

## ABSTRACT

Title of dissertation: CHARACTERIZATION AND MODELING OF OFF-SPECULAR NEUTRON SCATTERING FOR ANALYSIS OF TWO DIMENSIONALLY ORDERED STRUCTURES

Christopher Jason Metting, Doctor of Philosophy, 2011

Dissertation directed by: Professor Robert Briber  
Department of Materials Science and Engineering

Off-specular neutron reflectometry is an instrumental technique which can be utilized for the characterization of thin-film systems in the depth and in-plane directions simultaneously. Currently, its use is limited both experimentally by the available neutron flux at modern neutron facilities and theoretically by a lack of widely available, user friendly, and open-source modeling software. This thesis describes work carried out on the development of a software package which utilizes currently available mathematical approximations to characterize model systems and evaluates the abilities and deficiencies of each algorithm. The evaluation will be carried out within the framework of a well-structured, object oriented, Python software package which is versatile and extendable. As new approximations and mathematical treatments are developed, they can be incorporated into the software infrastructure and tested with minimal effort.

We show that, at high  $q$ , the Born approximation can be used to qualitatively model off-specular scattering data; however, it does not capture any of the dynamic effects observed in real data. Some dynamical effects can be captured by perturbing the wavefunction by interactions with the substrate/incident media interface; however, low  $q$  scattering as well as scattering at the 'horizons' is still inaccurately represented. Currently, the best interpretation of the off-specular scattering can be accomplished with the complete distorted wave Born approximation. This is shown to produce theory functions which match quite well with scattering data.

Neutron coherence length is an important parameter in off-specular reflectometry as it dictates the number of feature periods being probed by the neutron beam. To determine the coherence length, a series of magnetic gratings were fabricated. Specular and off-specular measurements were used to evaluate the shape of the neutron wave packet work is still on going for a complete interpretation of these results.

CHARACTERIZATION AND MODELING OF  
OFF-SPECULAR NEUTRON SCATTERING FOR ANALYSIS OF  
TWO DIMENSIONALLY ORDERED STRUCTURES

by

Christopher Jason Metting

Dissertation submitted to the Faculty of the Graduate School of the  
University of Maryland, College Park in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
2011

Advisory Committee:  
Professor Robert Briber, Chair/Advisor  
Dr. Brian Maranville  
Professor John Cumings  
Professor Ichiro Takeuchi  
Professor Srinivasa Raghavan

# Table of Contents

List of Figures	iv
List of Abbreviations	viii
1 Introduction	1
1.1 Neutron Properties . . . . .	2
1.2 Neutron Production . . . . .	5
1.2.1 Reactor Sources . . . . .	6
1.2.2 Spallation Sources . . . . .	7
1.3 Non-magnetic Neutron Specular Reflectometry . . . . .	8
1.3.1 Scattering Length Density . . . . .	9
1.3.2 Formalism . . . . .	11
1.4 Magnetic Neutron Specular Reflectometry . . . . .	21
1.4.1 Polarizing and Analyzing the Neutron Beam . . . . .	21
1.4.2 Interpreting Magnetic Scattering Data . . . . .	27
1.5 Off-Specular Scattering . . . . .	29
1.6 Off-specular Neutron Reflectometry Sensitivity and Applications . . . . .	33
2 Software	36
2.1 Overview . . . . .	36
2.2 Software Workflow . . . . .	37
2.3 Unit_Cell . . . . .	38
2.3.1 3D Modeling Software . . . . .	40
2.3.2 Mathematical Form Factors . . . . .	46
2.3.3 Image Loading . . . . .	51
2.4 Lattice . . . . .	54
2.5 Beam . . . . .	56
2.6 Object Oriented MicroMagnetic Framework . . . . .	58
3 Theory and Approximations	60
3.1 Overview . . . . .	60
3.2 First Order Born Approximations Form Factor . . . . .	61
3.3 Structure Factor . . . . .	66
3.4 Refractive Shift . . . . .	68
3.5 Substrate Modified Born Approximation Form Factor . . . . .	71
3.6 Distorted Wave Born Approximation . . . . .	74
3.7 Magnetic First Order Born Approximation . . . . .	81
4 Sample Fabrication and Data Acquisition	83
4.1 Overview . . . . .	83
4.2 Non-magnetic system . . . . .	83
4.3 Magnetic system . . . . .	88
4.4 Magnetic Gratings . . . . .	89

4.5	Data Conversion . . . . .	94
5	Modeling . . . . .	96
5.1	Overview . . . . .	96
5.2	Shape Differentiation . . . . .	96
5.3	Modeling using the Born Approximation . . . . .	99
5.4	Modeling using the Substrate Modified Born Approximation . . . . .	102
5.5	Modeling using the Distorted Wave Born Approximation . . . . .	103
5.6	Modeling Magnetic Samples . . . . .	107
6	Coherence Length Determination . . . . .	114
6.1	Overview . . . . .	114
6.2	Effect of Transverse Coherence on Specular Reflection . . . . .	115
6.3	Results of the Specular Reflection Measurements . . . . .	117
8	Future Work . . . . .	121
8.1	Summary . . . . .	121
8.2	Future Work on Software . . . . .	121
8.3	Future Work on Modeling . . . . .	123
8.4	Future Work on Samples . . . . .	124
8.5	Future Work on Coherence Length . . . . .	124
7	Conclusion . . . . .	126
7.1	Summary . . . . .	126
7.2	Final Thoughts . . . . .	128
7.3	Acknowledgments . . . . .	129
A	Overview . . . . .	130
A.1	Wave Transfer Matrix Derivation . . . . .	130
B	Overview . . . . .	134
B.1	Integral Form of the Reflectometry Calculation Derivation . . . . .	134
B.2	Green's Function Reference . . . . .	138
C	Overview . . . . .	141
C.1	Combining the wave equations . . . . .	141
C.2	Magnetic scattering potential in terms of $Nb$ and $Np$ . . . . .	142
C.3	Magnetic transfer matrix derivation . . . . .	143
C.4	Transfer matrix elements . . . . .	146
C.5	Polarized neutron reflection and transmission identities . . . . .	147
D	Software Manual . . . . .	148
E	Software Diagram . . . . .	177
	Bibliography . . . . .	181

## List of Figures

1.1	Select relative elemental scattering potentials for x-rays and neutrons. The scattering potential scales with circle diameter. . . . .	2
1.2	Schematic of scattering length. The blue planes indicate a plane wave representation of the neutron beam. O is the scattering center interacting with the wave. The other parameters are defined to solve for the scattered wave shown in the equation. . . . .	9
1.3	Schematic of the specular reflectometry geometry. The angle of the incoming wavevector is the same as the outgoing wavevector, resulting in a wavevector transfer which only has a z component. The coordinate system in this figure is the system most commonly used for the reflection geometry. . . . .	12
1.4	Schematic of a 1D plane wave interacting with a potential barrier. The solution to the wavefunction is solved at each interface. . . . .	15
1.5	Example of specular reflectometry simulation showing one scattering potential with a semi-infinite substrate. Present are the interference fringes associated with the layer thickness and the critical edge. (Inset) a schematic of the depth profile represented by the model. . . . .	18
1.6	Schematic of how samples with constantly varying SLD can be treated [4]. The integration is accomplished through approximating the system as a collection of small slabs. The size of these slabs can effect the model results and $dL$ must be chosen carefully. . . . .	19
1.7	Schematic and theory calculation for how neutron polarizers work. . . . .	23
1.8	Schematic of magnetic neutron reflectometry measurement. . . . .	24
1.9	Schematic of a spin flipper. An in-depth description can be found below. . . . .	25
1.10	figure from reference [26]. . . . .	31
1.11	An evaluation of the size scales able to be probed by the specular and off-specular scattering. The maximum q values determine the minimal feature size that can be resolved. . . . .	35
2.1	Class structure outline for the software. . . . .	37
2.2	K3D output and processing for a model system. . . . .	40
2.3	An example of the mathematical form factors. The shapes are instantiated using the syntax above each shape. The 3D plot illustrates the matrix produced by the instantiation. Finally, the features are related to each other using the syntax in the pink boxes to orient the shapes appropriately. The final shape is a complicated mixture of different objects which was produced with 8 lines of code. . . . .	47
2.4	Illustrates how the curvature parameter, curve, is calculated. $c$ is the curvature parameter. . . . .	52

2.5	A) The original .png image which will be loaded into the unit cell. B)Schematic showing the image extension in the y direction, which is how it will be represented in the unit cell. C)Slice of the matrix created after loading. Not the x-y axis which are pixel counts. D) The matrix has been reduced in size through a rebinning process. . . .	53
2.6	A) the formula for a rectilinear lattice with a square envelope, the real space schematic and the resulting calculation in $q_x$ and $q_y$ . B) the formula for the calculation of a body centered lattice by repeating a phase shifted rectilinear lattice. . . . .	54
2.7	Shows how undersampling the q map can lead to erroneous calculations. . . . .	55
2.8	A) Structure factor calculation carried out A) at discrete q values and B) by integrating of the the q range for which each pixel is representing.	57
2.9	A) a .mif file created by our software loaded into the oommf software. B) a .omf file loaded from an oommf output file to our software and viewed using our slice viewing panel. . . . .	59
3.1	Simulations of a 3.7 m parallelepiped Au system on Si with feature height of 200nm in the a) Born Approximation and b) Distorted Wave Born Approximation. . . . .	61
3.2	The wave transfer as done for the specular case versus the Born approximation. . . . .	62
3.3	A pictorial representation of how the model is treated. . . . .	63
3.4	Demonstration of the different structure factor calculations. . . . .	67
3.5	Illustration of the refractive shift on a data set taken on gold pillars on the AND/R reflectometer. . . . .	68
3.6	A real data set with the refractive shift marked. . . . .	69
3.7	A real data set with the refractive shift marked. . . . .	71
3.8	A direct comparison of the BA and the SMBA theory function results for a model of square pillars. . . . .	73
4.1	The results from the first sample measured for off-specular scattering. A = 2778 : height from top of glass substrate to top of film. B = 200.0 : twice the amplitude of the wave. C = 8733 : wavelength. . . .	84
4.2	The general progression of the Au sample fabrication. A) A schematic of the sample. B) An optical microscopy image of the mask used to pattern the sample. C) The S1813 layer after UV exposure. D) Optical image of the final sample. . . . .	85
4.3	The data reduction process. . . . .	87
4.4	Data from the Au sample at A) $\phi = 0$ ,B) $\phi = 45$ ,C) $\phi = 30$ . . . . .	88
4.5	SEM images of the permalloy samples. . . . .	90
4.6	The coercivity and hysteresis of the permalloy sample. . . . .	90
4.7	Schematic of the lift-off process. . . . .	91
4.8	Width variation calculated for the $5\mu m$ grating on the A) mask and B) actual Ni grating. . . . .	93

4.9	AFM of the nanoimprinted sample. . . . .	94
4.10	The GUI used to load data. A) first, the data files are selected. B) Then the specific values needed to rebin the data to Q space is entered. C) Finally, the user can select a subset of the data that he or she wishes to model. . . . .	95
5.1	The shapes tested all filled the same volume of the unit cell. A) an xy cross-section of the shape being modeled. B) Real data and the theory functions calculated for each shape. C) An enlarged image of the third diffraction peak showing differences in intensity. . . . .	97
5.2	Theory function calculation using the Born Approximation. . . . .	101
5.3	Modeling using the Born Approximation with slices in both $q_x$ and $q_z$ . . . . .	101
5.4	Modeling using the substrate modified Born approximation. The green box marks the integrated area for the $q_x$ slice, the red box indicates the integrated area for the specular $q_z$ data, and the orange box is the integrated $q_z$ peak for the first order diffraction peak. . . . .	104
5.5	Off-specular scattering calculation within the DWBA. The specular scattering is not included in this calculation. The calculation does not include any disorder. The averaged slices show good agreement between theory and experimentation. . . . .	105
5.6	A) An SEM image of the features produced by the fabrication process. B) an optical image of the features. C) a slice of the scattering length density array used for modeling D) a slice of the energy minimized magnetic moment array as produced by the OOMMF software for the saturated state and imported into the off-specular software. . . . .	108
5.7	This models the data using the features illustrated in figure 5.6 with a feature height of $1100\text{\AA}$ . A) ++ magnetic cross-section of the permalloy features under saturating magnetic field. B) Unpolarized DWBA results. C) a slice of the first order diffraction peak. . . . .	109
5.8	Spin flip and non spin flip cross-section data for the permalloy samples and their corresponding theory functions. A) with a data floor to match the instrumental data range and B) no data floor. . . . .	111
5.9	A vertical slice of the two non spin flip cross-sections. The data is off-set for clarity. . . . .	112
5.10	A horizontal slice of the two non spin flip cross-sections. . . . .	113
6.1	A schematic of the instrumental geometry used for the coherence length measurements. Two coordinate systems are present in this work. The blue x, y, and z coordinate system refers to the coherence length projection onto the sample where the y axis refers to the perpendicular projection component and x refers to the parallel projection component(referenced to the scattering plane) The red $x'$ , $y'$ , and $z'$ coordinate system refers to the components of the coherences length where $y'$ is the transverse direction and $x'$ is the longitudinal direction. . . . .	116

6.2	Model results for the expected difference between coherent and incoherent scattering from a system of nickel gratings. . . . .	118
6.3	Data for three representative case. the 800 micron grating exhibits incoherent scattering, the 25 micron is approximately the coherent limit, and the 10 micron grating is mostly coherent scattering. . . .	120
8.1	An experiment which would allow for the study of magnetic coherence length independently from the structural component. A) Gratings magnetization aligned parallel with the neutron polarization B) Gratings magnetized perpendicular to the neutron polarization direction. . . . .	125
E.1	UML diagram of the main calculation components involved in the theory function calculation. . . . .	178
E.2	UML diagram of the unit building classes. These classes can all be used to build the finite element models from which scattering may be calculated. . . . .	179
E.3	The Shape classes available to build a Scene for modeling. . . . .	180

## List of Abbreviations

NIST	National Institute of Standards and Technology
NCNR	NIST Center for Neutron Research
ANDR	Advanced Neutron Diffractometer Reflectometer
CNST	Center for Nanoscale Science and Technology
UMD	University of Maryland
SNS	Spallation Neutron Source
ORNL	Oak Ridge National Lab
HFIR	High Flux Isotope Reactor
LANSCCE	Los Alamos Neutron Science Center
PSD	position sensitive detector
AFM	atomic force microscopy
SEM	scanning electron microscopy
WDS	wave dispersion spectroscopy
BA	Born approximation
SMBA	substrate modified Born approximation
DWBA	distorted wave Born approximation
SLD	scattering length density
SANS	small angle neutron scattering
GISAXS	grazing incidence small angle X-ray scattering
FFT	fast Fourier transform
CZT	chirp-z transform
GUI	graphical user interface
API	application program interface
OOMMF	object oriented micro magnetic framework
HMDS	hexamethyldisilazane

# Chapter 1

## Introduction

This thesis describes work carried out on furthering the development of off-specular neutron scattering data acquisition, reduction and analysis. Chapter 1 will discuss neutron reflectometry as it applies to modern materials analysis, the advantages of using neutrons as a characterization probe, specular reflectometry mathematical derivations, and the current state of off-specular reflectometry modeling. Chapter 2 will describe the software development approach for the reduction and modeling of off-specular neutron reflectometry data. It will focus primarily on computational challenges and software infrastructure development. Chapter 3 will cover the formalism used for the modeling algorithms. Chapter 4 describes the fabrication process used to create the standard non-magnetic and magnetic samples utilized in both the modeling and the neutron coherence length studies. Chapter 5 will show modeling results for the fabricated standard samples and will discuss capability, applicability, and limitations of each approximation. Chapter 6 will discuss the neutron beam coherence length studies which were carried out the NG1 reflectometer at the NIST Center for Neutron Research. Finally, Chapter 7 will discuss the further work needed to make off-specular neutron reflectometry a widely utilized characterization technique.

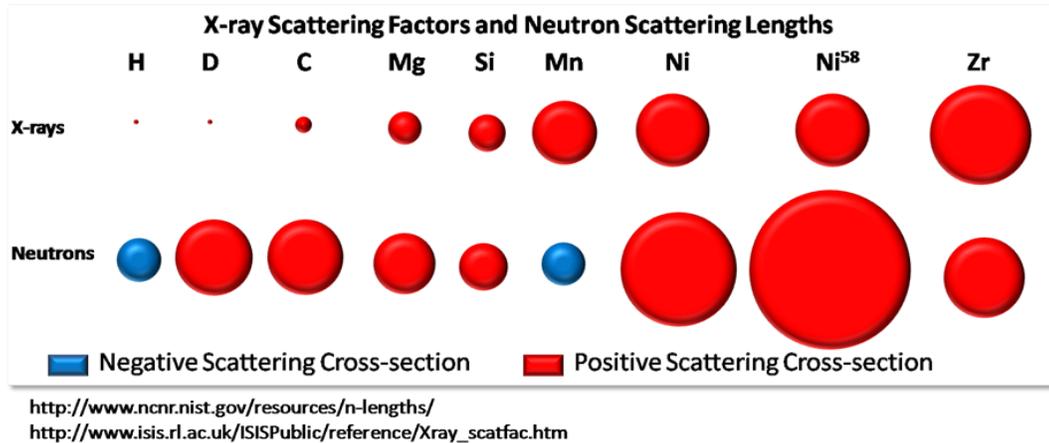


Figure 1.1: Select relative elemental scattering potentials for x-rays and neutrons. The scattering potential scales with circle diameter.

## 1.1 Neutron Properties

Neutrons may be used as a materials properties probe, providing many unique characterization abilities which cannot be achieved using other, more commonly used, probes such as X-ray and light scattering. Neutron scattering also has some limitations and complications which can result in significant challenges not inherent to other scattering probes (eg. low intensity, high facility costs, etc.). General knowledge of neutron scattering capabilities can help provide insight into what types of systems are best studied with a neutron probe. In practice, neutron scattering techniques are most commonly used in conjunction with other measurement techniques to elucidate sufficient information about a sample.

It is convenient to compare neutron scattering to X-ray scattering, although similar comparisons can be made with other scattering probes such as visible light. The most fundamental difference between neutron and X-ray probes is, while the

electromagnetic X-rays interact primarily with the electron cloud surrounding the atoms, the heavier neutrons interact with the atomic nuclei of a sample [9]. This difference in atomic interaction leads to some important consequences. First, by interacting with the nucleus, the elemental neutron scattering potential varies significantly as a function of atomic number. Figure 1.1 illustrates the relative scattering potentials for some select elements. With X-rays, the scattering potential is observed to be correlated with atomic number. This is not surprising as electron density also correlates closely with atomic number. With neutrons, scattering potential for the elements in figure 1.1 vary significantly even for atoms with similar atomic numbers. The figure also shows that even isotopes of the same element may have significantly different scattering potentials. This property is often exploited with hydrogen and deuterium isotopic substitution, which is routinely employed to intentionally manipulate contrast and simplify scattering profiles from complicated systems.

Due to this highly varying scattering potential across elements, neutrons are useful for seeing contrast between materials which would otherwise be difficult to differentiate between with other techniques [15]. It also allows for aspects of a sample to be highlighted using isotope exchange in select sample components [13]. For example, diblock copolymer systems are comprised of two polymer phases which, in general, have similar scattering potentials for most scattering probes. With neutron scattering, one of the phases can be deuterated to improve the contrast between the two polymer phases and provide a clearer scattering signal for determining the structure of the sample. In addition, isotopic solvent mixtures (eg.  $H_2O/D_2O$ ) may

be used to mask particular structures in a sample, allowing the neutron probe to effectively highlight specific aspects of a system [32]. A good example of this is in small angle neutron scattering (SANS) from core/shell micelles which are generally complicated to model. By varying the scattering potential of the solvent around the micelles to match the shell scattering potential, there will only be neutron scattering contrast between the micellar core and the surrounding media, providing a simple measurement of the micelle core diameter. For this type of contrast matching, mixing different volumetric ratios of  $H_2O$  ( $\rho = -0.0056 * 10^{-12} cm\text{\AA}^{-3}$ ) and  $D_2O$  ( $\rho = .06404 * 10^{-12} cm\text{\AA}^{-3}$ ) can produce a solvent of any scattering potential between the scattering length density (SLD) values of the pure components and can be calculated by:

$$\rho_{eff} = \rho_a w_a + \rho_b w_b \quad (1.1)$$

where  $w$  is the volume percent of solvent  $a$  and solvent  $b$  respectively [9]. The exact meaning of SLD is defined more clearly in section 1.3. To clarify, for a solvent with a scattering contrast of  $0.0 cm\text{\AA}^{-3}$ , one would use a solution of 92%  $H_2O$  and 8%  $D_2O$  [9].

The neutrons zero charge moment also provides neutron scattering techniques with additional capabilities. Because of their spin, neutrons can be used to probe a sample's structural and magnetic properties. This interaction makes neutrons incredibly valuable for elucidating information about a sample's magnetic characteristics [4]. How this interaction relates to reflectometry measurements will be

described in section 1.4. Finally, because of the weak nuclear scattering, neutrons can generally penetrate deep into condensed matter and probe a large sample volume [9].

The aforementioned properties indicate where neutron scattering can uniquely contribute to the understanding of materials properties. By utilizing these abilities, neutron scattering has contributed to advances in a wide variety of fields and continues to contribute heavily in important research areas such as hydrogen storage [27], fuel cells [33], solar cells [15]), battery technology [3], and computer memory [16] to name just a few.

## 1.2 Neutron Production

Unlike X-rays, where lab scale equipment can be readily acquired, neutron experiments must be carried out at facilities which are equipped with the ability to produce large fluxes of neutron radiation. This production is both complicated and expensive which limits access to instrumentation. Still, a number of facilities exist in the United States including the NIST Center for Neutron Research (NCNR) at the National Institute of Standards and Technology (NIST), the High Flux Isotope Reactor (HFIR) and the Spallation Neutron Source (SNS) at Oak Ridge National Lab, and the Los Alamos Neutron Science Center (LANSCE) at Los Alamos National Lab. Other, smaller sources also exist. These facilities do not all produce neutrons in the same way and an understanding of the neutron production method is important for data reduction and interpretation. There are two main neutron

production methods currently in use. The first is a reactor source and the second is a spallation source.

### 1.2.1 Reactor Sources

The NCNR and the HFIR are currently the two largest reactor sources operating in the United States. Reactor sources produce neutrons through the fission of  ${}_{92}^{235}\text{U}$  [23]. The net reaction can be written as:



where 0.5 of the neutrons are lost due to neutron absorption [9]. Reactor sources run at a constant power and the neutrons produced are very high energy [9]. To use the neutrons from this fission reaction in scientific instrumentation, they must be thermalized with a moderator material [9]. Some of the most common moderator materials are  $\text{H}_2\text{O}$ ,  $\text{D}_2\text{O}$ , graphite, or beryllium [9]. Different instrumentation requires different levels of moderations. Table 1 shows the classification scheme for neutrons and their corresponding characteristics. Once the neutron reaches the instrument, the beam still has a large distribution of energies and requires a monochromator to narrow the energy spread of the neutron [2].

Reactor sources can have much higher, time-averaged intensities than spallation sources; however, their maximum neutron flux is limited to  $10^{15}$  *neutrons*  $\text{cm}^{-2}\text{s}^{-1}$  due to reactor cooling requirements [2] [9]. In fact, the physical limitations presented by the heat production will most likely prevent any further increase

Table 1.1: Neutron classification and characteristics [2]

Neutron Classification	Energy(meV)	Velocity(m/s)	$\lambda$ (nm)
Ultra-cold	0.00025	6.9	57
Cold	1	437	0.9
Thermal	25	2,187	0.18
Epithermal	1,000	12,832	0.029

in neutron flux from reactor sources [9]. Higher flux production will have to be accomplished through the development of spallation sources.

### 1.2.2 Spallation Sources

In the United States, there are two main spallation neutron sources. The first is at Los Alamos Neutron Science Center (LANSCE) and the second is the relatively new Spallation Neutron Source (SNS). In this type of source, pulses of  $H^-$  ions are produced and accelerated down a linac at high energies where they collide with a target, releasing or spallating neutrons [9]. Most spallation sources utilize the time-of-flight (ToF) method of operating. In this configuration the neutron energy is determined by its location in the pulse distribution, which is a function of time from pulse genesis [2]. When the ion pulse interacts with the target, neutrons with a large distribution of energies are released [2]. As these neutrons are guided toward the instrumentation, the higher energy neutrons move faster than the slower energy neutrons, increasing the neutron spread [2]. Consequently, instrumental resolution

is highly dependent on flight path distance [2]. Moderators can also be used to alter the neutron spread and are generally used in the same way as for reactor sources.

The spallation sources biggest advantage are their low heat generation per neutron, allowing for a much higher peak neutron flux than reactor sources[2] [9]. Some instruments also gain notable advantages from ToF operation [2]. Depending on the operational configuration of the source (i.e. long versus short pulse sources), the full energy distribution may be utilized for more efficient data acquisition [8].

### 1.3 Non-magnetic Neutron Specular Reflectometry

In this section, the general physics and application of specular reflectometry is discussed. An understanding of the specular technique is important for determining how to extend reflectometry to the off-specular regime. It is important to note that, due to the nature of the measurement technique, reflectometry does not measure sample properties directly but rather infers the properties through an iterative process of calculating theory functions for a given representative system, or model, and then comparing the results to real data (*see 1.3.2 for explanation*). This can make the technique quite challenging to implement; however, with modern fitting software, the technique can be routinely used to determine compositional information about scientifically relevant systems.

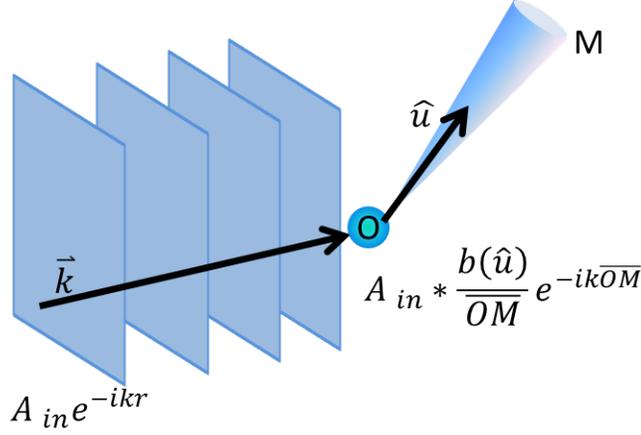


Figure 1.2: Schematic of scattering length. The blue planes indicate a plane wave representation of the neutron beam. O is the scattering center interacting with the wave. The other parameters are defined to solve for the scattered wave shown in the equation.

### 1.3.1 Scattering Length Density

In neutron reflectometry, the scattering contrast of a system is defined through the use of a scattering length density (SLD). The SLD is defined as [4]:

$$\rho = \sum_{j=1}^M N_j b_j \quad (1.3)$$

where  $b$  is the coherent scattering length of the atomic isotope,  $N$  is the number density of the isotope and  $M$  is the number of isotopes present in the material [4]. Essentially, this formula states that the scattering potential of a material is the sum of the scattering potentials of each individual component that makes up that material.

The scattering length is a measure of the scattering power. Figure 1.2 shows

a schematic of scattering length [5]. It illustrates that when a plane wave interacts with an object, the object will scatter the plane wave, which changes its amplitude [5]. The magnitude of this change in amplitude is dependent on the coefficient,  $b(\hat{u})$  [5]. In practice, values for the elemental scattering length have been experimentally measured and tabulated and can be found in many references [10]. It is important to note that, in the case of highly absorbing materials, the scattering length will have a non-trivial imaginary component [5]. Generally, neutron absorption is quite small and the imaginary component is negligible [5].

To understand the scattering produced from reflectometry we start with the Schrodinger wave equations:

$$\left[ -\frac{\hbar^2}{2m} \nabla^2 + V(\vec{r}) \right] \Psi = E\Psi \quad (1.4)$$

where the red portion of the equation is the kinetic energy of the wave, the blue is the potential energy and  $m$  is the neutron mass; simply stating the kinetic energy plus the potential energy is equal to the total energy [4]. For a wave in a vacuum (which we assume is outside of the sample) [4]:

$$V(\vec{r}) = 0 \quad (1.5)$$

so that all of the energy is kinetic and is given by:

$$E_0 = \frac{1}{2}mv_0^2 = \frac{\hbar^2 k_0^2}{2m} \quad (1.6)$$

The potential energy, after entering a new medium, is dependent on the scattering length density and can be written as [4]:

$$V = \frac{2\pi\hbar^2}{m}Nb = \frac{2\pi\hbar^2}{m}\rho \quad (1.7)$$

This means that the total energy inside a medium can be written as:

$$E = \frac{\hbar^2 k^2}{2m} + \frac{2\pi\hbar^2}{m}\rho \quad (1.8)$$

Through the conservation of energy it is necessary that:

$$E_0 = E \quad (1.9)$$

which allows us to reduce the previous equation 1.9:

$$\frac{\hbar^2 k_0^2}{2m} = \frac{\hbar^2 k^2}{2m} + \frac{2\pi\hbar^2}{m}\rho \quad (1.10)$$

to give:

$$k_0^2 = k^2 + 4\pi\rho \quad (1.11)$$

The refractive index for neutrons can be written [4] [5]:

$$n(k_0) \equiv \sqrt{1 - 4\pi\rho/k_0^2} \quad (1.12)$$

which can alternately be written as  $k = nk_0$ .

### 1.3.2 Formalism

Specular reflectometry is a small angle scattering technique which is traditionally used to determine the scattering length density depth profile of thin film

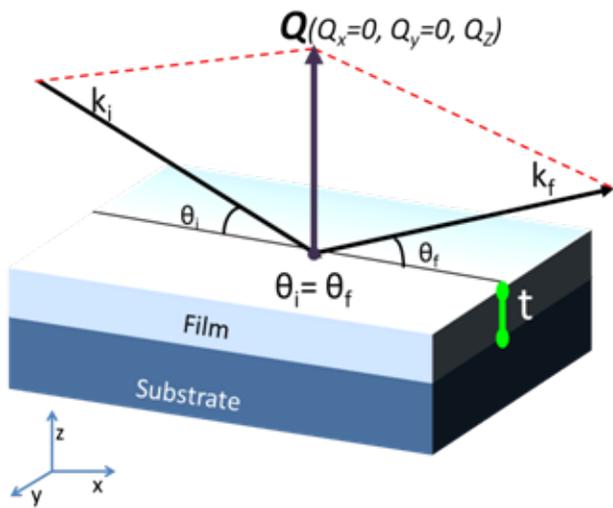


Figure 1.3: Schematic of the specular reflectometry geometry. The angle of the incoming wavevector is the same as the outgoing wavevector, resulting in a wavevector transfer which only has a  $z$  component. The coordinate system in this figure is the system most commonly used for the reflection geometry.

structures. This depth profile is then interpreted as a compositional profile through some knowledge of the initial composition and the processes being studied. When a sample (i.e. thin-film structures) has no in-plane variations in scattering length density the scattering is completely specular in nature and the wavevector transfer only occurs in the  $q_z$  direction. The geometry to measure such scattering is depicted in Figure 1.3.  $k_i$  is the incoming wave vector and  $k_f$  is the outgoing wave vector which leads to the scattering vector,  $\vec{Q}$ . The scattering vector can be written mathematically as [5]:

$$\vec{Q} = k_f - k_i \quad (1.13)$$

where the incoming wave vector is [5]:

$$k_i = \frac{2\pi}{\lambda} \quad (1.14)$$

and  $\lambda$  is the wavelength of the neutron beam.

In this geometry,  $\theta_i = \theta_f$  which means the vector,  $\vec{Q}$ , only has a z component, where z is the depth direction of the sample. This can be calculated by [5]:

$$q_z = \frac{4\pi}{\lambda} \sin(\theta_i) \quad (1.15)$$

The physics used to describe a reflectometry experiment is based on the quantum mechanical description of a plane wave interacting with a potential barrier. If it is assumed that the in-plane (x-y plane) structure is uniform, then there is no perturbation in the waves momentum in the plane of the film and the problem

simplifies significantly [4].

We can start by using the wave equation 1.4 and substituting the kinetic 1.6 and potential 1.7 energies to give:

$$[\nabla^2 + k^2]\Psi = 0 \quad (1.16)$$

which is comprised of:

$$\Psi_{\vec{r}} = \Psi_{x,y,z} = \phi_x \phi_y \phi_z \quad (1.17)$$

If the wave impinging on the sample is approximated as a plane wave, then the wavefunction is given by [4]:

$$\Psi_{\vec{r}} = e^{i\vec{k}\cdot\vec{r}} = e^{i(k_x x + k_y y + k_z z)} = e^{ik_x x} e^{ik_y y} e^{ik_z z} \quad (1.18)$$

Because the wavefunction can be broken down into its individual components, 1.11 may be written as:

$$k_x^2 + k_y^2 + k_z^2 + 4\pi\rho = k_{0x}^2 + k_{0y}^2 + k_{0z}^2 \quad (1.19)$$

Because it is assumed that there is no variation in scattering contrast in the plane of the film, there can be no wavevector transfer in either the x or y direction. Mathematically, this means  $k_x = k_{x0}$  and  $k_y = k_{y0}$  leaving only the z component [4]:

$$k_z^2 + 4\pi\rho = k_{0z}^2 \quad (1.20)$$

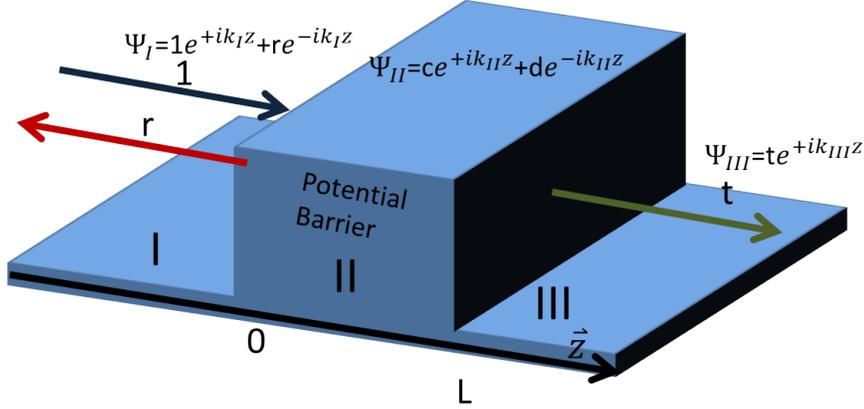


Figure 1.4: Schematic of a 1D plane wave interacting with a potential barrier. The solution to the wavefunction is solved at each interface.

To describe the reflected wave, equation 1.6 and 1.18 can be combined to form [4]:

$$\Psi_{vec_r} = e^{k_{0x}x} e^{k_{0y}y} \psi_z \quad (1.21)$$

Because the previous derivation shows that the reflected wave for a system of uniform scattering density in the plane of the film can be reduced to a one dimensional problem, the wave equation can be written as [4]:

$$\left[ \frac{\delta^2}{\delta z^2} + k_{0z}^2 - 4\pi\rho_z \right] \psi_z = 0 \quad (1.22)$$

This reduction to a one dimensional problem allows the system to be treated as a one dimensional plane wave impinging on a potential barrier, a problem that is easily treated[4]. The process is illustrated in figure 1.4 and can be found in introductory quantum mechanics books. In this schematic, the plane wave approaches a potential barrier with a normalized intensity of 1. When the wave hits the potential

barrier, some of the intensity ( $r$ ) is reflected from the interface and some is transmitted ( $t$ ) through the interface[4]. The wave equations for these scenarios are also shown in figure 1.4. Some absorption will occur during this process; however, the number of absorbed neutrons is small and will be left out for the sake of simplicity. Because energy and momentum are conserved in this process, we can assume that both the wavefunctions and their derivatives are equal or [4]:

$$1 + r = c + d \quad (1.23)$$

$$\frac{k_I}{k_{II}}(1 - r) = c - d \quad (1.24)$$

$$ce^{ik_{II}L} + de^{-k_{II}L} = te^{+k_{III}L} \quad (1.25)$$

$$ce^{ik_{II}L} - de^{-k_{II}L} = \frac{k_{III}}{k_{II}}te^{+k_{III}L} \quad (1.26)$$

where  $L$  is the thickness of the potential barrier. These formulas lead to the matrix:

$$\begin{pmatrix} t \\ ik_{III}t \end{pmatrix} e^{+k_{III}L} = \begin{pmatrix} \cos(k_{II}L) & \sin(k_{II}L)/k_{II} \\ -k_{II} \sin(k_{II}L) & \cos(k_{II}L) \end{pmatrix} \begin{pmatrix} 1 + r \\ ik_I(1 - r) \end{pmatrix} \quad (1.27)$$

The full derivation of this matrix may be found in appendix A.

The equations presented so far have an explicit dependence on  $\vec{k}$ ; however, we can also write these same equations as having a dependence on  $\vec{Q}$  using the relation shown in 1.13[4]. Because we are discussing specular reflectometry where the outgoing wavevector is the same magnitude as the incoming wavevector but the  $z$  component of the vector is reversed, this relationship means that the wave vector

transfer may be written as: [4]

$$k_f = -k_0 \quad (1.28)$$

$$\vec{Q} = k_0 - (-k_0) \quad (1.29)$$

$$\vec{Q} = 2k_0 \quad (1.30)$$

In addition, because the scattered data is a measure of the neutron beam intensity reflected off of a sample at a given position in space (the detector position), information corresponding to the phase of the neutron is lost. Mathematically, this means that instead of measuring the reflection as calculated by the transfer function, it is instead [4]:

$$I = |r|^2 \quad (1.31)$$

The consequence of this will be explained elsewhere; however, these two equations allow for the direct understanding of how the transfer function relates to the most common method for visualizing reflectometry data. In general, scattering data is plotted as  $I$  versus  $Q$  where  $I$  is the log scale intensity and  $Q$  is the  $Q_z$  inverse space.

