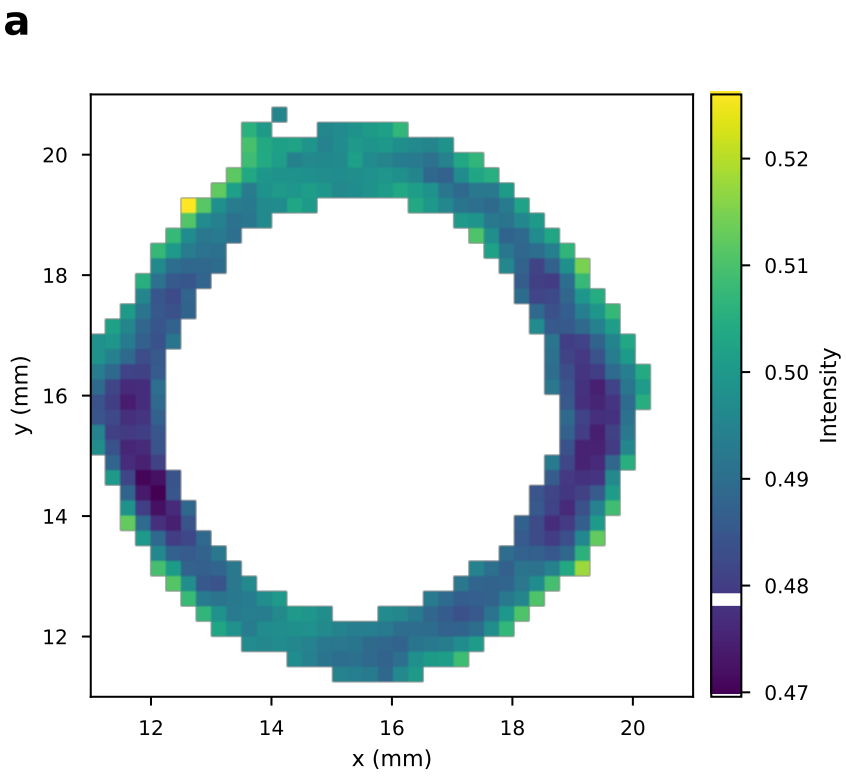


Figure 5.16: Tomogram of integrated intensity ratio between the stacking fault peak between 2.3-2.46 Åand the (002) peak between 2.78-2.99 Å. The tomogram shows non-azimuthally symmetric distribution of stacking fault density. The white bar in the color bar represents the stacking fault ratio of a SiC plate irradiated at .1 dpa dose from [Sprouster *et al.*, 2017]. Note that pixels off the tube were removed for clarity.

Mars Analog

Introduction

The NASA Mars 2020 mission will lay the groundwork for rocketing samples from the martian crust back to Earth. The main goal of this mission is to understand if the Martian surface could have supported life, in addition to providing otherwise unobtainable information on how Mars was formed and its geology evolved. Elucidation of the atomic and microscopic structure of these rocks will be critical to understanding Martian geology and potential for life [Hurowitz *et al.*, 2017].

Experimental Setup

Sample

The sample consisted of a Titanium Aluminum Vanadium alloy tube packed with three igneous and three sedimentary rocks. The rocks were obtained from the Mars 2020 Sample Caching System at the NASA Jet Propulsion Laboratory. The rocks were separated by foam filler and measured approximately 2 cm long and 1.5 cm in diameter [Hurowitz *et al.*, 2017].

ctXRD measurements

ctXRD experiments were performed at the XPDD beamline at the National Synchrotron Light Source-II. The x-ray beam was cut down to 200 um square by a set of four slits. X-ray scattering images were taken on a Dexela 2923 detector at a sample to detector distance of 762 mm with a wavelength of .1899 Å. A 2D tomography scan was performed across the tube with 136 rotations from -140 to 40 degrees, and 81 translations from 7.8 mm to 23.8 mm.

Data Processing

xpdAn was used to process the scattering images to 1D integrated patterns. The resulting patterns were then processed into a sinogram, where outliers were removed in a procedure similar to Section 5.4.3. The resulting data was reconstructed with the `ospml_hybrid` algorithm.

Results

The reconstructed cross sections are shown in Fig. 5.17. The reconstructions show a strong delineation between the titanium alloy tube and the martian analogue. The reconstructions also show the non-uniformities in the rock with Fig. 5.17 (a) showing more intensity in the middle of the rock, where (b) shows more evenly distributed intensity. Phase identification of the center of the rock yielded many different components including Anorthite, Albite, Quartz and Kyanite. This is consistent with the description of the phyllosilicate phase described in [Hurowitz *et al.*, 2017]. The tube was identified as mostly Titanium with some Aluminum Titanium alloy phase. As shown by the simulated reconstructions with noise the reconstructions are quite robust, so the variations shown in Fig. 5.17 are due to the variations of the rock itself.

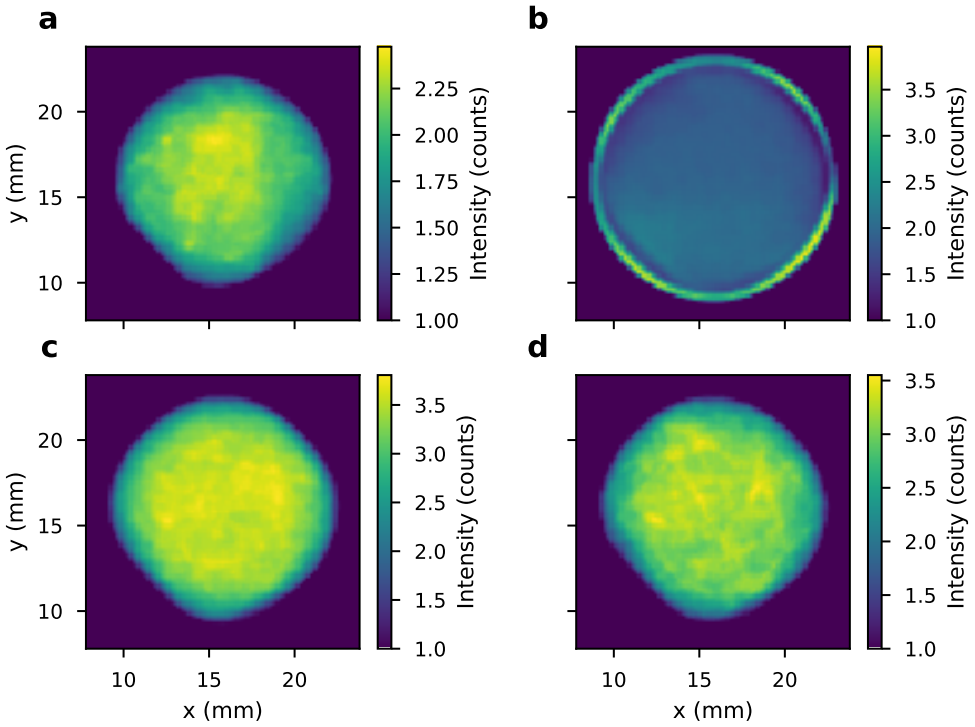


Figure 5.17: Tomographic reconstruction of (a) the Mars tube from $2\theta = 0.6°$ to $2\theta = 0.9°$ and (b) the Mars tube from $2\theta = 8.1°$ to $2\theta = 8.4°$ and (c) the Mars tube from $2\theta = 5.9°$ to $2\theta = 6.1°$. (a) shows some strong scattering areas, which may come from larger grains, and the general scattering from the rock while (b) shows the Titanium tube bounding the sample, with a small amount of scattering from the sample itself, showing the gap between the sample and the tube.

Summary

This chapter discussed the commissioning of an experimental setup for x-ray scattering tomography. Simulations were performed which showed that some reconstruction algorithms perform better at minimizing the deviation from true peak positions. The extent of these deviations were explored, showing that the effect is worse at higher scattering vectors. This work was then corroborated with experiments, which showed similar results. The tomography experiments and calculations were further commissioned with a phantom showing diffraction contrast. Tomography was then used to explore the stacking fault structure of nuclear grade SiC tubes. Finally, a mars regolith analogue was examined with scattering tomography showing different crystal grains present.

Chapter 6

Mapping the Melt

Introduction

In this chapter we report the results of ctXRD and scanning positional XRD analysis of ex-situ samples of TiO2 from a conventional optical float-zone furnace. The goal of this work is to perform in-situ structural characterization during a FZ growth, providing insight into the formation of crystals from the melt. Reaching this goal requires:

1. an in-situ furnace
2. a beamline with hardware set up for tomography and scanning experiments
3. software for experimental control and data acquisition
4. software for data reduction and analysis
5. algorithms for extracting information from the crystals, including: degree of crystallinity, strain, and orientation

Eventually this could be used to drive autonomous experiments to understand the exact conditions needed to produce certain crystal microstructures, like particular grain boundaries or strains. To further this goal the work presented in this chapter focuses on preliminary results of ex-situ testing of this experimental and computational infrastructure.

Scientific Questions

The main scientific goal of this work is to understand the growth of single crystals form polycrystalline feed and seed rods in an optical float zone furnace. Reaching this goal involves addressing the following questions:

1. What factors govern the competition between crystallites during growth?
2. What factors affect crystal quality as a function of growth?
3. How are crystal quality and competition related?

Answering these questions requires:

1. Characterization of uniformity, and graininess in the feed and seed rods
2. Mapping the position, orientation, strain, and mosaicity of crystallites in the recrystallized region

We expect that grain orientation will play an important role in determining which grains will grow and which will not. Lattice strain may play an important role in mediating propagation of grain boundaries and influence the mosaicity.

Required Measurement Capabilities

Reaching these characterization goals requires measuring powder ctXRD to understand the microstructure of the seed and feed rods and 2D maps of the powder and crystalline regions. The 2D maps will provide a diffraction pattern at each xy point, which will enable:

1. quantitative measures of graininess
2. mapping of individual crystal grains by associating sets of single crystal reflections with individual grains

3. mapping of d-spacings to get information about the lattice strain

4. mapping of reflection hkls and azimuthal angles

5. mapping of crystal grain orientations

These 2D maps could be further enhanced by simultaneously performing rocking curve measurements. These rocking measurements will shift the region of the Ewald sphere probed by the x-ray diffraction, helping to cover more of reciprocal space. The extra coverage will provide a better understanding of the mosaicity in the lattice by measuring the spread of intensity at each hkl, as shown in Fig. 2.1 as a function of xy position. Alternately, summing the scattering patterns across the rocking angles would provide a composite pattern where more points in the grain are in the diffraction condition, providing a better measurement of the d-spacings across a crystal.

Ideally, these experiments would be performed on a wide range of crystals, produced with different growth parameters, so a general pattern could be formed. Additionally, in-situ experiments could provide insight into the mechanism of growth, especially as the growth parameters could be tuned by computer to understand the response of the growth to changing parameters.

Scope of Current Work

The scope of this work is to develop the computational infrastructure to perform ctXRD measurements and data processing, and map the boules for their graininess, crystal grain locations, d-spacings, and hkl azimuthal angles, in a way that it can be reused when the in-situ furnace is built and for high throughput experiments where many crystals are characterized. This infrastructure was used on an ex-situ Rutile model system. The simplicity of the system, compared to some of the more chemically complex boules, allows for focusing on the effectiveness of the infrastructure.

Methods

Growth Details

Two boules were provided to be studied with our computational infrastructure. Boule A was made from anatase nanopowder (99.9% pure) from Alfa Aesar. The powder was dried at $\sim$ 1000°C in a box furnace. Rods were hydrostatically pressed at $\sim$ 65 MPa by sealing the powder in a rubber tube under vacuum. The rod was sintered in a box furnace at 1000 °C for 12 hours in air. The seed and feed rods were counter rotated at 10 rpm under O2 with a flow rate of 500mL/min. The feed and seed rods were moved downward at a rate of 6 and 5 mm/hr, respectively. The laser providing the heating was set at a 0 °angle. When the growth was finished the laser was turned off and the rods cooled in flowing O2 Boule A is cylindrical with a diameter of 5 mm and a length of at least 20 mm in the seed region and 15 mm in the crystalline zone. Boule B is a legacy sample, whose provenance is less well known, although it is a feed rod with a length of at least 16 mm and a diameter of 5 mm.

Experimental Setup

X-ray scattering measurements of boule A and B were performed at the PDF beamline of the National Synchrotron Light Source-II. The experiments included XY maps of both boule A and B and a ctXRD measurement of the powder region of boule A. The ctXRD measurement was done in tandem with a measurement of a Nickel standard. Both boules were held by a compression fit of Teflon tape in a Swagelok Teflon nut, which was affixed to a rotation stage. A Dexela 2923 detector used because of its resistance to beam damage caused by extremely bright single crystal scattering. The x-ray wavelength was 0.1688 Åwith a sample to detector distance of 539 mm. XpdAcq and xpdAn were used to collect and process the data.

Data Processing

ctXRD

Integration of the 2D data was performed by xpdAn. Tomopy found the center and did the reconstruction with the `ospml_hybrid` algorithm.

Graininess Metric

A simple metric was created to track if an image represented mostly powder or single crystal diffraction. The graininess metric counts the number of pixels which have values above 100 counts after dark, background and polarization corrections are applied. This is a simple to compute and unbiased metric that is large for a powder and small for a single crystal, and will interpolate between at all levels of graininess. The downside for this approach is that it can underestimate the amount of single crystal material, as a single crystal could be oriented in a way that no peaks fall on the detector, causing the metric to give similar results to when the beam is not on the sample.

Single Crystal Peak Tracking

Many parts of the proposed analysis rely on the identification, tracking, and indexing of the spots from single crystal scattering. To provide this information: dark field and polarization corrections were applied to the raw images. Trackpy was then used to identify single crystal peaks in the images [Allan *et al.*, 2016]. The peak tracking is most useful for samples that are not good powders and was performed on images whose powderness metric was between 4000 and 100000. This filtering helps to prevent false positives associated with the powder scattering rings and reduces computational overhead by not extracting peaks from images without diffraction. The identified peaks were downsampled by requiring the peak to exist in the crystalline region of the sample, as determined by the powderness metric. The tracked peaks were then combined into trajectories, associating peaks across multiple images and

providing a unique set of observed spots. The trajectories were downsampled to only include peaks which appeared in at least 30 images, allowing analysis on a non-trivial number of XY positions. This set of peaks were then used for subsequent analysis.

Heat Maps

Intensity heat maps were produced from the images by generating regions of interest (ROIs) around the average position of each identified peak on the detector. The ROIs were 10 pixels square and the values withing the ROI were summed for each XY position. This then created a heat map of the single crystal spot intensity for each spot.

Segmentation and Overlays

The intensity heat maps were segmented using a Sobel filter and waterfall based method from scikit-image [van der Walt *et al.*, 2014]. This segmentation was used to separate the individual crystal grains. Streaks in the heat maps, due to residual charge being measured by the detector, caused the segmentation to not yield perfect results, but it was able to separate the grains. Overlays of the heat maps were also created to show how the intense regions of the crystals matched with other crystals.

Indexing, Average d-spacing, Azimuthal Angle

Each of the single crystal spots was indexed, associating a set of hkl values with the spot. This was performed by extracting the average scattering vector for each spot using the pixel coordinates on the detector and the detector calibration. The measured scattering vector was then compared against expected values from PyMatGen for Rutile. The index with the smallest distance to the measured value was then assigned to the measured spot. The average d-spacing was then also associated with the spot, as was the average position of the spot in the azimuthal angle.

d-spacing and Azimuthal Angle Distributions

In addition to the average d-spacing and azimuthal angle extracted for each index a distribution across the crystals was also extracted. This data was produced by extracting the position of each spot on each image, for those image which had that particular reflection. The position of each spot was then used to calculate the d-spacing and azimuthal angle for that spot. While this analysis is limited to images which contain that particular spot, which is not necessarily the same as belonging to a particular grain due to changes in reflections across the grain, it is able to track the changes in the d-spacing and azimuthal angle across a single grain.

Results

Diffraction by Boule A

Three representative images from Boule A are shown in Fig. 6.1. The images show representative data from the crystalline, polycrystalline and powder regions with annotations showing the crystalline peaks which are tracked by trackpy. Fig. 6.2 shows the associated $I(Q)$ patterns.

ctXRD Results

The Fig. 6.3 is a tomogram of the rutile peak height, indicating that the seed rod is a uniform powder with no voids. The feed regime is expected to be the similarly powder in nature, since both rods were pressed using the same method.

Graininess

The graininess metrics shown in Fig. 6.4 show a strong delineation between the crystalline and powder regions in the seed and feed rods. The feed rod shows crystallization at the tip of

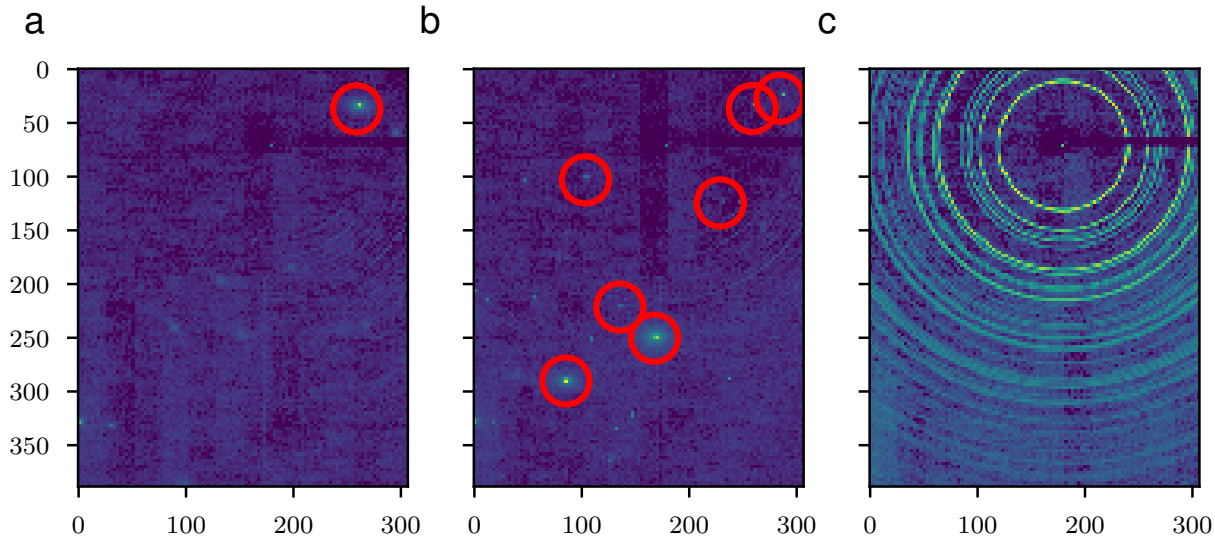


Figure 6.1: Representative scattering images from Boule A. The circles denote diffraction spots found from trackpy. a) shows a single crystal image b) shows a image with multiple single crystals c) shows a fully powder image

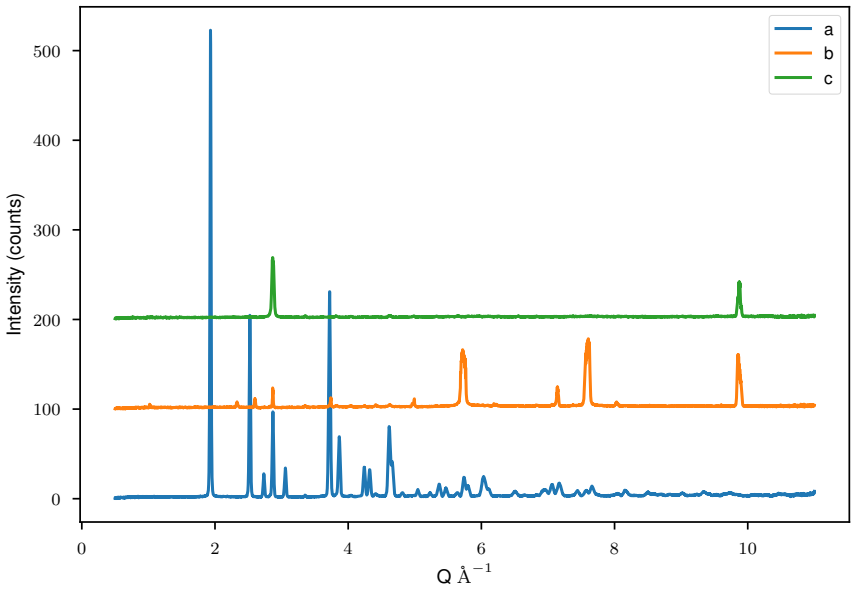


Figure 6.2: Diffraction patterns from Boule A. The lettering matches the images from Fig. 6.1

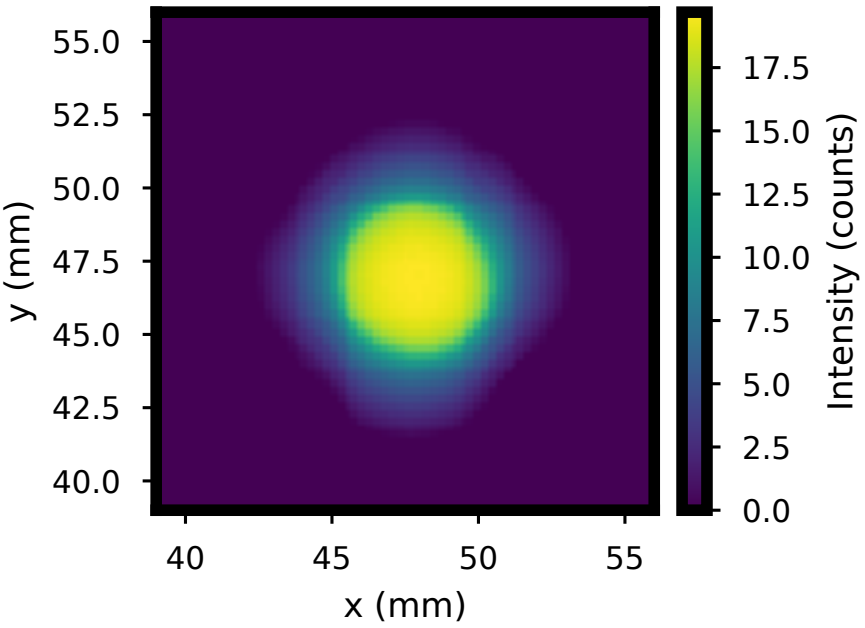


Figure 6.3: Tomographic reconstruction of the first rutile peak intensity for the Boule A seed region

the rod, most likely from the when the growth was finished and the two rods were separated, letting the liquid left on the feed to cool. The middle of the feed rod is polycrystalline, showing the impact of being close to the furnace hot spot causing the powder's grains to grow. The transition between the powder and polycrystalline regions has a parabolic shape, which is most likely due to the limited penetration of the heat into the boule so far away from the hot spot. The seed boule shows significant differences in graininess compared to the feed boule. The bottom of the seed boule is well powdered, with an abrupt transition to crystalline around 40 mm. This is most likely where the float zone started as a slight neck can be observed between 40 mm and 45 mm. Both boules show reduced powderness at the edges, most likely due to the powderness metric not being thickness invariant and there is less material to scatter through on the sides.