For a single film on a substrate, the interference fringe, sometimes referred to as Kiessig fringe, spacing is  $\frac{2\pi}{\Delta t}$  where  $t$  is the thickness of the film layer[4]. The critical edge is the point at which reflection changes from total external reflection (ie. the full intensity if reflected) to penetrating the sample[4]. This phenomenon can be explained by equation 1.12 which shows that, if  $k_0^2 < 4\pi\rho_z$  the right hand side

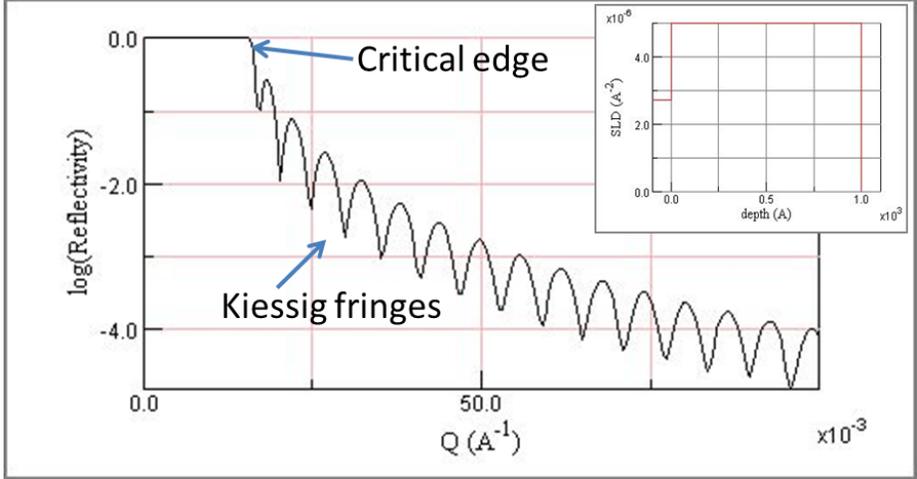


Figure 1.5: Example of specular reflectometry simulation showing one scattering potential with a semi-infinite substrate. Present are the interference fringes associated with the layer thickness and the critical edge. (Inset) a schematic of the depth profile represented by the model.

of the square root becomes  $> 1$  and  $n$  becomes purely imaginary [4]. This means the critical edge is at[4]:

$$Q_c = k_{0z}^2 = 4\pi\rho_z \quad (1.32)$$

Although the derived theory solves exactly the case of a single layer with sharp interfaces, most experimentally relevant systems are not isolated, sharply varying layers. To treat cases where multiple layers or non-sharp interfaces are involved, some extra steps are needed. Fortunately, the same matrix derived previously can be used to transfer the beam through multiple interfaces. Figure 1.6 shows a somewhat arbitrary case of a system with strongly varying scattering contrast[4]. The system can be estimated as a collection of sharply interfaced micro-slabs, which allows for

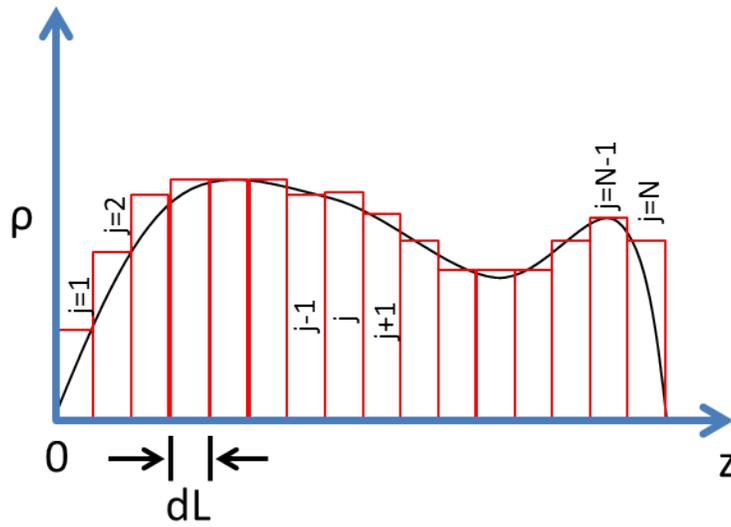


Figure 1.6: Schematic of how samples with constantly varying SLD can be treated [4]. The integration is accomplished through approximating the system as a collection of small slabs. The size of these slabs can effect the model results and  $dL$  must be chosen carefully.

the direct application of the transfer matrix [4]. If the matrix is represented as[4]:

$$M = \begin{pmatrix} \cos(k_{II}L) & \frac{1}{k_{II}} \sin(k_{II}L) \\ -k_{II} \sin(k_{II}L) & \cos(k_{II}L) \end{pmatrix} \quad (1.33)$$

Then, starting from the bottom of the stack, the wave may be propagated out toward the detector by [4]:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} = M_N M_{N-1} \cdots M_j \cdots M_2 M_1 \quad (1.34)$$

This matrix holds all of the information required to calculate the reflected intensity.

The reflectivity amplitude coefficients can be calculated as [9]:

$$r = \frac{B + C + i(D - A)}{B - C + i(D + A)} = \frac{B^2 + D^2 - A^2 - C^2 - 2i(AB + CD)}{A^2 + B^2 + C^2 + D^2 + 2} \quad (1.35)$$

Alternatively, the scattering may be written as [9]:

$$2 \frac{1 + |r|^2}{1 - |r|^2} = A^2 + B^2 + C^2 + D^2 \quad (1.36)$$

Determining a size for  $dL$  in figure 1.6 is not only important for specular but also off-specular reflectometry as the modeling for both scattering techniques is dependent on a discretized representation of the experimental system. Because  $\vec{Q}$  is in reciprocal space and  $L$  is a real space value, the measurement's sensitivity to  $L$  is dependent on the maximum measured  $\vec{Q}$  value. Figure 1.5 can be used as an example. If the theory function in this figure is estimated as a data set, the data

obtained would extend to  $0.1\text{\AA}^{-1}$ . This means that the the best resolution that can be recovered in real space is[4]:

$$\Delta t = \frac{2\pi}{\vec{Q}_{max}} = \frac{2\pi}{0.10\text{\AA}^{-1}} = 62.8\text{\AA} \quad (1.37)$$

So, when modeling this system, the maximum  $\Delta L$  is  $62.8\text{\AA}$ .

## 1.4 Magnetic Neutron Specular Reflectometry

Because neutron scattering has the ability to probe the magnetic properties of thin film samples as a function of depth, it is important to discuss the fundamental concepts involved in modeling these effects. The instrumental setup required to measure magnetic scattering in neutron reflectometry is somewhat more complicated than non-magnetic scattering, requiring additional components to prepare the neutron beam so that the initial neutron spin state is known and the finally spin state can be analyzed. This section will discuss the mathematical and physical principles by which these instrumental components operate, followed by how a magnetic sample interacts with the prepared neutron beam and how this interaction can be interpreted in the scattering data.

### 1.4.1 Polarizing and Analyzing the Neutron Beam

A schematic of the process is shown in 1.8. The neutron source produces both spin-up ( $\uparrow$  or  $+$ ) and spin-down ( $\downarrow$  or  $-$ ) neutrons which can interact with a magnetic field inside a thin film sample[4]. To measure the proportion of neutrons

which change their polarization after interacting with a magnetic sample to those which did not, the incoming beam must first be prepared so that only a single spin state impinges on the sample. This is accomplished through the use of a neutron spin state selector, or polarizer.

To select a single spin state, the polarizer uses the bias that the neutron's spin state has when interacting with a magnetic material. As will be discussed in more detail later, magnetic materials may be thought of as having two separate scattering potentials. The first is the structural scattering potential which was discussed in 1.3. The second is the magnetic scattering potential. The formula for the refractive index which includes the magnetic scattering potential can be written as[4]:

$$n_{\pm} = \sqrt{1 - 4\pi(\rho_N \pm \rho_M)/k_0^2} \quad (1.38)$$

which, in the case of specular reflectometry, reduces to only include the z component[4]:

$$n_{z\pm} = \sqrt{1 - 4\pi(\rho_N \pm \rho_M)/k_{0z}^2} \quad (1.39)$$

The neutron's interaction with this potential is dependent on the spin state. This difference in scattering potential leads to individual critical edges for each of the two spin states at different  $Q$  values. Using equation 1.32 and the magnetic scattering potential, the critical edges for the two spin states can be calculated with[4]:

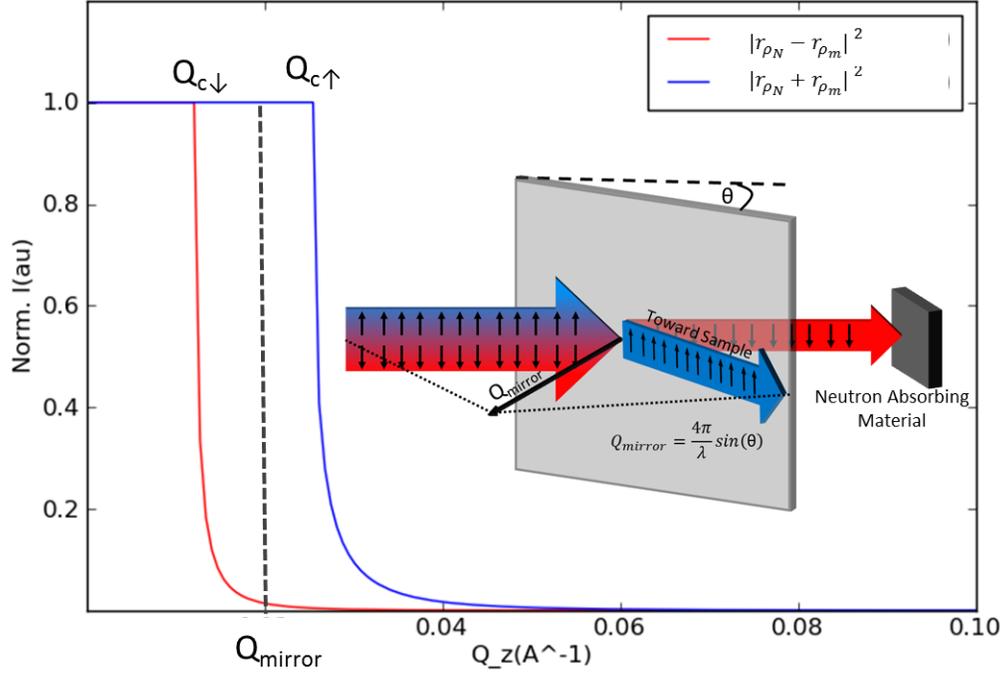


Figure 1.7: Schematic and theory calculation for how neutron polarizers work.

$$Q_{c\uparrow} = 4\pi(\rho_N + \rho_M) \quad (1.40)$$

$$Q_{c\downarrow} = 4\pi(\rho_N - \rho_M) \quad (1.41)$$

When an angle of reflection is chosen so that the  $Q$  vector falls between these two critical edges, the spin-up state will be totally reflected while the spin-down state will be scattered[4]. In the case of the iron polarizer shown in figure 1.7, the spin-up neutrons are reflected down the instrument toward the sample, while the spin-down neutrons are transmitted into a neutron absorbing materials to reduce its contribution to the background signal[4].

In this way, spin-up neutrons can be separated from the spin down neutrons

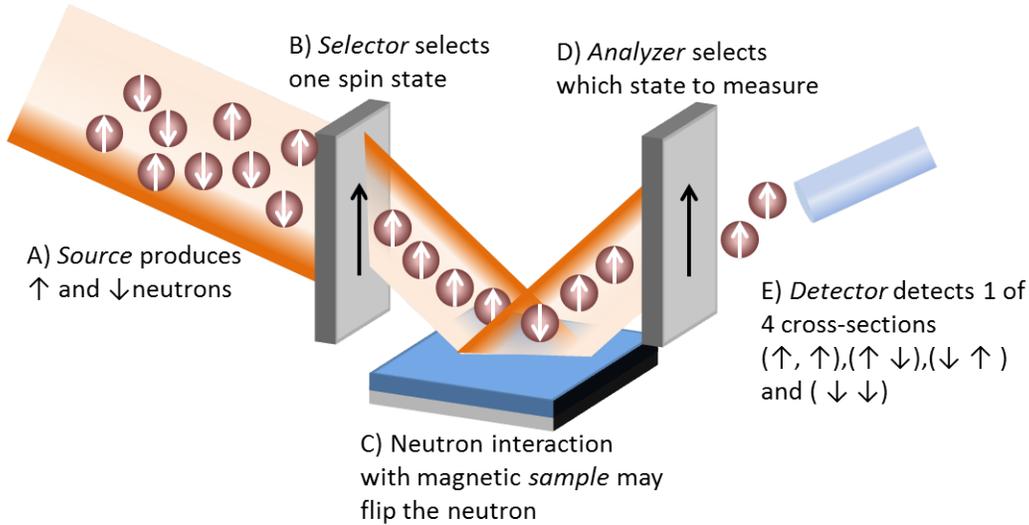


Figure 1.8: Schematic of magnetic neutron reflectometry measurement.

with efficiencies of up to 98%[4][1]. Once the spin-up neutrons are selected, they are reflected off of the sample. The reflected neutrons have a probability of changing their spin state depending on the sample's in-plane magnetization (the in-plane dependence is due to the specular geometry). This change can be determined using the same process which was used to polarize the beam. An analyzer selects the spin-up neutrons and reflects them toward the detector. This process is illustrated in figure 1.8[4].

Unfortunately, this only gives a quarter of the information needed to completely analyze the magnetic scattering. To understand and model the in-plane magnetization, data for both the flipped and non-flipped neutrons is needed. This leads to four different magnetic cross-sections, referred to as  $\downarrow\downarrow$ ,  $\downarrow\uparrow$ ,  $\uparrow\downarrow$ , and  $\uparrow\uparrow$ . The first state in each of the four cross-sections is the state of the neutron before it enters the sample. The second state is that of the neutron after it is reflected from

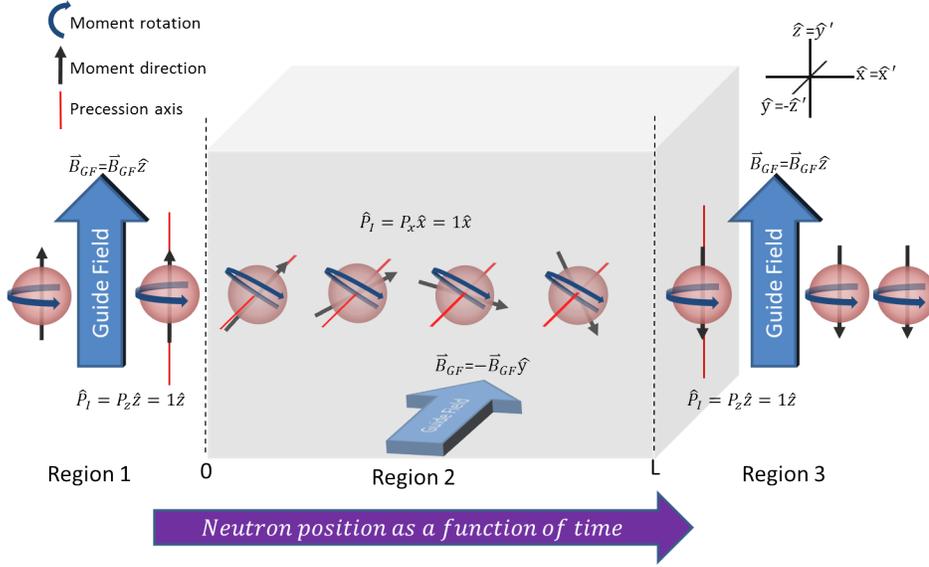


Figure 1.9: Schematic of a spin flipper. An in-depth description can be found below.

the sample. Because the polarizer and analyzer are sensitive to alignment and can be cumbersome to remove, the idea of switching  $\uparrow$  selecting polarizers for  $\downarrow$  selecting polarizers is a somewhat unrealistic procedure and would involve, not only the design and development of an  $\downarrow$  selecting polarizer but would also require time to setup which would otherwise be used for data acquisition. To resolve this issue and make neutron flipping a more reasonable experiment, a spin-flipper device, which is somewhat more complicated but far less labor intensive, has been developed.

Spin-flippers use electromagnetic fields to reverse the spin-state selected by the polarizer and the analyzer[4]. By having the ability to flip the selected neutrons after the polarizer and analyzer, all four magnetic cross-sections may be obtained. Figure 1.5 shows the principle behind the spin-flipper device[4].

The schematic starts at some point after the polarizer, where the spin-up state has already been selected. There is a small alignment field after the polarizer

which helps ensure that the neutrons do not lose polarity as they traverse the instrument. This magnetic field is applied in the the direction of the quantization axis, or the axis in which the spin-up neutrons are precessing around. Notice that this precession does not change the neutron's polarity. Schematically, this is illustrated by the blue arrow which shows the neutron precession around the quantization axis. As it precesses, there is no change to the moment (represented by the black arrow). When the neutron reaches position 0 at the boundary between region 1 and region 2, it immediately changes its quantization axis to the  $-\hat{y}$ [4]. This sharp and immediate transition results in a non-adiabatic process and is crucial for changing the quantization axis and, therefore, cause the neutron to precess in the  $x$  plane[4]. If the transition is too gradual, the neutron will instead follow the field gradient as it changes[4]. The size and magnetic field of region 2 is precisely chosen so that, by the time the neutron reaches location L, it has precessed 180 degrees. These design requirements may be calculated using the formula:

$$|\Delta\phi| \simeq \frac{2m\mu B}{(\hbar k_0)^2} k_0 y = \frac{2\mu B}{mv_0^2} k_0 y \quad (1.42)$$

where  $\mu$  is the magnitude of the neutron magnetic moment ( $\mu = -1.913 * 5.051 * 10^{-27} J/T$ ) and B is the magnetic induction. A nice example of this is given on page 421 of reference [4]. Finally, the individual polarization components for the x, y and z directions can be calculated using a rotation matrix:

$$\begin{pmatrix} P_{Fx} \\ P_{Fy} \\ P_{Fz} \end{pmatrix} = \begin{pmatrix} \cos(\Delta)\psi & -\sin(\Delta)\psi & 0 \\ \sin(\Delta)\psi & \cos(\Delta)\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_{Ix'} \\ P_{Iy'} \\ P_{Iz'} \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \quad (1.43)$$

Now that the neutron can be polarized and that polarization can be reversed, all four magnetic cross-section may be obtained.

## 1.4.2 Interpreting Magnetic Scattering Data

Now that the methods for neutron preparation have been explained, we can now discuss how to interpret magnetic scattering data. As in the non-magnetic case, the transfer matrix must be determined starting with the solution to the wavefunction. The process is similar; however, there is added complexity introduced by the magnetic scattering. It is important to note that, because we are still in the specular limit, there is only a  $z$  dependence in the formalism. Specifically, it takes the form[4]:

$$\left[ \frac{\delta^2}{\delta z^2} + \frac{Q^2}{4} - 4\pi\rho_{++z} \right] \Psi_{+z} - 4\pi\rho_{+-z} \Psi_{-z} = 0 \quad (1.44)$$

$$\left[ \frac{\delta^2}{\delta z^2} + \frac{Q^2}{4} - 4\pi\rho_{--z} \right] \Psi_{-z} - 4\pi\rho_{-+z} \Psi_{+z} = 0 \quad (1.45)$$

Just by observation, one can see the similarities between this equation and equation 1.22. Because both equations are equal to zero, we can combine them by multiplication to form[4]:

$$\left( \frac{\delta^4}{\delta z^4} + F \frac{\delta^2}{\delta z^2} + G \right) \Psi_{\pm z} = 0 \quad (1.46)$$

where:

$$F = \frac{Q^2}{2} - 4\pi(\rho_{++z} - \rho_{--z}) \quad (1.47)$$

$$G = \left( \frac{Q^2}{4} \right)^2 - Q^2\pi(\rho_{++z} - \rho_{--z}) + (4\pi)^2(\rho_{++z}\rho_{--z} - \rho_{+-z}\rho_{-+z}) \quad (1.48)$$

which is derived in C.1

This can be written more conveniently as:

$$S^4 + FS^2 + G = 0 \quad (1.49)$$

with roots (using the number density form of the scattering length density derived in C.2):

$$S_1 = \sqrt{4\pi(Nb + Np) - Q^2/4} \quad (1.50)$$

$$S_2 = -S_1$$

$$S_3 = \sqrt{4\pi(Nb - Np) - Q^2/4}$$

$$S_4 = -S_3$$

Now, solving the wavefunctions from equation 1.44 for each root[4]:

$$\Psi_{+z} = \sum_{j=1}^4 \mathcal{C}_j e^{S_j z} \quad (1.51)$$

$$\Psi_{-z} = \sum_{j=1}^4 \mathcal{D}_j e^{S_j z}$$

At this point, the same steps which were used to derive the non-magnetic scattering are followed for the magnetic derivation. This derivation is shown, in part, in C.3 and is given by[4]:

$$\begin{pmatrix} t_+ \\ t_- \\ \frac{iQ}{2}t_+ \\ \frac{iQ}{2}t_- \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix} \begin{pmatrix} I_+ + r_+ \\ I_- + r_- \\ \frac{iQ}{2}(I_+ + r_+) \\ \frac{iQ}{2}(I_- + r_-) \end{pmatrix} \quad (1.52)$$

where the values for  $A_{ij}$  are tabulated in C.4

## 1.5 Off-Specular Scattering

In the previous sections all mathematical derivations have assumed that the wave vector transfer only contains a  $q_z$  component. This is only the case where potential barriers are completely flat with no in-plane variation. In reality, this case rarely occurs and in-plane variations from fabrication, diffusion, phase separation, and other events are often present. These in-plane variations scatter the incoming neutron beam at angles outside of the specular scattering condition, resulting in a wave vector transfer with three dimensional components (scatters in  $q_x$ ,  $q_y$ , and  $q_z$ ). This results in a three dimensional problem rather than a one dimensional problem and creates a much more complicated scenario to model.

To date, no closed form solution to the off-specular scattering problem has been developed. Still, significant progress has been made in the formulation of approximations for modeling off-specular neutron scattering. This section describes

the most recent work on the appropriate ways to treat off-specular scattering data.

In 1988, Sinha et al. developed a closed form calculation for solving the diffuse scattering from films with 'special' types of interfacial roughness within the distorted wave Born approximation (DWBA) for systems where:

$$q_z \sigma \gg 1 \tag{1.53}$$

where  $\sigma$  is the root-mean-square roughness and  $q_z$  is the specular component of the scattering [26]. Most of his derivation is based on the work by Vineyard in 1982 [31]. He also presented the Born approximation (BA) results for a similar set of special cases. Of course, the approximations still contained inaccuracies and must be used with caution. The inaccuracies of both the BA and the DWBA which were implemented in the paper are well illustrated in figure 1.10. The BA cannot represent the critical edge and is observed to approach infinity as  $q_z \rightarrow 0$  while the DWBA proves to be inaccurate at higher  $q$  values.

Although his work was applicable for both X-rays and neutrons, it was only accurate for single films with special types of roughness. This was later extended by Ljungdahl [18] to include an autocorrelation function which allowed the formulation to be used for an even broader range of roughness types. In 2005, Rauscher et al. was able to use the DWBA with Gaussian roughness to model the off-specular scattering from a multilayer system [25]. Their results show Yoneda wings resulting from the interfacial roughness between the multilayers. Unfortunately, most of their results are shown as schematics and real systems were not modeled. Significant

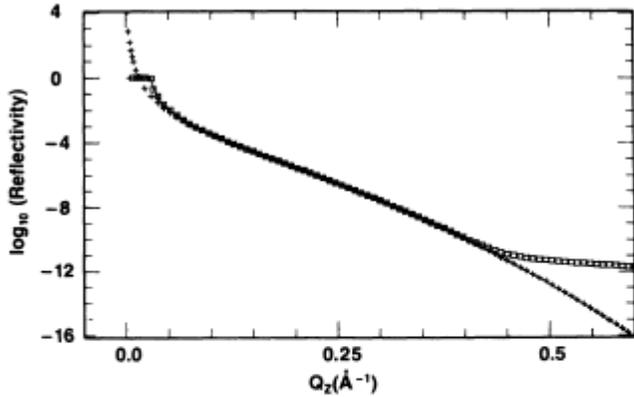


FIG. 5. The computed reflectivity of Pyrex at  $\lambda=1.46 \text{ \AA}$  with  $\sigma=8 \text{ \AA}$ , using the Born approximation (crosses), the DWBA (open squares), and the form given by Eq. (4.34) (solid line).

Figure 1.10: figure from reference [26].

contributions to the field have also been made by Toperverg [28, 29, 30]; however, these papers lack detail in their mathematical presentation and the source code used to perform the fundamental calculations is not available for review and verification. Perhaps the most well-presented and pertinent formalism is given by Kentzinger et al. [14]. As this result is the basis for many of the calculations implemented in our software, it is worthwhile to present these calculations in full. For this specific presentation, the magnetic component will be omitted. As with most DWBA formalisms, this version of the calculation starts with splitting the calculation into a reference potential and a perturbation on that potential. This can be written as [14]:

$$\hat{\mathcal{V}}_{\rho} = \hat{V}_l + \hat{V}_{l\rho} \quad (1.54)$$

The reference potential,  $\hat{V}_l$ , is the specular reflectivity and  $\hat{V}_{l\rho}$  is a perturbation

to that specular reflectivity. Because the full, three dimensional problem cannot be solved,  $\hat{\mathbb{V}}_{l\rho}$  is taken to be the average in-plane scattering potential [14]. This allows the perturbation to be calculated in a single (z) direction. This perturbation is then applied to the whole in-plane structure [14]. The assumption made here is that the out-of-plane dependence of the wavefunction is so small that it may be estimated as that calculated by the averaged the in-plane potential [14]. The wave equation may now be propagated down into this averaged sample to determine a solution to the wavefunction with which the scattering will be perturbed. There are two waves to be concerned with. The wavefunction for the incoming and outgoing wave can be written as [14]:

$$|\psi_{il}\rangle = e^{i\mathbf{k}_{il}\rho} \cdot \hat{S}_{il(z)} \cdot |\psi_{i0}(k_i,0)\rangle \quad (1.55)$$

$$\langle\psi_{fl}| = \langle\psi_{f0}(k_f,0) \cdot \hat{S}_{fl(z)} \cdot e^{i\mathbf{k}_{fl}} \quad (1.56)$$

respectively where the propagator functions are [14]:

$$\hat{S}_{il(z)} = e^{i\hat{p}_{il}(z-z_{l-1})}\hat{t}_{il}e^{-i\hat{p}_{il}(z-z_{l-1})}\hat{r}_{il} \quad (1.57)$$

$$\hat{S}_{fl(z)} = e^{i\hat{p}_{fl}(z-z_{l-1})}\hat{t}_{fl}e^{-i\hat{p}_{fl}(z-z_{l-1})}\hat{r}_{fl} \quad (1.58)$$

This is simply the solution to the wavefunction at each layer in the sample.

Using these values, the scattering amplitude may be calculated as [14]:

$$\hat{F}_{fi} = \sum_l \int dz \hat{S}_{fl}(z) \cdot \hat{F}_l(\mathbf{Q}_{||}) \cdot \hat{S}_{il}(z) \quad (1.59)$$

where  $\hat{F}_l(\mathbf{Q}_{||})$  is the in-plane Fourier transform.

By combining the scattering contributions from the reference and residual potential, one gets the resulting scattering from the system.

This formalism is similar to those used in previous work and is what we will adapt for our scattering calculations. The exact formula used in the software is described in chapter 3.

## 1.6 Off-specular Neutron Reflectometry Sensitivity and Applications

By measuring both specular and off-specular reflectometry simultaneously information about both the depth profile and in-plane structure may be determined. The size scales at which each of the probing directions is sensitive to; however, differ significantly. Figure 1.11 highlights the large difference in  $q$  range probed in the  $q_x$  and  $q_z$  directions. This is a practical limitation due to the instrumental geometry. In these plots, the maximum  $q$  indicates the smallest real space feature which is able to be resolved by that data. In the specific example of figure 1.11, the depth profile can resolve thicknesses on the order of  $209\text{\AA}$  while the in-plane structure can only resolve features of approximately  $3,121.6\text{\AA}$ . This means that, with current instrumentation, the in-plane ordering that can be practically evaluated by the off-specular scattering is on the order of microns.

This difference in real space sensitivity must be considered when determining the techniques usefulness when considering its use for modeling real samples.

Some useful cases are the use of determining magnetic exchange properties of large magnetic ellipses in a rectilinear lattice. Another use is presented by Kentzinger [14] evaluates large magnetic domains in permalloy supermirrors where there is no structural in-plane variation. With better signal to noise it may be possible to eventually characterize smaller in-plane feature which would allow for the study of phase separated diblock copolymers; however, current instrumentation does not have such capabilities.

In general, this technique will find use in any thin film systems that have large in-plane ordering. Although specialized, this technique provides many unique advantages over other techniques and may become a key characterization tool for many types of systems.

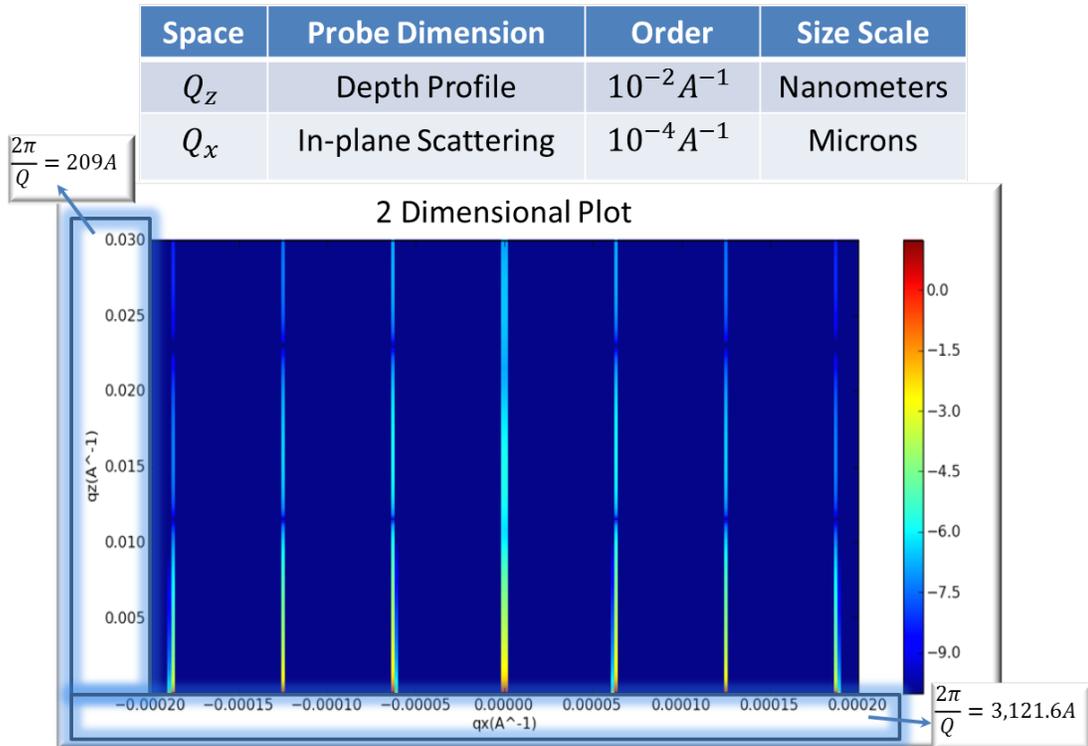


Figure 1.11: An evaluation of the size scales able to be probed by the specular and off-specular scattering. The maximum  $q$  values determine the minimal feature size that can be resolved.

## Chapter 2

### Software

#### 2.1 Overview

An important part of producing a widely usable and open software package is to ensure that the software is broad in its functionality and easily accessible to the target users. In general, specialized data analysis and modeling software presented in this thesis is written in either Mathematica or Matlab which are commercially available software packages that are designed for mathematical applications. Unfortunately, these packages also require licenses to use, which limits the software's user community to those which have licenses to the particular software package. Because one goal of this project is the development an open-source and broadly usable modeling software, other languages had to be utilized. The language chosen was Python [22], which is an open-source scripting language with a variety of mathematical (eg. `scipy`, `numpy`) [19], data visualization (eg. `matplotlib`, `pylab`)[12] and graphical user interface libraries(GUI)(eg. `wxpython`)[21]. By combining the power of these libraries (mostly written in C) with the development speed inherent to scripting languages, this is a good choice for scientific software development. More specific information about software installation, dependencies, and functionality may be found in Appendix D, which is the software instruction manual. The software is open source under a general usage license agreement. Users of this software should

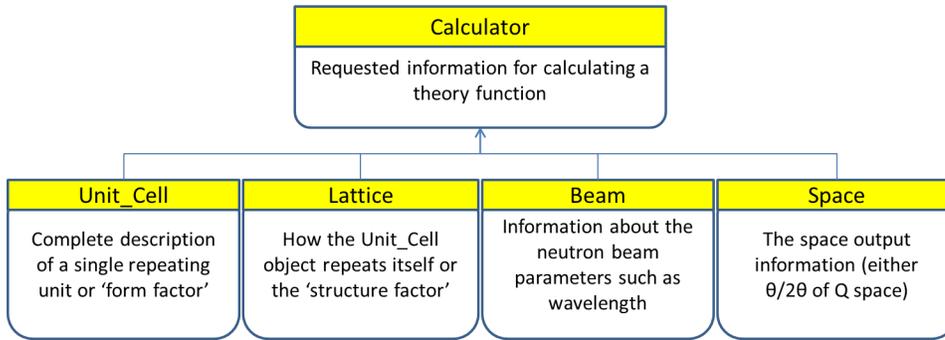


Figure 2.1: Class structure outline for the software.

reference this thesis.

## 2.2 Software Workflow

Software for the modeling of scattering data requires two basic components. The software must provide a user with the ability to build a model of their system. This can be done in a variety of ways and is where most of the user interaction occurs. The second component provides a user with a way to calculate what the scattering looks like from the model system and compare it to real data. To achieve this functionality, the software was developed with an object oriented design which is outlined in figure 2.1. This design allows for the easy development of additional components to the software without breaking other aspects of the calculations. Computational objects hold all of the required information and calculations which are specific to that set of information.

For example, figure 2.1 shows a Lattice object. This object holds all of the information required to calculate the structure factor as discussed in chapter 3. By making the structure factor calculation part of the Lattice object, a user or developer

could create a new structure factor calculation which could then be used in place of the one provided in the software. By following the overall application programming interface (API), the new structure factor calculation can utilize every other class (eg. `Unit_Cell` for feature building), without further modification. The advantage of this design becomes quite apparent when developing and testing different mathematical approximations for calculating theory functions.

The `Calculator` class holds all of the information that is required to calculate off-specular scattering theory functions. It is comprised entirely of individual objects of other classes and, based on those object types, chooses the appropriate calculation (which can also be defined by the user). The developed subclasses are complex and the details of each are explained below.

### 2.3 `Unit_Cell`

This is the most complicated class as it is the entire representation of a single unit. The primary representation of the unit cell is a matrix implemented with the `numpy` library, which holds the SLD values for a discretized model. This matrix can be produced through different methods depending on the the users desired model. The first unit building method imports a raw data file from an open source 3D modeling packages. The second allows the user to build a unit cell with parameterized shapes using mathematical form factors. The third model building method allows users to import a `.png` file of a model's cross-section. Each will be described in detail below.

Each of these methods provide a unique way to create the finite element matrix used to model scattering data. The important difference between these methods is how the models are parameterized. A system's parameterization is the numerical way in which a shape, feature, or structure is described. Parameterization is especially important in fitting, where an iterative change in parameter values is used to minimize the difference between scattering data and real data. Models should be parameterized in a way which allows for realistically constrained optimization. If the parameterization has too many degrees of freedom, the fitting process can take a prohibitively long time to minimize and can result in models which are non-physical. The parameterization must also have enough degrees of freedom that a fitting algorithm may be utilized. Of the three implemented methods, only the mathematical form factors method is parameterized in a way which allows for fitting. The 3D modeling software describes features using a collection of edges and nodes. These node positions could be used as parameters for a fitting engine; however, with so many nodes, and therefore so many parameters, it would be difficult to obtain realistic models from the minimization algorithm. The .png loader has the opposite issue. The image is fixed, and therefore does not allow for convenient parameterization. Each pixel could be treated as an individual parameter; however, this again leads to a prohibitively large number of parameters. Although only the mathematical form factors method is useful for fitting, the other modeling tools may be useful for more complicated systems, or for combining SEM images with scattering data.

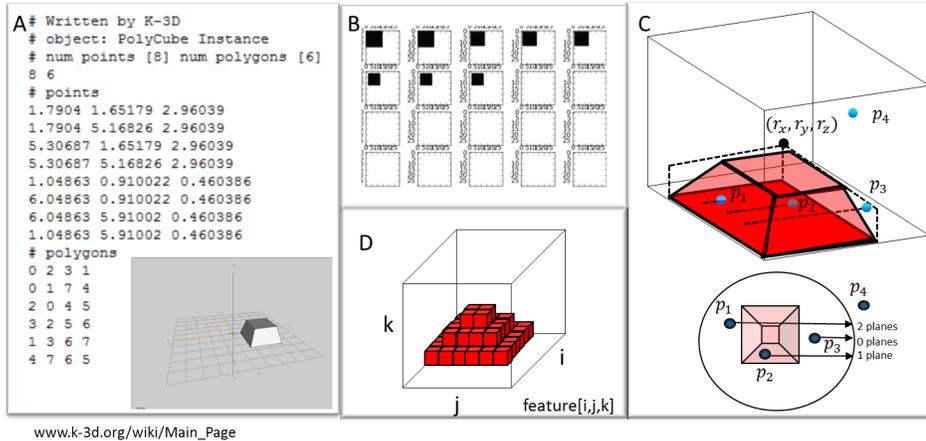


Figure 2.2: K3D output and processing for a model system.

### 2.3.1 3D Modeling Software

Among the open source 3D modeling software programs available, the one that best fit our needs was K3D [20]. This particular program allows for the output of a 3D model as a collection of polyhedron sides and nodes which make up the shape.

Figure 2.2A shows a trapezoid built in the 3D modeling software along with its raw data output. A discretization algorithm is applied so that a matrix is formed as the spy plot in figure 2.2B shows. The final matrix is schematically represented in figure 2.2D. The process of building the matrix in this manner is computationally expensive and time consuming. Some effort has been made to speed up this process; however, as this is not intended to be the main input mechanism, the current state has not been optimized.

The process starts with an empty matrix which is sized at the user's discretion. This matrix will be filled with the feature which was built in the 3D modeling software. The process relies on a ray tracing or shooting algorithm to determine

whether or not each of the points represented by the matrix fall within the feature which was created in the k3d software[11]. The process uses a ray to probe space and determine whether a point falls within a polyhedron by testing how many polygon faces the ray intersects with. The following math describes the steps taken by the algorithm to build a matrix using a K3D model.

The sample is assumed to be in the first octant of 3D space. First, the most distant point on the feature is determined by finding the maximum value for each dimension in the k3d node list. This point,  $(r_x, r_y, r_z)$  can be seen in figure 2.2 and defines a bounding radius for the feature. each point  $(p_i, p_j, p_k)$  is then run through a series of tests to conclusively determine whether it falls within a feature.

The first test determines whether:

$$(p_i - r_x)^2 + (p_j - r_y)^2 + (p_k - r_z)^2 > r_x^2 + r_y^2 + r_z^2 \quad (2.1)$$

which tests whether the point falls outside of the bounding sphere. This is computationally much less expensive than the shooting algorithm and, depending on the size of the feature, can significantly decrease calculation times. Point  $p_4$  in figure 2.2 illustrates this situation.