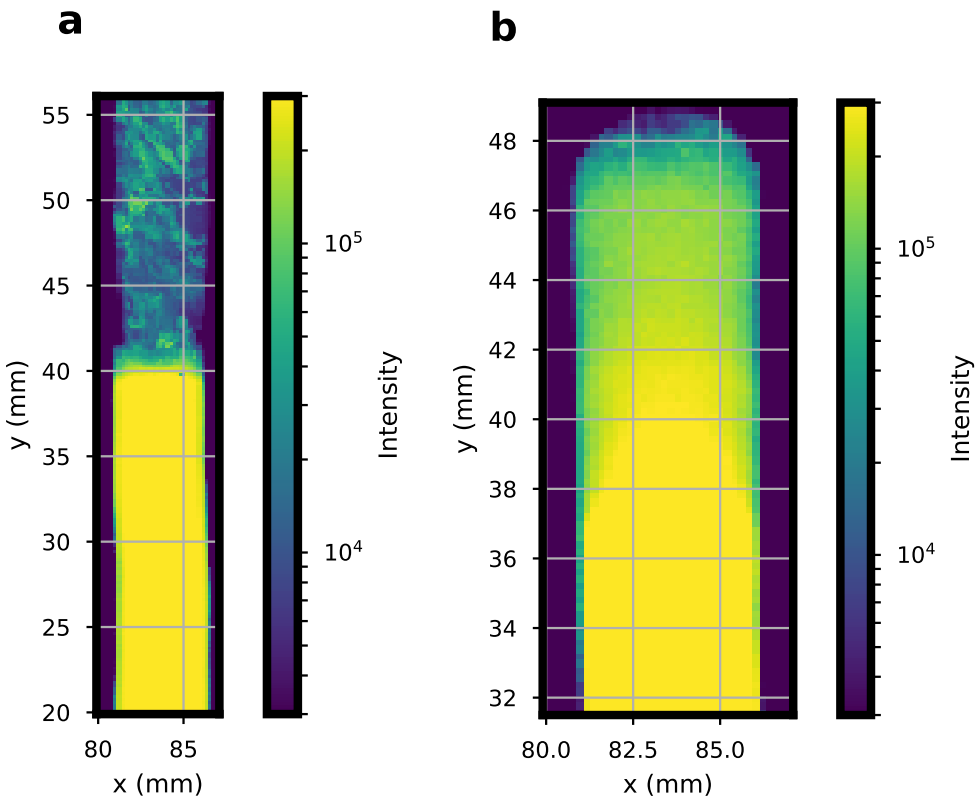


Figure 6.4: Powder metric as a function of XY position for Boule A, left and Boule B, right

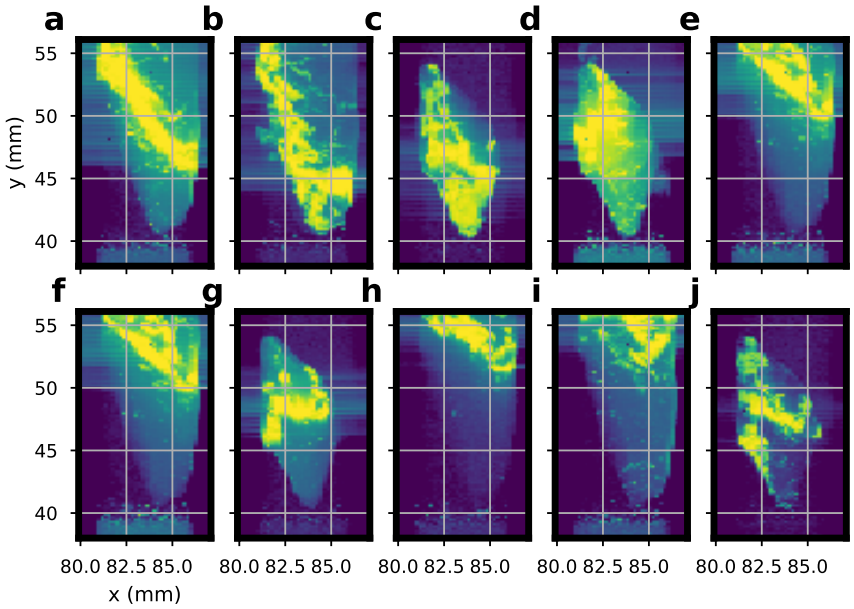
Heat Maps

The intensity heat maps, shown in Fig. 6.5 and Fig. 6.6, track the intensity of each single crystal spot as a function of XY position in the boules. The heat map for boule B shows the crystallization at the tip of the rod, which consists of small crystals around 2 mm to 500μm in size. Boule B also shows the formation of larger crystal grains in the polycrystalline region. These small grains, seem to have some influence on the crystallization of some of the larger grains. In Fig. 6.6 b-e, g and j there is a tail of intensity reaching from below the peak. This may indicate that a small grain from the polycrystalline region is influencing the crystallization from the melt.

Boule A shows many crystalline regions in Fig. 6.5. Interestingly the intensity of these crystals is not uniform, as would be expected for a perfect single crystal. This indicates that the crystal planes were not exactly in the diffracting condition across the bulk of the crystal. The large differences in intensity across the crystal may be due to a change in orientation across the crystal or due to strain.

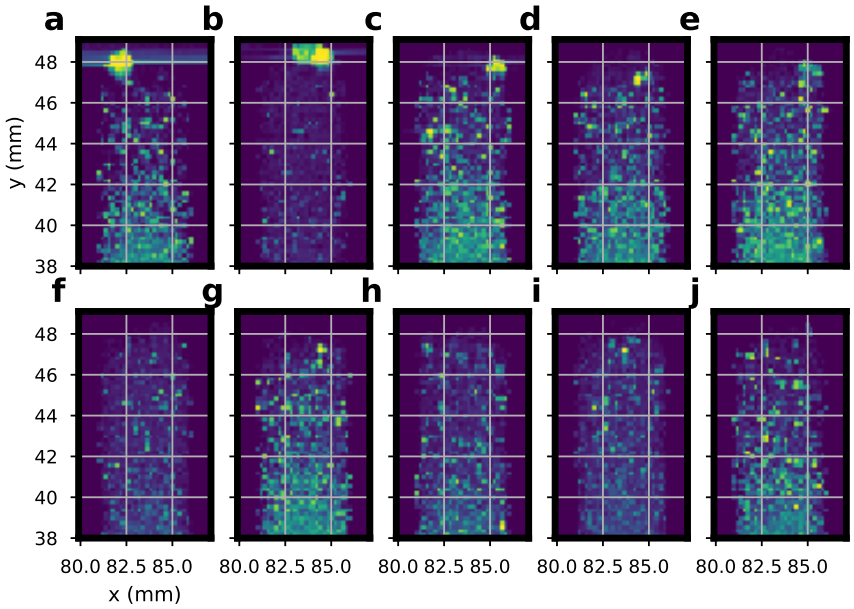
Overlays

Overlays of the heat maps from Fig. 6.5 associated with different crystals in boule A are shown in Fig. 6.7. The overlays show that the most intense regions of the single crystal scattering are closely related to where the crystals meet. This is supported by the other spots which are shown in Fig. 6.5 where d and e seem to share a boundary, as do c/i and j. This may imply that the meeting of the crystals is either causing a strain in the lattice or twisting in the crystal's orientations to bring the regions near the boundary into the diffraction condition.



index	*hkl*	azimuthal angle (degrees)	expected *d*-spacing (Å)	Δ *d*-spacing (mÅ)
a	(3, 3, 1)	-13.4	1.029	3.281
b	(2, 0, 2)	-174.5	1.252	19.192
c	(5, 2, 3)	-37.4	0.651	-0.598
d	(1, 1, 2)	-179.4	1.353	6.267
e	(1, 1, 2)	-67.3	1.353	-7.363
f	(1, 1, 2)	-65.6	1.353	5.517
g	(6, 1, 0)	-108.8	0.765	0.457
h	(6, 2, 0)	-94.6	0.736	0.877
i	(1, 0, 1)	32.7	2.503	-7.025
j	(2, 0, 2)	122.6	1.252	-9.887

Figure 6.5: Above: Selected ROI intensity maps for Boule A. These maps show separate crystal orientations as a function of position in the FZ grown crystal. Below: hkl and azimuthal angle assignments for Boule A



index	*hkl*	azimuthal angle (degrees)	expected *d*-spacing (Å)	Δ *d*-spacing (mÅ)
a	(1, 0, 1)	25.8	2.503	-10.57
b	(5, 1, 3)	-73.8	0.671	-3.918
c	(2, 1, 1)	-24.5	1.704	-15.308
d	(2, 1, 1)	-30.6	1.704	-23.931
e	(2, 1, 1)	-85.3	1.704	-16.365
f	(2, 1, 3)	22.6	0.894	-2.233
g	(2, 1, 1)	26.5	1.704	-19.235
h	(2, 1, 1)	-138.7	1.704	-28.766
i	(2, 2, 0)	-164.2	1.645	26.005
j	(2, 1, 1)	-48.4	1.704	-28.134

Figure 6.6: Above: Selected ROI intensity maps for Boule B. These maps show separate crystal orientations as a function of position in the FZ grown crystal. Below: hkl, azimuthal angle and d-spacing assignments for Boule B.

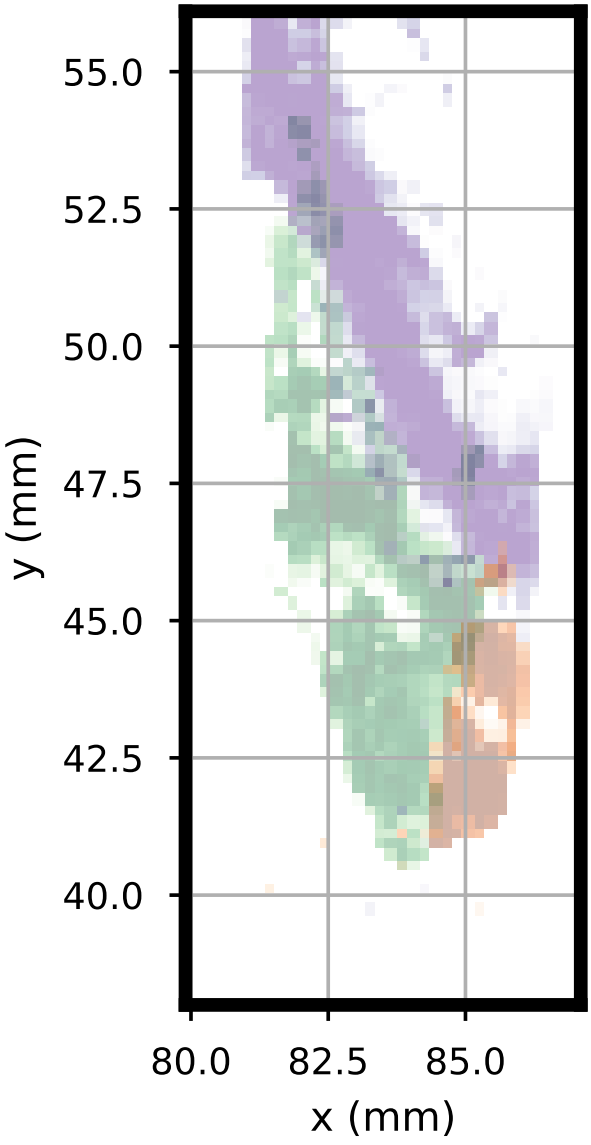


Figure 6.7: Overlay of selected heat maps from Boule A. Note that the most intense regions of the overlaid segments are overlapping, or nearly overlapping

Segmentation

The segmentation of the crystals enables the observation of how the crystals stack onto one another and enable the association of multiple reflections to the same crystal. Fig. 6.8 shows the segmentation of boule A, indicating that all the crystals start their growth from the top of the seed rod. Additionally the segmentation map shows that only one crystal of the 7 identified terminates within the crystal. All other crystals terminate their growth by expanding to the edge of the crystal and being pushed out from the interior with another crystal taking over the edge of the crystal. It is possible that the one outlier crystal has expanded to the edge, since this is a 2D projection of a 3D boule the map is insensitive to the depth dimension.

d-spacing and Angle Tracking

Fig. 6.9 and Fig. 6.10 show the mapping of the observed reflections deviation in d-spacing and azimuthal angle from the average for that reflection, and the intensity for reference. Fig. 6.9 shows an interesting disconnection between the azimuthal angle distribution and d-spacing distribution. The azimuthal distribution is quite continuous with a smooth gradient from the lower end of the crystal to the upper end. The d-spacing arrangement is much less gradual, with regions of high and low deviations right next to one another. Additionally, the d-spacing shows little middle ground with a bimodal distribution in the lattice parameter.

Fig. 6.10 displays an interesting relationship between the dark spot between 50.0 and 47.5 and 85.0, where that part of the crystal seems to be in compression compared to the rest of the crystal. There seems to be a similar shift to negative azimuthal angles in the same region, showing much more correlation than seen in Fig. 6.9 but less smooth gradients in the azimuthal angle than Fig. 6.9 Similar to Fig. 6.9 there seems to be a bimodal distribution in d-spacing deviation.

While d-spacing calculations can be influenced by sample to detector distance changes,

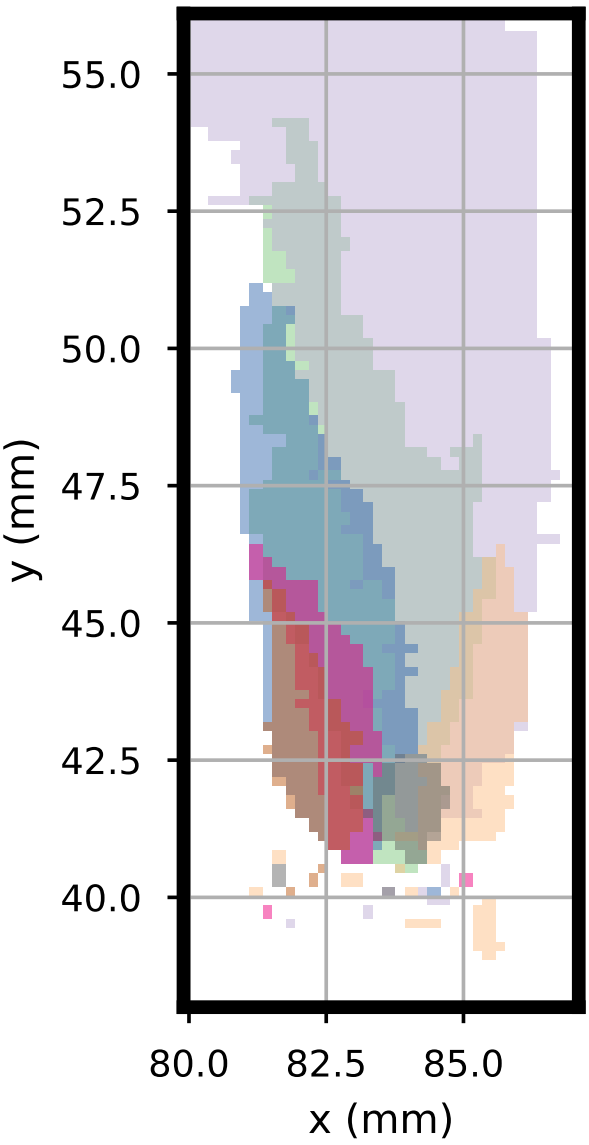


Figure 6.8: Selected crystal grains produced by image segmentation

potentially due to changes in the thickness of the particular crystal as it grows and competes with other crystals, the azimuthal angle would not be changed by the crystal thickness. This implies that the azimuthal angle changes may be more reliable than the d-spacing changes.

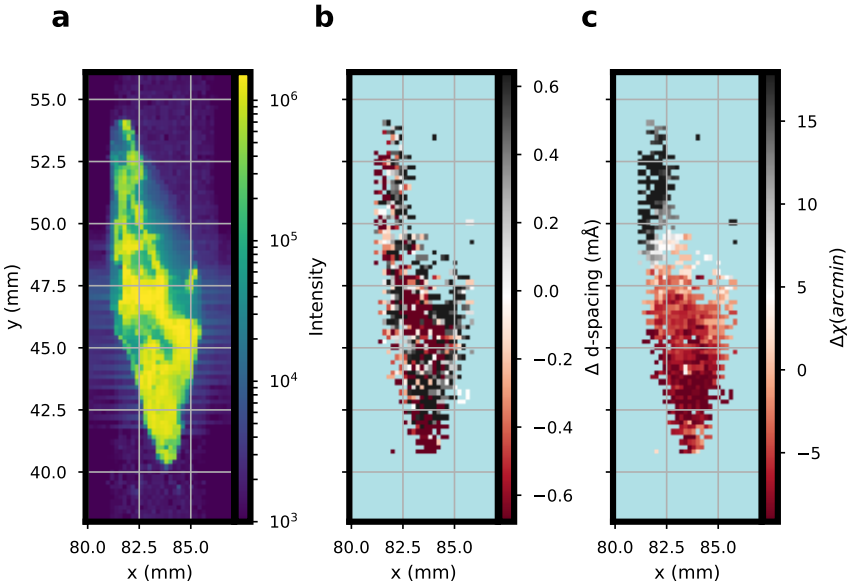


Figure 6.9: The (a) intensity, (b) deviation in d-spacing, and (c) deviation in azimuthal angle χ in crystallite α Note how the azimuthal angle deviation smoothly transitions from bottom to top.

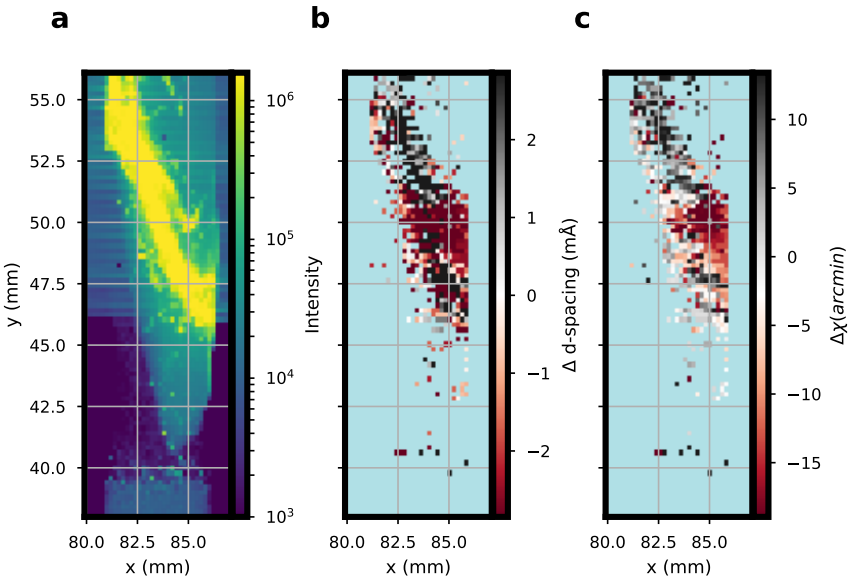


Figure 6.10: The (a) intensity, (b) deviation in d-spacing, and (c) deviation in azimuthal angle χ in crystallite β Note That the pocket of smaller d-spacings is located near the low intensity region between $y = 47.5, 50.0$ and $x = 85.0$.

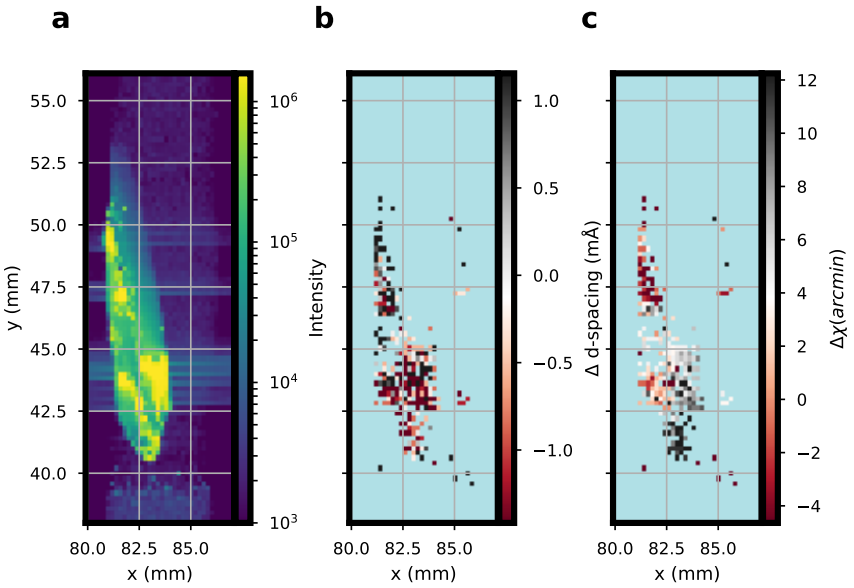


Figure 6.11: The (a) intensity, (b) deviation in d-spacing, and (c) deviation in azimuthal angle χ in crystallite γ Note how the two main regions have separate azimuthal angle and d spacing behavior.

Summary

X-ray diffraction measurements were performed on two float zone furnace grown rutile boules. ctXRD measurements showed that the seed rod was a uniform powder. XY positional mapping of the feed, seed and crystalline regions showed clear delineations between powder, polycrystalline and crystalline zones. These measurements also showed significant deviations from expected single crystal scattering behavior, exhibited by non uniform scattering from each single crystal, deviations in d-spacings across the crystal, and in the azimuthal spot position. These deviations are consistent with either strain in the lattice or mosaicity. Deviations in the intensity are located at the grain boundaries, implying that the boundary may be causing these deviations. Segmentation of the crystals showed that all but one crystal was not terminated at the edge of the boule. Overall the frameworks and tooling described in this and previous chapters have provided an unprecedented glimpse into the FZ growth of crystals, and will provide a strong foundation for future in-situ experiments.