Next, the special cases of the test point falling on a line of a planer face must be tested. Taking the two points which make up the line,  $p_1$  and  $p_2$ , and the point to be tested,  $p_0$ , the distance can be tested with:

$$d = \frac{|(p_2 - p_1) \times (p_1 - p_0)|}{|p_2 - p_1|} \quad (2.2)$$

and if  $d = 0$  then the point is accept as being inside the feature. This test must be run for every line which makes up the feature. Next, the point is tested to determine whether or not it falls on any of the planes. The plane is formulated as:

$$a(x - x_1) + b(y - y_1) + c(z - z_1) = 0 \quad (2.3)$$

where  $n = \langle a, b, c \rangle$  is a vector normal to the plane,  $p_1 = (x_1, y_1, z_1)$  is an arbitrary point contained on the plane, and  $p_1 = (x_0, y_0, z_0)$  is the point being tested. The plane equation can be written as:

$$ax + by + cz + d = 0 \quad (2.4)$$

where  $d = -ax_1 - by_1 - cz_1$ . With these equations, the distance formula may be written as:

$$d = \frac{ax_0 + by_0 + cz_0 + d}{\sqrt{a^2 + b^2 + c^2}} \quad (2.5)$$

and again, if  $d = 0$  then the point falls on a face of the shape and, therefore, is included in the shape. This test must be run for every plane which makes up the feature.

If these special situations are determined to be false, the shooting algorithm is applied. This algorithm relies on the principle that, by placing a tracer line from the test point to a point which is known to be outside of the feature, the number of planes the line crosses will elucidate the position of the point in space. Simply, if the line crosses an even number of planes, the point must lie outside of the feature

( $p_0$  and  $p_2$  in figure 2.2 C). If the line crosses an odd number of points ( $p_1$  in figure 2.2 C), it necessarily lies within the feature. Because this algorithm relies only on whether or not the number of crossed planes is even or odd, it will work for both concave and convex structures.

First, the tracer line must be chosen to ensure that it does not lie on top of any of the lines which make up the feature. We start with the equation of a line for each of the lines:

$$l_{trace} = \vec{a} + u_a(\vec{b} - \vec{a}) \quad (2.6)$$

$$l_{shape} = \vec{c} + u_b(\vec{d} - \vec{c}) \quad (2.7)$$

where  $l_a$  is the tracer line and  $l_b$  is the shape line segment and  $P$  are the two points on the lines. These points have an x, y and z component which need to be tested for intersection points. Breaking the equation into its individual components and setting them equal to each other, we can determine if the lines intersect:

$$a_x + (b_x - a_x)u_a = c_x + (d_x - c_x)u_b \quad (2.8)$$

$$a_y + (b_y - a_y)u_a = c_y + (d_y - c_y)u_b \quad (2.9)$$

$$a_z + (b_z - a_z)u_a = c_z + (d_z - c_z)u_b \quad (2.10)$$

The intersection calculation is now calculated for each plane. For example, to determine if the lines intersect in the x-y plane we use the matrix form:

$$x = bA^{-1} \quad (2.11)$$

which, for the x-y plane is:

$$\begin{pmatrix} u_a \\ u_b \end{pmatrix} = \begin{pmatrix} -a_x + c_x \\ -a_y + c_y \end{pmatrix} \begin{pmatrix} b_x - a_x & -d_x + c_x \\ b_y - a_y & -d_x + c_x \end{pmatrix}^{-1} \quad (2.12)$$

and we evaluate whether:

$$0 \leq u_a \quad (2.13)$$

$$0 \leq u_b \leq 1 \quad (2.14)$$

Equation 2.14 is open on the right hand limit because the tracer line is a ray which extends to the unit cell limit. The line segment (equation 2.14) which is being tested has two end points and so both inequalities are tested. If these two conditions are found to be true, then the x-z and y-z planes must all be tested. If their lines intersect in all three planes, then a new tracer line must be selected. The selection of a new tracer line is a trivial task because the discrete units represent a finite distance in space. Any new point which falls within this finite space may be used as the tracer line. The implemented algorithm starts at the center value of the discretized unit and takes:

$$p_{n+1} = \frac{p_n - p_0}{2} + p_0 \quad (2.15)$$

where  $p_0$  is the minimum value held by the discretization and  $n$  is the iteration number for tracer point selection. In practice,  $n$  is rarely greater than 2. Every line which makes up the feature must be tested to ensure that the trace will succeed.

Now that a tracer line has been selected, Each plane in the feature must be

tested to determine whether or not the trace line passes through it. We can define  $l_a$  and  $l_b$  as the points that make up the tracer line and:

$$p_0 + (p_1 - p_0)u + (p_2 - p_0)v \quad (2.16)$$

is the equation for the plane and  $p_0$ ,  $p_1$ , and  $p_2$  are three non-collinear points on the plane. Using the method applied earlier to determine the line intersection, we can set these two equations equal to each other to get:

$$l_a + (l_b - l_a)t = p_0 + (p_1 - p_0)u + (p_2 - p_0)v \quad (2.17)$$

$$l_a - (l_a - l_b)t = (p_1 - p_0)u + (p_2 - p_0)v \quad (2.18)$$

which can be written in the matrix form:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \begin{pmatrix} x_a - x_b & x_1 - x_0 & x_2 - x_0 \\ y_a - y_b & y_1 - y_0 & y_2 - y_0 \\ z_a - z_b & z_1 - z_0 & z_2 - z_0 \end{pmatrix}^{-1} \begin{pmatrix} x_a - x_0 \\ y_a - y_0 \\ z_a - z_0 \end{pmatrix} \quad (2.19)$$

The point of intersection is:

$$l_a + (l_b - l_a)t \quad (2.20)$$

Now that the point where the tracer line passes through the plane has been calculated, it must be determined if this point falls within the polygon which makes up the feature. To do this, the point and the polygon are projected onto each plane.

The point is then tested to see if it falls within the polygon projection. This is done for each plane which makes up the feature.

As mentioned previously, parameterization of this modeling would be prohibitively difficult. Therefore, a more convenient parameterization scheme would describe a sphere as a radius and a central point, rather than as a collection of polygons or matrix positions. Still, this modeling technique can be useful in predicting the scattering from very complex samples without the need to manually enter matrix values. To allow for parametrization of a system, mathematical form factors are needed.

### 2.3.2 Mathematical Form Factors

An infrastructure has been developed which allows for the addition of any form factor given there is an equation to carry out the desired discretization. As mentioned at the beginning, this is the only model building method that allows for fitting. Each shape definition ensures that the minimized model still contains the same form as the original shape. This limits the number of parameters and creates a more physical minimized feature (as apposed to a model which is less likely to physically exist). These predefined shapes may also be combined and constrained to produce more complicated features and the syntax simplifies model scripting. Figure 2.3 illustrates how to use these objects to build complicated unit cells.

A variety of different forms have already be developed and implemented. This section discusses the formalism for each form factor. For all formulas,  $C_x$ ,  $C_y$ , and

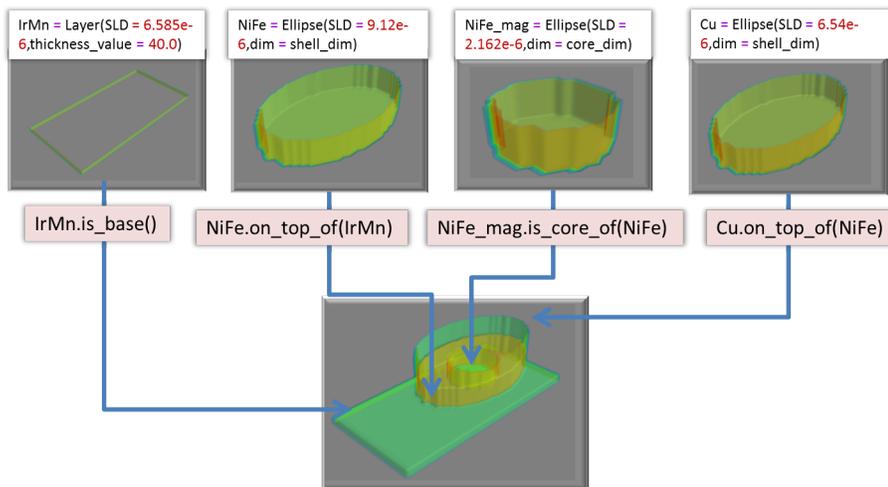


Figure 2.3: An example of the mathematical form factors. The shapes are instantiated using the syntax above each shape. The 3D plot illustrates the matrix produced by the instantiation. Finally, the features are related to each other using the syntax in the pink boxes to orient the shapes appropriately. The final shape is a complicated mixture of different objects which was produced with 8 lines of code.

$C_z$  denotes the center point and  $P_x, P_y,$  and  $P_z$  denote the point being tested.

## Sphere

$$(P_x - C_x)^2 + (P_y - C_y)^2 + (P_z - C_z)^2 \leq r^2 \quad (2.21)$$

where  $r$  is the radius of the sphere.

## Parallelepiped

For this calculation, first the minimum and maximum limits of the feature in each direction are calculated by:

$$M_{min} = C - \frac{D}{2} \quad (2.22)$$

$$M_{max} = C + \frac{D}{2} \quad (2.23)$$

where  $D$  is the dimension of the feature in the  $x, y$  and  $z$  directions. This is followed by the inequality tests:

$$M_{min_x} \leq x \leq M_{max_x} \quad (2.24)$$

$$M_{min_y} \leq y \leq M_{max_y} \quad (2.25)$$

$$M_{min_z} \leq z \leq M_{max_z} \quad (2.26)$$

## Ellipse

For the ellipse, the z-axis test is similar to that of the parallelepiped system:

$$M_{min_z} = C_z - \frac{D_z}{2} \quad (2.27)$$

$$M_{max_z} = C_z + \frac{D_z}{2} \quad (2.28)$$

and:

$$M_{min_z} \leq z \leq M_{max_z} \quad (2.29)$$

In the x-y plane, the test is carried out through the equation:

$$0.0 \leq \frac{(x - C_x)^2}{a^2} + \frac{(y - C_y)^2}{b^2} \leq 1.0 \quad (2.30)$$

where a and b are the equilateral radii in the x-y plane.

## Cone

The z-axis of the cone is calculated as was done in the ellipse

$$M_{min_z} = C_z - \frac{D_z}{2} \quad (2.31)$$

$$M_{max_z} = C_z + \frac{D_z}{2} \quad (2.32)$$

and:

$$M_{min_z} \leq z \leq M_{max_z} \quad (2.33)$$

There is an added ability here to make a truncated cone. This is done by parameter  $S$  which says that  $z < S$  must be true.

For the x-y plane we first define:

$$T_a = \tan^{-1} \left( \frac{z}{a} \right) \quad (2.34)$$

$$T_b = \tan^{-1} \left( \frac{z}{b} \right) \quad (2.35)$$

and then test:

$$0.0 \leq (x - C_x)^2 \frac{\tan(T_a)}{(C_z + z)/2 - z} + (y - C_y)^2 \frac{\tan(T_b)}{(C_z + z)/2 - z} \leq 1.0 \quad (2.36)$$

## Pyramid

The z dimension test is exactly the same as the cone, including the  $S$  parameter and so will not be repeated. Please see 2.3.2 for this calculation.

The pyramid calculation first defines:

$$T_a = \tan^{-1} \left( \frac{z}{a} \right) \quad (2.37)$$

$$T_b = \tan^{-1} \left( \frac{z}{b} \right) \quad (2.38)$$

This test is split into two components which are:

$$C_x - \frac{(C_z + D_z/2) - z}{(\tan(T_a)/2)} \leq x \leq C_x + \frac{(C_z + D_z/2) - z}{(\tan(T_a)/2)} \quad (2.39)$$

$$C_y - \frac{(C_z + D_z/2) - z}{(\tan(T_b)/2)} \leq y \leq C_y + \frac{(C_z + D_z/2) - z}{(\tan(T_b)/2)} \quad (2.40)$$

## Ellipsoid

The ellipsoid shape is much like a lens shape which is not to be confused with the ellipse. In the case of the ellipse, the scattering length density is constant in the z direction whereas the test for the ellipsoid in the z direction is written as:

$$0.0 \leq \frac{(x - C_x)^2}{a^2} + \frac{(y - C_y)^2}{b^2} + \frac{(z - C_z)^2}{c^2} \leq 1.0 \quad (2.41)$$

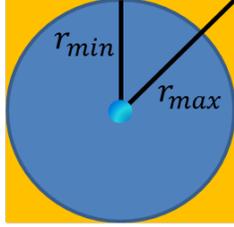
where a, b and c are equilateral radii. if  $a = b = c$  then this produces a sphere.

## RoundedParallelepiped

This version of the the parallelepiped feature representation combines the parallelepiped feature with the ellipse calculation. It was created to model gold featured samples discussed in chapter 3 because wet etching results in rounded corners and is discussed in more detail in chapter 3 This object has an additional parameter called 'curve', which allows the user to choose a percent rounding of the parallelepiped features corners. The value of the parameter is set from 0 to 1 where 0 is a purely parallelepiped feature and 1 is a purely ellipse feature. The ratio's meaning is derived by taking the length of each side and the length of the diagonal. This is illustrated for in figure 2.4.

### 2.3.3 Image Loading

Often a user has an image of the sample they wish to model, such as an optical or SEM cross-sectional micrography, of their structure. The software has an



$$r_{round} = [c(r_{max} - r_{min})] + r_{min}$$

Figure 2.4: Illustrates how the curvature parameter, curve, is calculated.  $c$  is the curvature parameter.

image loader ability which can use these micrographs as input for model building to calculate the expected scattering. The color scale in the image must represent the desired SLD to within some scaling factor. For example, figure 2.5A has grey scale values which are multiplied by  $1 * 10^{-5}$  to match the desired SLD. The image is then read into the program and a matrix is created by assuming that one axis is constant in the depth direction of the photo. This is illustrated in figure 2.5B.

Because the pixel density of these images are often much higher than is required for the needs of our unit cell object, a rebinning algorithm was developed. This uses a traditional rebinning process where the volume of pixels in the original image (2.5C) is averaged and entered into the courser discretized matrix(2.5D). Because the chosen resizing factor is generally not an exactly divisible number, often the new pixel volume is not an integer value of pixels from the original image. To resolve this, the pixel contribution is weighted by the amount of the original pixel that is included in the new pixel area. The result of this can be observed in figure 2.5D where the interface between two SLD values is an average of the two SLD values which make up that interface. Although this averaging causes a loss in information,

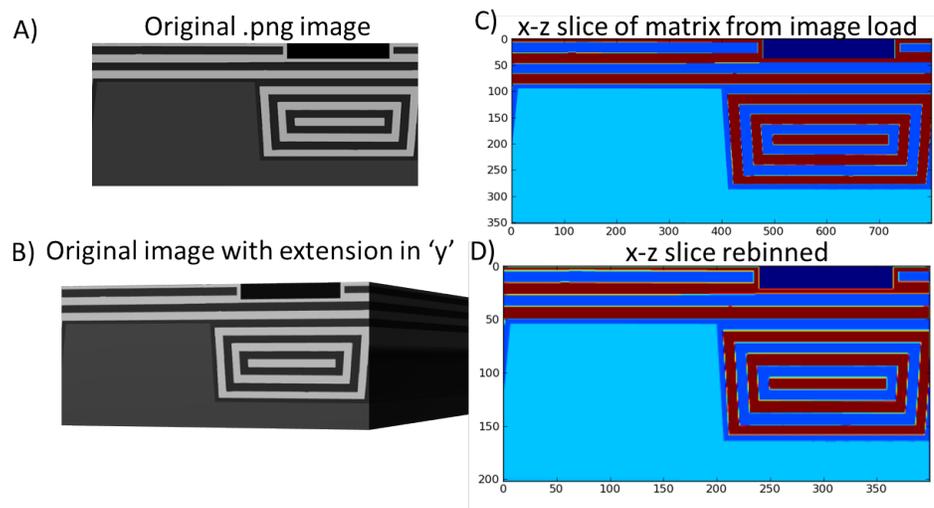


Figure 2.5: A) The original .png image which will be loaded into the unit cell. B) Schematic showing the image extension in the y direction, which is how it will be represented in the unit cell. C) Slice of the matrix created after loading. Not the x-y axis which are pixel counts. D) The matrix has been reduced in size through a rebinning process.

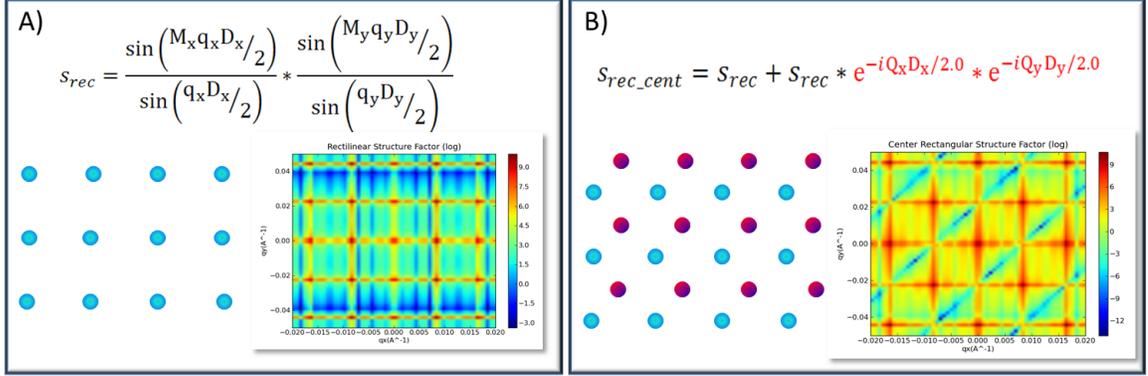


Figure 2.6: A) the formula for a rectilinear lattice with a square envelope, the real space schematic and the resulting calculation in  $q_x$  and  $q_y$ . B) the formula for the calculation of a body centered lattice by repeating a phase shifted rectilinear lattice. the effect on the theory function is unnoticeable for sufficiently dense unit cells.

## 2.4 Lattice

The lattice object holds all of the information required to describe how the unit cell object repeats itself across the sample. This object also contains the methods for calculating the scattering from this repeat structure. There are three lattice representations available. For now, we will discuss the case where the repeat structure has a coherence length convolved with a box envelope. The difference between the box envelope and the Gaussian envelope are discussed in more detail in chapter 3. The first is the square lattice, which is shown in figure 2.6A. 2.6B shows how the body centered lattice (depicted in the first half of 2.6B) may use the rectilinear lattice calculation with a phase shift to add a repeat structure. Figure 2.6B shows two separate rectilinear lattices represented by the blue and red dots respectively.

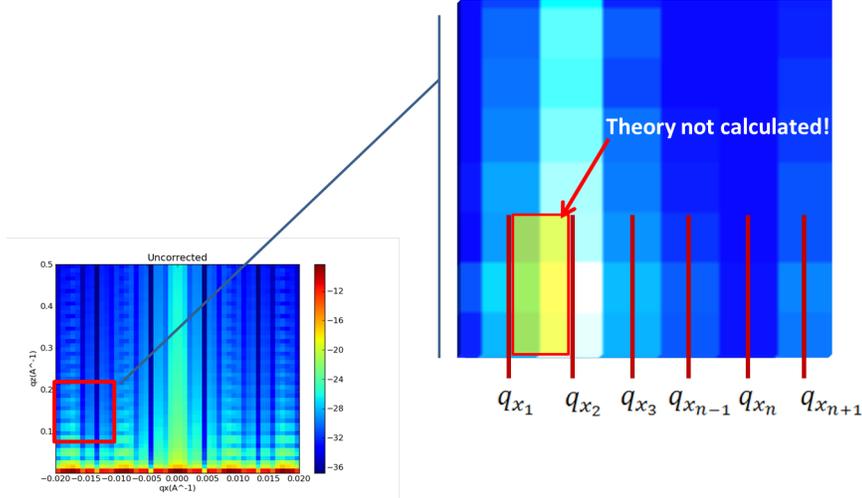


Figure 2.7: Shows how undersampling the  $q$  map can lead to erroneous calculations. The red dots are phase shifted to fall half way between the blue dots at a 45 degree angle. It is evident from the equation at the top of figure 2.6B that the phase shift is not limited to 0.5 period shift (see the exponential components) and can be adjusted to position the second lattice set anywhere in space, although in the current version, this repeat is fixed to a 0.5 shift in the x-y direction.

The software has a pre-calculated hexagonal lattice structure which takes in one value for the repeat distance( $D_x$ ) and calculates the corresponding  $D_y$  distance needed to produce a hexagonal lattice. It uses the formula:

$$D_y = 2D_x \cos(30 \text{ deg}) \quad (2.42)$$

Although the formalism used to calculate the structure factor is derived elsewhere, there is a computational concern which needs to be addressed for accurate calculations. The structure factor is calculated for each discrete point represented by the individual array elements. Unfortunately, this result is for a single  $q$  value

and does not represent the full pixel for which the calculation is being carried out. If the desired  $q$  spacing requested by the user is sufficiently large, then the calculation will not solve for  $q$  values which contain important features in the scattering, inaccurately representing key features in the scattering. This type of artifact is called aliasing. Figure 2.7 shows the region of  $q$  space where no calculation is occurring and, therefore, not represented in the calculated results. When the sampled  $q$  values are spaced far apart, very sharp scattering features, such as the specular scattering, will most likely be under-sampled and, therefore, appear not to exist as is seen in figure 2.8A. Another artifact evident in figure 2.8A is that the first order diffraction peak appears to be much weaker than that of the second order diffraction peak. Although this phenomenon is sometimes possible, in this case the lower intensity is due to the fact that the first order diffraction peak is substantially more defined than the second order peak, which means the coarse sampling in  $q$  misses most of the scattered intensity. The problem can be resolved by integrating over the range of  $q$  values for which each pixel is representing. Figure 2.8B shows that, by integrating over the  $q$  range which contains the specular scattering,

## 2.5 Beam

The Beam object holds all of the information about properties of the probing beam. Specifically, it contains the wavelength, wavelength divergence, angular divergence, and background. Because this information is needed to calculate the resolution of the instrument, this object along with the Space object and the result-

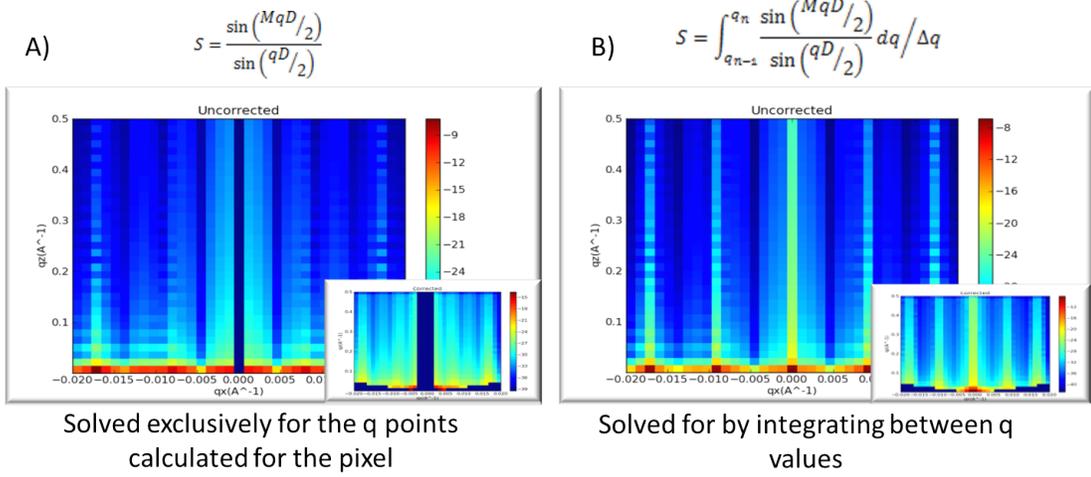


Figure 2.8: A) Structure factor calculation carried out A) at discrete q values and B) by integrating of the the q range for which each pixel is representing.

ing scattering, which is held by the Calculator object, are used to apply a resolution correction. This correction includes the wavelength divergence and the angular divergence and is applied as a Gaussian convolution of these two effects. The resolution can be approximately written as [10]:

$$\left(\frac{\delta\vec{Q}}{\vec{Q}}\right)^2 = \left(\frac{\delta\theta}{\tan(\theta)}\right)^2 + \left(\frac{\delta\lambda}{\lambda}\right)^2 \quad (2.43)$$

The wavelength divergence originates from the finite wavelength selected by the monochromator. Ideally, the monochromator would select neutrons of a specific wavelength and remove the others. In practice some finite range of wavelengths are selected by the monochromator, affecting the resulting scattering from the sample. The angular divergence originates from the spread of angles at which the neutrons leave the guide and enter the sample. Although the instrumental geometry is set for a specific angle, this spread in angle results in measuring scattering from multiple

angles. This also affects the neutron scattering data. Of course, because  $\vec{Q}$  is a vector, this implementation is somewhat complicated and is described in more detail in chapter 3.

## 2.6 Object Oriented MicroMagnetic Framework

Magnetics systems can be modeled by take advantage of already existing magnetic minimization software which are used to find the minimum-energy configuration of a magnetic system. the Object Oriented MicroMagnetic Framework (OOMMF) software[6], which is an industry-standard tool developed at NIST, uses a finite element approach similar to that used in the off-specular modeling software to represent their systems [6]. By allowing this software to minimize the magnetic moments in the given model and then importing that result into our software infrastructure, we can calculate scattering from magnetic features.

The OOMMF software uses a Landau-Lifshitz ordinary differential equation solver to relax the spin states in a model system. The model system is represented by a finite element matrix of the samples starting magnetization state and is relaxed under conditions parameterized in the model. This minimization approach can result in different final magnetic states, the scattering of which can all be modeled in the off-specular software.

First a specialized file writer takes the magnetic information given to the Shape objects and creates an OOMMF model input file. This file is then imported and modeled using the OOMMF software, resulting in an .omf file which can then be

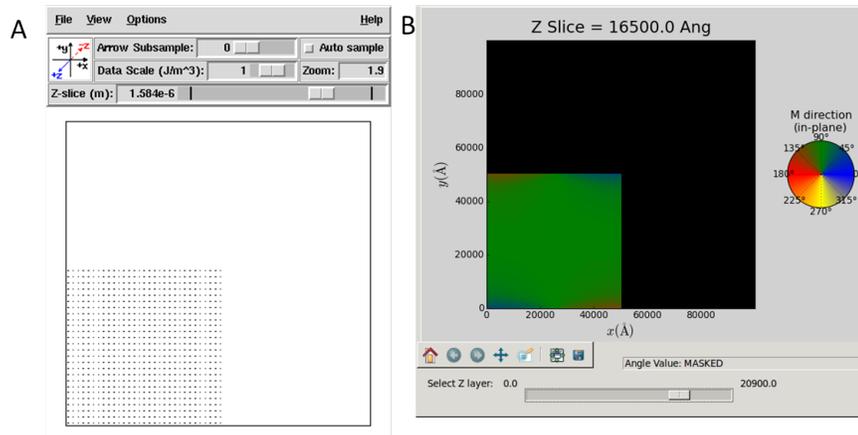


Figure 2.9: A) a .mif file created by our software loaded into the oommf software. B) a .omf file loaded from an oommf output file to our software and viewed using our slice viewing panel.

loaded by the off-specular modeling software (and the scattering calculated). Figure 2.9 shows an example of this procedure.

Figure 2.9B shows a slice viewer panel which is based off of a pre-existing viewer but adds the ability to scroll through the depth profile of the magnetic sample. This viewer can also be used for viewing the structural part of the sample. Alternatively, the full range of shape-building tools available for making 3D-models of nuclear SLD can be used to make magnetic SLD unit cells.

## Chapter 3

### Theory and Approximations

#### 3.1 Overview

This chapter will cover the formalism developed and included in the modeling software. It shows the derivations of the formalism, discusses the usefulness of each calculations, and shows examples of theory functions produced from models. The first attempts at modeling involved tweaking algorithms implemented in currently available software. The only real software that employed mathematical formalism which could be used to model off-specular scattering was IsGISAXS[17]. This software is designed to model grazing incidence small angle X-ray scattering (GISAXS), which is very similar in geometry to neutron off-specular reflectometry. Because GISAXS data is generally viewed in a different subset of reciprocal space ( $q_x$  versus  $q_y$ ) than off-specular reflectometry ( $q_x$  versus  $q_z$ ), the Fortran scattering kernel was removed from the software and wrapped with Matlab code to produce the desired scattering plot. The advantages of the distorted wave Born approximation (DWBA) became clear as the resulting model contains many features similar to those observed in real data, while the Born approximation (BA) does not. This kernel had some very problematic limitations. First, it utilized mathematical form factors to calculate the scattering. This limited its application to a single feature in a given system. While this feature could be buried in a layer, multilayered fea-

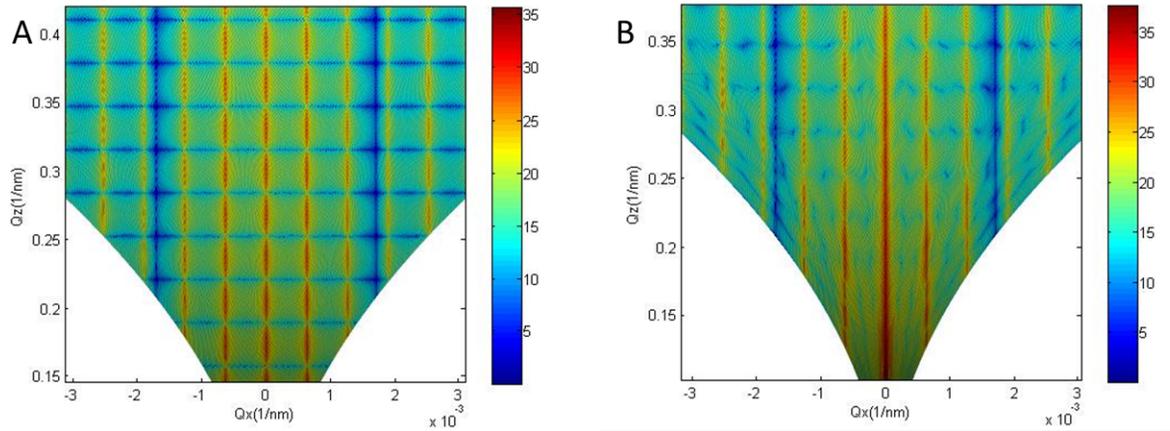


Figure 3.1: Simulations of a 3.7  $\mu\text{m}$  parallelepiped Au system on Si with feature height of 200nm in the a) Born Approximation and b) Distorted Wave Born Approximation. The Born approximation was not possible (a key class of systems for neutron reflectometry). Second, although the scattering qualitatively matched some key features in the data, there was evidence that it was not calculating certain scattering conditions appropriately. Using knowledge gained from the IsGISAXS kernel, new modeling formalism was developed.

### 3.2 First Order Born Approximations Form Factor

The kinematic or Born approximation is used in cases where dynamical scattering effects, such as multiple scattering events, do not appreciably contribute to the observed scattering. This approximation is most valid at larger  $\vec{Q}$  values. Because the described wave in this system is assumed to have little to no distortion by the sample media, the wave inside the media can be approximated to be the same as that of vacuum. This is illustrated in figure 3.2

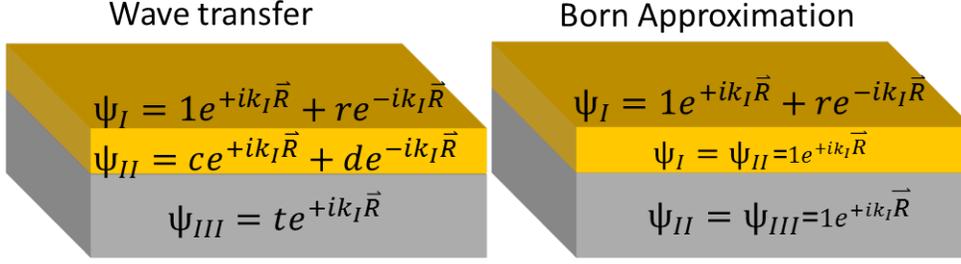


Figure 3.2: The wave transfer as done for the specular case versus the Born approximation.

As shown in appendix B, the reflection amplitude can be written as an integral:

$$r_{\vec{q}_z} = \frac{4\pi}{i\vec{q}} \int_{-\infty}^{\infty} \Psi_{\vec{q}_z} \rho_z e^{ik_{0z}z} dz \quad (3.1)$$

The approximation shown in figure 3.2 which sets the wave inside the sample to a simple plane wave:

$$\Psi_{\vec{q}_z} = e^{ik_{0z}z} \quad (3.2)$$

so:

$$r_{\vec{q}_z} = \frac{4\pi}{i\vec{q}} \int_{-\infty}^{\infty} e^{ik_{0z}z} \rho_z e^{ik_{0z}z} dz \quad (3.3)$$

$$r_{\vec{q}_z} = \frac{4\pi}{i\vec{q}} \int_{-\infty}^{\infty} \rho_z e^{2*k_{0z}iz} dz \quad (3.4)$$

$$r_{\vec{q}_z} = \frac{4\pi}{i\vec{q}} \int_{-\infty}^{\infty} \rho_z e^{iq_zz} dz \quad (3.5)$$

This result shows that, to this approximation, the scattering is the Fourier transform of the scattering potential. This can be expanded to the three dimensional case [4]:

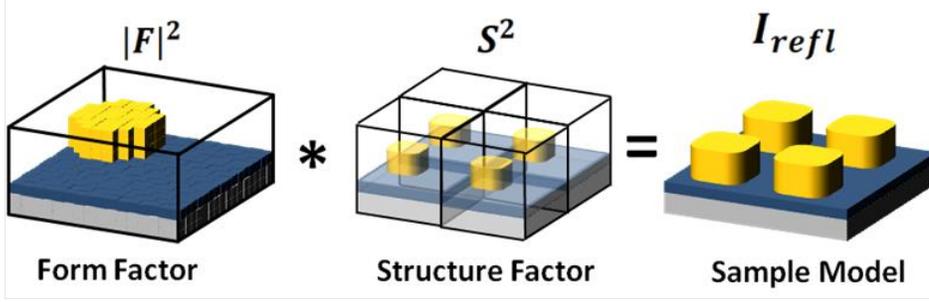


Figure 3.3: A pictorial representation of how the model is treated.

$$r_{\vec{Q}} = \frac{4\pi}{iq_z S_{xy}} \iiint_{-\infty}^{\infty} \rho_{\vec{R}} e^{-i\vec{Q} \cdot \vec{R}} \quad (3.6)$$

where  $S_{xy}$  is the the surface area of the sample probed by the neutron.

Because this work focuses on the specific case of repeated structures on thin film surface, we can treat the system as two components. As such, the calculation can be split into two separate pieces. The first component will be referred to as the form factor and will represent the single repeat unit of the sample. For convenience, reference to the form factor in its entirety will be denoted as  $\mathfrak{F}$ . The second will be referred to as the structure factor and will represent how that unit cell repeats across a sample surface. In formalism it will be denoted as  $\mathfrak{S}$ . Figure 3.3 shows the two components which make up a model.

$$\mathfrak{F} = \left( \frac{1}{iq_x} \right) \left( \frac{1}{iq_y} \right) \left( \frac{1}{iq_z} \right) (1 - e^{-iq_x x}) (1 - e^{-iq_y y}) (1 - e^{-iq_z z}) CZT(\rho_{lmn}) \quad (3.7)$$

To model data, this integral must be solved computationally. First, it will be easiest to break the integral into the two components based on the modeling shown in the previous section. For now,  $\mathfrak{S}$  will be used as a placeholder for the structure

factor calculation. The integral may now be written as:

$$\left(\frac{q_z S_{xy}}{4\pi}\right)^2 |r_{\vec{Q}}|^2 = \left| \int_0^{D_x} \int_0^{D_y} \int_0^{D_z} \rho_{xyz} e^{iq_z z} e^{iq_y y} e^{iq_x x} dz dy dx \right|^2 * |\mathfrak{G}|^2 \quad (3.8)$$

Taking only the z component of this integral:

$$\mathfrak{F}_z = \int_0^{D_z} \rho_{xyz} e^{iq_z z} dz \quad (3.9)$$

where  $D$  is the unit cell size in each of the three dimensions. As seen in figure 3.3, the integral will be solved over discrete units much like figure 1.6. By the nature of the discretization, each sub-unit of the model will have the same SLD value. This means that the integral may be written as:

$$\mathfrak{F}_z = \sum_{n=1}^N \int_{(n-1)\Delta z}^{n\Delta z} \rho_{lmn} e^{iq_z z} dz \quad (3.10)$$

makes  $\rho$  a constant and can be removed from the integral.

$$\mathfrak{F}_z = \sum_{n=1}^N \rho_{lmn} \int_{(n-1)\Delta z}^{n\Delta z} e^{iq_z z} dz \quad (3.11)$$

Because we have now presented the formalism for the discretized system and, over each discretized unit, the  $\rho_{lmn}$  is constant, the integral may now be written as a summation over all of the discretized units:

$$\mathfrak{F}_z = \sum_{n=1}^N \rho_{lmn} \left[ \frac{e^{iq_z n\Delta z}}{iq_z} - \frac{e^{iq_z (n-1)\Delta z}}{iq_z} \right] \quad (3.12)$$

Now, to reduce the number of operations, we can pull out commonalities within the summation to give:

$$\mathfrak{F}_z = \left( \frac{1}{iq_z} \right) (1 - e^{-iq_z z}) \left[ \sum_{n=1}^N \rho_{lmn} e^{iq_z n \Delta z} \right] \quad (3.13)$$

Because each component can be treated separately, the demonstrated treatment of  $\mathfrak{F}_z$  can be applied to  $\mathfrak{F}_x$  and  $\mathfrak{F}_y$  which yields the form factor:

$$\mathfrak{F} = \left( \frac{1}{iq_x} \right) \left( \frac{1}{iq_y} \right) \left( \frac{1}{iq_z} \right) (1 - e^{-iq_x x}) (1 - e^{-iq_y y}) (1 - e^{-iq_z z}) \left[ \sum_{n=1}^N \rho_{lmn} e^{iq_x l \Delta x} e^{iq_y m \Delta y} e^{iq_z n \Delta z} \right] \quad (3.14)$$

It now becomes clear that the calculation may be treated as the Fourier transform of the scattering potential. Computationally, this simplifies the calculation by utilizing build-in mathematical function like the `numpy.fft()` which can be found in the `numpy` library and solves the fast Fourier transform (FFT). While the FFT significantly reduces calculation time by utilizing the Fourier transform symmetry, the resulting matrix is both sparse in reciprocal space and solves an incredibly large region of  $Q$ , most of which is unnecessary when modeling real scattering data. The matrix sparseness can potentially be resolved by zero padding the function; however, the zero padding required to obtain solutions for  $Q$  points needed to model data creates a matrix so large that the computational resources required to store and interact with it are prohibitively limiting. To quickly and efficiently solve the Fourier transform for a specific region of  $Q$  space, we utilize the Chirp-z transform [24]. To be thorough, the formula can now write as:

$$\mathfrak{F} = \left( \frac{1}{iq_x} \right) \left( \frac{1}{iq_y} \right) \left( \frac{1}{iq_z} \right) (1 - e^{-iq_x x}) (1 - e^{-iq_y y}) (1 - e^{-iq_z z}) * CZT \rho_{lmn} \quad (3.15)$$

This result is used in the Born approximation form factor calculation. Now that a form factor has been developed we may turn our attention to the structure factor,  $\mathfrak{S}$ , which will be used to describe how the form repeats itself across the sample.

### 3.3 Structure Factor

Mathematically, the scattering is calculated over all space. In practice; however, scattering only occurs over the probed sample volume which is limited to the size of the beam footprint. Furthermore, if only coherent scattering is considered, which is the volume over which the neutron coherence length extends in the x, y and z directions, then the integral volume is decreased even further (assuming that the beam footprint is larger than the neutron beam coherence length, which is generally the case. In this work, we assume that, under the probed area, the features are uniformly repeating which leads to the simple formula:

$$\mathfrak{S} = \left[ \frac{\sin((N_x q_x D_x)/2)}{\sin((q_x D_x)/2)} \right]^2 \left[ \frac{\sin((N_y q_y D_y)/2)}{\sin((q_y D_y)/2)} \right]^2 \left[ \frac{\sin((N_z q_z D_z)/2)}{\sin((q_z D_z)/2)} \right]^2 \quad (3.16)$$

where  $D$  is the the size of the unit cell and  $N$  is the number of repeats which are scattering coherently. Figure 3.4A shows the results for this calculation. Between the diffraction peaks there is calculated scattering which is not observed in the real scattering data. This is due to the implicit assumption that the system is coherently scattering evenly across the sample until the coherence limit where it sharply becomes zero. This treatment not only results in a theory function exhibiting

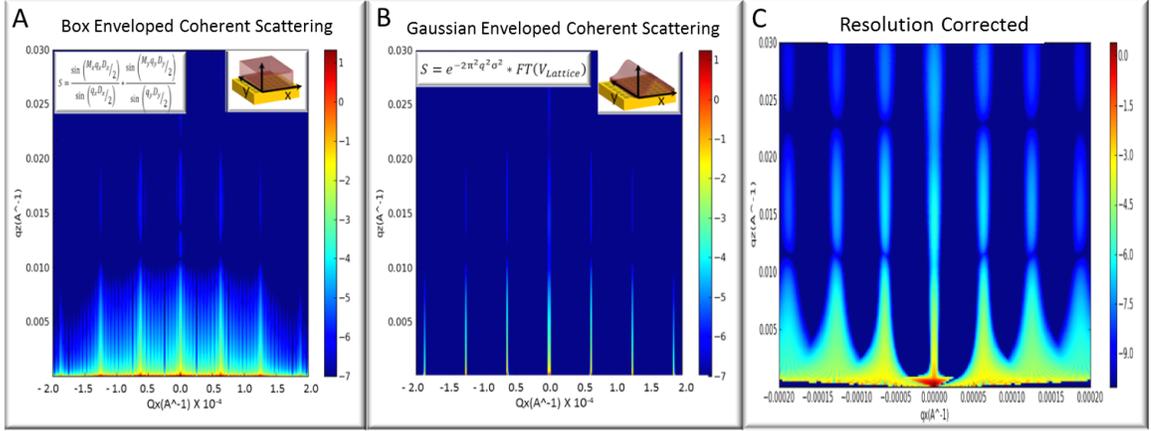


Figure 3.4: Demonstration of the different structure factor calculations.

scattering that does not exist, but also does not fundamentally represent the actual scattering.

In reality, the coherent scattering is a convolution of a Gaussian with the diffraction rods of the repeat structure, or:

$$\mathfrak{S} = e^{-2\pi q^2 \sigma^2} * FT(V_{lattice}) \quad (3.17)$$

where the exponential is the Fourier transform of the Gaussian,  $\sigma$  is the length over which the system is coherently scattering, and  $V$  represents the lattice structure. The results of this calculation are shown in figure 3.4B and the final resolution corrected scattering calculation (in the Born approximation) is shown in figure 3.4C. This structure factor is a much more realistic representation of the scattering from a repeating unit cell and will be used for the modeling in chapter 5.

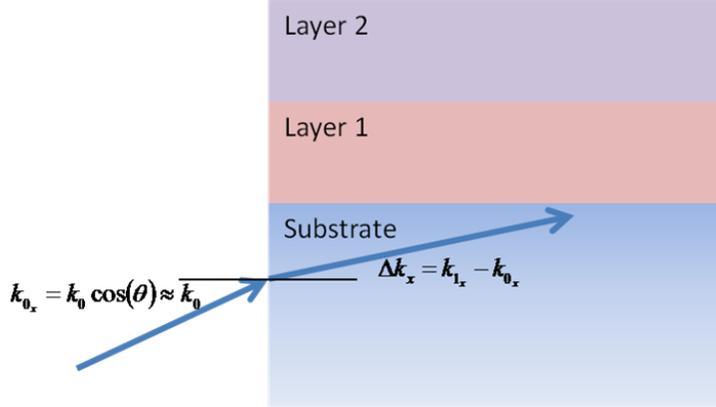


Figure 3.5: Illustration of the refractive shift on a data set taken on gold pillars on the AND/R reflectometer.

### 3.4 Refractive Shift

An interesting and somewhat surprising experimental observation was a refractive shift due to the neutron beam impinging on the substrate at a near orthogonal incident angle below the 'horizon' of the sample. The 'horizon' is defined as the points where  $\theta_i = 0$  and  $\theta_f = 0$ . The refractive shift can be observed in real data and is shown in figure 3.6.

This refraction ends up being simple to derive.

starting with the equation:

$$k_1 = nk_0 = \sqrt{\left(\frac{4\pi\rho}{k_0^2}\right)}k_0 \quad (3.18)$$

which can be rearranged to:

$$k_1 = \sqrt{k_0^2 - 4\pi\rho} \quad (3.19)$$

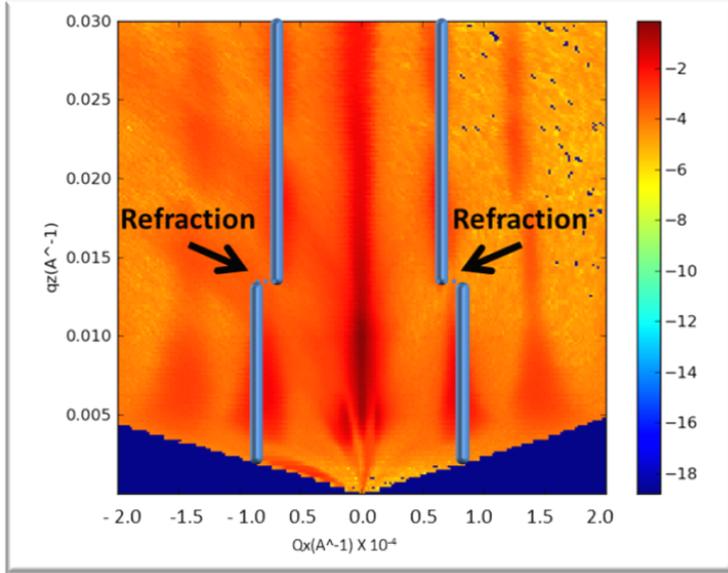


Figure 3.6: A real data set with the refractive shift marked.

As shown in figure 3.5 we need to solve:

$$\Delta k_x = k_{1_x} - k_{0_x} \quad (3.20)$$

This can be substituted into the previous equation to get:

$$\Delta k_x = \sqrt{(k_0^2 - 4\pi\rho)} - k_{0_x} \quad (3.21)$$

The factoring here is complicated but  $k_0 \gg \rho$  so doing a series expansion at  $k_0 = \infty$  gives:

$$\sqrt{k_0^2 - 4\pi\rho} \approx k_{0_x} - \frac{2\pi\rho_1}{k_{0_x}} \quad (3.22)$$

and can write:

$$\Delta k_x \approx \left( k_{0_x} - \frac{2\pi\rho_1}{k_{0_x}} \right) - k_{0_x} \quad (3.23)$$

This can be simplified to:

$$\Delta k_x \approx k_{0_x} - \frac{2\pi\rho_1}{k_{0_x}} - k_{0_x} \quad (3.24)$$

$$\Delta k_x \approx - \left( \frac{2\pi\rho_1}{k_{0_x}} \right) \quad (3.25)$$

because its is known that:

$$k_{0_x} = \frac{2\pi}{\lambda} \quad (3.26)$$

the final result is:

$$\Delta k_x \approx - \left( \frac{2\pi\rho_1}{\frac{2\pi}{\lambda}} \right) \quad (3.27)$$

$$\Delta k_x \approx -\lambda\rho_1 \quad (3.28)$$

where the red indicates cancellations in the equations.

Here the resulting refractive shift for the  $q_x$  direction is observed. Furthermore, it is somewhat evident that, when the beam exits the substrate at a nearly orthogonal angle, the opposite shift will occur which can be calculated as:

$$\Delta k_x \approx \lambda\rho_1 \quad (3.29)$$

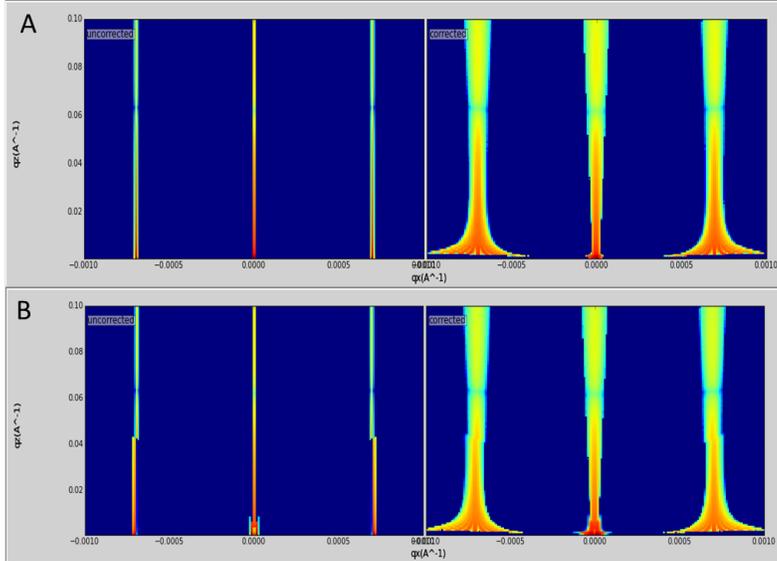


Figure 3.7: A real data set with the refractive shift marked.

### 3.5 Substrate Modified Born Approximation Form Factor

As discussed in 3.2, the assumptions made in the first order kinematic approximation are only accurate at large  $Q$  vectors where the dynamic scattering contributes less significantly to the overall scattering. In addition, other effects such as a suppression of the neutron beam intensity at the 'horizons'<sup>1</sup> cannot be reproduced by the BA. One step toward improving the modeling is to assume that some perturbation of the wavefunction occurs as a result of interaction with the sample. The substrate modified Born approximation (SMBA) is a distorted wave Born approximation whereby the wavefunction is perturbed by the substrate/incident media interface. This calculation takes the form:

<sup>1</sup>The 'horizons' are defined as the position in  $q$  where either the incoming or outgoing angle are orthogonal to the substrate. This regime generally exhibits significant dynamical scattering effects which need to be captured by the modeling.

$$\mathfrak{F} = \iiint_0^{D_{xyz}} \Psi_{i_z} \rho (e^{iq_x x} e^{iq_y y} e^{iq_z z}) \Psi_{f_z}^* dD_{xyz} \quad (3.30)$$

where:

$$\Psi_i = t e^{ik_{i_z} z} \quad k_{i_z} < 0 \quad (3.31)$$

$$\Psi_i = e^{ik_{i_z} z} + r e^{-ik_{i_z} z} \quad k_{i_z} > 0 \quad (3.32)$$

$$\Psi_f^* = e^{-ik_{f_z} z} + r e^{ik_{f_z} z} \quad k_{f_z} > 0 \quad (3.33)$$

$$\Psi_f^* = t e^{ik_{f_z} z} \quad k_{f_z} < 0 \quad (3.34)$$

where the solution to the wavefunction only considers the effects caused by the incident media and substrate. Because this particular approximation does not concern itself with what happens inside the sample, only the total reflected and transmitted intensity needs to be solve. Starting with the standard matrix form,  $r$  and  $t$  may be solved for using the  $k_i$  or  $k_f$  as the wave vector depending on which wavefunction is being solved. In this form we extract a factor of  $k_0$  so the matrix is in terms of the refractive index  $n$ . The algebra can be carried out starting with:

$$\begin{pmatrix} t \\ in_{sub} t \end{pmatrix} e^{ik_{sub} L} = M \begin{pmatrix} 1 + r \\ in_{inc} (1 - r) \end{pmatrix} \quad (3.35)$$

$r$  must now be extracted from the matrix. This can be rewrite as:

$$M_{11}(1 + r) + M_{12} in_{inc}(1 - r) = t e^{ik_{sub} L} \quad (3.36)$$

$$M_{21}(1 + r) + M_{22} in_{inc}(1 - r) = in_{sub} t e^{ik_{sub} L} \quad (3.37)$$

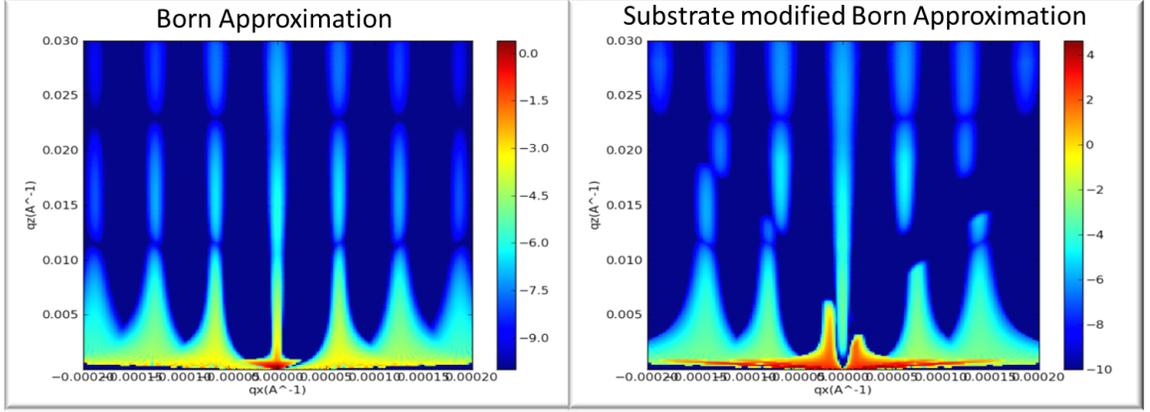


Figure 3.8: A direct comparison of the BA and the SMBA theory function results for a model of square pillars.

The  $t$  dependence may be removed by setting the two equations equal to each other, leaving only  $r$ :

$$M_{11}(1 + r) + M_{12}in_{inc}(1 - r) = te^{ik_{sub}L} = \frac{M_{21}(1 + r)}{in_{sub}} + \frac{M_{22}in_{inc}(1 - r)}{in_{sub}} \quad (3.38)$$

Now the equation may be distributed to get all of the  $r$  values to one side.

Once rearranged and canceled the result is:

$$r = \frac{M_{11} + in_{inc}M_{12} + \frac{-1}{in_{inc}} (M_{21} + in_{inc}M_{22})}{-M_{11} + in_{inc}M_{12} + \frac{1}{in_{sub}} (M_{21} - in_{inc}M_{22})} \quad (3.39)$$

With this equation, the solution to the wavefunction may be solved and used to perturb the scattering calculated by the Born approximation.

Figure 3.8 shows the improvements after perturbing the BA by the wavefunction. The 'horizons' now are represented; however, they are different widths which does not match the data obtained from the samples fabricated in chapter 4. Also, the peak locations are the same as those calculated for the BA, which are known to

be incorrect. Still, The application of the wavefunction shows how it contributes to the scattering results.

### 3.6 Distorted Wave Born Approximation

The math involved for the DWBA calculation is substantially more complicated than that used in any of the other approximations, but is currently the most accurate treatment for calculating off-specular scattering. This section will cover the math used for the DWBA implemented in this software. for the most part, notation in this section will be borrowed from Kentzinger et al. [14] and one may use this reference to supplement understanding of the formalism [14]. It starts by envisioning the sample as a set of two different scattering potentials. written as:

$$\hat{\mathbb{V}}_{\rho} = \hat{V}_l + \hat{V}_{l_{\rho}} \tag{3.40}$$

The  $\rho$  in this equation represents a dependence on the in-plane coordinates of layer,  $l$ .  $\hat{V}_l$  is called the reference potential and is the in-plane average of layer,  $l$ , which, as expected, has no dependence on the in-plane coordinates, ie. the average is the same throughout the plane. The vector,  $\hat{V}_{l_{\rho}}$  is called the residual potential and is the difference between the full potential and the reference potential or:

$$\hat{V}_{l_{\rho}} = \hat{\mathbb{V}}_{\rho} - \hat{V}_l \tag{3.41}$$

which can easily be seen from rearranging the previous equation. The importance of this potential lies in it being the contributor to the in-plane and, therefore,

the off-specular scattering. This will be seen in the formalism. Also, the residual potential is defined so that:

$$\left\langle \hat{V}_{l\rho} \right\rangle_{\rho} = 0 \quad (3.42)$$

which just says that the in-plane average of the residuals is 0.

Now:

$$\hat{\mathbb{F}} = \left\langle \psi_{f0}(\mathbf{k}_f, 0) | \hat{F}_{fi} | \psi_{i0}(\mathbf{k}_i, 0) \right\rangle \quad (3.43)$$

must be solved for, which is written here in the bra-ket notation. In this case, the bra and ket pertain to the spin states of the neutron and can be ignored, which means:

$$\mathbb{I} = |\hat{\mathbb{F}}|^2 \quad (3.44)$$

where  $\mathbb{I}$  is the intensity of the scattering cross-section. This is more complicated than it looks. We can break up  $\hat{\mathbb{F}}$  into:

$$\hat{\mathbb{F}} = \sum_l \hat{c}_{fl} F_{fi}^{tt} \hat{c}_{il} + \hat{c}_{fl} F_{fi}^{tr} \hat{d}_{il} + \hat{d}_{fl} F_{fi}^{rt} \hat{c}_{il} + \hat{d}_{fl} F_{fi}^{rr} \hat{d}_{il} \quad (3.45)$$

This equations deserves some explanation. First, this is the sum over all of the layers in the sample. For each layer,  $l$ , we want to sum over all of the factors contributing to the scattering. The subscript  $f$  is designating the outgoing wave and  $i$  is designating the incoming wave. The  $F$  components contain the information about the sample which is interacting with the wavefunction. So the the first term can be said to be representing the scattering events whereby the transmitted wave

in layer  $l$  of the incoming wave ( $c_{fl}$ ) interacts with the sample,  $F$ , and scatters into an outgoing wave of  $c_{fl}$ . Each of the rest of the terms can be defined similarly. The  $c$  and  $d$  components are simply the amplitudes of the wavefunction. We have seen this before in chapter 1 as the  $c$  and  $d$  in 1.23. By following the derivations both in chapter 1 and appendix A for the matrix in 1.27, One can use the specular reflectivity calculation to determine almost directly the values for  $c$  and  $d$ .

If the results of the matrix transfer are defined more appropriately in terms of the solution to the wavefunction and the derivative:

$$\begin{pmatrix} \Psi_{l+1} \\ \frac{1}{k_0} \Psi'_{l+1} \end{pmatrix} = M_l \begin{pmatrix} \Psi_{l+1} \\ \frac{1}{k_0} \Psi'_{l+1} \end{pmatrix} \quad (3.46)$$

then the wavefunction and its derivative may be written (as is shown in the derivation in 1):

$$\Psi = ce^{ikz} + de^{-ikz} \quad (3.47)$$

$$\Psi' = ikce^{ikz} - ikde^{-ikz} \quad (3.48)$$

Now the algebra can be carried out to solve for  $c$  and  $d$  (not forgetting the  $\frac{1}{k_0}$  term). Rearranging the wavefunction to solve for  $c$  produces:

$$c = \Psi e^{-ikz} - de^{-2ikz} \quad (3.49)$$

and solving for  $d$  in the derivative gives:

$$d = ce^{2ikz} - \frac{k_0\Psi'e^{ikz}}{ik} \quad (3.50)$$

The two rearranged equations can now be combined to form:

$$c = \Psi e^{-ikz} - \left( ce^{2ikz} - \frac{k_0\Psi'e^{ikz}}{ik} \right) e^{-2ikz} \quad (3.51)$$

and distributing the exponential:

$$c = \Psi e^{-ikz} - ce^{2ikz} e^{-2ikz} - \frac{k_0\Psi'e^{ikz}}{ik} e^{-2ikz} \quad (3.52)$$

The exponentials cancel leaving:

$$c = \Psi e^{-ikz} - c - \frac{k_0\Psi'}{ik} e^{-ikz} \quad (3.53)$$

or

$$2c = \Psi e^{-ikz} - \frac{k_0\Psi'}{ik} e^{-ikz} \quad (3.54)$$

From 1 we recall that:

$$k_l = k_0 n_l \quad (3.55)$$

so  $k_0$  can be removed leaving:

$$2c = \Psi e^{-ikz} - \frac{\Psi'}{in} e^{-ikz} \quad (3.56)$$

Cleaning up this equation gives:

$$c = .5e^{-ikz} \left( \Psi - \frac{\Psi'}{in} \right) \quad (3.57)$$

Carrying out the same procedure for d results in:

$$d = .5e^{-ikz} \left( \Psi + \frac{\Psi'}{in} \right) \quad (3.58)$$

Interestingly, the derivations seen in appendix A are simply used to factor out the values which we now need for this equation. we can use equation 3.57 and 3.58 to solve the c and d values in this equation for each layer. Now that the formalism needed to extract the wavefunction amplitudes has been derived, the F component must be solved. This function is defined as:

$$\hat{F}_{fil}^{\alpha\beta} = e^{\pm ik_{\alpha l} z} g_{fil}^{\alpha\beta} \hat{F}_l(Q_{||}) e^{\pm ik_{\beta l} z} \quad (3.59)$$

where  $\alpha$  and  $\beta$  are each of the respective c and d values and the  $\pm$  is + for the transmitted component and - for the reflected component. g is the Laue function written as:

$$g_{fil}^{\alpha\beta} = \frac{e^{iq_{fil}^{\alpha\beta} D_l} - 1}{iq_{fil}^{\alpha\beta}} \quad (3.60)$$

where  $D_l$  is the thickness of layer l. There are a large number of superscripts and subscripts here but all have the traditional meaning and are only trying to indicate that q is being specified very rigidly. The  $fi$  dependence can be observed in the calculation of q as:

$$q_{fil}^{cc} = p_{fl} + p_{il} \quad (3.61)$$

$$q_{fil}^{cd} = p_{fl} - p_{il} \quad (3.62)$$

$$q_{fil}^{dc} = -p_{fl} + p_{il} \quad (3.63)$$

$$q_{fil}^{dd} = -p_{fl} - p_{il} \quad (3.64)$$

Here, the p variables are just reformed versions of the wave vectors and can be solved as:

$$\hat{p}_{al} = \sqrt{(k_d \sin(\theta_d))^2 - (\sqrt{4\pi\rho_l})^2} \quad (3.65)$$

where d is being used to designate either i or f and the  $\theta$  is either the incoming or outgoing angle. These two components have also been observed in various forms throughout chapter 1. One difference is that the z component of the wave vector now depends on the layer depth. The second difference is an obvious cancellation of the square and square root. This part of the equation was left in expanded form to show that it is the formula for the location of the critical edge as was written in the specular reflectivity portion of chapter 1. Using this equation, all of the q values and, therefore, all of the components in the equation 3.61 may be solved for. The last piece of the equation needed to be solve is  $\hat{F}_l(Q_{||})$ .

This component is where all of the scattering potential information is provided. As an important reminder, the scattering potential is split into two components. Both will be used to solve this function. The equation can be written as:

$$\hat{F}_l(Q_{\parallel}) = \rho_l \hat{1} F_l(Q_{\parallel}) \quad (3.66)$$

where  $F_l(Q_{\parallel})$  is the lateral Fourier transform of the residual potential and can be written as:

$$F_l(Q_{\parallel}) = -\frac{m}{2\pi\hbar^2} \int d\rho e^{-iQ_{\parallel}\rho} \frac{\tilde{\rho}_l(\rho)}{\rho_l} \quad (3.67)$$

Unfortunately, keeping with the convention used in the cited paper means the value for  $\rho$  used in these equations become quite confusing. Equation 3.67 indicates that the in-plane Fourier transform of the residual potential for a given layer ( $\tilde{\rho}_l$ ) relative to the reference potential for that layer ( $\rho_l$ ) is needed. This result is then multiplied by the reference potential in equation 3.66 to give the potential needed for equation 3.61. Conceptually, the  $z$  dependence in the potential is removed, the in-plane Fourier transform taken, and then the  $z$  dependent potential is brought back for the rest of the scattering calculation. This is how the  $q_z$  scattering calculated by this approximation becomes more accurate than the traditional BA. With all of these functions defined, the non-magnetic off-specular scattering can be calculated. It is important to note that the specular reflectivity will not be produced from this calculation and must be explicitly placed into the scattering results at  $q_x = 0$ . This presents somewhat of an issue with matching the relative intensities of the specular and off-specular results and is the most obvious deficiency of the DWBA. Still, the ability to represent key features in scattering data make it a very attractive candidate for modeling off-specular scattering.

### 3.7 Magnetic First Order Born Approximation

The first order Born approximation discussed in 3.2 can be modified to include magnetic scattering. Because neutron reflectometry is only sensitive to in-plane magnetization (magnetization which is perpendicular the the wave vector transfer direction) the Halpern-Johnson vector can be used to remove the magnetization component in the direction of the wave vector transfer from the total magnetization. This vector is defined as:

$$\hat{\mathbf{q}} = \hat{M} - \hat{Q}(\hat{Q} \cdot \hat{M}) \quad (3.68)$$

where  $\hat{Q}$  is the unit vector of the wave vector transfer and  $\hat{M}$  is the magnetization vector. As with any vector, the Halpern-Johnson vector can be broken up into its 3 directional components. Because the three orthogonal coordinates are defined by the neutron polarization direction, they will be defined as  $\hat{\mathbf{q}}_a$ ,  $\hat{\mathbf{q}}_b$  and  $\hat{\mathbf{q}}_c$ , where  $\hat{\mathbf{q}}_a$  is the vector direction aligned with the external field, H. First, the Halpern-Johnson vector is calculated for the x, y and z coordinates:

$$\hat{\mathbf{q}}_x = \hat{M}_x - \hat{Q}_x(\hat{Q}_x \cdot \hat{M}_x) \quad (3.69)$$

$$\hat{\mathbf{q}}_y = \hat{M}_y - \hat{Q}_y(\hat{Q}_y \cdot \hat{M}_y) \quad (3.70)$$

$$\hat{\mathbf{q}}_z = \hat{M}_z - \hat{Q}_z(\hat{Q}_z \cdot \hat{M}_z) \quad (3.71)$$

Once these vectors are calculated, they can be transformed to match the orientation of the neutron reflectometer. If the magnetic moment,  $\hat{M}$ , is aligned in the

y direction then we can use the vectors:

$$a = [010] \quad (3.72)$$

$$b = [001] \quad (3.73)$$

$$c = [100] \quad (3.74)$$

results in:

$$\hat{\mathbf{q}}_a = \hat{\mathbf{q}}_x * a_1 + \hat{\mathbf{q}}_y * a_2 + \hat{\mathbf{q}}_z * a_3 \quad (3.75)$$

$$\hat{\mathbf{q}}_b = \hat{\mathbf{q}}_x * b_1 + \hat{\mathbf{q}}_y * b_2 + \hat{\mathbf{q}}_z * b_3 \quad (3.76)$$

$$\hat{\mathbf{q}}_c = \hat{\mathbf{q}}_x * c_1 + \hat{\mathbf{q}}_y * c_2 + \hat{\mathbf{q}}_z * c_3 \quad (3.77)$$

Once this vector is obtained, the reflectivity for the non-spin-flip cross-sections can be written as:

$$r^{\uparrow\uparrow, \downarrow\downarrow} = \sum_l \left[ (\rho_{nuc,l} \mp \mathbf{q}_a \rho_{mag,l}) e^{i\vec{Q} \cdot \vec{R}_l} \right] \quad (3.78)$$

and the spin-flip cross-sections, which only depend on the magnetic scattering length density, can be obtained with:

$$r^{\uparrow\downarrow, \downarrow\uparrow} = \sum_l \left[ (\mathbf{q}_b \pm i\mathbf{q}_c) \rho_{mag,l} e^{i\vec{Q} \cdot \vec{R}_l} \right] \quad (3.79)$$

By using this simplified calculation for modeling magnetic data, the qualitative differences in scattering depending on magnetic field alignment may be determined.

## Chapter 4

### Sample Fabrication and Data Acquisition

#### 4.1 Overview

Because the calculations employed in this software are all approximations and hold inherent (and sometimes unrealistic) assumption about the propagation of the wave, experimental systems are required to understand both the capabilities and the limitations of each approximation. The fabricated systems were designed to scatter where the instrumentation is most sensitive, providing an ideal system for use in analyzing the models. In addition, a collection of Ni gratings were fabricated to elucidate information about the neutron coherence length. All of these samples were fabricated using standard lithography processes in either the University of Maryland FabLab or the NIST Center for Nanoscale Science and Technology. Presented here are the details of the fabrication process.

#### 4.2 Non-magnetic system

The project started by looking at systems that were commercially available. An optical grating was purchased from Edmond Scientific to see if the scattering from such a system would yield a well defined off-specular pattern. The results from this system are shown in figure 4.1. The system is a sinusoidal grating with epoxy

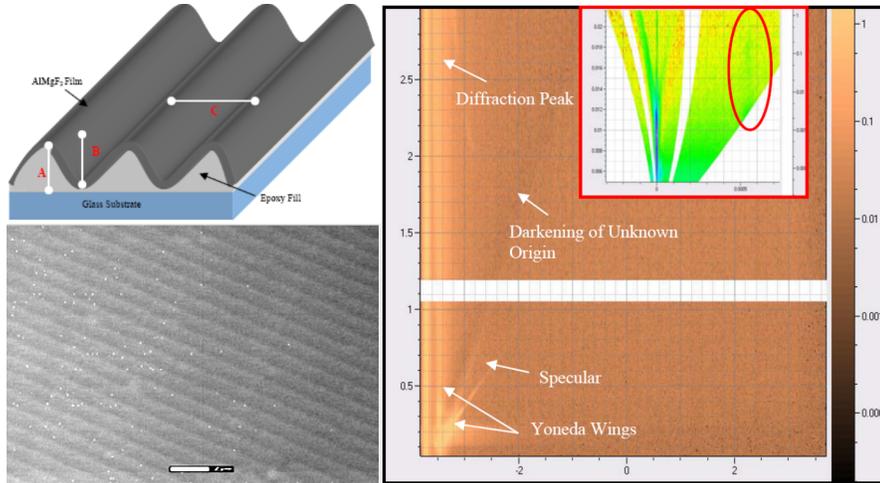


Figure 4.1: The results from the first sample measured for off-specular scattering.  $A = 2778$  : height from top of glass substrate to top of film.  $B = 200.0$  : twice the amplitude of the wave.  $C = 8733$  : wavelength.

filler and a  $AlMgF_2$  coating. This design presented a variety of issues. First, the sinusoidal shape resulted in an effective density gradient in the  $q_z$  direction, which produces complicated scattering. Second, the hydrogen rich epoxy filling caused the system to scatter incoherently. Third, the shape lends itself to complicated dynamical scattering effects which are not so prevalent in other shape forms. Finally, the grating period of  $873.3nm$  is relatively small, resulting in off-specular scattering at very high  $q_x$ , where the instrumental sensitivity is low. As can be seen in figure 4.1, some off-specular scattering was measured; however, it became clear that no commercial product fitting our criteria were available. We had to fabricate our own.

The first fabricated system was a collection of Au pillars on a Cr/Si substrate. This sample was used to gain information about the non-magnetic off-specular scat-

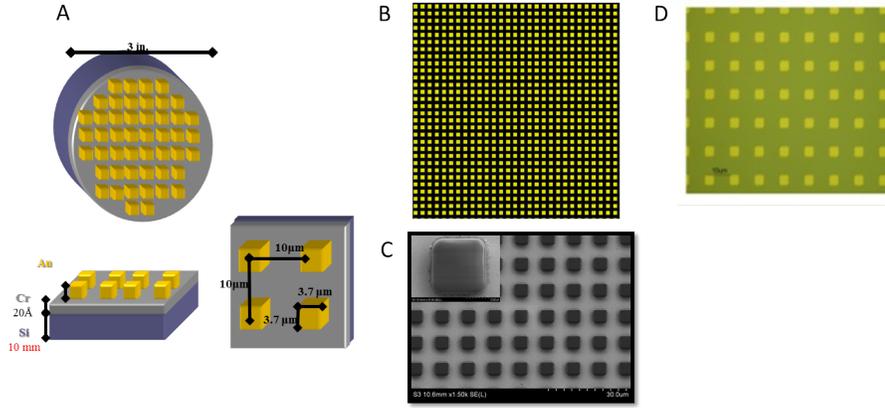


Figure 4.2: The general progression of the Au sample fabrication. A) A schematic of the sample. B) An optical microscopy image of the mask used to pattern the sample. C) The S1813 layer after UV exposure. D) Optical image of the final sample.

tering. The general fabrication process is shown in figure 4.2A. The sample was fabricated on a 3 inch, 10mm silicon substrate to ensure that the sample did not bow once in the sample holder. Figure 4.2B shows an image of the mask used to create the sample. The spacing was chosen to be  $10\mu m$  because in reciprocal space:

$$\frac{2\pi}{1.0 * 10^5 \text{Å}} = 6.28 * 10^{-5} \text{Å}^{-1} \quad (4.1)$$

which is the location of the first order diffraction peak. This  $q_x$  position is well above the instrumental resolution, which makes it easy to resolve from the specular peak, but is close enough to  $q_x = 0.0$  that substantial scattered intensity is observed.

A  $20\text{Å}$  chromium adhesion layer was deposited using a Denton Discovery 550 at the CNST. This was followed by a layer of gold deposited by the same method. A layer of hexamethyldisilazane (HMDS) was spin-coated onto the substrate to promote mask adhesion. Negative photo resist was then spincoated onto the metal.

The resist was exposed using a Suss Microtec M6 aligner, also located at the CNST. Figure 4.2C shows the photoresist after development. It is important to note that, due to light leakage at the corners, the parallelepiped systems has rounded corners. This is unavoidable with the available mask and must be accounted for in the modeled.

Once exposed, the resist was developed in the appropriate developer and rinsed thoroughly. The Au was then etched using an aqua regia solution ( $HNO_3 : 3HCl : 20H_2O$ ) until the Cr adhesion layer was visible. Wave dispersive spectroscopy was used to ensure that the Cr layer did not etch with the Au. Because this process involved a wet etch, there was substantial feature size variation across the 3 inches of substrate. This is primarily due to solution saturation where larger amounts of etch metal are located. At the center of the wafer, a larger amount of Au is being etched but fresh etchant cannot be cycled as quickly because of etch solution on the outer area of the wafer. This results in a slower etch rate at the center of the wafer, which is common for large area wet etch processes. Agitation can mitigate some of this rate differential; however, it can never be fully eliminated. This is less than ideal, and the issue is solved in later samples with a lift-off process. Figure 4.2D shows an optical microscopy image of the sample after etching.

Once the sample was fabricated, scattering data was taken on the Advanced Neutron Diffractometer/Reflectometer (AND/R) and the NIST Center for Neutron Research (NCNR)[7] using a position sensitive detector (PSD).

Figure 4.3 shows the process of data reduction. First, a single data point is measured on the PSD. This is a  $\theta_{out}$  plot for a given  $\theta_{in}$  corresponding to the x and

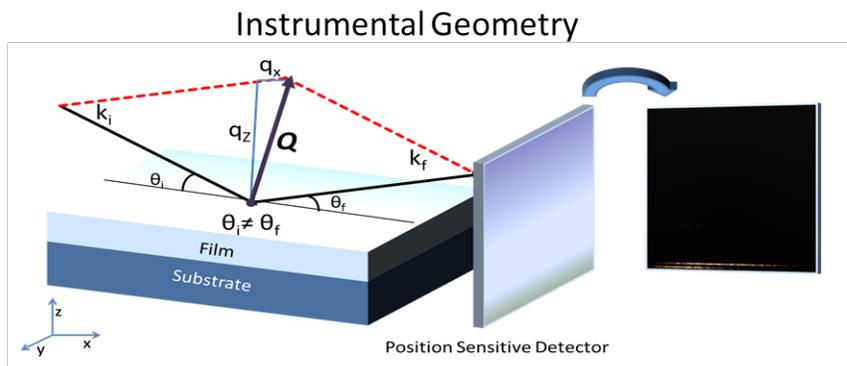
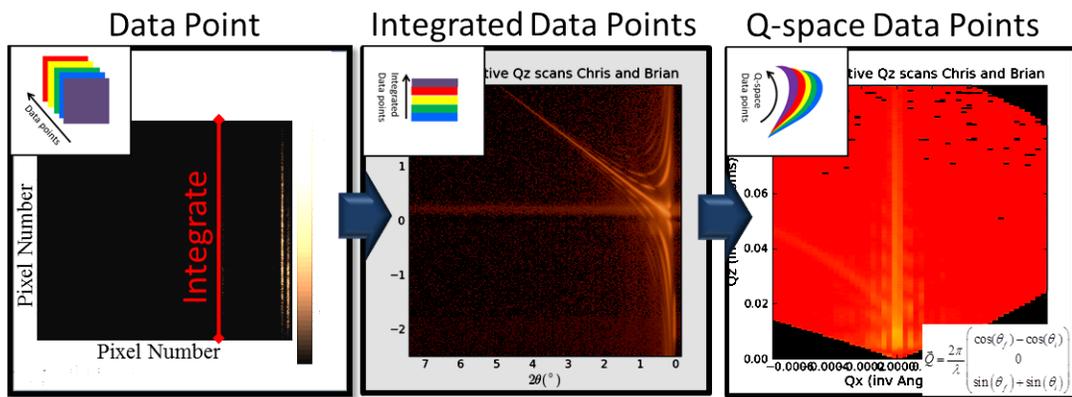


Figure 4.3: The data reduction process.