Chapter 7

Conclusion

The goal of this work was to understand the growth of single crystals in an optical float zone furnace and to build the tooling needed to support in-situ furnace experiments.This goal was reached by combining use x-ray scattering tomography and x-ray diffraction crystal mapping to provide insight into the microstructure of the grown crystal. These techniques required the development of significant experimental, and computational infrastructure. Developments in computational infrastructure occurred in three major thrusts, construction of a framework for the processing of streaming heterogeneous data, the implementation of that framework for x-ray scattering and tomography data processing, and the development of ctXRD simulation software. The experimental developments required the commissioning of the ctXRD technique and hardware via multiple phantom measurements, and the measurement of float zone grown boules.

Bibliography

[Aczel *et al.*, 2008] A. A. Aczel, H. A. Dabkowska, P. R. Provencher, and G. M. Luke. Crystal growth and characterization of the new spin dimer system Ba3cr2o8. *Journal of Crystal Growth*, 310(4):870–873, February 2008.

[Akashi *et al.*, 1969] T. Akashi, K. Matumi, T. Okada, and T. Mizutani. Preparation of ferrite single crystals by new floating zone technique. *IEEE Transactions on Magnetics*, 5(3):285–289, September 1969.

[Allan *et al.*, 2016] Daniel Allan, Thomas Caswell, Nathan Keim, and Casper van der Wel. Trackpy: Trackpy V0.3.2, August 2016.

[Allan *et al.*, 2019] Daniel Allan, Thomas Caswell, Stuart Campbell, and Maksim Rakitin. Bluesky's Ahead: A Multi-Facility Collaboration for an *a la Carte* Software Project for Data Acquisition and Management. *Synchrotron Radiation News*, 32(3):19–22, May 2019.

[Anand *et al.*, 2018] V. K. Anand, A. T. M. N. Islam, A. Samartzis, J. Xu, N. Casati, and B. Lake. Optimization of single crystal growth of candidate quantum spin-ice Pr2hf2o7 by optical floating-zone method. *Journal of Crystal Growth*, 498:124–129, September 2018.

[Behr *et al.*, 1999] G Behr, W Lser, G Graw, H Bitterlich, J Freudenberger, J Fink, and L Schultz. Growth of RENi2b2c single crystals by RF-zone melting. *Journal of Crystal Growth*, 198-199:642–648, March 1999.

[Behr *et al.*, 2010] G. Behr, W. Lser, N. Wizent, P. Ribeiro, M.-O. Apostu, and D. Souptel. Influence of heat distribution and zone shape in the floating zone growth of selected oxide compounds. *Journal of Materials Science*, 45(8):2223–2227, April 2010.

[Bethel *et al.*, 2016] E. W. Bethel, M. Greenwald, K. K. van Dam, M. Parashar, S. M. Wild, and H. S. Wiley. Management, analysis, and visualization of experimental and observational data The convergence of data and computing. In *2016 IEEE 12th International Conference on e-Science (e-Science)*, pages 213–222, October 2016.

[Bicer *et al.*, 2017] T. Bicer, D. Gursoy, R. Kettimuthu, I. T. Foster, B. Ren, V. De Andrede, and F. De Carlo. Real-Time Data Analysis and Autonomous Steering of Synchrotron Light Source Experiments. In *2017 IEEE 13th International Conference on e-Science (e-Science)*, pages 59–68, October 2017.

[Blaiszik *et al.*, 2019] Ben Blaiszik, Kyle Chard, Ryan Chard, Ian Foster, and Logan Ward. Data automation at light sources. *AIP Conference Proceedings*, 2054(1):020003, January 2019.

[Bridgman, 1925] P. W. Bridgman. Certain Physical Properties of Single Crystals of Tungsten, Antimony, Bismuth, Tellurium, Cadmium, Zinc, and Tin. *Proceedings of the American Academy of Arts and Sciences*, 60(6):305, 1925.

[Busing and Levy, 1967] W. R. Busing and H. A. Levy. Angle calculations for 3- and 4-circle X-ray and neutron diffractometers. *Acta Crystallographica*, 22(4):457–464, April 1967.

[Campbell, 2019] Stuart Campbell. DAMA Update, March 2019.

[Chang *et al.*, 2013] Kelvin B. Chang, Laszlo Frazer, Johanna J. Schwartz, John B. Ketterson, and Kenneth R. Poeppelmeier. Removal of Copper Vacancies in Cuprous Oxide Single Crystals Grown by the Floating Zone Method. *Crystal Growth & Design*, 13(11):4914–4922, November 2013.

[Chupas *et al.*, 2003] P. J. Chupas, X. Qiu, J. C. Hanson, P. L. Lee, C. P. Grey, and S. J. L. Billinge. Rapid-acquisition pair distribution function (RA-PDF) analysis. *Journal of Applied Crystallography*, 36(6):1342–1347, December 2003.

[Davison *et al.*, 2014] Andrew P. Davison, Michele Mattioni, and Dmitry Samarkanov. Sumatra: A Toolkit for Reproducible Research, April 2014.

[Davison, 2012] A. Davison. Automated Capture of Experiment Context for Easier Reproducibility in Computational Research. *Computing in Science Engineering*, 14(4):48–56, July 2012.

[Dectris,] Corp Dectris. PILATUS3 X CdTe 2m.

[Defense Technical Information Center, 1966] Defense Technical Information Center. *DTIC AD0378363: ENGINE PROPOSAL FOR PHASE 3 OF THE SUPERSONIC TRANSPORT DEVELOPMENT PROGRAM. VOLUME 3. TECHNICAL ENGINE. REPORT B. ENGINE DESIGN*. September 1966.

[du Plessis and Boshoff, 2019] Anton du Plessis and William P. Boshoff. A review of X-ray computed tomography of concrete and asphalt construction materials. *Construction and Building Materials*, 199:637–651, February 2019.

[du Plessis and Rossouw, 2015] Anton du Plessis and Pierre Rossouw. X-ray computed tomography of a titanium aerospace investment casting. *Case Studies in Nondestructive Testing and Evaluation*, 3:21–26, April 2015.

[Dbkowska *et al.*, 2015] Hanna Dbkowska, Antoni Dbkowski, Regina Hermann, Janis Priede, and Gunter Gerbeth. 8 - Floating Zone Growth of Oxides and Metallic Alloys. In Peter Rudolph, editor, *Handbook of Crystal Growth (Second Edition)*, Handbook of Crystal Growth, pages 281–329. Elsevier, Boston, January 2015.

[Eckert, 2012] M. Eckert. Max von Laue and the discovery of X-ray diffraction in 1912. *Annalen der Physik*, 524(5):A83–A85, May 2012.

[Finegan *et al.*, 2019] Donal P. Finegan, Antonis Vamvakeros, Lei Cao, Chun Tan, Thomas M. M. Heenan, Sohrab R. Daemi, Simon D. M. Jacques, Andrew M. Beale, Marco Di Michiel, Kandler Smith, Dan J. L. Brett, Paul R. Shearing, and Chunmei Ban. Spatially Resolving Lithiation in SiliconGraphite Composite Electrodes via in Situ High-Energy X-ray Diffraction Computed Tomography. *Nano Letters*, 19(6):3811–3820, June 2019.

[Frank, 2006] Joachim Frank. Introduction: Principles of Electron Tomography. In Joachim Frank, editor, *Electron Tomography: Methods for Three-Dimensional Visualization of Structures in the Cell*, pages 1–15. Springer New York, New York, NY, 2006.

[Frazer *et al.*, 2015] Laszlo Frazer, Kelvin B Chang, Kenneth R Poeppelmeier, and John B Ketterson. Cupric oxide inclusions in cuprous oxide crystals grown by the floating zone method. *Science and Technology of Advanced Materials*, 16(3), May 2015.

[Granda *et al.*, 2018] Jarosaw M. Granda, Liva Donina, Vincenza Dragone, De-Liang Long, and Leroy Cronin. Controlling an organic synthesis robot with machine learning to search for new reactivity. *Nature*, 559(7714):377, July 2018.

[Grosse *et al.*, 2013] Mirco K. Grosse, Juri Stuckert, Martin Steinbrck, Anders P. Kaestner, and Stefan Hartmann. Neutron Radiography and Tomography Investigations of the Secondary Hydriding of Zircaloy-4 during Simulated Loss of Coolant Nuclear Accidents. *Physics Procedia*, 43:294–306, January 2013.

[group, 2019] DAMA group. Data Model Bluesky Event Model 1.11.1 documentation, August 2019.

[Grsoy *et al.*, 2014] D. Grsoy, F. De Carlo, X. Xiao, and C. Jacobsen. TomoPy: a framework for the analysis of synchrotron tomographic data. *Journal of Synchrotron Radiation*, 21(5):1188–1193, September 2014.

[Hammersley *et al.*, 1996] A. P. Hammersley, S. O. Svensson, M. Hanfland, A. N. Fitch, and D. Hausermann. Two-dimensional detector software: From real detector to idealised image or two-theta scan. *High Pressure Research*, 14(4-6):235–248, January 1996.

[Harding *et al.*, 1987] G. Harding, J. Kosanetzky, and U. Neitzel. X-ray diffraction computed tomography. *Medical Physics*, 14(4):515–525, 1987.

[He, 2018] Bob Baoping He. Miscellaneous Applications. In *Two-dimensional X-ray Diffraction*, pages 395–432. John Wiley & Sons, Inc., Hoboken, NJ, USA, June 2018.

[Herman, 2009] Gabor T. Herman. *Fundamentals of Computerized Tomography: Image Reconstruction from Projections*. Advances in Computer Vision and Pattern Recognition. Springer-Verlag, London, 2 edition, 2009.

[Higuchi and Kodaira, 1992] Mikio Higuchi and Kohei Kodaira. Effect of ZrO2 addition on FZ growth of rutile single crystals. *Journal of Crystal Growth*, 123(3):495–499, October 1992.

[Higuchi *et al.*, 2000] Mikio Higuchi, Kazuhito Hatta, Junichi Takahashi, Kohei Kodaira, Hideaki Kaneda, and Junji Saito. Floating-zone growth of rutile single crystals inclined at 48 to the c-axis. *Journal of Crystal Growth*, 208(1):501–507, January 2000.

[Hurowitz *et al.*, 2017] J A Hurowitz, J Thieme, J Bai, E Dooryhee, E Fogelqvist, J Gregerson, K A Farley, and S Sherman. PREPARING FOR MARS SAMPLE RETURN: IN-SITU X-RAY DIFFRACTION MEASUREMENTS USING THE NATIONAL SYNCHOTRON LIGHT SOURCE-II AT BROOKHAVEN NATIONAL. page 2, 2017.

[Ihli *et al.*, 2017] J. Ihli, R. R. Jacob, M. Holler, M. Guizar-Sicairos, A. Diaz, J. C. da Silva, D. Ferreira Sanchez, F. Krumeich, D. Grolimund, M. Taddei, W. C. Cheng, Y. Shu, A. Menzel, and J. A. van Bokhoven. A three-dimensional view of structural changes caused by deactivation of fluid catalytic cracking catalysts. *Nature Communications*, 8(1), December 2017.

[Jacques *et al.*, 2011] Simon D. M. Jacques, Marco DiMichiel, Andrew M. Beale, Taha Sochi, Matthew G. O'Brien, Leticia EspinosaAlonso, Bert M. Weckhuysen, and Paul Barnes. Dynamic X-Ray Diffraction Computed Tomography Reveals Real-Time Insight into Catalyst Active Phase Evolution. *Angewandte Chemie International Edition*, 50(43):10148–10152, 2011.