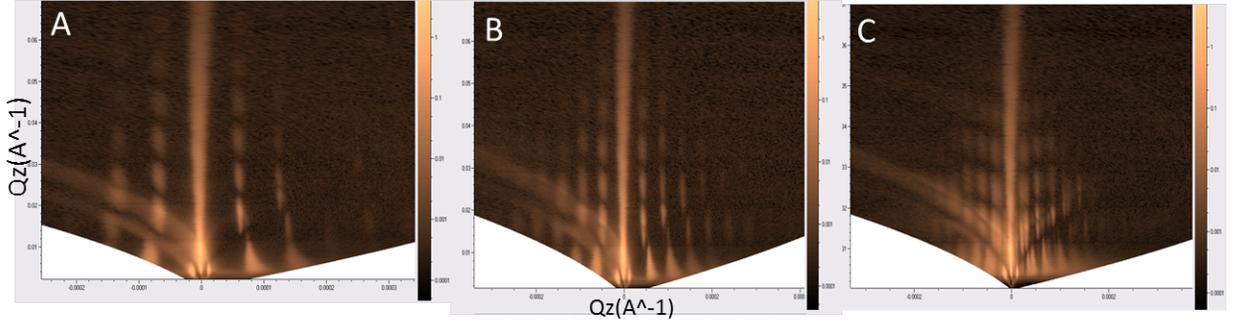


Figure 4.4: Data from the Au sample at A)  $\phi = 0$ , B)  $\phi = 45$ , C)  $\phi = 30$ .

y real space directions. The figure shows that in the vertical direction the intensity extends a significant distance on the detector. This is due to the poor resolution in the y direction, which increases the beam intensity impinging on the sample and reduces count times. Because the resolution is poor in this direction, the data point is integrated. This set of integrated points is plotted as a function of  $\theta_{in}$  versus  $\theta_{out}$ . Because data interpretation is much more intuitive in a  $q_x$  versus  $q_z$  plot, the data is then converted to reciprocal space.

A variety of measurements were taken on the Au films. Figure 4.4 shows a selection of data taken from the Au sample as a function of in-plane rotation,  $\phi$ .

### 4.3 Magnetic system

A permalloy sample of the same type as the Au was fabricated to measure the magnetic scattering. For this sample, the same mask was used; however, a lift-off process was applied. First, a layer of MicroChem LOR-2A was spin-coated onto a 10mm silicon wafer. This layer was baked, followed by a layer of Shipley S1813 positive photoresist. The resist was then exposed and developed in a similar

manner as the Au sample. The LOR layer develops away, leaving a S1813 lip which was used to help ensure complete liftoff. Next, a  $1050\text{\AA}$  layer of permalloy was deposited onto the sample at the University of Maryland FabLab using the Denton Ebeam/thermal evaporator. A crystal monitor was used to track the metal deposition thickness. Finally, the sample was soaked in PG stripper. This removed the polymer and any permalloy which was deposited on it, leaving only the permalloy features which were deposited where the photomask had been removed. This method of lithography ensures that there is minimal size variation between features across the wafer by removing the metal wet etch step. Unfortunately, as seen in figure 4.5, the mask design did not take into account over-deposition from the somewhat anisotropic sputtering deposition. This was a trade-off for mask re-usability for both lift-off and standard lithography methods as each method would require different mask corrections. Still, both the Object Oriented MicroMagnetic Framework loader found in section 2.6 and the software infrastructure have the ability to handle such complexities. For a more accurate representation, 4 overlaid Cylinder objects were used. The modeling is shown in chapter 5.

## 4.4 Magnetic Gratings

For all models used in this software, knowledge about the neutron beam coherence length is required. The coherence length dictates the number of features the neutron beam is probing and is important for modeling coherent versus incoherent scattering. A more complete description of the coherence length and the studies

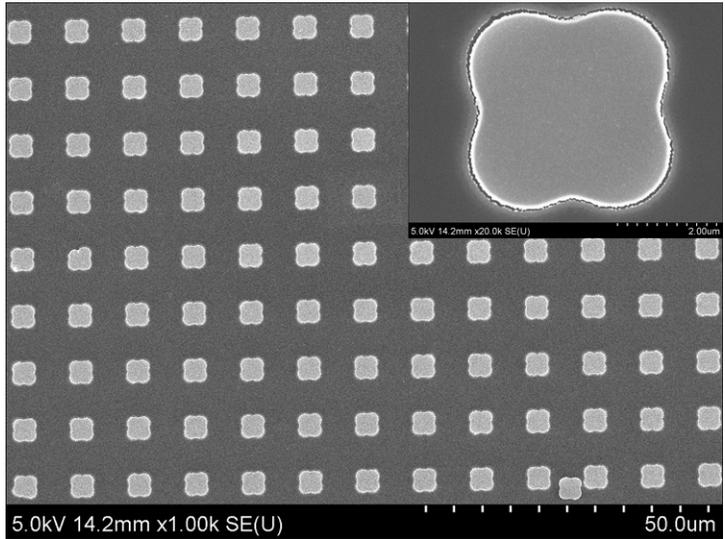


Figure 4.5: SEM images of the permalloy samples.

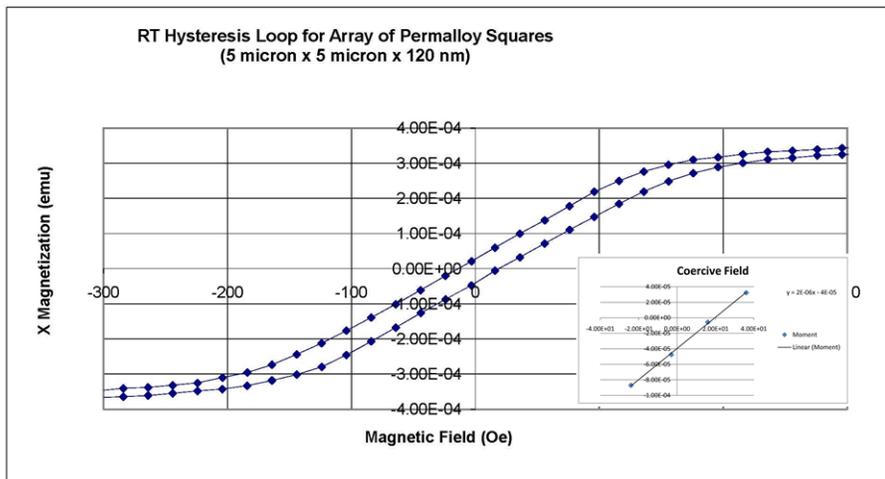


Figure 4.6: The coercivity and hysteresis of the permalloy sample.

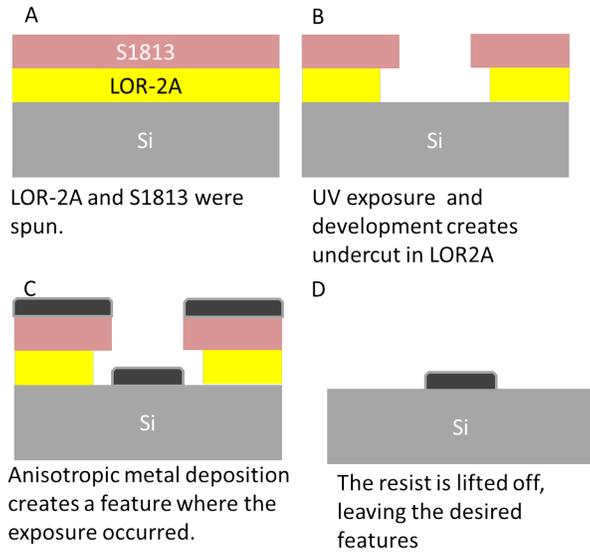


Figure 4.7: Schematic of the lift-off process.

carried out to determine it may be found in chapter 6.

For the coherence length studies carried out toward the end of this project, grating samples were required. The principles used to determine the coherence length can be found in chapter 6. Because the data analysis involved with the determination of the neutron beam coherence length assumes a perfect repeating structure, sample uniformity was critical. The lift-off process had proven to be a successful fabrication technique for such uniformity requirements and was used in the construction of Ni gratings. The grating sizes were fabricated from  $1600\mu m$  down to  $.606\mu m$  (all values are the grating period). They were produced in 4 different sets with two different methods. The  $1600\mu m$ ,  $800\mu m$ ,  $400\mu m$ ,  $200\mu m$ ,  $100\mu m$ ,  $50\mu m$ , and  $25\mu m$  were fabricated using plastic masks from Output City inc. and were exposed on the Karl Suss MJB-3 Mask aligner in the UMD FabLab. LOR-2A was used as the spacing layer with S1813 as the photosensitive layer. Deposition

for the  $1600\mu m$  to the  $100\mu m$  were made using the Denton Discovery 550 at the CNST because the sputtered films were determined by X-ray reflectometry to be much smoother than those from e-beam evaporation. The Denton Discovery sputtering system produced films with approximately  $30\text{\AA}$  of roughness while the e-beam evaporation system had roughnesses of  $50\text{\AA}$  to  $60\text{\AA}$ . Unfortunately, the isotropic deposition of the sputtering deposition technique prevented the smaller gratings from being fabricated in this manner. Because the lift-off technique is dependent on the the deposition being somewhat anisotropic (preferentially directional in its deposition: see figure 4.7) if the deposition is too isotropic, the metal film does not create two separate pieces (deposition on the polymer and on the substrate) but rather one continuous film which will either lift the feature off the substrate or prevent the polymer from lifting off the unwanted metal. The limit for the CNST sputtering system was found to be about  $100\mu m$ . For smaller grating periods, Ni was deposited by either the Denton ebeam/thermal evaporator or the Temescal ebeam deposition system at the UMD FabLab. Although these film qualities were not as good as the sputtering system, they were sufficiently smooth and showed good scattering results.

Although the plastic masks were claimed to be good down to  $10\mu m$ , by  $12.5\mu m$  the feature roughness introduced by the printing quality (dots per square inch) resulted in very poor gratings. To make smaller gratings, quartz masks were needed. a  $10\mu m$  and  $5\mu m$  mask was purchased from compugraphix. The exposure was carried out using hard and vacuum contact on the EVG 620 mask aligner at the UMD FabLab. Other than increased rinse times, the procedure was the same as for the larger masks. At this size the gratings started to deviate somewhat from the

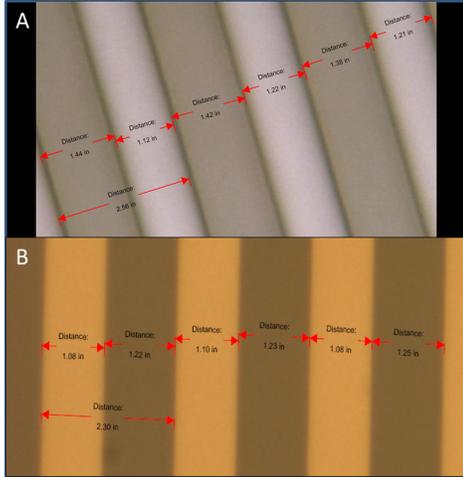


Figure 4.8: Width variation calculated for the  $5\mu\text{m}$  grating on the A) mask and B) actual Ni grating.

intended 50/50 ratio of Ni to empty space. This is illustrated in figure 4.8. The next mask included both  $2\mu\text{m}$  and  $1\mu\text{m}$  gratings and attempted to compensate for the deviation due to exposure edge effects by having  $2.2\mu\text{m}$  to  $1.8\mu\text{m}$  and  $1.2\mu\text{m}$  to  $0.8\mu\text{m}$  compositions. Figure 4.8 is representative of the grating quality for all gratings and so only the  $5\mu\text{m}$  is shown explicitly.

The final sample was a  $606\text{nm}$  grating which had to be fabricated using nanoimprinting. the nanoimprint stamp was purchased from Lightsmyth inc. and was coated with an anti-stick layer using the Nanonex Ultra-100. The imprint was then performed using the Nanonex NX-2000. Both systems are located at the CNST. The nanoimprint technique generally utilizes a dry etch rather than lift-off to fabricate the system and, because Ni does not have a good dry etchant, SiN was used as the grating layer. As seen in figure 4.9, the gratings were far lower in quality than those produced by lift-off; a factor of both the technique and the size

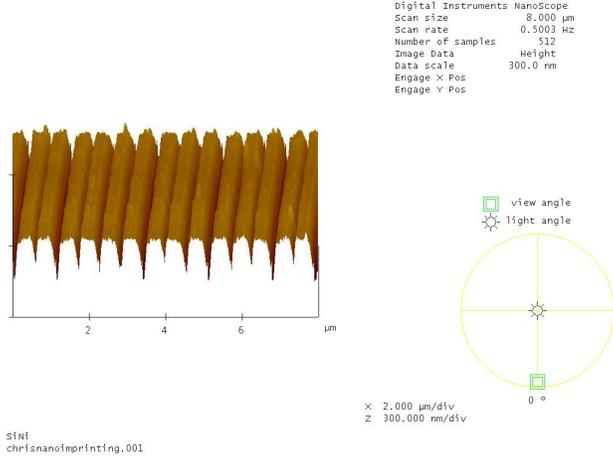


Figure 4.9: AFM of the nanoimprinted sample.

scale. Although, these gratings could not provide quantitative information about the coherence length, they were of sufficient quality to determine whether or not the coherence limit had been crossed.

## 4.5 Data Conversion

The data reduction process is somewhat crude at this point but still allows the user to load .cg1 files from AND/R and convert the data to Q space.

The rebinning process micro-slices the theta space array to allow for a more accurate rebinning procedure. It uses the formula:

$$\vec{Q} = \frac{2\pi}{\lambda} \begin{pmatrix} \cos(\theta_f) - \cos(\theta_i) \\ 0 \\ \sin(\theta_f) + \sin(\theta_i) \end{pmatrix} \quad (4.2)$$

where the middle component of the matrix is 0 because there is no  $q_y$  component. The data loading process is shown in figure 4.10

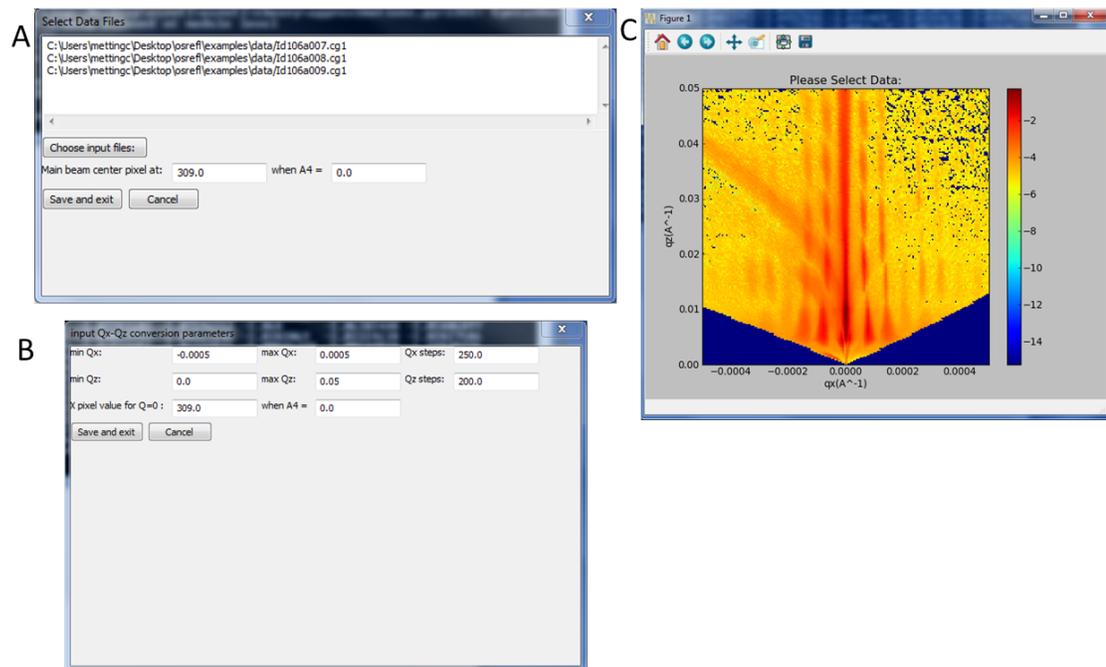


Figure 4.10: The GUI used to load data. A) first, the data files are selected. B) Then the specific values needed to rebin the data to Q space is entered. C) Finally, the user can select a subset of the data that he or she wishes to model.

## Chapter 5

### Modeling

#### 5.1 Overview

Now that a software infrastructure has been developed, the formalism presented in chapter 3 can be used to model the real systems which were fabricated in chapter 4. The real system referred to in the following models is the 5 micron x 5 micron Au pillar system in the 10 micron x 10 micron lattice. The sample was fabricated through standard lithography techniques and was etched using aqua regia.

Because of the broad range of models supported by the software, the sensitivity of the algorithms to specific parameters can be determined. Although the fitting infrastructure has not been implemented, the software is designed to allow for the implementation of fitting algorithms with relatively little effort.

#### 5.2 Shape Differentiation

In samples with larger repeating structures, it is often important to differentiate between feature shapes. For example, in the case of the wet etched Au parallelepiped pillars described in section 5.1, the degree of corners rounding due to the wet etch process should manifest itself in the resulting scattering data. To

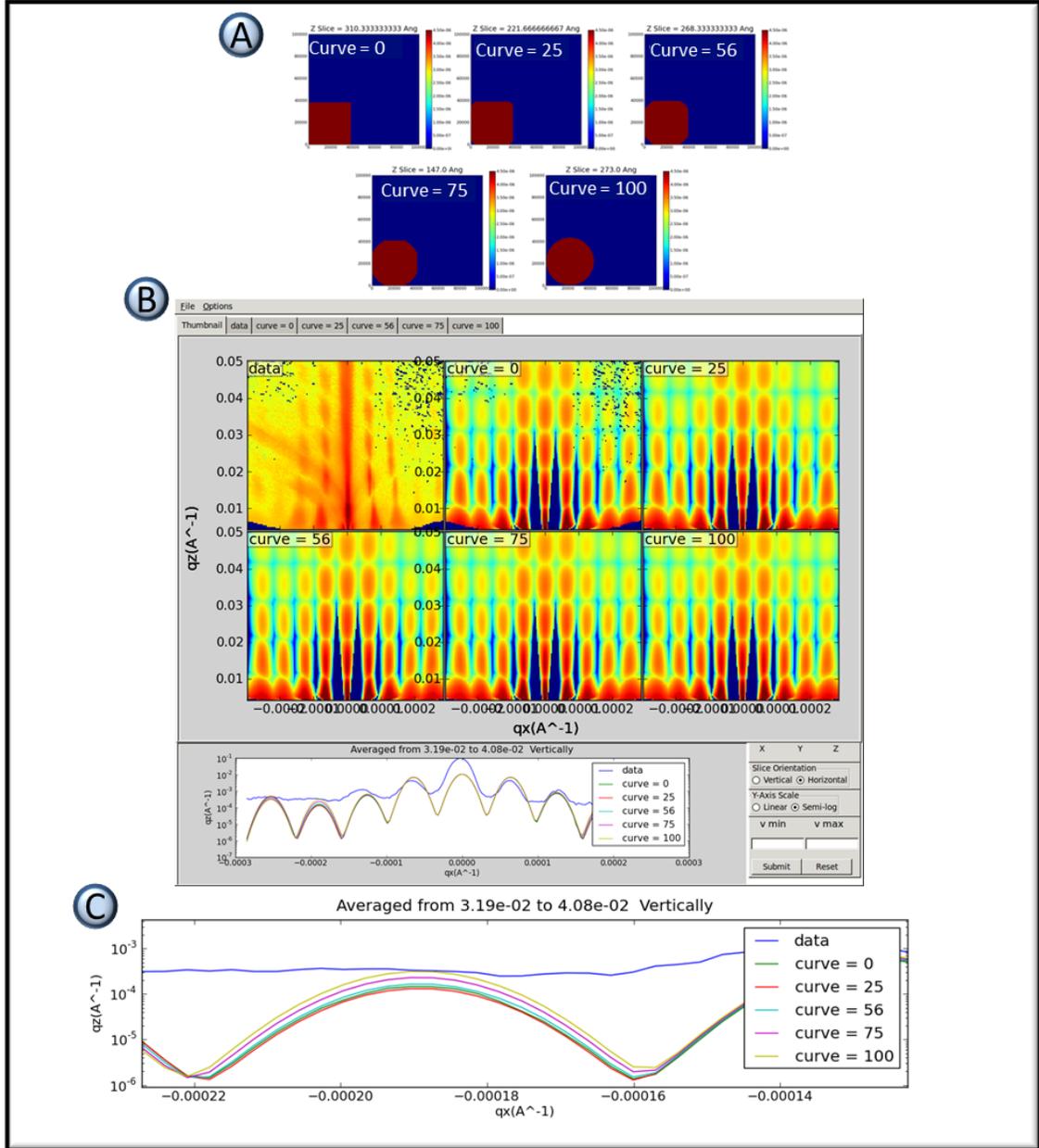


Figure 5.1: The shapes tested all filled the same volume of the unit cell. A) an xy cross-section of the shape being modeled. B) Real data and the theory functions calculated for each shape. C) An enlarged image of the third diffraction peak showing differences in intensity.

determine the sensitivity of off-specular reflectometry to these small variations, simulations within the Born approximation were calculated. Figure 5.1 shows the results of this study. Figure 5.1A shows five models with varying degrees of curvature. For each of these models, the discretization size is kept constant to eliminate the influences of element (in the finite element process) resolution on the scattering. As the corners are rounded, the feature's total volume fill decreases. Because this decrease in volume fill can also change the calculated scattering results and adds a variable which is not explicitly being tested, the size of the features are progressively increased so that all model features have the same total material volume fill. The 'curve' value expressed in figure 5.1A refers to a normalized degree of curvature which is defined during the model building process. The possible values go from 0, totally square, to 100, totally rounded. More details on this parameter can be found in section 2.3.

Once the models were built, they were compared to the gold pillar scattering data collected of the samples discussed in section 5.1. Figure 5.1B shows the resulting theory function calculations and a  $q_x$  slice for comparison. The scattering is scaled to the first order diffraction peak and the  $q_x$  slice is taken well above the 'horizons' <sup>1</sup>. In general, the models produce similar results; however, in the third order diffraction peak, there is a significant degree of variation between the five models. Figure 5.1B shows a magnified view of this peak. As the curvature decreases, the third order diffraction peak becomes increasingly suppressed. The third order diffraction peak is also found to be below the background of the instrumentation. This indicates that, even for such large feature spacings, it is not possible

to distinguish between the different features experimentally. As the feature spacing decreases, the diffraction peaks would push to larger  $q_x$  values, making shape differentiation even more difficult.

The conclusion from this study is that, for feature curvature, it is not possible with the currently available reflectometry to easily distinguish between different models at the feature sizes and spacings discussed in this model system. It may, however, be possible with future reflectometers or features with larger spacings to distinguish between different degrees of curvature.

### 5.3 Modeling using the Born Approximation

The first algorithm to evaluate is also the simplest. The kinematic or Born approximation assumes no perturbation of the wavefunction inside the sample as is discussed in 3.2. The modeling was carried out and can be seen in figure 5.2.

Some important features of the model are highlighted and labeled using lettered circles. Figure 5.2B is indicating where the 'horizons' are on the data. In the data, there is a suppression of the scattered intensity along the 'horizons' which is not observed in the BA modeling results. This is expected as the intensity suppression is primarily a result of wave perturbation by the sample. Figure 5.2A indicates the region which is being averaged over in the 1D plot. This box shows that the averaged intensity is over both refracted and non-refracted data as discussed in section

---

<sup>1</sup>The 'horizons' are defined as the position in  $q$  where either the incoming or outgoing angle are orthogonal to the substrate. This regime generally exhibits significant dynamical scattering effects which need to be captured by the modeling.

3.4. The corresponding area in the 1D plot can be seen at figure 5.2E. The averaging produces an effective splitting of the peaks; however, looking at the 2D data the peaks are clearly an artifact produced by the refractive shift. The average is taken over such a large area because the fringing in the  $Q_z$  direction do not match well at low  $Q$ , which can make model comparisons difficult. Although the refractive shift is not inherent to the Born approximation it must still be considered in the modeling. Figure 5.2F shows the peak splitting, which is a result of the refractive shift. By figure 5.2G the scattering is almost completely due to the refracted scattering and the peak positions no longer match.

Figure 5.2H shows the background level from that data. The background is preventing the modeling of the sixth and seventh diffraction peak. This modeling provides a glimpse of how much information is lost under the instrumental background. It is clear that this technique is significantly limited by this background and that further work is needed to improve the instrumentation.

Figure 5.3 shows modeling of the sample data modeled in figure 5.2; however, in this model the refractive shift is now applied below the 'horizons'. Because the refraction caused by entering and leaving the substrate at approximately an orthogonal angle to the substrate sidewall below the 'horizons' can be calculated separately from the scattering calculation, the refraction can be applied directly to the BA without altering the approximation formalism. Once applied, the diffraction peak position below the 'horizons' are positioned properly in  $q_x$ . One distinct error is that the peaks seem to overlap in  $q_z$  at the 'horizons' which is not how they appear in the data. This is because at the 'horizons' the scattering intensity is supposed to

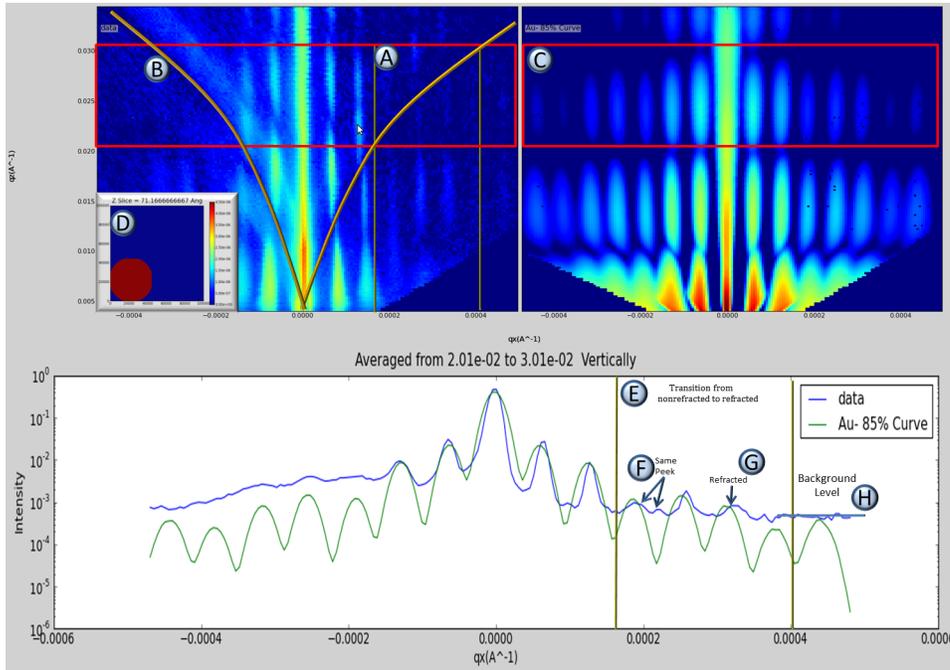


Figure 5.2: Theory function calculation using the Born Approximation.

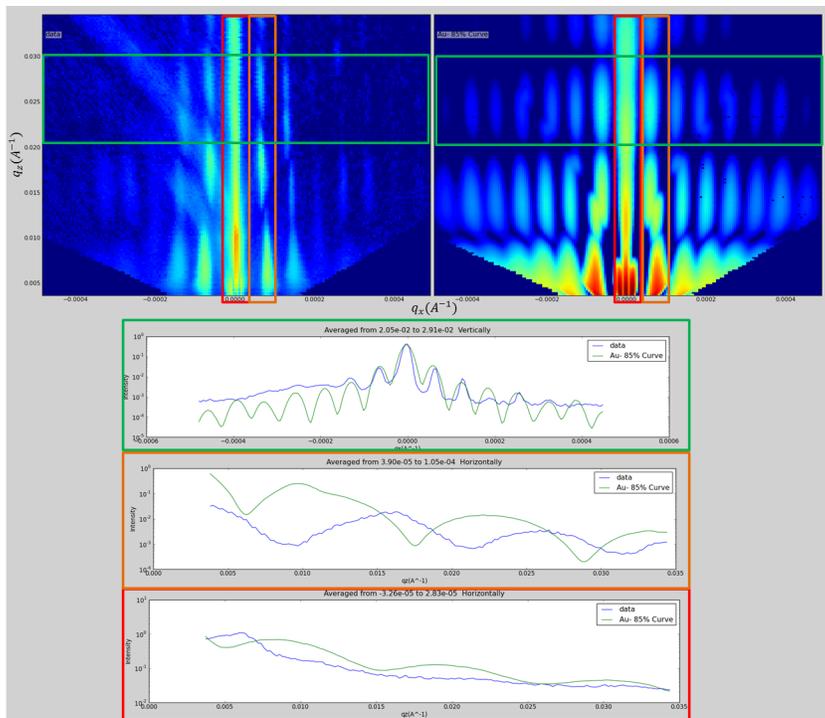


Figure 5.3: Modeling using the Born Approximation with slices in both  $q_x$  and  $q_z$ .

be suppressed; a phenomenon that the BA cannot capture. To appropriately match this scattering characteristic, a more robust calculation is needed.

Figure 5.3 also provides slices in both the  $q_x$  and  $q_z$  directions. The model feature height is determined experimentally from profilometry measurements taken on the sample and was validated with the DWBA calculation. Although the peak fringing in the  $q_z$  direction does not align with the data, the peak spacings are as expected for a feature of this thickness. The mismatch between the data and the theory is primarily due to the BA inability to reproduce substrate effects such as the critical edge and total reflection. The  $q_x$  slice now matches the diffraction peak positions where scattering is primarily caused by refracted intensity. The point at which the data is transitioning between scattered and refracted scattered intensity in the averaged slice is still not well modeled because the 'horizons' are not being suppressed.

Some of the deficiencies seen in the Born approximation modeling may be overcome by applying a simple perturbation to the wavefunction. The results of this approximation are evaluated in the next section.

## 5.4 Modeling using the Substrate Modified Born Approximation

As described in chapter 4, the SMBA is the Born approximation algorithm where the wavefunction is perturbed by the neutron beam interaction with the incident media/substrate interface. The most notable result of this perturbation is the effects observed at the 'horizons' as is illustrated in figure 5.4. Comparing figure

5.4 with figure 5.3, the duplicated peaks in  $q_x$  slices near the 'horizon' are no longer observed.

This model is more representative of the real system but there are clearly inaccuracies in this approximation. The 'horizon' asymmetry indicated by the brown areas marked in figure 5.4 in the theory function calculation is a result of the critical edge. At positive  $q_x$  scattering, the 'horizon' is the result of an air-to-silicon critical edge. at negative  $q_x$  the sample stack is inverted and the critical edge is now the result of silicon-to-air interface. The extreme asymmetry in the theory function is not observed in the data. Comparing the results of the slices in  $q_x$  and  $q_z$  to those in the BA, the same inaccuracies are observed. The  $q_z$  marked in figure 5.4 with an orange box exhibits the same phase misalignment observed in the BA and the  $q_x$  slice indicated in figure 5.4 by the green box match the correct peak positions but still do not match the scattering intensity and structure where the intensity is averaging over both refracted and non-refracted scattering (at approximately  $q_x = 0.0003A$ ). A more rigorous treatment is needed to better match these areas.

## 5.5 Modeling using the Distorted Wave Born Approximation

As described in chapter 3, the full distorted wave Born approximation implementation involves perturbing the off-specular scattering by the solution to the wavefunction inside each layer of the sample. Because the solution cannot be solved explicitly at the off-specular angles, the solution for the wavefunction at the specular scattering is used instead. This calculation cannot reproduce the specular scatter-

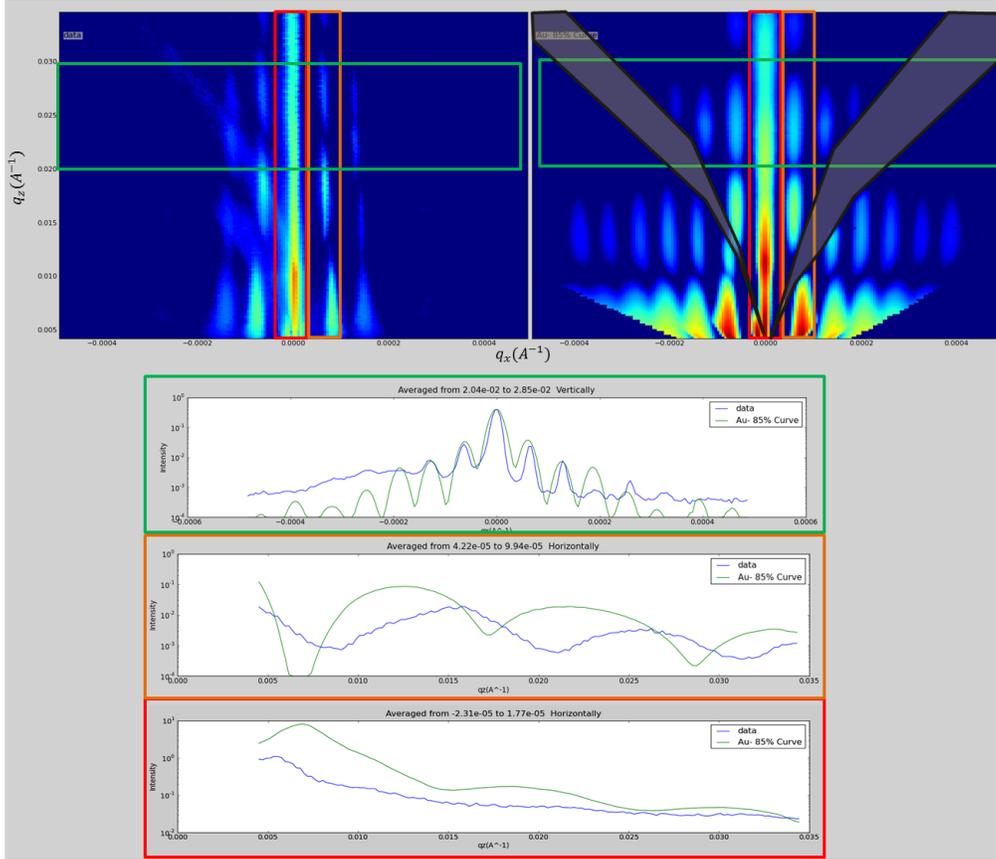


Figure 5.4: Modeling using the substrate modified Born approximation. The green box marks the integrated area for the  $q_x$  slice, the red box indicates the integrated area for the specular  $q_z$  data, and the orange box is the integrated  $q_z$  peak for the first order diffraction peak.

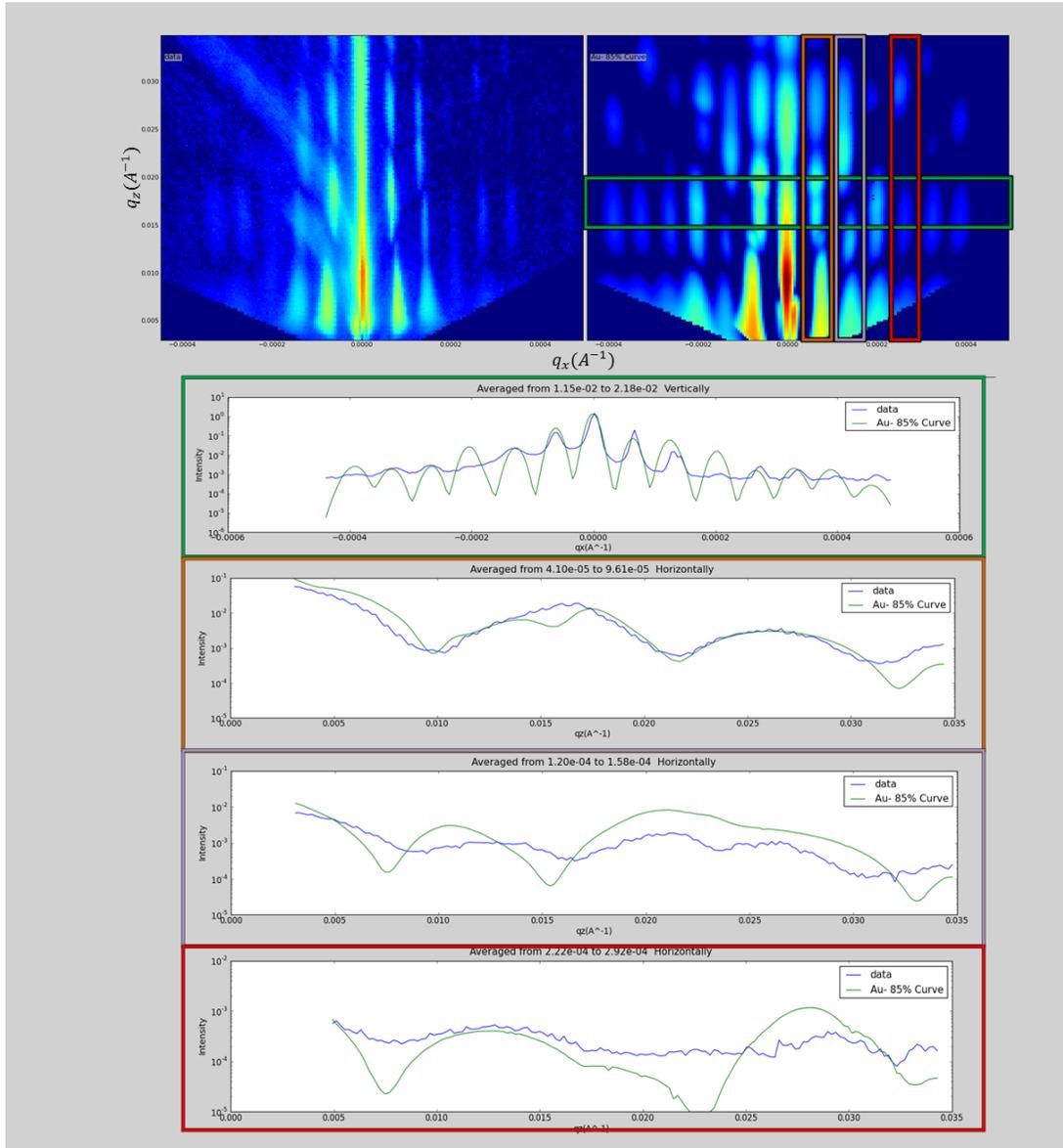


Figure 5.5: Off-specular scattering calculation within the DWBA. The specular scattering is not included in this calculation. The calculation does not include any disorder. The averaged slices show good agreement between theory and experimentation.

ing and so the specular scattering must be included independently. To clarify, the scattering at  $q_x = 0$  in the calculated theory function used in this software is really the in-plane scattering which has been resolution corrected. The software does not yet include the specular scattering because current forms of the theory do not provide a way to explicitly scale the off-specular scattering to the specular scattering. In general, empirical methods are used to appropriately scale the two components [14]. Figure 5.5 shows the results for the off-specular scattering calculation without the specular scattering included. The scattering is empirically scaled to the second order diffraction peak in the data. The parameters used to solve this scattering are the same as those used in the BA and the SMBA. The substantial improvement in agreement between data and theory at low  $q$  illustrates the advantages of using the DWBA. The  $q_z$  slice long the first order diffraction peak (orange box) are well-aligned and even show some minor structure ( $q_z = 0.015$ ) observed in the data. Similar results are observed in the second order diffraction peak (purple box) where, although the theory produces much more well-defined peaks, the profiles match quite well. The third order diffraction peak (red box) starts to be significantly effected by the instrumental background. The  $q_x$  matches well until the third order diffraction peak. This third order peak intensity mismatch is present in all of the modeling and is difficult capture. The nature of the third order peak suppression is unclear and needs further work for an explanation. Because the models in this software do not yet include parameters for disorder, many of the deviation between the theory and data may be due to disorder in the feature lattice and roughness between the substrate/feature and feature/air interfaces. Also, given more time and computational

resources, it is possible that better fit could be obtained.

## 5.6 Modeling Magnetic Samples

Because neutrons are sensitive to magnetic fields, it is important to be able to model magnetic samples. The magnetic scattering is more complicated and requires much more resources to carry out. To test the magnetic modeling capabilities a square lattice of permalloy parallelepiped features was fabricated as discussed in chapter 4. The fabrication processes produced rounded edges and dimples on the sides of the features. Fortunately, the versatility of this software allows this rounding and the dimples to be modeled. Figure 5.6 shows the fabricated sample and how the sample was modeled in the software.

The software will utilize functionality from the Object Oriented Micro Magnetic Framework (OOMMF) software package as discussed in chapter 2. The ability to create OOMMF input files and load OOMMF output files provides a way to model magnetic samples based on powerful magnetic moment minimization software. The software takes the minimized unit cell and translates the resulting moments into a magnetic scattering length density which may then be used by the modeling algorithm. More details on this process may be found in section 2.6.

The DWBA may be used to accurately determine the feature height through modeling. The DWBA results shown in figure 5.7 agree with the crystal monitor and indicate the feature heights are  $1100\text{\AA}$ . The model also shows the data to be of poor quality which will make modeling difficult.

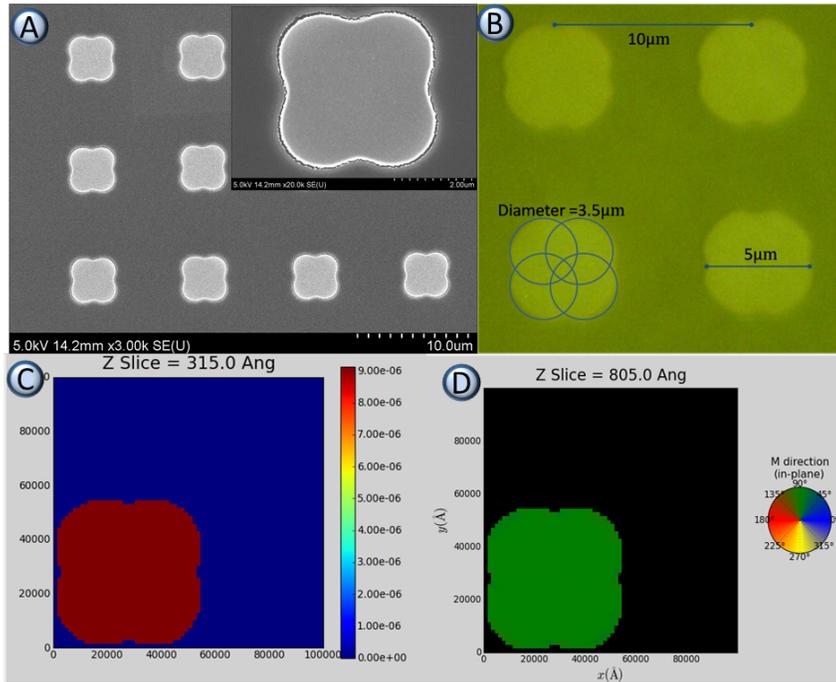


Figure 5.6: A) An SEM image of the features produced by the fabrication process. B) an optical image of the features. C) a slice of the scattering length density array used for modeling D) a slice of the energy minimized magnetic moment array as produced by the OOMMF software for the saturated state and imported into the off-specular software.

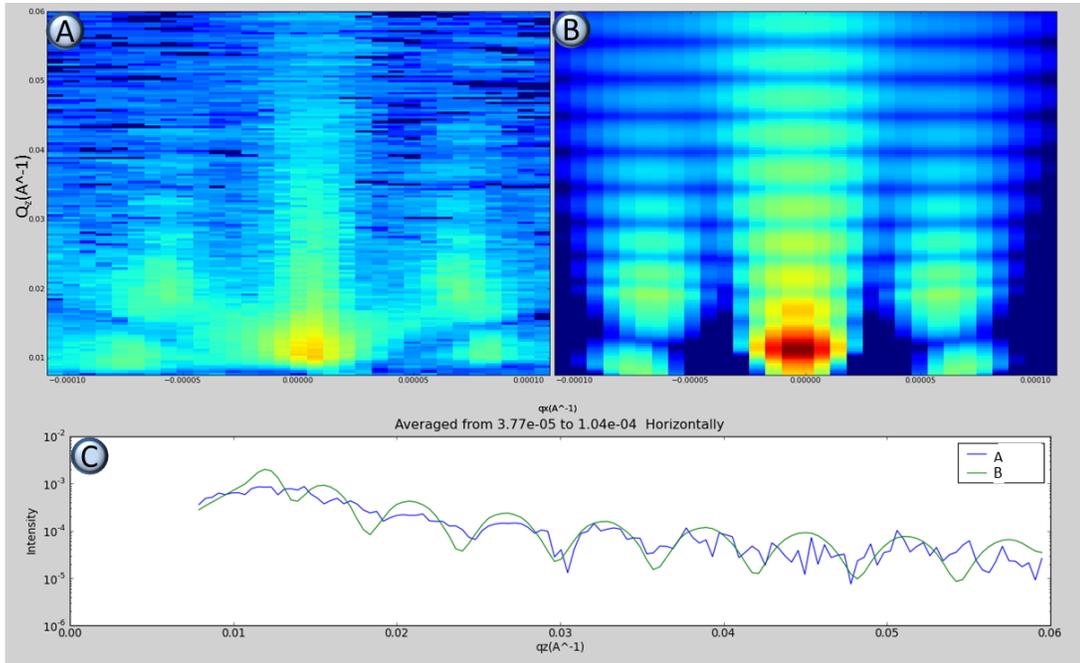


Figure 5.7: This models the data using the features illustrated in figure 5.6 with a feature height of  $1100\text{Å}$ . A) ++ magnetic cross-section of the permalloy features under saturating magnetic field. B) Unpolarized DWBA results. C) a slice of the first order diffraction peak.

Now that the appropriate parameters have been accurately determined using the DWBA, the magnetic BA can be solved for. Figure 5.8 shows the results from the model. Because there is not a large spin-flip contribution to the scattering, the theory function does not indicate significant scattering, as is observed in the data. In general, the theory functions used to model the real data have a data 'floor' or lower limit placed on them because the  $q$  range for which the instrumentation is sensitive to is much narrower than that which can be calculated. To better illustrate that scattering is being calculated for the spin-flip states but at intensity that the instrument is not sensitive to (below background), figure 5.8B shows the calculation without adding a lower limit to the data values.

To better understand the accuracies and deficiencies of the resulting calculation, vertical and horizontal slices can be compared. Figure 5.9 shows a vertical slice of the data. This comparison shows that the data is of fairly poor quality and does not give a good indication of the differences between the  $++$  and  $-$  scattering. Also, the scattering for this data is measured at lower  $q_z$  values which means the BA is far worse at representing the data results than that of the Au features. This sample modeling would benefit significantly from a magnetic DWBA calculation which has yet to be implemented.

The horizontal slice shown in figure 5.10 indicates that the magnetic calculation does provide some insight into the sample's magnetic characteristics. The differences between the  $++$  and  $--$  cross-sections are captured by the theory function and compare quite well to each other. This agreement shows that OOMMF has minimized the saturated magnetic state appropriately. It also shows that the

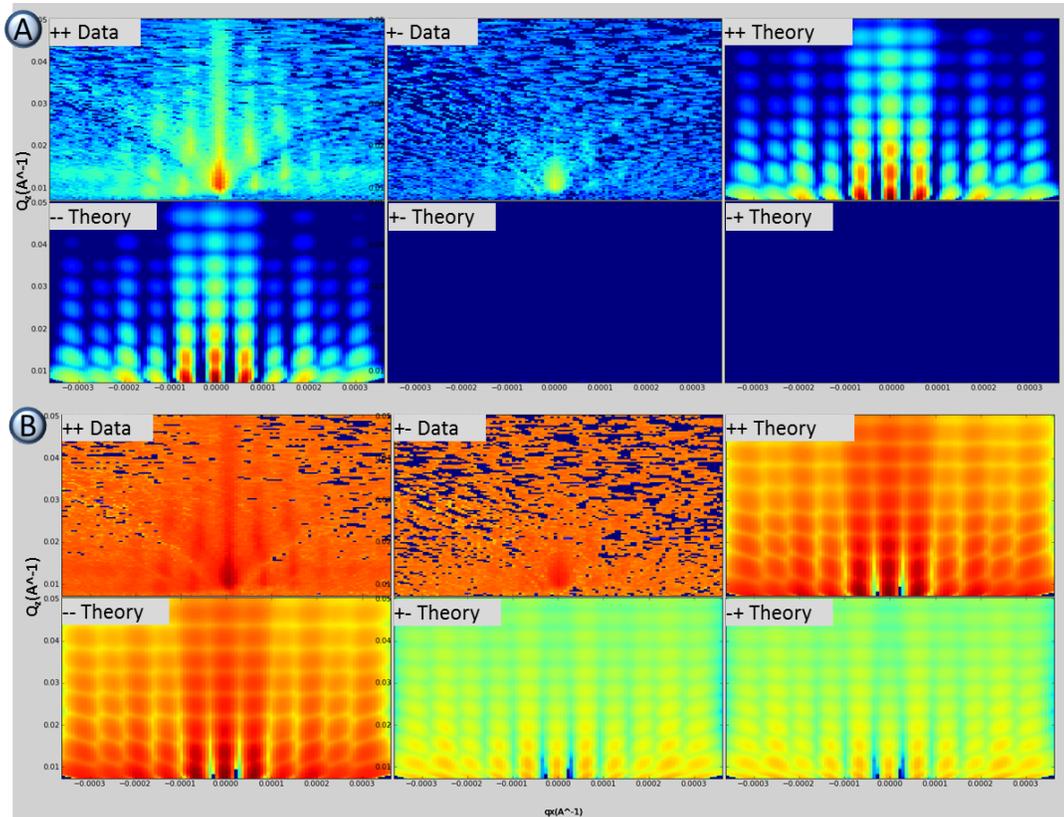


Figure 5.8: Spin flip and non spin flip cross-section data for the permalloy samples and their corresponding theory functions. A) with a data floor to match the instrumental data range and B) no data floor.

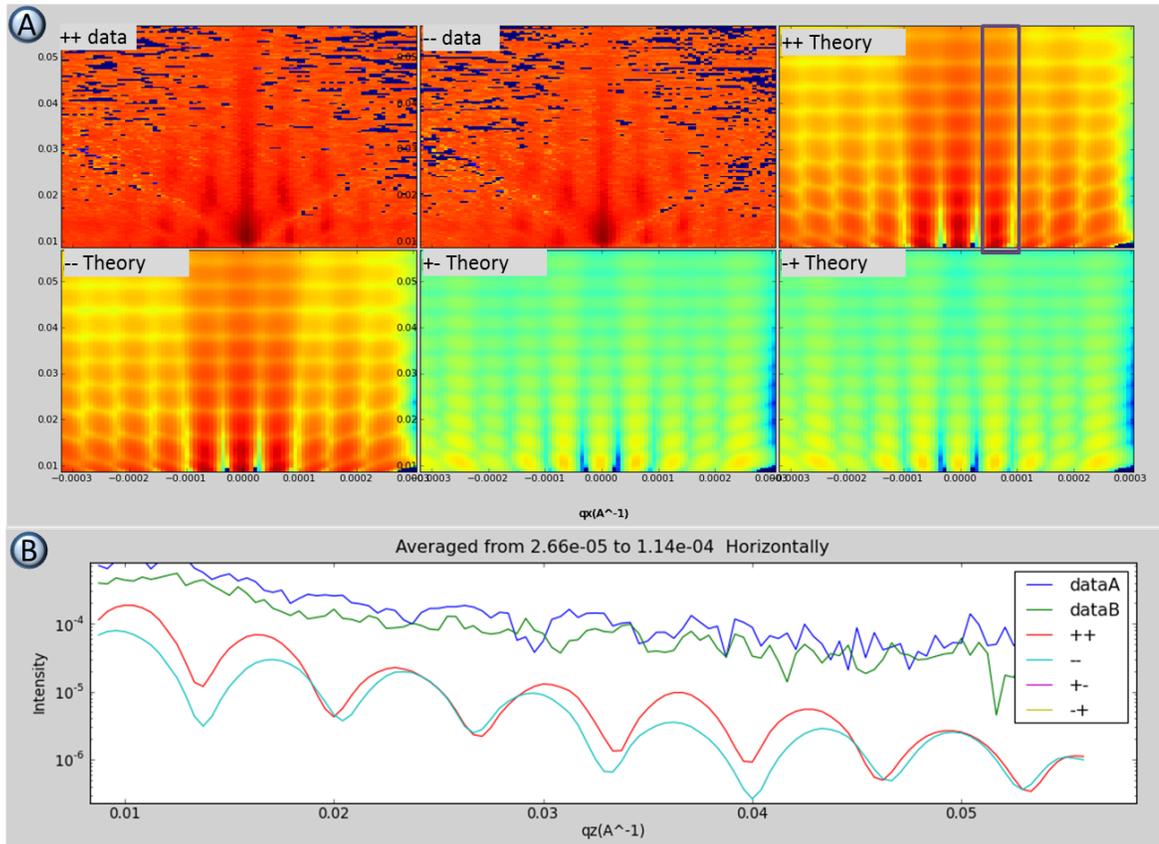


Figure 5.9: A vertical slice of the two non spin flip cross-sections. The data is off-set for clarity.

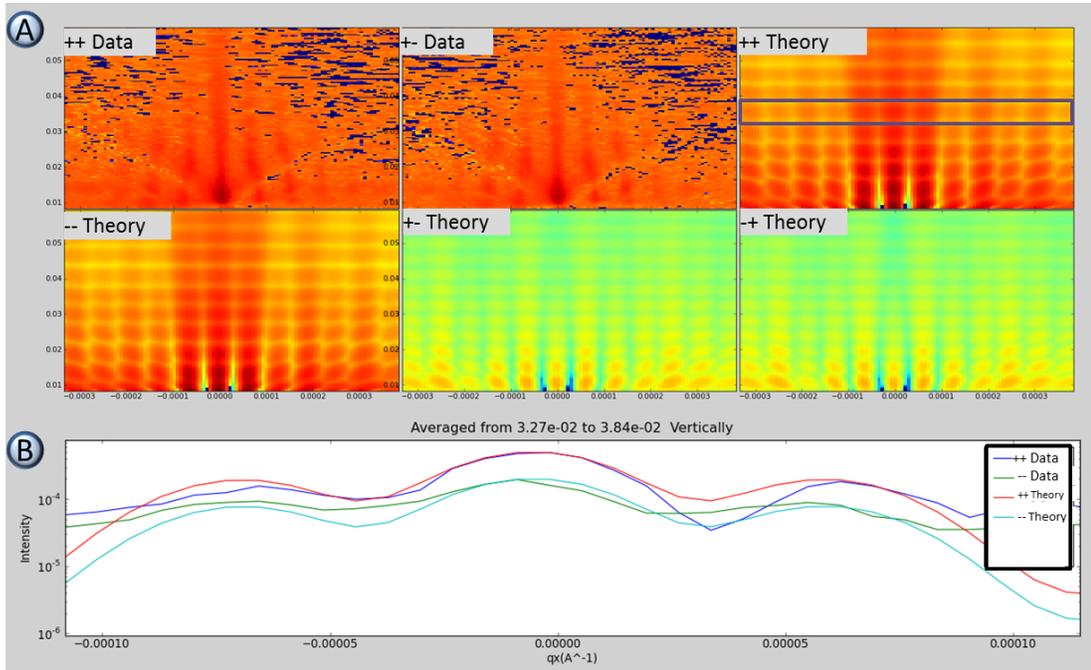


Figure 5.10: A horizontal slice of the two non spin flip cross-sections.

formalism used to convert the individual moments in the finite element matrix to a magnetic SLD is also accurate. The sensitivity of the instrumentation to magnetic variation in the the plane of the film has yet to be fully evaluated but, in this work, it is assumed to be similar to the structural sensitivity.

Although the magnetic modeling shown in these results is fairly crude, it shows two important features of the software. First, by tying our software to OOMMF, we are able to model magnetic systems which would otherwise be complicated to model. Second, the infrastructure design allows for the inclusion of magnetic modeling and, with better approximations, will be able to handle even more complicated systems.

## Chapter 6

### Coherence Length Determination

#### 6.1 Overview

The coherence length of the neutron is the length over which the neutron will interfere with variations in scattering potential simultaneously, allowing for the characterization of individual features as they relate to the entire system. For variations in scattering potential over length scales larger than the neutron coherence length, the neutron will interact incoherently with the sample. The effective in-plane neutron coherence length can be an important factor in interpreting off-specular reflectivity data as the measurement is sensitive to the in-plane structure. To accurately model off-specular scattering data, knowledge of the neutron coherence length properties is required. Part of the work in this thesis focuses on determining these properties for neutrons as prepared in a typical reactor source neutron reflectometer.

The neutron beam employed for diffraction in a standard instrumental configuration consists of a non-interacting collection, or ensemble, of individual neutron wave packets, each with a characteristic transverse coherence length perpendicular to its normal wavevector,  $k$ . Of particular consequence is the projection of the transverse coherence length onto the sample plane. Each neutron wave packet is composed of a distribution of plane wave momentum eigenstates which, when summed together to form a packet, localize the neutrons in space. Although representing

a wave packet as a single plane wave can be useful for describing some scattering phenomenon, it is neither physically realistic nor sufficiently adequate to accurately describe all scattering behavior. One example where the plane wave approximation is inadequate occurs when the effects of the coherent distribution of wavevectors comprising an individual packet must be separated from effects associated with the incoherent angular distribution of normal wavevectors of the packet forming a beam. For our discussion and analysis of this subject, we will assume that all scattering is elastic.

## 6.2 Effect of Transverse Coherence on Specular Reflection

The projection of the transverse coherence length can be split into a parallel and a perpendicular component as defined in figure 6.1B. For the geometry and beam configuration used in neutron reflectometry, only study of the parallel projection component is possible as the beam resolution in the perpendicular direction is so poor that an accurate determination would be unfeasible. One approach for determining the transverse coherence length of the neutron wave packet independently from the incoherent angular divergence of the beam is to investigate the extent to which an individual neutron effectively averages over in-plane variations in the scattering length density during the process of specular reflection. If we employ a well-characterized set of diffraction gratings which possess sufficient long-range order, we can semi-quantitatively determine the extent of the neutron transverse coherence length. Specifically, if the projection of the neutron transverse coherence

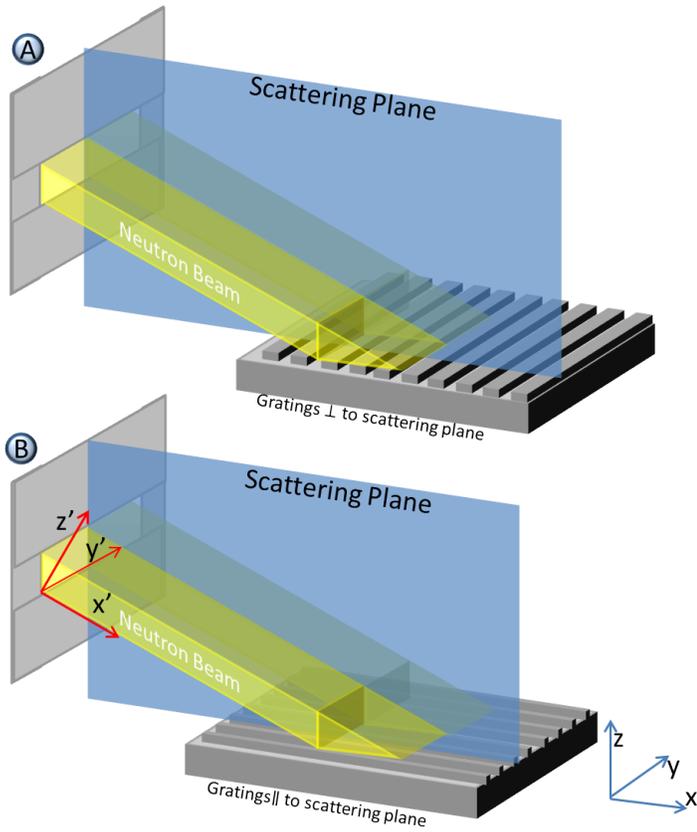


Figure 6.1: A schematic of the instrumental geometry used for the coherence length measurements. Two coordinate systems are present in this work. The blue  $x$ ,  $y$ , and  $z$  coordinate system refers to the coherence length projection onto the sample where the  $y$  axis refers to the perpendicular projection component and  $x$  refers to the parallel projection component (referenced to the scattering plane). The red  $x'$ ,  $y'$ , and  $z'$  coordinate system refers to the components of the coherence length where  $y'$  is the transverse direction and  $x'$  is the longitudinal direction.

length onto the grating structure, as shown in figure 6.1A is large enough to average over a number of periods (i.e., one period being equal to the sum of the widths of one Ni stripe plus intervening space), then the effective SLD of the film on the substrate probed by the specular scattering is equal to the average in-plane scattering potential resulting from the stripe and the spacing. Only a single critical  $Q$  will be observed, corresponding to the specular reflectivity curve shown in figure 6.2A. If the projected transverse coherence length of the neutron is significantly less than either the stripe or spacing width, the observed specular reflectivity will be an area-weighted incoherent sum of the reflectivity for the Ni (on Si substrate) and the bare Si substrate, as illustrated in 6.2B. The specular reflectivity for both of the limiting cases (where the feature spacing is much greater or much smaller than the coherence length) is negligibly affected by angular beam divergence typically employed in such measurements. Consequently, the effect of a coherent distribution of wavevectors in each neutron wave packet can be distinguished from that of the incoherent ensemble of nominal neutron wave vectors within the beam.

### 6.3 Results of the Specular Reflection Measurements

The ferromagnetic Ni gratings were saturated in an applied field, and measured with a polarized beam on the NG1 reflectometer at the NCNR. These measurements resulted in a splitting of the critical  $Q$  associated with the specular reflection, depending on whether the neutron spin eigenstate was plus or minus. This splitting indicates that the scattering measured from the system was due to the magnetic

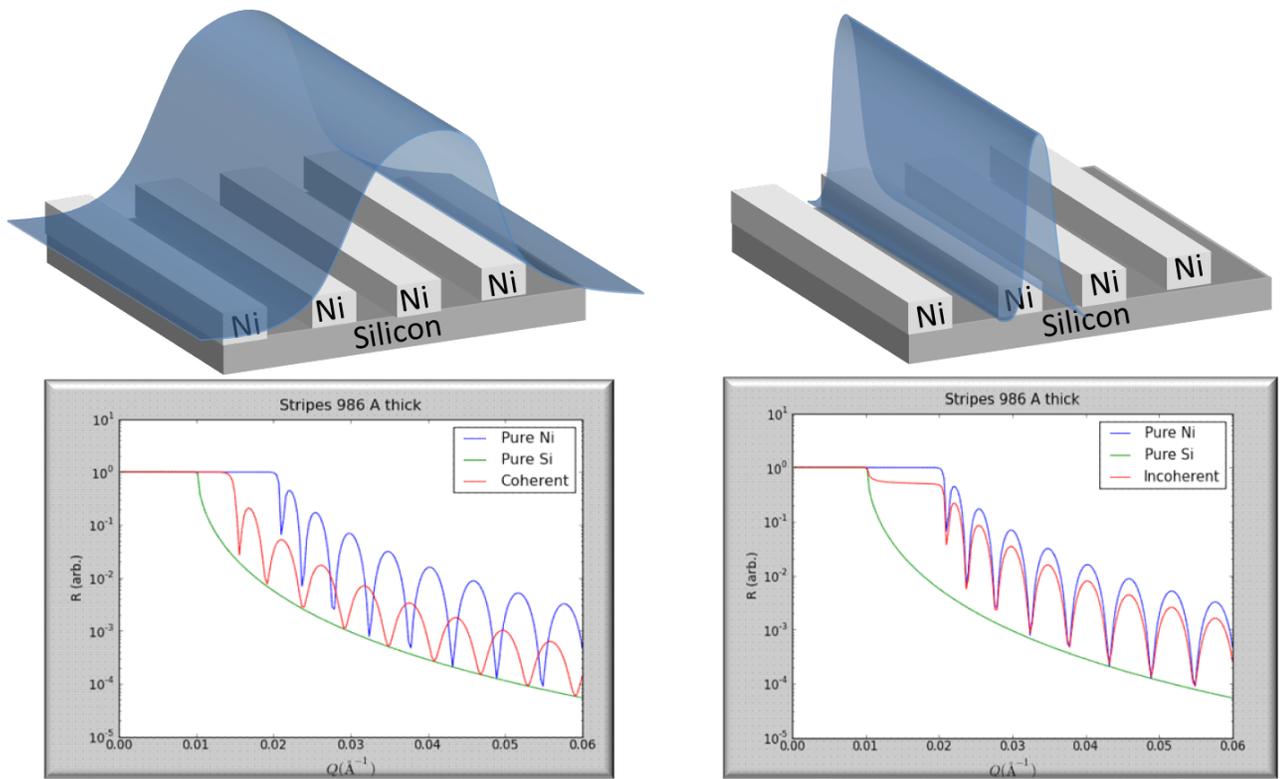


Figure 6.2: Model results for the expected difference between coherent and incoherent scattering from a system of nickel gratings.

material	Ni/Air(50:50)	Ni	Si	Air
density( $g/cm^3$ )	—	8.912	2.329	0.0
SLD( $\text{Å}^{-2}$ )	4.71e-6	9.42e-6	2.07e-6	0.0
Critical Edge	1.54e-2	2.18e-2	1.02e-2	0.0

gratings and not some other non-magnetic structure but is of no consequence in the present consideration and can be ignored.

In the large grating period limit, where the neutron wave packet fails to adequately average the in-plane variations in SLD associated with the grating structure, the specular reflectivity appears as in figure 6.3, (for an 800 micron Ni stripe width), The data show two distinct critical angles corresponding to the scattering potential of the Ni stripe and the bare Si substrate. In the other limiting case, e.g., for a stripe width of 10 microns the neutron wave packet effectively averages over the in-plane variations in the SLD associated with the grating structure and only a single critical Q is observed, corresponding to a SLD that is the average of Ni and the air in the space between adjacent Ni stripes, as shown in figure 6.3. The exact values for the SLD are provided in the table. Also shown in figure 6.3 is the specular reflectivity curve for an intermediate case, where more complicated scattering processes are observed. The transverse coherence length can be estimated by rotating the 10 micron grating to produce an effective periodicity based on the rotation angle. For example, the effective grating periods at 45, 60, and 75 degree rotations is 28, 40, and 77 microns respectively. The transverse coherence length of a neutron wave

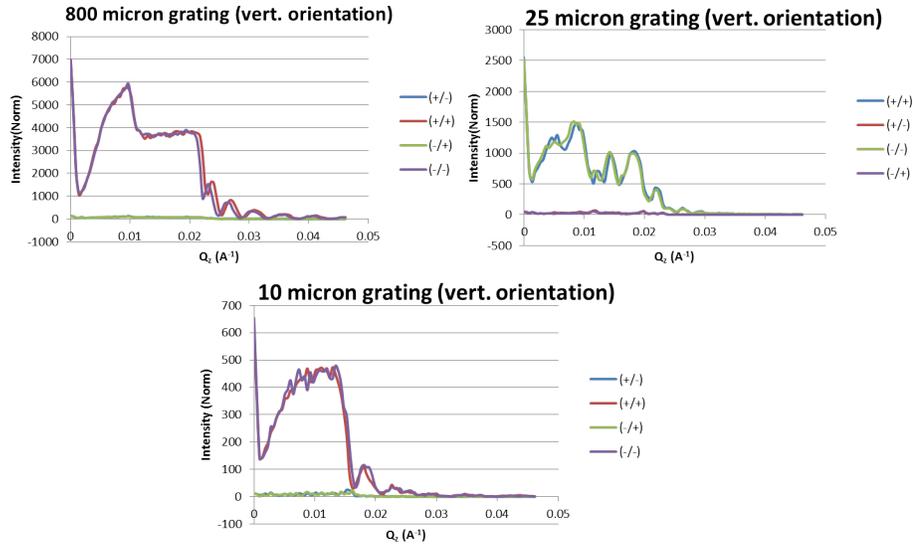


Figure 6.3: Data for three representative case. the 800 micron grating exhibits incoherent scattering, the 25 micron is approximately the coherent limit, and the 10 micron grating is mostly coherent scattering.

packet within the context of the instrumentation used for this study is estimated to be on the order of 1 micron. More rigorous and quantitative analysis is currently underway in preparation for a comprehensive publication reporting the results of this study. This reference also contains the derivation for the 1 micron value estimate in this thesis. The specifics will not be discussed here as the results have not yet been published.

## Chapter 8

### Future Work

#### 8.1 Summary

This section will discuss some aspects of off-specular reflectometry modeling which still need to be addressed before the technique can be utilized as a highly reliable characterization tool. One of the greatest limitations not discussed in this thesis is instrumentation. The samples used in this work are highly idealized to scatter where the instrumentation is most sensitive. resolution and background limit the instrument capabilities and still requires significant improvement. Still, the available modeling software is also lacking and this chapter will discuss the software requirements.

#### 8.2 Future Work on Software

Because modeling reflectometry is a an iterative process, the current software would benefit from a fitting engine. A fitting engine is a module which can automatically iterate over the model parameters to minimize the differences between the theory function and data. By automating this process, the user would be able to save significant time over running each individual model and changing the parameters manually. The implementation of a fitting engine requires a thoughtful and

creative means of parameterizing the model systems to ensure quick convergence and that the resulting model is physically possible. Some design has been discussed for use in this software; however, there was not enough time to implement these features.

Although many of the calculations used in this software are parallelized, they are still too slow for practical data fitting. The calculation speed can be improved by both increasing the efficiency of the code (removing superfluous loops, storing information that is needed in the future, etc.) and by increasing the number of nodes or processors which the calculation may utilize. The later improvement can be accomplished through the use of supercomputers, clusters, or commercial services which use cloud computing. Implementation of both the coding efficiencies multi-processing could require some significant code restructuring.

Because experimentalists are generally not well-adept at computer programming (and are the primary users of neutron scattering instruments) the software would benefit from a robust graphical user interface (GUI) which would make modeling and data analysis easier to carry out. Although this software has some graphical interface elements, there are places, such as model building, where a GUI could significantly improve the modeling process. In addition, the current GUI applications should be collected and implemented into one piece of software. The current method of GUI control is mostly script based and can lead to confusion.

For magnetic modeling, this software interfaces nicely with the OOMMF software package. Unfortunately, the importing and exporting required to use OOMMF modeling is cumbersome and confusing. This process can be simplified by utiliz-

ing an OOMMF function which handles batch processes through scripts. Although this process was investigated, combining our Python code with the tcl back-end of OOMMF was complicated and the idea was abandon.

### 8.3 Future Work on Modeling

The models implemented in this software show that qualitative modeling of simple features can be accomplished using finite element analysis. Still, additional calculations and features must be added to the software before it can truly capture all of the features in off-specular scattering.

Although the BA and SMBA calculations have been very useful under certain conditions, the DWBA has shown superior modeling capabilities. The DWBA implementation in this software still requires the addition of the specular scattering to model the data completely. This is non-trivial as the off-specular and specular calculations do not scale the same and are generally combined empirically using a real data set. The DWBA also needs the full magnetic calculation implemented. This can be found in reference [14] but was not implemented due to time constraints.

Some samples may require a study of how sample angle effects the scattering data. To model this, the software needs a matrix rotation function. This is complicated by the fact that the calculation is split into two components and needs to be carefully implement. Similarly, a more robust lattice builder function which allows the user to build arbitrary lattice structures would provide additional flexibility to the software. An example of this feature can be found in the isGISAXS software

[17].

All real systems will have some disorder which will effect the scattering. Currently, the software has no way to model this disorder. To quantitatively model experimentally relevant systems, this disorder will have to be modeled. Implementing this parameter will also require an in-depth understanding of the neutron beam coherence length which has yet to be fully characterized.

## 8.4 Future Work on Samples

Off-specular reflectometry would benefit from a greater variety of standard systems to allow for a broader understanding of the software modeling capabilities. For example, it is still unclear how well this scattering technique can differentiate between differently shaped features with similar volume fills. Creating a set of pillars with progressively more rounded edges would help answer this question. Also, sample features with an in-plane averaged density gradient could also provide some interesting scattering properties. The finite element method of calculation may not be able to reproduce all of the scattering effects.

## 8.5 Future Work on Coherence Length

Initial steps have been taken to fully characterized the coherence length properties of the neutron beam. These results have provided some interesting findings and warrant further investigation. The coherence length has been determined in the direction parallel to the beam; however, the nature of the coherence length in the y

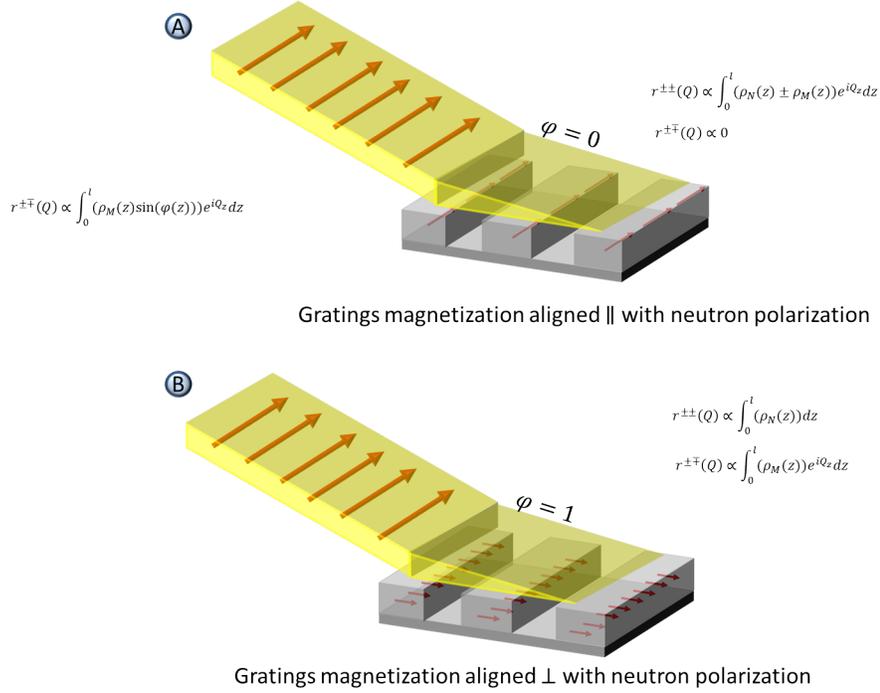


Figure 8.1: An experiment which would allow for the study of magnetic coherence length independently from the structural component. A) Gratings magnetization aligned parallel with the neutron polarization B) Gratings magnetized perpendicular to the neutron polarization direction.

direction is still unclear. Other phenomenon such as a dependence of the coherence length on slit width and high dynamic scattering effects at feature spacings close to the coherence length still need to be explored.

Another possible experiment is the study of the magnetic coherence length. Figure 8.1 illustrates an experiment which would allow for the study of the purely magnetic coherence length by studying the polarized reflection. The spin flip cross-sections are only affected by the magnetic scattering and are independent of the nuclear scattering component.

## Chapter 7

### Conclusion

#### 7.1 Summary

A software infrastructure has been developed using the Python computer programming language for the modeling of off-specular neutron reflectometry scattering data in a broad and extendable format which provides a generalized modeling environment with an openly reviewable and revisable set of scattering calculations. This software was used to model the standard samples which were fabricated at the NIST Center for Nanoscale Technology and the University of Maryland's NanoFab center.

This thesis focuses on comparing different algorithms for the modeling of off-specular data from two-dimensionally ordered structures. The algorithms are extended from the specular reflectometry formalism and work carried out previously on off-specular scattering data interpretation previously developed, as discussed in chapter 1. First, a software infrastructure was developed which could handle fast prototyping of both model representation methods and scattering calculations as outlined in chapter 2. Once this infrastructure was developed, different algorithms and mathematical treatments for the scattering calculations were formulated in terms of how they would be coded into the finite element software and are presented in chapter 3. To test the calculations, real scattering data was required. For

this, large area patterns of simple parallelepiped features were fabricated using the methods discussed in chapter 4. Once all of the required components were completed, the real data was modeled using different scattering algorithms to determine each method's accuracies and deficiencies, as discussed in chapter 5. Finally, all of the calculations contained parameters for the neutron beam coherence length which were arbitrarily set based on crude knowledge about the neutron beam properties. For more accurate calculations, a value for the coherence must be determined. This beam dependent parameter is studied in chapter 6.

The first approximation used to model these systems was the Born approximation. Even with this simple approximation, many mathematical treatments were formulated to improve modeling accuracy. First, a Gaussian envelop was placed on the structure factor calculation to more accurately represent the neutron beam coherence length. This eliminated artifacts produced between the off-specular diffraction peaks. For treatment of scattering calculations below the 'horizons' refraction effects due to the incident beam impinging on the substrate at a nearly orthogonal angle. Finally, resolution effects from deviations in the incoming angle and neutron beam wavelength were added to the calculation. The resulting theory function calculations qualitatively matched the data at high  $Q$  values but were not able to capture the scattering at low  $Q$  values or near the 'horizons'.

To improve the results obtained from the Born approximation, the wavefunction was perturbed by the neutron interaction with the interface between the substrate and the incident media. This perturbation made the theory function calculation much more complicated and increased the calculation time significantly.

The resulting theory functions exhibited differences from the BA calculation at the 'horizons'; however, the calculation still show similar inaccuracies at low  $q$  values. The final formalism implemented in this software was the distorted wave Born approximation which perturbs the wave function by neutron interactions with the substrate, incident media, and sample. This improved the accuracy of the calculation and qualitatively match the data much better then previous approximations.

We have found that, although BA does not capture all of the scattering details observed in real scattering data, its speed relative to other calculations provides a useful tool for obtaining quick estimates for parameter values. The substrate modified Born approximation provides one more step toward modeling quality by representing scattering effects at the horizons. This calculation was further improved by calculating the refractive shift which occurs below the horizons of the sample.

Neutron coherence length studies were carried out to determine the surface area over which a neutron beam would simultaneously probe. These properties are dependent on how the neutron wave packet was "prepared" before it reaches the sample. The neutron coherence length in the  $z$  direction was found to be approximately 1 micron. The coherence length of the neutron in the  $x$  direction was so small that it could not be determined.

## 7.2 Final Thoughts

As the off-specular neutron reflectometry technique continues to mature, it is clear that its contribution to materials characterization will be significant. Still, it

is a technique still in its infancy, and will require strong commitment from both the neutron and general scientific community to fully realize its potential. This project represents a modest attempt to apply the latest computational, experimental, and mathematical methods to evaluate the technique's current status, capabilities, and limitations.