[Jacques *et al.*, 2013] Simon D. M. Jacques, Marco Di Michiel, Simon A. J. Kimber, Xiaohao Yang, Robert J. Cernik, Andrew M. Beale, and Simon J. L. Billinge. Pair distribution function computed tomography. *Nature Communications*, 4:2536, September 2013.

[Janssen *et al.*, 2013] Yuri Janssen, Dhamodaran Santhanagopalan, Danna Qian, Miaofang Chi, Xiaoping Wang, Christina Hoffmann, Ying Shirley Meng, and Peter G. Khalifah. Reciprocal Salt Flux Growth of LiFePO $_4$ Single Crystals with Controlled Defect Concentrations. *Chemistry of Materials*, 25(22):4574–4584, November 2013.

[Jensen *et al.*, 2015] Kirsten M. Jensen, Xiaohao Yang, Josefa Vidal Laveda, Wolfgang G. Zeier, Kimberly A. See, Marco Di Michiel, Brent C. Melot, Serena A. Corr, and Simon J. L. Billinge. X-Ray Diffraction Computed Tomography for Structural Analysis of Electrode Materials in Batteries. *Journal of The Electrochemical Society*, 162(7):A1310–A1314, January 2015.

[Juhs *et al.*, 2013] P. Juhs, T. Davis, C. L. Farrow, and S. J. L. Billinge. PDFgetX3: a rapid and highly automatable program for processing powder diffraction data into total scattering pair distribution functions. *Journal of Applied Crystallography*, 46(2):560–566, April 2013.

[Keck and Golay, 1953] Paul H. Keck and Marcel J. E. Golay. Crystallization of Silicon from a Floating Liquid Zone. *Physical Review*, 89(6):1297–1297, March 1953.

[Kieffer and Wright, 2013] J. Kieffer and J. P. Wright. PyFAI: a Python library for high performance azimuthal integration on GPU. *Powder Diffraction*, 28(S2):S339–S350, September 2013.

[Koohpayeh *et al.*, 2013] S. M. Koohpayeh, J. J. Wen, M. Mourigal, S. E. Dutton, R. J. Cava, C. L. Broholm, and T. M. McQueen. Optical floating zone crystal growth and magnetic properties of MgCr2o4. *Journal of Crystal Growth*, 384:39–43, December 2013.

[Levin, 1913] I. H. Levin. Synthesis of Precious Stones. *Journal of Industrial & Engineering Chemistry*, 5(6):495–500, June 1913.

[LIGO,] Group LIGO. LIGO Technology.

[Loeliger *et al.*, 2012] T. Loeliger, C. Brnnimann, T. Donath, M. Schneebeli, R. Schnyder, and P. Trb. The new PILATUS3 ASIC with instant retrigger capability. In *2012 IEEE Nuclear Science Symposium and Medical Imaging Conference Record (NSS/MIC)*, pages 610–615, October 2012.

[Luckow *et al.*, 2018] Andr Luckow, Georgios Chantzialexiou, and Shantenu Jha. Pilot-Streaming: A Stream Processing Framework for High-Performance Computing. *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 177–188, 2018.

[Miao *et al.*, 1999] Jianwei Miao, Pambos Charalambous, Janos Kirz, and David Sayre. Extending the methodology of X-ray crystallography to allow imaging of micrometre-sized non-crystalline specimens. *Nature*, 400(6742):342–344, July 1999.

[Misture and Snyder, 2001] S. T. Misture and R. L. Snyder. X-ray Diffraction. In K. H. Jrgen Buschow, Robert W. Cahn, Merton C. Flemings, Bernhard Ilschner, Edward J. Kramer, Subhash Mahajan, and Patrick Veyssire, editors, *Encyclopedia of Materials: Science and Technology*, pages 9799–9808. Elsevier, Oxford, January 2001.

[Muiznieks *et al.*, 2015] Andris Muiznieks, Janis Virbulis, Anke Ldge, Helge Riemann, and Nico Werner. 7 - Floating Zone Growth of Silicon. In Peter Rudolph, editor, *Handbook of Crystal Growth (Second Edition)*, Handbook of Crystal Growth, pages 241–279. Elsevier, Boston, January 2015.

[Mrer *et al.*, 2018] Fredrik K. Mrer, Sophie Sanchez, Michelle lvarez Murga, Marco Di Michiel, Franz Pfeiffer, Martin Bech, and Dag W. Breiby. 3d Maps of Mineral Composition and Hydroxyapatite Orientation in Fossil Bone Samples Obtained by X-ray Diffraction Computed Tomography. *Scientific Reports*, 8(1), December 2018.

[Nikolaev *et al.*, 2016] Pavel Nikolaev, Daylond Hooper, Frederick Webber, Rahul Rao, Kevin Decker, Michael Krein, Jason Poleski, Rick Barto, and Benji Maruyama. Autonomy in materials research: a case study in carbon nanotube growth. *npj Computational Materials*, 2:16031, October 2016.

[Noyan and Cohen, 2013] Ismail C. Noyan and Jerome B. Cohen. *Residual Stress: Measurement by Diffraction and Interpretation*. Springer, March 2013. Google-Books-ID: EaVtCQAAQBAJ.

[Pablo *et al.*, 2019] Juan J. de Pablo, Nicholas E. Jackson, Michael A. Webb, Long-Qing Chen, Joel E. Moore, Dane Morgan, Ryan Jacobs, Tresa Pollock, Darrell G. Schlom, Eric S. Toberer, James Analytis, Ismaila Dabo, Dean M. DeLongchamp, Gregory A. Fiete, Gregory M. Grason, Geoffroy Hautier, Yifei Mo, Krishna Rajan, Evan J. Reed, Efrain Rodriguez, Vladan Stevanovic, Jin Suntivich, Katsuyo Thornton, and Ji-Cheng Zhao. New frontiers for the materials genome initiative. *npj Computational Materials*, 5(1):41, April 2019.

[Palancher *et al.*, 2011] Herv Palancher, Rmi Tucoulou, Pierre Bleuet, Anne Bonnin, Elonore Welcomme, and Peter Cloetens. Hard X-ray diffraction scanning tomography

with sub-micrometre spatial resolution: application to an annealed -U $_{0.85}$ Mo $_{0.15}$ particle. *Journal of Applied Crystallography*, 44(5):1111–1119, October 2011.

[Palmer *et al.*, 2018] Jeremy C. Palmer, Amir Haji-Akbari, Rakesh S. Singh, Fausto Martelli, Roberto Car, Athanassios Z. Panagiotopoulos, and Pablo G. Debenedetti. Comment on The putative liquid-liquid transition is a liquid-solid transition in atomistic models of water [I and II: J. Chem. Phys. 135, 134503 (2011); J. Chem. Phys. 138, 214504 (2013)]. *The Journal of Chemical Physics*, 148(13):137101, April 2018.

[Patterson, 1939] A. L. Patterson. The Scherrer Formula for X-Ray Particle Size Determination. *Physical Review*, 56(10):978–982, November 1939.

[Perez and Granger, 2007] F. Perez and B. E. Granger. IPython: A System for Interactive Scientific Computing. *Computing in Science Engineering*, 9(3):21–29, May 2007.

[Pollak, 1953] B. Pollak. Experiences with Planography* *From the Fort William Sanatorium, Fort William, Ontario, Canada. *Diseases of the Chest*, 24(6):663–669, December 1953.

[Prabhakaran *et al.*, 2003] D. Prabhakaran, F. R. Wondre, and A. T. Boothroyd. Preparation of large single crystals of ANb2o6 (A=Ni, Co, Fe, Mn) by the floating-zone method. *Journal of Crystal Growth*, 250(1):72–76, March 2003.

[Radon, 1986] J. Radon. On the determination of functions from their integral values along certain manifolds. *IEEE Transactions on Medical Imaging*, 5(4):170–176, December 1986.

[Ren *et al.*, 2017] Fang Ren, Ronald Pandolfi, Douglas Van Campen, Alexander Hexemer, and Apurva Mehta. On-the-Fly Data Assessment for High-Throughput X-ray Diffraction Measurements. *ACS Combinatorial Science*, 19(6):377–385, June 2017.

[Revcolevschi and Jegoudez, 1997] A. Revcolevschi and J. Jegoudez. Growth of large high-Tc single crystals by the floating zone method: A review. *Progress in Materials Science*, 42(1):321–339, January 1997.

[Robinson *et al.*, 2019] Niall H. Robinson, Joe Hamman, and Ryan Abernathey. Science needs to rethink how it interacts with big data: Five principles for effective scientific big data systems. *arXiv:1908.03356 [cs]*, August 2019. arXiv: 1908.03356.

[Rocklin, 2017] Matthew Rocklin. Streaming Python Prototype, April 2017.

[Roncallo *et al.*, 2010] Scilla Roncallo, Omeed Karimi, Keith D. Rogers, John M. Gregoire, David W. Lane, Jonathan J. Scragg, and Salman A. Ansari. High Throughput X-ray Diffraction Analysis of Combinatorial Polycrystalline Thin Film Libraries. *Analytical Chemistry*, 82(11):4564–4569, June 2010.

[Saito, 1986] M. Saito. Growth process of gas bubble in ruby single crystals by floating zone method. *Journal of Crystal Growth*, 74(2):385–390, February 1986.

[Sauter *et al.*, 2014] Nicholas K. Sauter, Johan Hattne, Aaron S. Brewster, Nathaniel Echols, Petrus H. Zwart, and Paul D. Adams. Improved crystal orientation and physical properties from single-shot XFEL stills. *Acta Crystallographica Section D Biological Crystallography*, 70(12):3299–3309, December 2014.

[Schmidt, 2014] Sren Schmidt. *GrainSpotter* : a fast and robust polycrystalline indexing algorithm. *Journal of Applied Crystallography*, 47(1):276–284, February 2014.

[scikit-beam team, 2019] scikit-beam team. Data analysis tools for X-Ray, Neutron and Electron sciences: scikit-beam/scikit-beam, July 2019. original-date: 2014-07-10T04:44:35Z.

[Selinger, 2018] Peter Selinger. English: Illustration of how to compute the 2-dimensional Radon transform in terms of several Fourier transforms., September 2018.

[Shi *et al.*, 2013] X. Shi, S. Ghose, and E. Dooryhee. Performance calculations of the X-ray powder diffraction beamline at NSLS-II. *Journal of Synchrotron Radiation*, 20(2):234–242, March 2013.

[Silva *et al.*, 2007] C. T. Silva, J. Freire, and S. P. Callahan. Provenance for Visualizations: Reproducibility and Beyond. *Computing in Science Engineering*, 9(5):82–89, September 2007.

[Smart, 2018] Ashley G. Smart. The war over supercooled water. August 2018.

[Somogyi *et al.*, 2005] A. Somogyi, R. Tucoulou, G. Martinez-Criado, A. Homs, J. Cauzid, P. Bleuet, S. Bohic, and A. Simionovici. ID22: a multitechnique hard X-ray microprobe beamline at the European Synchrotron Radiation Facility. *Journal of Synchrotron Radiation*, 12(2):208–215, March 2005.

[Sottmann *et al.*, 2017] Jonas Sottmann, Marco DiMichiel, Helmer Fjellvg, Lorenzo Malavasi, Serena Margadonna, Ponniah Vajeeston, Gavin B. M. Vaughan, and David S. Wragg. Chemical Structures of Specific Sodium Ion Battery Components Determined by Operando Pair Distribution Function and X-ray Diffraction Computed Tomography. *Angewandte Chemie International Edition*, 56(38):11385–11389, 2017.