### 7.3 Acknowledgments

The author would like to acknowledge the multitude of intellectuals without whom this would would would not be possible. Dr. Robert Briber for guiding the academic goals and responsibilities at the University of Maryland. Paul Kienzle for the superior guidance on scientific and general programming techniques. Dr. Brian Maranville for his work on theory development and cross-checking. Dr. Joseph Dura for guidance on lithographic fabrication processes. Dr. Charles Majkrzak for guidance on neutron reflectometry instrumentation, data acquisition and theory development. The author would also like to thank the Briber and Kofinas laboratory groups and the whole neutron reflectometry group for stimulating conversation and input. Finally, the author would like to thank friends and family who provided moral support through the entire project. This project was funded through the NSF Grant #: DMR-0529547

## Appendix A

### Overview

Although the derivation of the transfer matrix is somewhat straightforward, it is worth recording because a similar process is needed when providing the corrections used in the theories.

#### A.1 Wave Transfer Matrix Derivation

To derive:

$$\begin{pmatrix} t \\ ik_{III}t \end{pmatrix} e^{+k_{III}L} = \begin{pmatrix} \cos(k_{II}L) & \sin(k_{II}L)/k_{II} \\ -k_{II} \sin(k_{II}L) & \cos(k_{II}L) \end{pmatrix} \begin{pmatrix} 1 + r \\ ik_I(1 - r) \end{pmatrix} \quad (\text{A.1})$$

we start with the 4 boundary conditions:

$$1 + r = c + d \quad (\text{A.2})$$

$$\frac{k_I}{k_{II}}(1 - r) = c - d \quad (\text{A.3})$$

$$ce^{ik_{II}L} + de^{-k_{II}L} = te^{+k_{III}L} \quad (\text{A.4})$$

$$ce^{ik_{II}L} - de^{-k_{II}L} = \frac{k_{III}}{k_{II}}te^{+k_{III}L} \quad (\text{A.5})$$

Equation A.3 can be rearranged to:

$$c = (1 + r) - d \quad (\text{A.6})$$

which can be substituted into equation A.4 to yield:

$$\frac{k_I}{k_{II}}(1-r) = 1+r-d-d \quad (\text{A.7})$$

$$2d = (1+r) - \frac{k_I}{k_{II}}(1-r) \quad (\text{A.8})$$

With d we can now solve for c:

$$2c = 2\frac{k_I}{k_{II}}(1-r) + 2d \quad (\text{A.9})$$

$$2c = 2\frac{k_I}{k_{II}}(1-r) + (1+r) - \frac{k_I}{k_{II}}(1-r) \quad (\text{A.10})$$

$$2c = (1+r) + \frac{k_I}{k_{II}}(1-r) \quad (\text{A.11})$$

These can be brought into A.5:

$$\left[ (1+r) + \frac{k_I}{k_{II}}(1-r) \right] e^{+ik_{III}L} + \left[ (1+r) - \frac{k_I}{k_{II}}(1-r) \right] e^{-ik_{III}L} = 2te^{+k_{III}L} \quad (\text{A.12})$$

it is convenient here to substitute:

$$A = (1+r) \quad (\text{A.13})$$

$$B = (1-r) \quad (\text{A.14})$$

Which gives:

$$\left[ A + \frac{k_I}{k_{II}} B \right] e^{+ik_{II}L} + \left[ A - \frac{k_I}{k_{II}} B \right] e^{-ik_{II}L} = 2te^{+k_{III}L} \quad (\text{A.15})$$

Distributing the exponential gives:

$$Ae^{+ik_{II}L} + \frac{k_I}{k_{II}} Be^{+ik_{II}L} + Ae^{-ik_{II}L} - \frac{k_I}{k_{II}} Be^{-ik_{II}L} = 2te^{+k_{III}L} \quad (\text{A.16})$$

Here we can apply Euler's formula:

$$\underbrace{Ae^{+ik_{II}L} + Ae^{-ik_{II}L}}_{2A \cos(k_{II}L)} + \underbrace{\frac{k_I}{k_{II}} Be^{+ik_{II}L} - \frac{k_I}{k_{II}} Be^{-ik_{II}L}}_{2 \frac{k_I}{k_{II}} B i \sin(k_{II}L)} = 2te^{+k_{III}L} \quad (\text{A.17})$$

resulting in:

$$A \cos(k_{II}L) + \frac{k_I}{k_{II}} B i \sin(k_{II}L) = te^{+k_{III}L} \quad (\text{A.18})$$

The same substitution can be applied to equation A.5 to yield:

$$\left[ A + \frac{k_I}{k_{II}} B \right] e^{ik_{II}L} - \left[ A - \frac{k_I}{k_{II}} B \right] e^{-k_{II}L} = 2 \frac{k_{III}}{k_{II}} te^{+k_{III}L} \quad (\text{A.19})$$

$$Ae^{ik_{II}L} + \frac{k_I}{k_{II}} Be^{ik_{II}L} - Ae^{-k_{II}L} + \frac{k_I}{k_{II}} Be^{-k_{II}L} = 2 \frac{k_{III}}{k_{II}} te^{+k_{III}L} \quad (\text{A.20})$$

$$\underbrace{Ae^{ik_{II}L} - Ae^{-k_{II}L}}_{2Ai \sin(ik_{II}L)} + \underbrace{\frac{k_I}{k_{II}} Be^{ik_{II}L} + \frac{k_I}{k_{II}} Be^{-k_{II}L}}_{\frac{k_I}{k_{II}} B 2 \cos(ik_{II}L)} = 2 \frac{k_{III}}{k_{II}} te^{+k_{III}L} \quad (\text{A.21})$$

$$Ai \sin(k_{II}L) + \frac{k_I}{k_{II}} B \cos(k_{II}L) = \frac{k_{III}}{k_{II}} te^{+k_{III}L} \quad (\text{A.22})$$

It is convenient to multiply through by  $k_{II}$ :

$$k_{II}Ai \sin(k_{II}L) + k_{II} \frac{k_I}{k_{II}} B \cos(k_{II}L) = k_{II} \frac{k_{III}}{k_{II}} t e^{+k_{III}L} \quad (\text{A.23})$$

$$k_{II}Ai \sin(k_{II}L) + k_I B \cos(k_{II}L) = k_{III} t e^{+k_{III}L} \quad (\text{A.24})$$

We can also multiply by  $i$  to get:

$$ik_{II}Ai \sin(k_{II}L) + ik_I B \cos(k_{II}L) = ik_{III} t e^{+k_{III}L} \quad (\text{A.25})$$

$$-k_{II}A \sin(k_{II}L) + ik_I B \cos(k_{II}L) = ik_{III} t e^{+k_{III}L} \quad (\text{A.26})$$

Now we can reform equation A.18 and A.26 into a matrix:

$$\begin{pmatrix} t e^{+k_{III}L} \\ ik_{III} t e^{+k_{III}L} \end{pmatrix} = \begin{pmatrix} \cos(k_{II}L)A & \frac{k_I}{k_{II}} \sin(k_{II}L)iB \\ -k_{II} \sin(k_{II}L)A & \cos(k_{II}L)k_I iB \end{pmatrix} \quad (\text{A.27})$$

The common multiplies in column 1 and column 2 can be taken out and solved using a matrix multiply. This is written as:

$$\begin{pmatrix} t e^{+k_{III}L} \\ ik_{III} t e^{+k_{III}L} \end{pmatrix} = \begin{pmatrix} \cos(k_{II}L) & \frac{1}{k_{II}} \sin(k_{II}L) \\ -k_{II} \sin(k_{II}L) & \cos(k_{II}L) \end{pmatrix} \begin{pmatrix} A \\ ik_I B \end{pmatrix} \quad (\text{A.28})$$

We can replace A and B and pull out the common turn on the left hand side to yeild:

$$\begin{pmatrix} t \\ ik_{III} t \end{pmatrix} e^{+k_{III}L} = \begin{pmatrix} \cos(k_{II}L) & \frac{1}{k_{II}} \sin(k_{II}L) \\ -k_{II} \sin(k_{II}L) & \cos(k_{II}L) \end{pmatrix} \begin{pmatrix} (1+r) \\ ik_I(1-r) \end{pmatrix} \quad (\text{A.29})$$

## Appendix B

### Overview

The Born approximation starts with the integral form of the reflection calculation. Because this is important to the rest of the derivation, but somewhat long, it is derived in its completion in this appendix.

### B.1 Integral Form of the Reflectometry Calculation Derivation

It is convenient to derive the integral form for the one dimensional case. Starting from 1.22:

$$\left[ \frac{\delta^2}{\delta z^2} + k_{0z}^2 - 4\pi\rho_z \right] \psi_z = 0 \quad (\text{B.1})$$

Before the wave enters the sample, it is described as an unperturbed plane wave as seen in region I of figure 1.4:

$$\psi_z = e^{ik_0z} + r e^{-ik_0z} \quad (\text{B.2})$$

The scattering length density  $\rho_z$  can be written as the sum of the scattering length density felt by the unscattered wave plus the that of which the wave is perturbed by:

$$\rho_z = \rho_{0z} + \rho_{1z} \quad (\text{B.3})$$

To combine equation B.1 and equation B.3 we create:

$$L = \frac{\delta^2}{\delta z^2} + k_{0z}^2 - 4\pi\rho_{0z} \quad (\text{B.4})$$

which gives:

$$\left[ \frac{\delta^2}{\delta z^2} + k_{0z}^2 - 4\pi(\rho_{0z} + \rho_{1z}) \right] \psi_z = 0 \quad (\text{B.5})$$

$$\left[ \frac{\delta^2}{\delta z^2} + k_{0z}^2 - 4\pi\rho_{0z} - 4\pi\rho_{1z} \right] \psi_z = 0 \quad (\text{B.6})$$

$$[L - 4\pi\rho_{1z}] \psi_z = 0 \quad (\text{B.7})$$

$$L\psi_z - 4\pi\rho_{1z}\psi_z = 0 \quad (\text{B.8})$$

$$L\psi_z = 4\pi\rho_{1z}\psi_z \quad (\text{B.9})$$

We can make  $\psi_{z0}$  the unperturbed wave equation which follows equation B.1 to give:

$$L\psi_{0z} = 0 \quad (\text{B.10})$$

Defining the Green's function for the equation:

$$LG(z|z') = 4\pi\delta_{z-z'} \quad (\text{B.11})$$

Allows for the equation to be rewritten in terms of the Green's function. As Discussed in section B.2, We know that LG should satisfy:

$$LG = \delta_{z-z'} \quad (\text{B.12})$$

We have defined our  $LG(z|z')$  to be  $4\pi(z - z')$ . So we can deduce that:

$$\frac{LG}{4\pi} = \delta_{x-x'} \quad (\text{B.13})$$

It was also shown that:

$$f_z = \int_{-\infty}^{\infty} \delta_{z-z'} f_{z'} dz' \quad (\text{B.14})$$

where the  $f_{x'}$  is B.9 with an  $x'$  dependence. We can put these into the equation to form:

$$f_z = \int_{-\infty}^{\infty} \frac{LG}{4\pi} 4\pi \rho_{1z'} \psi_{z'} dz' \quad (\text{B.15})$$

$$f_z = \int_{-\infty}^{\infty} LG \rho_{1z'} \psi_{z'} dz' \quad (\text{B.16})$$

Because of B.10, the equation can also be written:

$$f_z = L\psi_{0z} + \int_{-\infty}^{\infty} LG \rho_{1z'} \psi_{z'} dz' \quad (\text{B.17})$$

carrying out the rest of the steps outlined in B.2, we get:

$$L\psi_z = L\psi_{0z} + \int_{-\infty}^{\infty} LG \rho_{1z'} \psi_{z'} dz' \quad (\text{B.18})$$

The L has no  $z'$  dependence so:

$$L\psi_z = L\psi_{0z} + L \int_{-\infty}^{\infty} G \rho_{1z'} \psi_{z'} dz' \quad (\text{B.19})$$

$$\psi_z = \psi_{0z} + \int_{-\infty}^{\infty} G \rho_{1z'} \psi_{z'} dz' \quad (\text{B.20})$$

It is somewhat straight forward to show that, because  $\rho \rightarrow 0$  as  $z \rightarrow -\infty$ :

$$\Psi_z \rightarrow e^{ik_z z} + r e^{-ik_z z} \text{ as } z \rightarrow -\infty \quad (\text{B.21})$$

which is just the equation for a plain wave in free space, as discussed in 1. At this point it becomes useful to look at the limit of equation B.9 and equation B.11 as  $z \rightarrow \infty$ . They are:

$$\Psi_{0z} \rightarrow e^{ik_z z} + r_0 e^{-ik_z z} \text{ as } z \rightarrow -\infty \quad (\text{B.22})$$

and

$$G(z|z') \rightarrow \frac{2\pi}{ik_z} e^{-ik_z z} \Psi_{0z'} \text{ as } z \rightarrow -\infty \quad (\text{B.23})$$

respectively.

Now, given the three previous equations, the asymptotic solution may be derived.

These three equations can be put into equation B.20 to give:

$$e^{ik_z z} + r e^{-ik_z z} = e^{ik_z z} + r_0 e^{-ik_z z} + \int_{-\infty}^{\infty} \frac{2\pi}{ik_z} e^{-ik_z z} \Psi_{0z'} \rho_{1z'} \psi_{z'} dz' \quad (\text{B.24})$$

or, pulling out the independent portion of the integral:

$$e^{ik_z z} + r e^{-ik_z z} = e^{ik_z z} + r_0 e^{-ik_z z} + \frac{2\pi}{ik_z} e^{-ik_z z} \int_{-\infty}^{\infty} \Psi_{0z'} \rho_{1z'} \psi_{z'} dz' \quad (\text{B.25})$$

Now we can reduce the equation to:

$$e^{ik_z z} + r e^{-ik_z z} = e^{ik_z z} + r_0 e^{-ik_z z} + \frac{2\pi}{ik_z} e^{-ik_z z} \int_{-\infty}^{\infty} \Psi_{0z'} \rho_{1z'} \psi_{z'} dz' \quad (\text{B.26})$$

$$r \frac{e^{-ik_z z}}{e^{-ik_z z}} = r_0 \frac{e^{-ik_z z}}{e^{-ik_z z}} + \frac{2\pi}{ik_z} \frac{e^{-ik_z z}}{e^{-ik_z z}} \int_{-\infty}^{\infty} \Psi_{0z'} \rho_{1z'} \psi_{z'} dz' \quad (\text{B.27})$$

$$r = r_0 + \frac{2\pi}{ik_z} \int_{-\infty}^{\infty} \Psi_{0z} \rho_{1z} \psi_z dz \quad (\text{B.28})$$

Two last alterations put this equation in the form that is started with in 3. In the specular case:

$$\vec{Q} = 2k_{0z} \quad (\text{B.29})$$

and the wavefunction for the initial wave is a plane wave:

$$\Psi_{0z} = e^{ik_{0z} z} \quad (\text{B.30})$$

which can be incorporated into equation B.28 to make:

$$r = r_0 + \frac{2\pi}{(1/2)Q} \int_{-\infty}^{\infty} e^{ik_{0z} z} \rho_{1z} \psi_z dz \quad (\text{B.31})$$

$$r = r_0 + \frac{4\pi}{Q} \int_{-\infty}^{\infty} e^{ik_{0z} z} \rho_{1z} \psi_z dz \quad (\text{B.32})$$

## B.2 Green's Function Reference

Because it can be somewhat unfamiliar, Green's functions are a way to solve inhomogeneous differential equations through the use of known boundary conditions.

In general, an operator and Green's function are defined such that:

$$LG_{x,x'} = \delta_{x-x'} \quad (\text{B.33})$$

where  $L$  is the defined operator (for our purposes it is equation B.4) and  $G$  is the Green's function. The  $\delta$  is the Dirac delta function which necessarily has the identity:

$$\int_{-\infty}^{\infty} \delta_x dx = 1 \quad (\text{B.34})$$

It also has an important translation property which is utilized in this application. In the case where the integration of a  $\delta$  function with a  $(x - x')$  dependence and a function with a dependence on an individual component leads to:

$$\int_{-\infty}^{\infty} \delta_{x-x'} f_{x'} dx' = f_x \quad (\text{B.35})$$

which means that:

$$\int_{-\infty}^{\infty} LG_{x,x'} f_{x'} dx' = \int_{-\infty}^{\infty} \delta_{x-x'} f_{x'} dx' = f_x \quad (\text{B.36})$$

We have intentionally set up our equations so that we can solve for  $\Psi_z$  and so that:

$$L\Psi_z = f_x \quad (\text{B.37})$$

which can be inserted into equation B.36 to give:

$$L\Psi_z = \int_{-\infty}^{\infty} LG_{x,x'} f_{x'} dx' \quad (\text{B.38})$$

Finally, the  $L$  is not dependent on  $x'$  so it can be pulled out of the integral to give:

$$L\Psi_x = L \int_{-\infty}^{\infty} G_{x,x'} f_{x'} dx' \quad (\text{B.39})$$

$$\Psi_x = \int_{-\infty}^{\infty} G_{x,x'} f_{x'} dx' \quad (\text{B.40})$$

This is what is derived in B.1.

## Appendix C

### Overview

This appendix holds the long form algebra or the magnetic reflectometry derivations. The tables are taken from [4].

### C.1 Combining the wave equations

We start with:

$$\left[ \frac{\delta^2}{\delta z^2} + \frac{Q^2}{4} - 4\pi\rho_{+++} \right] \Psi_{+z} - 4\pi\rho_{+-z} \Psi_{-z} = 0 \quad (\text{C.1})$$

$$\left[ \frac{\delta^2}{\delta z^2} + \frac{Q^2}{4} - 4\pi\rho_{--z} \right] \Psi_{-z} - 4\pi\rho_{-+z} \Psi_{+z} = 0 \quad (\text{C.2})$$

Because the result of both equations equal zero, these two equations may be multiplied to form an uncoupled fourth order equation. It is a simple algebraic distribution but is complicated somewhat by the fact that only the  $\Psi_{\pm}$  components contribute to the equation.

$$\left( \left[ \frac{\delta^2}{\delta z^2} + \frac{Q^2}{4} - 4\pi\rho_{+++} \right] \Psi_{+z} - 4\pi\rho_{+-z} \Psi_{-z} \right) \left( \left[ \frac{\delta^2}{\delta z^2} + \frac{Q^2}{4} - 4\pi\rho_{--z} \right] \Psi_{-z} - 4\pi\rho_{-+z} \Psi_{+z} \right) \quad (\text{C.3})$$

Now, distributing out the wave function:

$$\left( \frac{\delta^2}{\delta z^2} \Psi_{+z} + \frac{Q^2}{4} \Psi_{+z} - 4\pi \rho_{+++} \Psi_{+z} - 4\pi \rho_{+-z} \Psi_{-z} \right) \left( \frac{\delta^2}{\delta z^2} \Psi_{-z} + \frac{Q^2}{4} \Psi_{-z} - 4\pi \rho_{--z} \Psi_{-z} - 4\pi \rho_{-+z} \Psi_{+z} \right) \quad (C.4)$$

This can be distributed which combines the wave functions to form  $\Psi_{\pm z}$ . Each row in the equation array is the included results from each distribution:

$$\frac{\delta^2}{\delta z^2} \frac{\delta^2}{\delta z^2} \Psi_{\pm z} + \frac{\delta^2}{\delta z^2} \frac{Q^2}{4} \Psi_{\pm z} - \frac{\delta^2}{\delta z^2} 4\pi \rho_{--z} \Psi_{\pm z} \quad (C.5)$$

$$\frac{\delta^2}{\delta z^2} \frac{Q^2}{4} \Psi_{\pm z} + \frac{Q^2}{4} \frac{Q^2}{4} \Psi_{\pm z} - 4\pi \rho_{--z} \frac{Q^2}{4} \Psi_{\pm z} \quad (C.6)$$

$$-\frac{\delta^2}{\delta z^2} 4\pi \rho_{+++} \Psi_{\pm z} - 4\pi \rho_{+++} \frac{Q^2}{4} \Psi_{\pm z} + (4\pi)^2 \rho_{+++} \rho_{--z} \Psi_{\pm z} \quad (C.7)$$

$$-(4\pi)^2 \rho_{+-z} \rho_{-+z} \Psi_{\pm z} \quad (C.8)$$

Now we can pull out the  $\Psi_{\pm z}$  component and combine like terms to yeild:

$$\frac{\delta^4}{\delta z^4} \quad (C.9)$$

$$\frac{Q^2}{2} - 4\pi(\rho_{+++} - \rho_{--z}) \frac{\delta^2}{\delta z^2} \quad (C.10)$$

$$\left( \frac{Q^2}{4} \right)^2 - Q^2 \pi(\rho_{+++} - \rho_{--z}) + (4\pi)^2 (\rho_{+++} \rho_{--z} - \rho_{+-z} \rho_{-+z}) \quad (C.11)$$

We can now make the prefix on the second order derivative  $F$  and the non-derivative dependence  $G$  to yield:

$$\left( \frac{\delta^4}{\delta z^4} + F \frac{\delta^2}{\delta z^2} + G \right) \Psi_{\pm z} = 0 \quad (C.12)$$

## C.2 Magnetic scattering potential in terms of $Nb$ and $Np$

Starting with the wave equations in terms of their basic scattering potential:

$$\left[-\frac{\hbar^2}{2m} \frac{\delta^2}{\delta z^2} + V_{++z} - E\right] \Psi_{+z} + V_{+-z} \Psi_{-z} = 0 \quad (\text{C.13})$$

$$\left[-\frac{\hbar^2}{2m} \frac{\delta^2}{\delta z^2} + V_{--z} - E\right] \Psi_{-z} + V_{-+z} \Psi_{+z} = 0 \quad (\text{C.14})$$

Which can be converted to a matrix form:

$$\left[ -\frac{\hbar^2}{2m} \frac{\delta^2}{\delta z^2} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} V_{++z} & V_{+-z} \\ V_{-+z} & V_{--z} \end{pmatrix} - E \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right] \begin{pmatrix} \Psi_+ \\ \Psi_- \end{pmatrix} = 0 \quad (\text{C.15})$$

Where the magnetic potential can be written in terms of the magnetic number density and the scattering length:

$$V_M = \frac{2\pi\hbar^2}{m} \begin{pmatrix} Np_z & Np_x - iNp_y \\ Np_x - iNp_y & -Np_z \end{pmatrix} \quad (\text{C.16})$$

and the nuclear potential can be written as:

$$V_N = \frac{2\pi\hbar^2}{m} \begin{pmatrix} Nb & 0 \\ 0 & Nb \end{pmatrix} = \frac{2\pi\hbar^2}{m} \begin{pmatrix} \rho_N & 0 \\ 0 & \rho_N \end{pmatrix} \quad (\text{C.17})$$

so the total combined scattering potential is:

$$V = \frac{2\pi\hbar^2}{m} \begin{pmatrix} Nb + Np_z & Np_x - iNp_y \\ Np_x + iNp_y & Nb - Np_z \end{pmatrix} = \frac{2\pi\hbar^2}{m} \begin{pmatrix} \rho_{++} & \rho_{+-} \\ \rho_{-+} & \rho_{--} \end{pmatrix} \quad (\text{C.18})$$

### C.3 Magnetic transfer matrix derivation

We can start with the magnetic wave function described in C.21 and the roots in C.20.

$$S_1 = \sqrt{4\pi(Nb + Np) - Q^2/4} \quad (\text{C.19})$$

$$S_2 = -S_1$$

$$S_3 = \sqrt{4\pi(Nb - Np) - Q^2/4}$$

$$S_4 = -S_3$$

$$\Psi_{+z} = \sum_{j=1}^4 \mathcal{C}_j e^{S_j z} \quad (\text{C.20})$$

$$\Psi_{-z} = \sum_{j=1}^4 \mathcal{D}_j e^{S_j z}$$

we solve for  $\mathcal{D}$  in terms of  $\mathcal{C}$  to get:

$$\mathcal{D}_j = \mathcal{C}_j \frac{S_j^2 + Q^2/4 - 4\pi(\rho_{++} - \rho_{-+})}{S_j^2 + Q^2/4 - 4\pi(\rho_{--} - \rho_{+-})} = \mathcal{C}_j \mu_j \quad (\text{C.21})$$

where  $\mu_j$  can be written out for each root as:

$$\mu_1 = \mu_2 = \frac{Np - Np_z + Np_x + iNp_y}{Np + Np_z + Np_x - iNp_y} \quad (\text{C.22})$$

$$\mu_3 = \mu_4 = \frac{Np + Np_z - Np_x - iNp_y}{Np + Np_z - Np_x + iNp_y} \quad (\text{C.23})$$

When  $z = 0$ :

$$\Psi_{+0} = \mathcal{C}_1 + \mathcal{C}_2 + \mathcal{C}_3 + \mathcal{C}_4 \quad (\text{C.24})$$

$$\Psi_{-0} = \mu_1 \mathcal{C}_1 + \mu_2 \mathcal{C}_2 + \mu_3 \mathcal{C}_3 + \mu_4 \mathcal{C}_4 \quad (\text{C.25})$$

and their derivatives with respect to  $z$  are:

$$\Psi'_{+0} = S_1 \mathcal{C}_1 + S_2 \mathcal{C}_2 + S_3 \mathcal{C}_3 + S_4 \mathcal{C}_4 \quad (\text{C.26})$$

$$\Psi'_{-0} = S_1 \mu_1 \mathcal{C}_1 + S_2 \mu_2 \mathcal{C}_2 + S_3 \mu_3 \mathcal{C}_3 + S_4 \mu_4 \mathcal{C}_4 \quad (\text{C.27})$$

This leads to the matrix:

$$\begin{pmatrix} \Psi_{+z} \\ \Psi_{-z} \\ \Psi'_{+z} \\ \Psi'_{-z} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix} \begin{pmatrix} \Psi_{+0} \\ \Psi_{-0} \\ \Psi'_{+0} \\ \Psi'_{-0} \end{pmatrix} \quad (\text{C.28})$$

where the values for  $A_{ij}$  have been tabulated in C.4

Using the identities in C.5, the final transfer, matrix may be written as:

$$\begin{pmatrix} t_+ \\ t_- \\ \frac{iQ}{2} t_+ \\ \frac{iQ}{2} t_- \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix} \begin{pmatrix} I_+ + r_+ \\ I_- + r_- \\ \frac{iQ}{2} (I_+ + r_+) \\ \frac{iQ}{2} (I_- + r_-) \end{pmatrix} \quad (\text{C.29})$$

## C.4 Transfer matrix elements

---

$$A_{11} = 2\Delta[\mu_3 \cosh(S_1\delta_z) - \mu_1 \cosh(S_3\delta_z)]$$

$$A_{21} = 2\Delta[\mu_1\mu_3 \cosh(S_1\delta_z) - \mu_1\mu_3 \cosh(S_3\delta_z)]$$

$$A_{31} = 2\Delta[\mu_3 \cosh(S_1\delta_z) - \mu_1 \cosh(S_3\delta_z)]$$

$$A_{41} = 2\Delta[\mu_1\mu_3 \sinh(S_1\delta_z) - \mu_1\mu_3 \sinh(S_3\delta_z)]$$


---

$$A_{12} = -2\Delta[\cosh(S_1\delta_z) - \cosh(S_3\delta_z)]$$

$$A_{22} = -2\Delta[\mu_1 \cosh(S_1\delta_z) - \mu_3 \cosh(S_3\delta_z)]$$

$$A_{32} = -2\Delta[\sinh(S_1\delta_z) - \sinh(S_3\delta_z)]$$

$$A_{42} = -2\Delta[\mu_1 \sinh(S_1\delta_z) - \mu_3 \sinh(S_3\delta_z)]$$


---

$$A_{13} = 2\Delta[\mu_3 \sinh(S_1\delta_z)/S_1 - \mu_1 \sinh(S_3\delta_z)/S_3]$$

$$A_{23} = 2\Delta[\mu_1\mu_3 \sinh(S_1\delta_z)/S_1 - \mu_1\mu_3 \sinh(S_3\delta_z)/S_3]$$

$$A_{33} = 2\Delta[\mu_3 \cosh(S_1\delta_z)/S_1 - \mu_1 \cosh(S_3\delta_z)/S_3]$$

$$A_{43} = 2\Delta[\mu_1\mu_3 \cosh(S_1\delta_z)/S_1 - \mu_1\mu_3 \cosh(S_3\delta_z)/S_3]$$


---

$$A_{14} = -2\Delta[\sinh(S_1\delta_z)/S_1 - \sinh(S_3\delta_z)/S_3]$$

$$A_{24} = -2\Delta[\mu_1 \sinh(S_1\delta_z)/S_1 - \mu_3 \sinh(S_3\delta_z)/S_3]$$

$$A_{34} = -2\Delta[\cosh(S_1\delta_z)/S_1 - \cosh(S_3\delta_z)/S_3]$$

$$A_{44} = -2\Delta[\mu_1 \cosh(S_1\delta_z)/S_1 - \mu_3 \cosh(S_3\delta_z)/S_3]$$


---

## C.5 Polarized neutron reflection and transmission identities

---



---


$$\Psi_{+I_z} = I_+ = \mathcal{I}_+ e^{iQ_z/2} \quad \Psi'_{+I_z} = \frac{iQ}{2} I_+$$

$$\Psi_{-I_z} = I_- = \mathcal{I}_- e^{iQ_z/2} \quad \Psi'_{-I_z} = \frac{iQ}{2} I_-$$


---

$$\Psi_{+r_z} = r_+ = \mathcal{R}_+ e^{iQ_z/2} \quad \Psi'_{+r_z} = \frac{iQ}{2} r_+$$

$$\Psi_{-r_z} = r_- = \mathcal{R}_- e^{iQ_z/2} \quad \Psi'_{-r_z} = \frac{iQ}{2} r_-$$


---

$$\Psi_{+t_z} = t_+ = \mathcal{T}_+ e^{iQ_z/2} \quad \Psi'_{+t_z} = \frac{iQ}{2} t_+$$

$$\Psi_{-t_z} = t_- = \mathcal{T}_- e^{iQ_z/2} \quad \Psi'_{-t_z} = \frac{iQ}{2} t_-$$


---



---

## Appendix D

### Software Manual

This appendix contains the manual for the software. It is generated using Sphinx from the in-code documentation. It also include installation instructions and how to use the methods illustrated in the UML diagram in appendix E. The documentation only includes descriptions of the most pertinent modules.

---

# **OsRefi Documentation**

*Release 1.1.1*

**Christopher Metting**

October 03, 2011

# CONTENTS

<b>1</b>	<b>User's Guide</b>	<b>3</b>
1.1	Introduction	3
<b>2</b>	<b>Reference</b>	<b>9</b>
2.1	Model Creation	9
2.2	Calculations	35
2.3	Data Load	42
2.4	.omf File Loader	42
<b>3</b>	<b>Indices and tables</b>	<b>45</b>
	Python Module Index	47
	Index	49

Contents:

---

CONTENTS	1
----------	---

2	CONTENTS
---	----------

---

# USER'S GUIDE

This section gives an overview of the Off-Specular reflectometry modeling software. Read this to have an idea how the modeling software works and how it can be used to model scattering from a sample.

## 1.1 Introduction

This is an instruction manual on how to use the current infrastructure developed to model off-specular neutron scattering data. The manual covers the basic attributes of the software as well as how to build and model a specific sample.

### 1.1.1 Installing the Software

There are many scientific libraries which are needed to run this code. All of the libraries are free and can easily be installed simultaneously by going to [Link pythonx](#) and installing their product. In addition, if a Cuda compatible Nvidia GPU device is available, pyCuda must also be installed which may be downloaded at [Link pyCuda](#). Once these packages are installed, the osrefl package needs to be installed.

The software package may be downloaded at: <http://www.reflectometry.org/danse/software.html>

A link to the source code may also be found on this site.

**Windows:** For a windows install, download the osrefl executable, double click to run it, and follow the on screen instructions.

**Linux:** Download the source code from the link on the reflectometry.org website or go to:

<http://danse.us/trac/reflectometry/browser/trunk/osrefl>

Once the software is in the desired location, open up a command line prompt (either a terminal in linux or command prompt in windows). Change the directory to the top level osrefl folder and type:

```
python setup.py build_ext --inplace
```

This will build the C code in the appropriate place and allow the software to run.

### 1.1.2 Running Examples

The examples included in this software are located in the top level osrefl folder in a folder called 'examples'. To run the default example, change the directory to the top level osrefl folder and use the command:

```
python osrefl.py
```

This will run AuFit.py which is the model for an Au pillar system. This can easily be changed for any of the example file by:

```
python osrefl.py examples/<filename>
```

For example:

```
python osrefl.py examples/Ba_demo.py
```

This is also the procedure for running your own modeling scripts.

### 1.1.3 To Begin Modeling

This documentation provides instructions on how to write a simple model script. The first requirement is the import statements. Only two import statements are needed for the script to run. They should look like:

```
import scatter, sample_prep
```

sample\_prep holds all of the code for creating a model. Scatter hold all of the information about the different approximations that can be used.

### 1.1.4 Creating a Unit Cell

This section reviews how to create a model to be scattered off of. There are for main model creation tools: GeomUnit, K3D, OOMFUnit, and GrayInqUnit. Each of these can be used to produce a discretized unit cell.

#### GeomUnit

GeomUnit uses a mathematical description of the shapes in a sample to produce the unit scale to be scattered from. The first step is to create the shapes that are in the sample. A full list is documented in the code and an example is show below:

```
Au = Ellipse(SID = 4.506842e-6, dim=[3.75e4, 3.75e4, 700.0])  
Cr = Layer(SID = 3.01e-6, thickness_value = 20.0)
```

Here, a gold ellipse and a chrome adhesion layer are produced.

The modelling software allows the user to specify a centre value for manipulation of shape location in 3D space. To make the shape localisation easier, some tools have been created to orient shapes relative to other shapes. For example:

```
Au.on_top_of(Cr)
```

Will place the centre of the gold feature such that the ellipse base is flush with the adhesion layer. Other tools like this help orient the model and are explicitly defined in the documentation.

Once the shapes are created and oriented appropriately, they can be added to a Scene. A Scene is a container class which holds all of the shapes that make up a model. By allowing for the addition of an arbitrary amount of shapes, arbitrarily complex systems can be produced. The Scene is simply created by:

```
scene = Scene([Au,Cr])
```

Now we can produce the GeomUnit object. This objects contains the rest of the pertinent unit cell information such as dimension and discretization count:

```
GeomUnit = GeomUnit(Dxyz = [10.0e4,10.0e4, 1000.0], n = [50,51,52], scene = scene)
```

Finally, we need to run a producer command that will tie the GeomUnit object to the infrastructure that handles all discretized unit cells:

```
unit = GeomUnit.buildUnit()
```

### GrayImgUnit

A unit cell can also be created using the GrayImgUnit. This loader inputs a grey scale .png image file whose grey scale values are related to the SLD of that layer. When an object is created:

```
a = GrayImgUnit()
```

A file loader will open asking the user to choose the images file. The file name may also be scripted into the call:

```
img = sample_prep.GrayImgUnit(newres = numpy.array([150,400]), filename = '/Downloads/sample1_sld.png')
```

Once the object has been created, the universal 'Unit' must be created. For this, the software needs to know the rest of the unit cell information such as unit cell dimensions, discretization count and image scaling factor:

```
unit = img.unitBuild(Dxyz = [8480.0,8480.0,3500.0], scale = 1.0e-5, inc_sub=[0.0,2.0784314e-6])
```

#### Note:

- This unit building method assumes the image is extended infinity in the y direction which is the direction into the image, i.e. the image is of the x-z plane of the sample and the direction into the image is y.

### K3DUnit

This unit is created from the K-3D software. This software allows an output file that contains a list of points and plains that make up the shapes in the 3D model. This loader parses through these shapes using a point tracer method to determine whether or not a point falls inside the polyhedron. Although slow and limited in its modeling capability relatively complicated structures can be created easily using this method.

### OOMMFUnit

This unit loader creates a magnetic sample using the magnetic minimization software call Object Oriented MicroMagnetic Framework. This allows for both the flexibility of a discretized system with an simple way to produce magnetic structures.

## 1.1.5 Creating a Model

A unit is only one piece of the information needed to produce a scattering model. The model must also have a lattice which contains the information about the repeat structure:

```
Lattice = Rectilinear([20,20,1],unit)
```

A Q\_space object which tells the model where to calculate the scattering in reciprocal space:

```
q_space = Q_space([-0.001,-0.001,0.000021],[.0001,0.001,0.04],[200,50,200])
```

and a Beam object which provides the model with information about the probing beam:

## 1.1. Introduction

5

```
beam = Beam(5.0,None,None,0.05,None)
```

Once these objects are created they can be combined to form a Calculator object. This class is made to:

- Ensure that the user has provided all of the necessary pieces to calculate the scattering.
- Makes calculating scattering using different theories convenient.

This is created by:

```
sample = Calculator(Lattice,beam,q_space,unit)
```

## 1.1.6 Theory Function

Now that the software has everything it needs to calculate off-specular scattering, a modeling formalism must be chosen. The option here can be found elsewhere in the documentation but the modeling itself is easily run by the convention:

```
sample.BA()
```

Each theory calculation is a method on the calculator object. The user can now specify if they would like to run a resolution correction on the sample. This is done by:

```
sample.resolution_correction()
```

## 1.1.7 Viewing

To view the scattering, the user simply needs to script:

```
sample.viewUncorr()
```

to view the uncorrected scattering or:

```
sample.viewUncorr()
```

To view both the corrected and uncorrected plots side-by-side use:

```
sample.viewCorrCor()
```

to view the output plots.

## 1.1.8 Modeling Data

In the examples folder is a python script called AuFit.py. This is an example of how to compare a fit to real data using this software. This will go through the steps taken in this file.

### Data Loading

First, a model must be created as was shown in the previous section. The data included for this example was taken from Au pillars on a Si/Cr substrate. The data loading is all completed through GUI interfaces and only requires one line of code in the script. First, the data is loaded using the:

```
AuMeasurements = Data()
```

call which is found in the osrefl.loaders.andr\_load module.

## 6

This call will bring up a file selector where multiple .cgl data files may be loaded and combined. Use the "Choose input files" button

The next window will be the data selection window. This allows the user to select a specific subset of their data to model. This is important as modeling can be long and areas that don't have data should not have models calculated for it.

### Model Building

The models are build in the sample way as described in the model building section of this manual. One key additional command that is useful is:

```
q_space = Au_measurements.space
```

This command takes the q space values and point count from the selected data q space and uses it as the points to solve the model for. This is convenient for calculating models in the most efficient manner.

### Model/Data Interactor

There is now a view and GUI interactor for the data and model. This can be used by:

```
test_data.fitCompare(Au_measurements,titles = ['data','Model Label'])
```

where the method is run on the model and given the data as a parameter. Other options can be found in the method description in this documentation.