[Sprouster *et al.*, 2017] D. J. Sprouster, T. Koyanagi, E. Dooryhee, S. K. Ghose, Y. Katoh, and L. E. Ecker. Microstructural evolution of neutron irradiated 3c-SiC. *Scripta Materialia*, 137:132–136, August 2017.

[Starr,] Michelle Starr. Less Than 1% of Large Hadron Collider Data Ever Gets Looked At.

[Tabor *et al.*, 2018] Daniel P. Tabor, Loc M. Roch, Semion K. Saikin, Christoph Kreisbeck, Dennis Sheberla, Joseph H. Montoya, Shyam Dwaraknath, Muratahan Aykol, Carlos Ortiz, Hermann Tribukait, Carlos Amador-Bedolla, Christoph J. Brabec, Benji Maruyama, Kristin A. Persson, and Aln Aspuru-Guzik. Accelerating the discovery of materials for

clean energy in the era of smart automation. *Nature Reviews Materials*, 3(5):5–20, May 2018.

[Takeya *et al.*, 2001] H Takeya, E Habuta, H Kawano-Furukawa, T Ooba, and K Hirata. Magnetization isotherms on ErNi2b2c, Er0.8tb0.2ni2b2c and Er0.8lu0.2ni2b2c single crystals. *Journal of Magnetism and Magnetic Materials*, 226-230:269–271, May 2001.

[Tanaka *et al.*, 1975] Takaho Tanaka, Eisuke Bannai, Shichio Kawai, and Tsuneko Yamane. Growth of high purity LaB6 single crystals by multi-float zone passage. *Journal of Crystal Growth*, 30(2):193–197, September 1975.

[Thayer *et al.*, 2017] J. Thayer, D. Damiani, C. Ford, M. Dubrovin, I. Gaponenko, C. P. OGrady, W. Kroeger, J. Pines, T. J. Lane, A. Salnikov, D. Schneider, T. Tookey, M. Weaver, C. H. Yoon, and A. Perazzo. Data systems for the Linac coherent light source. *Advanced Structural and Chemical Imaging*, 3(1), 2017.

[Theuerer, 1962] Henry C. Theuerer. Method of processing semiconductive materials, October 1962.

[Toby *et al.*, 2009] B. H. Toby, Y. Huang, D. Dohan, D. Carroll, X. Jiao, L. Ribaud, J. A. Doebbler, M. R. Suchomel, J. Wang, C. Preissner, D. Kline, and T. M. Mooney. Management of metadata and automation for mail-in measurements with the APS 11-BM high-throughput, high-resolution synchrotron powder diffractometer. *Journal of Applied Crystallography*, 42(6):990–993, December 2009.

[Tremsin *et al.*, 2017] Anton S. Tremsin, Didier Perrodin, Adrian S. Losko, Sven C. Vogel, Takenao Shinohara, Kenichi Oikawa, Jeff H. Peterson, Chang Zhang, Jeffrey J. Derby, Alexander M. Zlokapa, Gregory A. Bizarri, and Edith D. Bourret. In-Situ Observation of Phase Separation During Growth of Cs2lilabr6:Ce Crystals Using Energy-Resolved Neutron Imaging. *Crystal Growth & Design*, 17(12):6372–6381, December 2017.

[Vamvakeros *et al.*, 2016] A. Vamvakeros, S. D. M. Jacques, M. Di Michiel, P. Senecal, V. Middelkoop, R. J. Cernik, and A. M. Beale. Interlaced X-ray diffraction computed tomography. *Journal of Applied Crystallography*, 49(2):485–496, April 2016.

[van der Walt *et al.*, 2014] Stfan van der Walt, Johannes L. Schnberger, Juan Nunez-Iglesias, Franois Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in Python. *PeerJ*, 2:e453, June 2014.

[Walt *et al.*, 2011] S. van der Walt, S. C. Colbert, and G. Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering*, 13(2):22–30, March 2011.

[Wang *et al.*, 2015] Jiyang Wang, Guochun Zhang, Haohai Yu, Yan Wang, and Chuantian Chen. 5 - Czochralski and Flux Growth ofCrystals for Lasers andNonlinearOptics. In Peter Rudolph, editor, *Handbook of Crystal Growth (Second Edition)*, Handbook of Crystal Growth, pages 169–208. Elsevier, Boston, January 2015.

[Windus *et al.*, 2017] Theresa Windus, Michael Banda, Thomas Devereaux, Julia C. White, Katie Antypas, Richard Coffey, Eli Dart, Sudip Dosanjh, Richard Gerber, James Hack, Inder Monga, Michael E. Papka, Katherine Riley, Lauren Rotman, Tjerk Straatsma, Jack Wells, Tunna Baruah, Anouar Benali, Michael Borland, Jiri Brabec, Emily Carter, David Ceperley, Maria Chan, James Chelikowsky, Jackie Chen, Hai-Ping Cheng, Aurora Clark, Pierre Darancet, Wibe DeJong, Jack Deslippe, David Dixon, Jeffrey Donatelli, Thomas Dunning, Marivi Fernandez-Serra, James Freericks, Laura Gagliardi, Giulia Galli, Bruce Garrett, Vassiliki-Alexandra Glezakou, Mark Gordon, Niri Govind, Stephen Gray, Emanuel Gull, Francois Gygi, Alexander Hexemer, Christine Isborn, Mark Jarrell, Rajiv K. Kalia, Paul Kent, Stephen Klippenstein, Karol Kowalski, Hulikal Krishnamurthy, Dinesh Kumar, Charles Lena, Xiaosong Li, Thomas Maier, Thomas Markland, Ian McNulty, Andrew Millis, Chris Mundy, Aiichiro Nakano, A. M. N. Niklasson, Thanos Panagiotopoulos, Ron Pandolfi, Dula Parkinson, John Pask, Amedeo Perazzo,

John Rehr, Roger Rousseau, Subramanian Sankaranarayanan, Greg Schenter, Annabella Selloni, Jamie Sethian, Ilja Siepmann, Lyudmila Slipchenko, Michael Sternberg, Mark Stevens, Michael Summers, Bobby Sumpter, Peter Sushko, Jana Thayer, Brian Toby, Craig Tull, Edward Valeev, Priya Vashishta, V. Venkatakrishnan, C. Yang, Ping Yang, and Peter H. Zwart. Basic Energy Sciences Exascale Requirements Review. An Office of Science review sponsored jointly by Advanced Scientific Computing Research and Basic Energy Sciences, November 3-5, 2015, Rockville, Maryland. Technical report, US Department of Energy, Washington, DC (United States). Advanced Scientific Computing Research and Basic Energy Sciences, February 2017.

[Wright and Zhou, 2017] C. J. Wright and X.-D. Zhou. Computer-assisted area detector masking. *Journal of Synchrotron Radiation*, 24(2):506–508, March 2017.

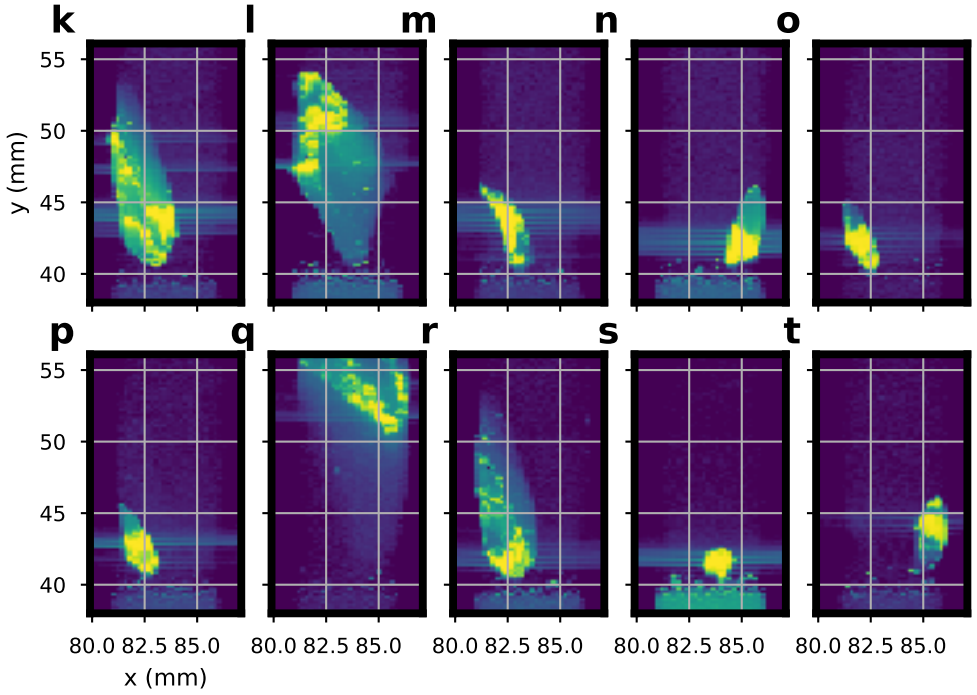
[Yager and Majewski, 2014] K. G. Yager and P. W. Majewski. Metrics of graininess: robust quantification of grain count from the non-uniformity of scattering rings. *Journal of Applied Crystallography*, 47(6):1855–1865, December 2014.

[Yang *et al.*, 2017] Yongsoo Yang, Chien-Chun Chen, M. C. Scott, Colin Ophus, Rui Xu, Alan Pryor, Li Wu, Fan Sun, Wolfgang Theis, Jihan Zhou, Markus Eisenbach, Paul R. C. Kent, Renat F. Sabirianov, Hao Zeng, Peter Ercius, and Jianwei Miao. Deciphering chemical order/disorder and material properties at the single-atom level. *Nature*, 542(7639):75–79, February 2017.

[Zaharia *et al.*, 2016] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM*, 59(11):56–65, October 2016.

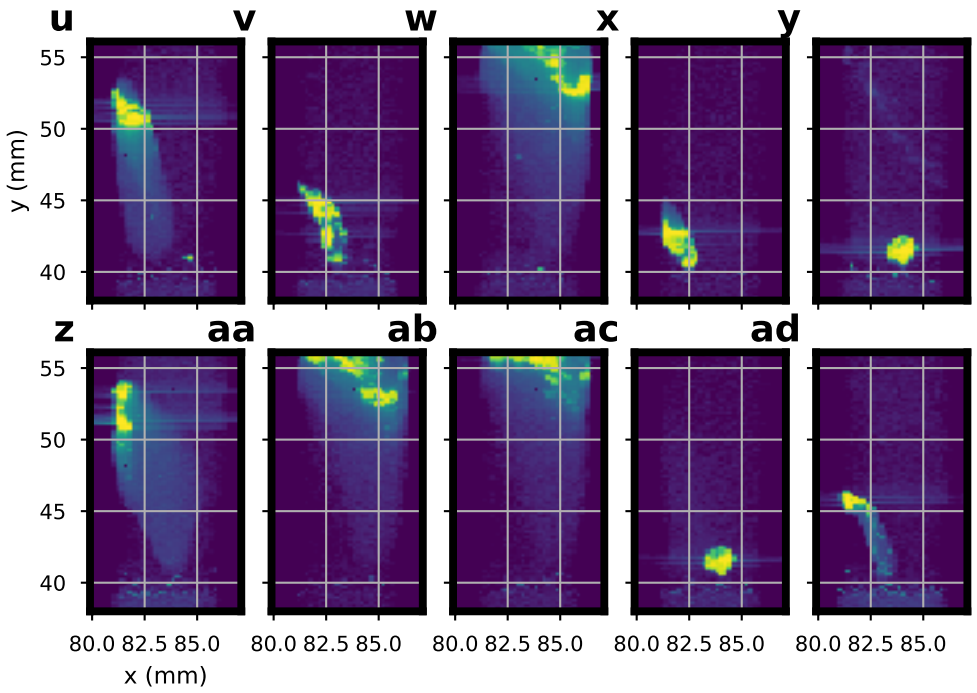
Appendix : Grain Maps

Boule A



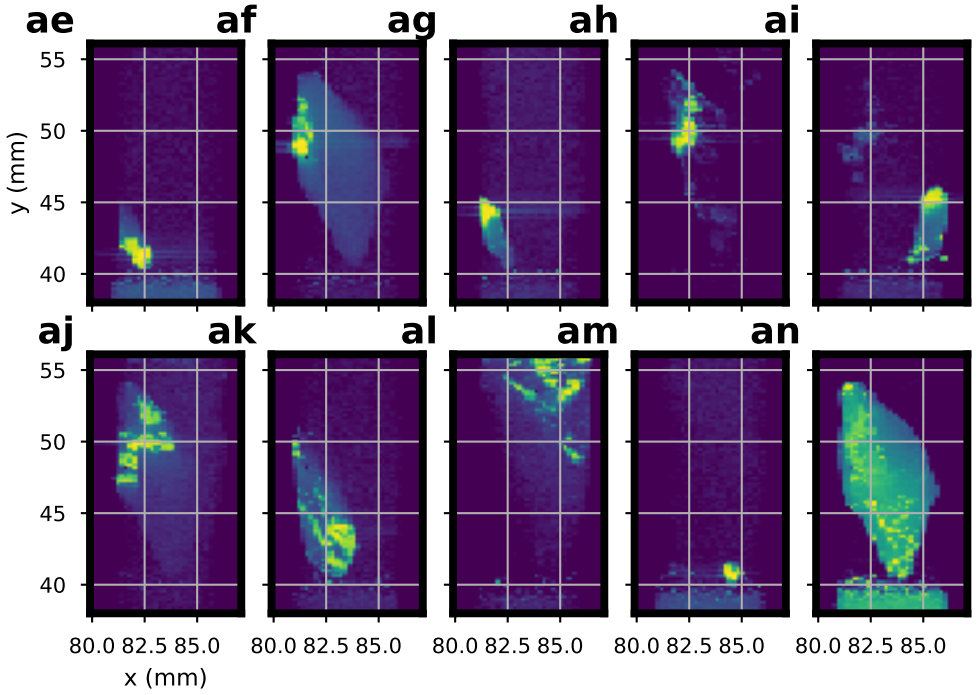
index	*hkl*	azimuthal angle (degrees)	expected *d*-spacing (Å)	Δ *d*-spacing (mÅ)
k	(5, 0, 1), (4, 3, 1)	-60.9	0.888	0.015
l	(3, 3, 1)	71.7	1.029	4.993
m	(5, 1, 3)	-107.4	0.671	1.809
n	(3, 3, 1)	51.8	1.029	4.272
o	(3, 3, 4)	122.8	0.615	1.657
p	(3, 3, 2)	64.6	0.882	-4.466
q	(7, 0, 1)	68.5	0.649	-1.531
r	(1, 0, 1)	124.4	2.503	-8.301
s	(2, 1, 1)	104.9	1.704	-16.342
t	(1, 0, 3)	-122.4	0.968	-6.476

Figure .1: Above: ROI intensity maps for Boule A, with spot size 5 pixels. These maps show separate crystal orientations as a function of position in the FZ grown crystal. Below: hkl and azimuthal angle assignments for Boule A



index	*hkl*	azimuthal angle (degrees)	expected *d*-spacing (Å)	Δ *d*-spacing (mÅ)
u	(4, 4, 2)	109.2	0.72	1.061
v	(2, 0, 2)	62.3	1.252	-5.85
w	(8, 2, 0)	-46.6	0.564	0.011
x	(4, 3, 4), (5, 0, 4)	67.8	0.58	-0.303
y	(3, 3, 0)	-18.8	1.097	-4.271
z	(2, 2, 3)	-152.4	0.848	-4.415
aa	(9, 1, 1)	49.6	0.506	-13.683
ab	(9, 1, 1)	50.8	0.506	-11.585
ac	(5, 3, 1)	119.5	0.771	-1.789
ad	(2, 0, 2)	-131.2	1.252	-8.792

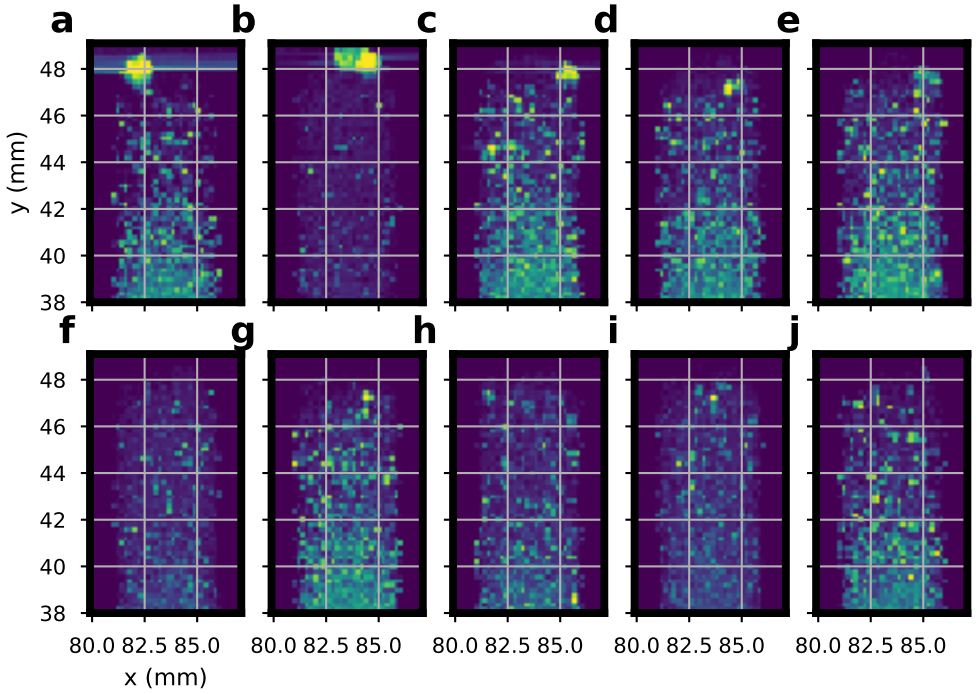
Figure .2: Above: ROI intensity maps for Boule A, with spot size 5 pixels. These maps show separate crystal orientations as a function of position in the FZ grown crystal. Below: hkl and azimuthal angle assignments for Boule A



index	*hkl*	azimuthal angle (degrees)	expected *d*-spacing (Å)	Δ *d*-spacing (mÅ)
ae	(4, 1, 2)	-160.7	0.898	-1.096
af	(7, 1, 0), (5, 5, 0)	88.1	0.658	-1.723
ag	(6, 2, 0)	-140.1	0.736	2.121
ah	(1, 1, 0)	36.2	3.29	659.087
ai	(1, 1, 2)	175.1	1.353	6.191
aj	(6, 3, 4)	51.0	0.507	-0.142
ak	(6, 1, 0)	77.8	0.765	2.112
al	(1, 1, 1)	39.0	2.204	-20.123
am	(5, 0, 1), (4, 3, 1)	-37.9	0.888	0.385
an	(1, 1, 0)	-45.6	3.29	-23.477

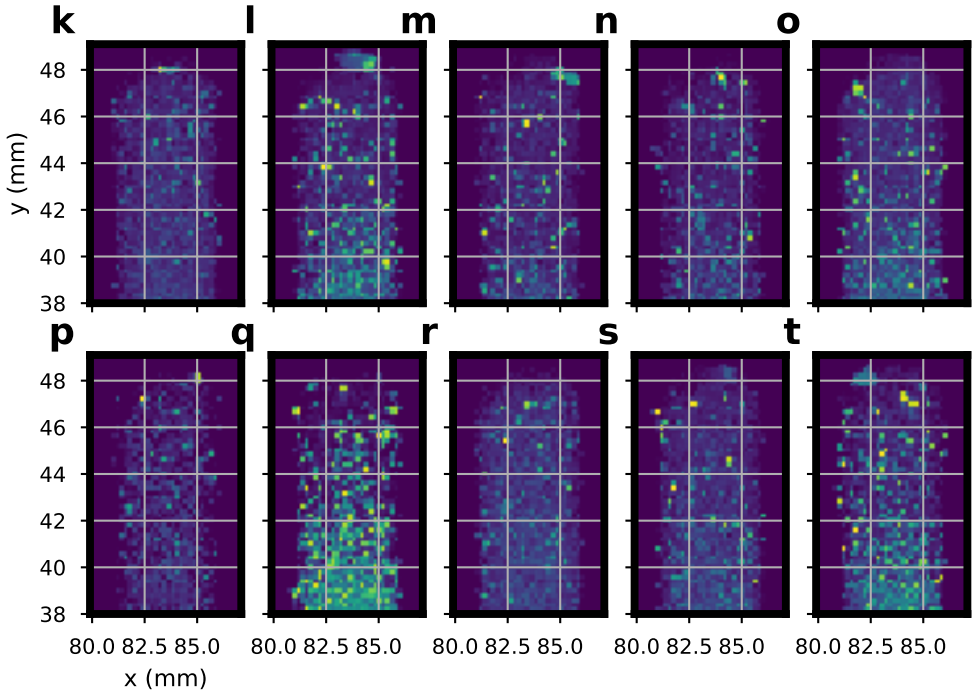
Figure .3: Above: ROI intensity maps for Boule A, with spot size 5 pixels. These maps show separate crystal orientations as a function of position in the FZ grown crystal. Below: hkl and azimuthal angle assignments for Boule A

Boule B



index	*hkl*	azimuthal angle (degrees)
a	(1, 1, 0)	40.8
b	(1, 0, 1)	146.4
c	(5, 4, 3)	-56.2
d	(3, 0, 1)	-80.9
e	(2, 1, 1)	-41.5
f	(2, 1, 1)	-147.7
g	(1, 0, 1)	-55.2
h	(2, 1, 1)	-24.6
i	(1, 0, 3)	156.2
j	(2, 1, 1)	63.7

Figure .4: Above: ROI intensity maps for Boule B. These maps show separate crystal orientations as a function of position in the FZ grown crystal. Below: hkl and azimuthal angle assignments for Boule B



index	*hkl*	azimuthal angle (degrees)
k	(3, 3, 2)	-120.4
l	(3, 3, 2)	-158.5
m	(2, 1, 1)	-143.5
n	(2, 1, 1)	-172.6
o	(1, 1, 1)	169.1
p	(1, 1, 2)	21.7
q	(2, 2, 0)	88.1
r	(4, 2, 0)	53.2
s	(3, 3, 1)	10.1
t	(1, 0, 1)	47.3

Figure .5: Above: ROI intensity maps for Boule B. These maps show separate crystal orientations as a function of position in the FZ grown crystal. Below: hkl and azimuthal angle assignments for Boule B