---

CHAPTER  
TWO

---

# REFERENCE

## 2.1 Model Creation

### 2.1.1 osrefl.model.sample\_prep

```
class osrefl.model.sample_prep.Beam (wavelength=None, angular_dth=None, background=None,
                                     wavelength_dth=None, resolution=None)
```

Bases: object

**Overview:**

Hold the beam information. These are all of the instrument characteristics the have an effect on the scattering.

**Parameters(\_\_\_\_init\_\_\_\_):**

*wavelength:* (float)angstroms) For reactor source, the wavelength is used to calculate the resolution of the instrument.

*angular\_dth:* (float)degrees) The angular divergence of the beam

*background:* (float)intensity) This is the dark counts on the detector

*resolution:* (float) Generally, spallation sources have a resolution that they use as a beam parameter.

**Note:**

•This class is primarily developed for a reactor source but is open to parameters needed for a spallation source.

```
class osrefl.model.sample_prep.Cone (SLD, dim, stub=None, center=(None, None, None),
                                     Ms=0.0)
```

Bases: osrefl.model.sample\_prep.Shape

**Overview:**

Uses the generic formula for a cone feature to create a cone object. Also allows for the creation of a truncated cone by providing a cut-off parameter for the thickness.

**Parameters(\_\_\_\_init\_\_\_\_):**

*SLD:* (float)angstroms<sup>-2</sup>) The scattering length density of the cone.

*dim:* (float,[3]angstroms) The x component, y component and thickness of the cone respectively.  
x is the radius of the cone base in the x direction and b is the radius of the cone base in the y direction.

*stub:* (float)angstroms) Provides a hard cut-off for the thickness of the cone. this allows for the creation of a truncated cone object who side slope can be altered by using different z component values while keeping the stub parameter fixed.

*center:* (float,[3]angstroms) The x, y, and z component of the central point of the cone. In the case that the center is set to [None,None,None] the shape will be put in the bottom corner of the unit cell (the bounding box will start at (0,0,0)).

*Ms:* (float)angstroms) The magnetic SLD of the material for this shape.

*discretize* (x\_points, y\_points, z\_points, cell\_to\_fill, mag\_to\_fill)

**Overview:**

This module takes in x,y, and z points and fills the matrix array with the SLD of the shape for the points that fall within the shape

**Parameters:**

*x\_points:* (float)angstroms) An array of x points to be determined if they fall within the cone.

*y\_points:* (float)angstroms) An array of y points to be determined if they fall within the cone.

*z\_points:* (float)angstroms) An array of z points to be determined if they fall within the cone.

*cell\_to\_fill:* (float,array)angstroms) This is the SLD matrix of the unit cell. It is filled by the render function.

*mag\_to\_fill:* (float,array)angstroms) This is the Ms matrix of the unit cell. It is filled by the render function.

**Returns:**

*cell\_to\_fill:* (array)angstroms) The discretized unit of the form factor built unit cell.

*height* ()

**Overview:**

Returns the total height of the cone. This differs from thickness which only describes the thickness of the individual cone whereas this method returns the maximum z-value of the shape in the unit cell.

**Returns:**

*height:* (float)angstroms) The total height of the cone object (measures the top most part of the cone in z)

*is\_core\_of* (element, offset=(0, 0, 0))

**Overview:**

This method is used to place the Shape 'self' into the center of the Shape element. This creates a core shell type geometry.

**Parameter:**

*element:* (Shape) The Shape object that is being embedded into the selected Shape object 'self' (The shapes who's center value is being altered).

**Note:**

•In the case of use with a Layer Object, the Shape object will only be placed in the center in the z-direction because it's location in the x-y plane does not make a difference.

*length* ()

**Overview:**

Returns the maximum length of the cone (x direction).

**Returns:**

*length:* (`floatlangstroms`) The total length of the cone object (absolute distance in x)

*on\_top\_of (element)*

**Overview:** This method alters the center value of a shape object so that the Shape who's *on\_top\_of* method was called is located on top of the Shape 'element'.

**Parameter:**

*element: (Shape)* The Shape object that is being put on top of the selected Shape object  
*self* (The shapes who's z-component center value is being altered).

*thickness ()*

**Overview:**

Returns the total thickness of the cone.

**Returns:**

*thickness: (floatlangstroms)* The total thickness of the cone object (absolute thickness)

*width ()*

**Overview:**

Returns the maximum width of the cone (y direction).

**Returns:**

*width: (floatlangstroms)* The total width of the cone object (absolute distance in y)

```
class osrefl.model.sample_prep_Ellipse(SLD, dim, center=(None, None, None), Ms=0.0)
```

Bases: `osrefl.model.sample_prep.Shape`

**Overview:**

Uses the generic formula for an Ellipse feature to create a Ellipse object:  $(x^2/a^2) + (y^2/b^2) = 1$ . This dim variable will be in the form [a,b,z]. This class can also be used to make a cylinder by setting `dim[0] = dim[1]`

**Parameters(\_\_\_\_):**

*SLD: (floatlangstroms^2)* The scattering length density of the Ellipse.

*dim: (float,[3]langstroms)* The 'a' component, 'b' component and thickness of the Ellipse respectively, 'a' is the radius of the Ellipse in the x direction and 'b' is the radius of the ellipsoid in the y direction.

*center: (float,[3]langstroms)* The x, y, and z component of the central point of the Ellipse. In the case that the center is set to [None, None, None] the shape will be put in the bottom corner of the unit cell (the bounding box will start at (0,0,0)).

*Ms: (floatlangstroms)* The magnetic SLD of the material for this shape.

**Note:**

\*This class is different than Ellipsoid which builds a lenticular shaped object where as this class produces a pillar shaped object.

*discretize (x\_points, y\_points, z\_points, cell\_to\_fill, mag\_to\_fill)*

**Overview:** This module takes in x,y, and z points and fills the matrix array with the SLD of the shape for the points that fall within the shape.

**Parameters:**

*x\_points: (floatlangstroms)* an array of x points to be determined if they fall within the Ellipse.

*y\_points: (floatlangstroms)* an array of y points to be determined if they fall within the Ellipse.

*z\_points: (floatlangstroms)* an array of z points to be determined if they fall within the Ellipse.

*cell\_to\_fill: (floatarraylangstroms)* This is the SLD matrix of the unit cell. It is filled by the render function.

*mag\_to\_fill: (floatarraylangstroms)* This is the Ms matrix of the unit cell. It is filled by the render function.

*height ()*

**Overview:**

Returns the total height of the ellipsoid. This differs from thickness which only describes the thickness of the individual Ellipse whereas this method returns the maximum z-value of the shape in the unit cell.

*is\_core\_of (element, offset=(0, 0, 0))*

**Overview:**

This method is used to place the Shape 'self' into the center of the Shape element. This creates a core shell type geometry.

**Parameter:**

*element: (Shape)* The Shape object that is being embedded into the selected Shape object  
*self* (The shapes who's center value is being altered).

**Note:**

\*In the case of use with a Layer Object, the Shape object will only be placed in the center in the z-direction because it's location in the x-y plane does not make a difference.

*length ()*

**Overview:**

Returns the maximum length of the Ellipse (x direction).

*on\_top\_of (element)*

**Overview:** This method alters the center value of a shape object so that the Shape who's *on\_top\_of* method was called is located on top of the Shape 'element'.

**Parameter:**

*element: (Shape)* The Shape object that is being put on top of the selected Shape object  
*self* (The shapes who's z-component center value is being altered).

*thickness ()*

**Overview:**

Returns the total thickness of the Ellipse.

*width ()*

**Overview:**

Returns the maximum width of the Ellipse (y direction).

```
class osrefl.model.sample_prep_Ellipsoid(SLD, dim, center=(None, None, None), Ms=0.0)
```

Bases: `osrefl.model.sample_prep.Shape`

**Overview:**

Uses the generic formula for a Ellipsoid feature to create a Ellipsoid object. This object can be used to create a sphere by setting `dim[0] = dim[1] = dim[2]`

**Parameters**(`__init__`):

*SLD*: (`float`|`3`)`langstroms`\*2) The scattering length density of the Ellipsoid.

*dim*: (`float`|`3`)`langstroms`) The 'a' component, 'b' component and 'c' component of the Ellipsoid respectively. 'a' is the radius of the Ellipsoid in the x direction, 'b' is the radius of the Ellipsoid in the y direction, and 'c' is the radius of the Ellipsoid in the z direction.

*center*: (`float`|`3`)`langstroms`) The x, y, and z component of the central point of the ellipsoid. In the case that the center is set to [`None`,`None`,`None`] the shape will be put in the bottom corner of the unit cell (the bounding box will start at (0,0,0)).

*M*: (`float`|`langstroms`) The magnetic SLD of the material for this shape.

**Note:**

•This is a lenticular shaped object.

**deserialize** (`x_points`, `y_points`, `z_points`, `cell_to_fill`, `mag_to_fill`)

**Overview:**

This module takes in x,y, and z points and fills the matrix array with the SLD of the shape for the points that fall within the shape

**Parameters:**

*x\_points*: (`float`|`langstroms`) an array of x points to be determined if they fall within the Ellipsoid.

*y\_points*: (`float`|`langstroms`) an array of y points to be determined if they fall within the Ellipsoid.

*z\_points*: (`float`|`langstroms`) an array of z points to be determined if they fall within the Ellipsoid.

*cell\_to\_fill*: (`float`|`array`|`langstroms`) This is the SLD matrix of the unit cell. It is filled by the render function.

*mag\_to\_fill*: (`float`|`array`|`langstroms`) This is the M<sub>s</sub> matrix of the unit cell. It is filled by the render function.

**height** ()

**Overview:** Returns the total height of the layer. This differs from thickness which only describes the thickness of the individual layer whereas this method returns the maximum z-value of the shape in the unit cell.

**is\_core\_of** (`element`, `offset`=(`0`, `0`))

**Overview:**

This method is used to place the Shape 'self' into the center of the Shape element. This creates a core shell type geometry.

**Parameter:**

*element*: (`Shape`) The Shape object that is being embedded into the selected Shape object 'self' (The shapes who's center value is being altered).

**Note:**

•In the case of use with a Layer Object, the Shape object will only be placed in the center in the z-direction because it's location in the x-y plane does not make a difference.

**on\_top\_of** (`element`)

## 2.1. Model Creation

13

**Overview:** This method alters the center value of a shape object so that the Shape who's `on_top_of` method was called is located on top of the Shape 'element'.

**Parameter:**

*element*: (`Shape`) The Shape object that is being put on top of the selected Shape object 'self' (The shapes who's z-component center value is being altered).

**thickness** ()

**Overview:**

Returns the total thickness of the layer.

**width** ()

**Overview:** Returns the maximum width of the Pyramid (y direction)

```
class osrefl.model.sample_prep.GeoMetric (Dxy=None, None, None, n=None, None, None,
                                           scene=None, inc_sub=None, None)
```

Bases: `object`

**Overview:**

This is a producer of a Unit Cell object. Given a Scene of Shape objects and other key parameters (defined below), this class will render a three dimensional numpy array of the SLD of the unit cell along with the magnetic SLD in the case the it is defined.

**Parameters:**

*Dxy*: (`float`|`3`)`langstroms`) The x,y and z real space size of the unit cell.

*n*: (`int`|`3`)`count`) The number of elements the x, y and z axis of the unit cell will be divided into. This is how coarse/finer the unit cell is discretized into.

*scene*: (`Scene`) A scene object which holds the collection of shapes to be rendered into the unit cell. The render renders shapes in the order they are provided. Multiple shape are rendered by only filling unit cell array values where they have not yet been changed by previous shapes. For example, in the case of a core/shell scenario, the shapes should be listed so that the core is listed before the shell.

*inc\_sub*: (`float`|`2`) [SLD of incident media, SLD of Substrate]. This holds the scattering length density for the incident and substrate media respectively. The attribute does not currently hold the neutron absorption of the materials which is negligible.

**Parameters(Class):**

*value\_list*: (`float`|`3`)`langstroms`)

This is a list of arrays for the x,y and z directions. Each array contains the real space distance of the array element from the origin. (eg. for 4 points at a step size of .2 angstroms in the x direction, `value_list[0] = array([0,0.2,0.4,0.6])`)

*unit*: (`float`|`3D array`|`angstroms`\*2)

This is the discretized representation of the structural unit cell. This is the array for which the scattering is calculated.

*mag\_unit*: (`float`|`3D array`|`langstroms`\*2)

This is the discretized representation of the magnetic unit cell. This is for the case of unpolarized neutrons where a value is given for the magnetic SLD which differs from the structural SLD.

*step*: (`float`|`3`)`langstroms`)

14

This is the real space step size for the unit cell in the x, y and z direction. It is the total real space increment that a single array value represents.

**Note:**

- The renderer uses the mathematical formulas provided in the calculation.py file for each shape class to determine if each point in the 3D array falls within the shape.
- If Dxyz[2] is not defined, the class chooses a value that will end just above the top of the tallest feature in the unit cell (adds approximately one layer of incident media).
- Currently, the x,y and z values represent the real space value at the beginning of the discretized unit rather than the value of the unit at the center. This must be revised.

**buildDrate ()**

**Overview:**

Producer method. This method produces a Unit\_Cell object from the rendered geomUnit object. Because this is the original development of Unit\_Cell, the conversion is pretty simple and most of the parameters are in the exact form needed by Unit\_Cell.

**Note:**

- The GeomUnit object must be rendered first. If it has not been rendered the method will do it automatically.

**render ()**

**Overview:**

Uses the discretized method contained in each Shape object in a Scene to create a 3D numpy array of SLD values which can be used to solve the scattering from.

**Note:**

- This module fills the unit cell array with Shape objects in the order that they are listed in the Scene object. Shapes that are later in the list will write over those that came earlier. This means, for example, in the case of a core-shell sample, the outer shell object must be entered before the core.

**value\_extend ()**

**Overview:**

This module takes the individual values of step and length and creates a list of three arrays [x,y,z] that contains the real space value for each discrete piece of the unit cell array:

```
class osrefl.model.sample_prep.GrayImageUnit (newres=None, filename=None)
```

Bases: object

**Overview:**

This class creates a Unit\_Cell object from a gray scale image by loading an image and the parameters that are correlated with that image. This class assumes that the direction into the picture is the same straight through. The user may choose this axis.

**Parameters:**

*newres (float,[2]count)* Sometimes, the .png file is much higher in resolution than is needed for the scattering calculation. The user has the option of choosing a new resolution to down-scale the image file to.

**Note:**

- Currently, this only supports images that are colored to be on scale with the scattering length density values of the profile.
- This file load system takes in .png files.

**2.1. Model Creation**

15

- When loading the image, it assumes that the image is a three channel load but that each channel is equal. This can be much improved by counting channels and handling RGB files.

```
unitBuild (Dxyz, scale, inc_sub=(None, None))
```

**Overview:**

Producer method. This method produces a Unit\_Cell object from the parameters that are carried over from the .omf loaded by the OMMFUnit class. The object that this method produces is needed to work with the calculation API.

**Parameters:**

*Dxyz: (float,[3]angstroms)*

The real space length, width and height represented by the image. Because it is assumed that the image is the x z direction, the Dxyz[1] or Dy is meaningless here. It is left in only to allow for the consistency of Dxyz.

*scale: (float(factor))*

This parameter allows the user to uniformly scale the image values by a factor. This is to allow for directly scaling the image to SLD values.

**Note:**

- The user should be allowed to choose the axes on the image.
- Some structure should be put together for manually assigning SLD values to colors on the map. This will only be needed if image loading becomes a highly used load system.

```
class osrefl.model.sample_prep.Hexagonal (repeat, unit)
```

Bases: osrefl.model.sample\_prep.Lattice

**Overview:**

A lattice structure that is packed in a hexagonal ordering. This is produced by solving the nonlinear structure factor for two phases of repeating units and adding these phases together.

0 = feature phase 1 0 = feature phase 2 - = spacing

Hexagonal:

0-0-0-0-0-0

0-0-O-0-O-0

0-0-0-0-0-0

0-O-0-O-0-O-0

16

*Q*: (*q\_space*) a *Q\_space* object.

Note:

- Used to superimpose two rectilinear lattice structures over each other.

**rect\_ft** (*Q*, *repeat\_mod*=[*None*, *None*, *None*])

Overview:

Solves the structure factor for the *Q* points in the *q\_space* object. Using this method solves the structure factor before integrating over the *q* steps. If used explicitly without integrating over the *q* steps aliasing errors can be introduced into the scattering. This is especially true where the *q*-step is coarse in the *qx* direction which can lead to mismatch in intensities between the negative and positive *qx* diffraction peaks.

Parameters:

*Q*: (*q\_space*) a *Q\_space* object.

*repeat\_mod*: (*float*,*3*),*count*) A repeat modifier for the repeat attribute of a Lattice object. This is necessary when the effective repeat of a specific lattice type is different than the lattice spacing requested by the user.

Note:

- This calculation should be used in conjunction with integration over the *x* direction.

**struc\_calc** (*Q*)

Overview:

This is the calculation of the structure factor for a hexagonal lattice. It returns the structure factor integrated over the *qx* direction by solving the scattering for two phases of rectilinear scattering and adds the results.

Parameters:

*Q*: (*q\_space*) a *Q\_space* object.

**x\_calc\_sfx** (*qx*)

Overview: Used internally to solve the structure factor for a given *qx* value in the *qx* direction. This is possible because the *qx*, *qy*, and *qz* components are separable.

Parameters:

*qx*: (*float*) a *qx* value.

Note:

- This method is used in conjunction with the integration call to obtain the structure factor that is returned.

**x\_calc\_sfx\_shift** (*qx*)

Overview:

Used internally to solve the structure factor for a given *qx* value in the *qx* direction. This solution applies a 0.5 phase shift to the wave solution which can be combined with the unshifted solution to give a solution to the scattering from a hexagonal lattice.

Parameters:

*qx*: (*float*) a *qx* value.

Note:

- This method is used in conjunction with the integration call to obtain the structure factor that is returned.

**x\_gauss\_sfx** (*qx*, *args*)

**y\_calc\_sfx** (*qy*)

Overview:

Used internally to solve the structure factor for a given *qy* value in the *qy* direction. This is possible because the *qx*, *qy*, and *qz* components are separable.

Parameters:

*qy*: (*float*) a *qy* value.

Note:

- This method is used in conjunction with the integration call to obtain the structure factor that is returned.

**y\_calc\_sfx\_shift** (*qy*)

Overview:

Used internally to solve the structure factor for a given *qy* value in the *qy* direction. This solution applies a 0.5 phase shift to the wave solution which can be combined with the unshifted solution to give a solution to the scattering from a hexagonal lattice.

Parameters:

*qy*: (*float*) a *qy* value.

Note:

- This method is used in conjunction with the integration call to obtain the structure factor that is returned

```
class osrefl.model.sample_prep.K3DUnit (filename, k3d_scale=1.0, SLD_list=None)
```

Base: object

Overview:

This class is for a shape that is entered as a list of polygons and points from k3d modeling software.

Parameters:

*filename*(*str*): The name of the file that was exported from the k3d model software.

*SLD\_list*(*ll*): The list of scattering length densities. there should be the same number of SLDs as there are shapes in filename(A^2).

```
class K3D_Shape (vertices=None, edges=None, numpoly=None, numpoint=None)
```

Overview:

This contains variables to define a shape

Parameters:

*vertices*: the points in space that make up a shape

*edges*: Array containing the thicknesses of all layers in the substrate given in the order: Sample Bottom -> Feature/Substrate Interface

```
class K3DUnit.K3D_Shape_Collection (description_list=None, correction_scaling=1)
```

Overview:

This contains the information for the description of multiple features

**Parameters**

*feature[K]* is an object of type Shape

*description\_list* a list of objects of type shape that holds the edges, vertices and Surf all of the features

K3DUnit.**discretize** (*x\_points, y\_points, z\_points, cell\_to\_fill, mag\_to\_fill*)

**Overview:**

This method will turn a given K3D file of shapes into a numpy matrix of scattering length densities.

**Note:**

•This is done for a given Unit\_Cell Object

K3DUnit.**height** ()

**Overview:**

This module takes in a K3D\_Shape object and determines its height in real space

**Note:**

•features is of type K3D\_Shape

K3DUnit.**k3d\_listform** (*point\_array, poly\_array, num\_polygons, num\_points, shapeList*)

**Overview:**

This method forms the list of features needed to create the scattering matrix.

**Parameters**

*point\_array* An array of points in 3d space that make up a feature.

*poly\_array* for numbers that represents the points that make up a polyhedran face.

*num\_polygons* total number of polygons that make up a feature

*num\_points* number of points that make up a feature

*shapeList* the list of features that the new feature is being added to

class osrefl.model.prepare.**Lattice**

Bases: object

**Overview:** *Abstract Class:* This class is an abstract class which holds objects which describe the lattice structure of the repeating feature. These objects also hold the calculations required to determine the structural contribution to the scattering. Structure factors solved using these methods can be solved by integrating over the course q-spacings to reduce errors introduced by aliasing. Currently supported classes are:

*Rectilinear:* A lattice structure that is spaced evenly in the x and y direction and is aligned with these directions:

```
00000000
```

```
00000000
```

```
00000000
```

*Hexagonal:* A lattice structure that is packed in a hexagonal ordering.

```
_00000000
```

```
0000000000
```

```
_000000000
```

**gauss\_normalize** (*args*)

**phase\_shift** (*Q*)

**Overview:** Used internally to solve a 0.5 phase shift for both the x and y directions.

**Parameters:**

*Q:* (*q\_space*) a *Q-space* object.

**Note:**

•Used to superimpose two rectilinear lattice structures over each other.

**rect\_ft** (*Q, repeat\_mod=None, None, None*)

**Overview:**

Solves the structure factor for the *Q* points in the *q-space* object. Using this method solves the structure factor before integrating over the *q* steps. If used explicitly without integrating over the *q* steps aliasing errors can be introduced into the scattering. This is especially true where the *q*-step is coarse in the *qx* direction which can lead to mismatch in intensities between the negative and positive *qx* diffraction peaks.

**Parameters:**

*Q:* (*q\_space*) a *Q-space* object.

*repeat\_mod:* (*float,[3],count*) A repeat modifier for the repeat attribute of a Lattice object. This is necessary when the effective repeat of a specific lattice type is different than the lattice spacing requested by the user.

**Note:**

•This calculation should be used in conjunction with integration over the *x* direction.

**x\_calc\_sfx** (*qx*)

**Overview:** Used internally to solve the structure factor for a given *qx* value in the *qx* direction. This is possible because the *qx*, *qy*, and *qz* components are separable.

**Parameters:**

*qx:* (*float*) a *qx* value.

**Note:**

•This method is used in conjunction with the integration call to obtain the structure factor that is returned.

**x\_calc\_sfx\_shift** (*qx*)

**Overview:**

Used internally to solve the structure factor for a given *qx* value in the *qx* direction. This solution applies a 0.5 phase shift to the wave solution which can be combined with the unshifted solution to give a solution to the scattering from a hexagonal lattice.

**Parameters:**

*qx:* (*float*) a *qx* value.

**Note:**

•This method is used in conjunction with the integration call to obtain the structure factor that is returned.

**x\_gauss\_sfx** (*qx, args*)

`y_calc_sfx(qy)`

**Overview:**

Used internally to solve the structure factor for a given qy value in the qy direction. This is possible because the qx, qy, and qz components are separable.

**Parameters:**

*qy*: (*float*) a qy value.

**Note:**

•This method is used in conjunction with the integration call to obtain the structure factor that is returned.

`y_calc_sfx_shift(qy)`

**Overview:**

Used internally to solve the structure factor for a given qy value in the qy direction. This solution applies a 0.5 phase shift to the wave solution which can be combined with the unshifted solution to give a solution to the scattering from a hexagonal lattice.

**Parameters:**

*qy*: (*float*) a qy value.

**Note:**

•This method is used in conjunction with the integration call to obtain the structure factor that is returned

```
class osrefl.model.sample_prep.Layer(SLD, thickness_value, center=(None, None, None),
                                     Ms=0.0)
Bases: osrefl.model.sample_prep.Shape
```

**Overview:**

Creates an object that extends the length and width of the unit cell but is parameterized in the thickness direction.

**Parameters**(*\_\_init\_\_*):

*SLD*: (*floatlangstroms^2*) The scattering length density of the Pyramid.

*thickness\_value*: (*floatlangstroms*) The thickness of the layer.

*center*: (*float,3langstroms*) The x, y, and z component of the central point of the layer. Although allowed to be provided, the x and y component play no role in the layer location, the pertinent parameter here is only the z component.

*Ms*: (*floatlangstroms*) The magnetic SLD of the material for this shape.

*describe*(*x\_points, y\_points, z\_points, cell\_to\_fill, mag\_to\_fill*)

**Overview:** This module takes in x,y, and z points and fills the matrix array with the SLD of the shape for the points that fall within the shape

**Parameters:**

*x\_points*: (*floatlangstroms*) an array of x points to be determined if they fall within the layer.

*y\_points*: (*floatlangstroms*) an array of y points to be determined if they fall within the layer.

*z\_points*: (*floatlangstroms*) an array of z points to be determined if they fall within the layer.

*cell\_to\_fill*: (*floatarraylangstroms*) This is the SLD matrix of the unit cell. It is filled by the render function.

*mag\_to\_fill*: (*floatarraylangstroms*) This is the Ms matrix of the unit cell. It is filled by the render function.

`height()`

**Overview:**

Returns the total height of the layer. This differs from thickness which only describes the thickness of the individual layer whereas this method returns the maximum z-value of the shape in the unit cell.

`is_core_of(element, offset=(0, 0, 0))`

**Overview:**

This method is used to place the Shape 'self' into the center of the Shape element. This creates a core shell type geometry.

**Parameter:**

*element*: (*Shape*) The Shape object that is being embedded into the selected Shape object 'self' (The shapes who's center value is being altered).

**Note:**

•In the case of use with a Layer Object, the Shape object will only be placed in the center in the z-direction because it's location in the x-y plane does not make a difference.

`on_top_of(element)`

**Overview:** This method alters the center value of a shape object so that the Shape who's on\_top\_of method was called is located on top of the Shape 'element'.

**Parameter:**

*element*: (*Shape*) The Shape object that is being put on top of the selected Shape object 'self' (The shapes who's z-component center value is being altered).

`thickness()`

**Overview:**

Returns the total thickness of the layer.

```
class osrefl.model.sample_prep.COMMUNIT
Bases: object
```

**Overview:**

This class is used to create a Unit\_Cell object from a .onit file output from the Object Oriented Micromagnetic Framework (OOMMF) software. It will produce a unit cell which has the structure unit array as well as a list of three arrays of the same size as the unit array which contains each of the three magnetic vector components. This information allows for the calculation of the four magnetic scattering cross-sections for the given system.

**Note:**

•Currently, this only supports systems that are constant in the z-direction, (eg. an SEM image of the feature is used as a mask to create a 2D image that OOMMF then calculates the minimized magnetic character for. The results are assumed to be consistent through the depth of the shape. This also only supports single feature unit cells.

`unitBuild(SLD, inc_sub=(None, None))`

**Overview:**

Producer method: This method produces a Unit\_Cell object from the parameters that are carried over from the .omf loaded by the OOMMFUnit class. The object that this method produces is needed to work with the calculation API.

**Parameters:**

*SLD: (floatangstroms^2)* When doing an OOMMF simulation, the software does not care about the structural SLD, however, to calculate the full off-specular scattering this information is needed. At the time of Unit\_Cell creation the user must specify the structural SLD for the magnetic feature being loaded by OOMMFUnit.

**Note:**

- Produces a Unit\_Cell object.
- This may not be the best place for the SLD input. This will have to be evaluated.

```
class osrefl.model.sample_prep.ParallelLapped (SLD, dim, center=[None, None, None],
                                             Ms=0.0)
Bases: osrefl.model.sample_prep.Shape
```

**Overview:** Uses the generic formula for a parallelapped feature to create a parallelapped object

**Parameters( \_\_init\_\_ ):**

*SLD: (floatangstroms^2)* The scattering length density of the sphere.

*dim: (float,3)langstroms)* x, y and z dimensions of the feature.

*center: (float,3)langstroms)* The x, y, and z component of the central point of the sphere. In the case that the center is set to [None,None,None] the shape will be put in the bottom corner of the unit cell (the bounding box will start at (0,0,0)).

*Ms: (floatangstroms)* The magnetic SLD of the material for this shape.

**discritize (x\_points, y\_points, z\_points, cell\_to\_fill, mag\_to\_fill)**

**Overview:**

This module takes in x,y, and z points and fills the matrix array with the SLD of the shape for the points that fall within the shape

**Parameters:**

*x\_points (floatangstroms)* an array of x points to be determined if they fall within the parallelapped.

*y\_points (floatangstroms)* an array of y points to be determined if they fall within the parallelapped.

*z\_points(floatangstroms)* an array of z points to be determined if they fall within the parallelapped.

*cell\_to\_fill (float,arraylangstroms)* This is the SLD matrix of the unit cell. It is filled by the render function.

*mag\_to\_fill(float,arraylangstroms)* This is the Ms matrix of the unit cell. It is filled by the render function.

**height ()**

**Overview:**

Returns the total height of the parallelapped. This differs from thickness which only describes the thickness of the individual sphere whereas this method returns the maximum z-value of the shape in the unit cell.

```
is_core_of (element, offset=(0, 0, 0))
```

**Overview:**

This method is used to place the Shape 'self' into the center of the Shape element. This creates a core shell type geometry.

**Parameter:**

*element: (Shape)* The Shape object that is being embedded into the selected Shape object 'self' (The shapes who's center value is being altered).

**Note:**

- In the case of use with a Layer Object, the Shape object will only be placed in the center in the z-direction because it's location in the x-y plane does not make a difference.

**length ()**

**Overview:**

Returns the maximum length of the parallelapped (x direction)

**on\_top\_of (element)**

**Overview:** This method alters the center value of a shape object so that the Shape who's on\_top\_of method was called is located on top of the Shape 'element'.

**Parameter:**

*element: (Shape)* The Shape object that is being put on top of the selected Shape object 'self' (The shapes who's z-component center value is being altered).

**thickness ()**

**Overview:**

Returns the total thickness of the parallelapped.

**width ()**

**Overview:**

Returns the maximum width of the parallelapped (y direction)

```
class osrefl.model.sample_prep.Pyramid (SLD, dim, sub=None, center=[None, None, None],
                                       Ms=0.0)
Bases: osrefl.model.sample_prep.Shape
```

**Overview:** Uses the generic formula for a Pyramid feature to create a Pyramid object.

**Parameters( \_\_init\_\_ ):**

*SLD: (floatangstroms^2)* The scattering length density of the Pyramid.

*dim: (float,3)langstroms)* The x component, y component and thickness of the cone respectively, x is the length of the Pyramid base and y is the width of the Pyramid base.

*sub: (floatangstroms)* provides a hard cut-off for the thickness of the Pyramid, this allows for the creation of a trapezoidal object who side slope can be altered by using different z component values while keeping the sub parameter fixed.

*center: (float,3)langstroms)* The x, y, and z component of the central point of the Pyramid. In the case that the center is set to [None,None,None] the shape will be put in the bottom corner of the unit cell (the bounding box will start at (0,0,0)).

*Ms: (floatangstroms)* The magnetic SLD of the material for this shape.

**discritize (x\_points, y\_points, z\_points, cell\_to\_fill, mag\_to\_fill)**

**Overview:**

This module takes in x,y, and z points and fills the matrix array with the SLD of the shape for the points that fall within the shape

**Parameters:**

**x\_points: (float|langstroms)** an array of x points to be determined if they fall within the Pyramid.

**y\_points: (float|langstroms)** an array of y points to be determined if they fall within the Pyramid.

**z\_points: (float|langstroms)** an array of z points to be determined if they fall within the Pyramid.

**cell\_to\_fill: (float|array|langstroms)** This is the SLD matrix of the unit cell. It is filled by the render function.

**mag\_to\_fill: (float|array|langstroms)** This is the Ms matrix of the unit cell. It is filled by the render function.

**height ()**

**Overview:**

Returns the total height of the Pyramid. This differs from thickness which only describes the thickness of the individual Pyramid whereas this method returns the maximum z-value of the shape in the unit cell.

**is\_core\_of (element, offset=(0, 0, 0))**

**Overview:**

This method is used to place the Shape 'self' into the center of the Shape element. This creates a core shell type geometry.

**Parameter:**

**element: (Shape)** The Shape object that is being embedded into the selected Shape object 'self' (The shapes who's center value is being altered).

**Note:**

\*In the case of use with a Layer Object, the Shape object will only be placed in the center in the z-direction because it's location in the x-y plane does not make a difference.

**length ()**

**Overview:**

Returns the maximum length of the Pyramid (x direction)

**on\_top\_of (element)**

**Overview:** This method alters the center value of a shape object so that the Shape who's on\_top\_of method was called is located on top of the Shape element.

**Parameter:**

**element: (Shape)** The Shape object that is being put on top of the selected Shape object 'self' (The shapes who's z-component center value is being altered).

**thickness ()**

**Overview:**

Returns the total thickness of the Pyramid.

**width ()**

**Overview:**

Returns the maximum width of the Pyramid (y direction)

class osrefl.model.sample\_prep.Q\_space (*minimums, maximums, points*)

Bases: osrefl.model.sample\_prep.Space

**Overview:**

Holds all of the information for the q-space output for which the scattering will be solved. Many of the attributes provided in this class make access to information about the scattering easier.

**Parameters: (self, ...):**

**minimums: (float|3|langstroms)** The minimum q values that the user would like solved. The data is in the form: [minimum x, minimum y, minimum z]

**maximums: (float|3|langstroms)** The maximums q values that the user would like solved. The data is in the form: [maximums x, maximums y, maximums z]

**points: (float|3|langstroms)** The number of points that the user would like the provided q space (defined by the minimums and maximums) split into. The data is in the form: [x points, y points, z points]

**Parameters: (Class):**

**q\_step: (float|3|langstroms^-1)** The reciprocal space step size for the xy and z dimensions.

**q\_list: (float|3|array|langstroms^-1)** The total list of values being solved for in the x, y and z directions.

**q\_refract: (float|array|langstroms^-1)** When the neutron beam is transmitted through a substrate, the beam refracts, altering the effective qx value. This is recorded in this variable. Its value is dependent on the ki and ko values for a specific qx,qy, qz combination.

**k\_space: (float|array|langstroms)** This is the equivalent k-space values for the given set of q values.

**getExtent ()**

**Overview:**

This method is used to get the minimum and maximum plot area of the Q\_space object which can be directly fed to a pylab plotting object.

**Returns:**

(array|langstroms^-1) Returns an array in the form [ $Q_x^{min}$ ,  $Q_x^{max}$ ,  $Q_z^{min}$ ,  $Q_z^{max}$ ]

**getkSpace (wavelength)**

**Overview:**

This method creates an attribute which holds the equivalent k-space values for a given set of Qs.

**Returns: (array|langstroms)**

**normalize ()**

**Overview:**

Creates 3 arrays which contain the qx, qy, and qz value which are normalized by the total q magnitude.

**Returns:**

(list,3D array|langstroms^-1) The normalized Q values.

**vectorize (type='float')**

**Overview:**

Turns the `q` information given by a `Q_space` object into vectors to allow for vector math. Uses the `numpy` reshape functionality.

**Parameters:**

*types(r)*: Allows the user to define the type of the numbers that `q` is. (eg. float, complex)

```
class osrefl.model.sample_prep.Rectilinear (repeat, unit)
```

```
Bases: osrefl.model.sample_prep.Lattice
```

**Overview:**

A lattice structure that is spaced evenly in the `x` and `y` direction and is aligned with these directions. This is essentially a gridded ordering. The ASCII art represents this structure.

0 = feature

~ = spacing

```
0-0-0-0-0
```

```
0-0-0-0-0
```

```
0-0-0-0-0
```

**Parameters:**

*Repeat* (`float`,`3`,`count`) The number of repeats of the unit cell in the `x`, `y` and `z` direction.

*Unit*: (`Unit_Cell`) a `Unit_Cell` object from which the unit cell length, width and height parameters can be obtained.

**Note:**

- This is a Lattice object and can use methods from Lattice.

```
gauss_normalize (args)
```

```
gauss_struct_calc(Q, struct/fract=False)
```

**Overview:**

This structure calculation applies a gaussian convolution to the delta function diffraction peaks produced by the structure factor to produce a more accurate theory function. The convolution represents the combination of the diffraction from the lattice with the coherence length of the probing beam.

**Parameters:**

*Q*: (`Q_space`) The scattering produced by the structure factor of the model is calculated for the `q` range supplied by this `Q_space` object.

```
phase_shift_ft(Q)
```

**Overview:** Used internally to solve a 0.5 phase shift for both the `x` and `y` directions.

**Parameters:**

*Q*: (`q_space`) a `Q_space` object.

**Note:**

- Used to superimpose two rectilinear lattice structures over each other.

```
rect_ft(Q, repeat_mod=(None, None, None))
```

**Overview:**

Solves the structure factor for the `Q` points in the `q_space` object. Using this method solves the structure factor before integrating over the `q` steps. If used explicitly without integrating over the `q` steps aliasing errors can be introduced into the scattering. This is especially true where the `q-step` is coarse in the `qx` direction which can lead to mismatch in intensities between the negative and positive `qx` diffraction peaks.

**Parameters:**

*Q*: (`q_space`) a `Q_space` object.

*repeat\_mod*: (`float`,`3`,`count`) A repeat modifier for the repeat attribute of a Lattice object. This is necessary when the effective repeat of a specific lattice type is different than the lattice spacing requested by the user.

**Note:**

- This calculation should be used in conjunction with integration over the `x` direction.

```
struct_calc(Q)
```

**Overview:**

Returns the structure factor for a rectilinear lattice integrated over the `qx` steps.

**Parameters:**

*Q*: (`q_space`) = a `Q_space` object.

**Note:**

- The direct solution is calculated to get the `y` and `z` structural components. The integrated solution for the `qx` direction is then solved and applied over the direct solution. This is somewhat inefficient and can be streamlined.

```
theta_struct_calc(theta)
```

```
x_calc_sfx(qx)
```

**Overview:** Used internally to solve the structure factor for a given `qx` value in the `qx` direction. This is possible because the `qx`, `qy`, and `qz` components are separable.

**Parameters:**

*qx*: (`float`) a `qx` value.

**Note:**

- This method is used in conjunction with the integration call to obtain the structure factor that is returned.

```
x_calc_sfx_shift(qx)
```

**Overview:**

Used internally to solve the structure factor for a given `qx` value in the `qx` direction. This solution applies a 0.5 phase shift to the wave solution which can be combined with the unshifted solution to give a solution to the scattering from a hexagonal lattice.

**Parameters:**

*qx*: (`float`) a `qx` value.

**Note:**

- This method is used in conjunction with the integration call to obtain the structure factor that is returned.

```
x_gauss_sfx(qx, args)
```

`y_calc_sfx(qy)`

**Overview:**

Used internally to solve the structure factor for a given qy value in the qy direction. This is possible because the qx, qy, and qz components are separable.

**Parameters:**

*qy*: (float) a qy value.

**Note:**

•This method is used in conjunction with the integration call to obtain the structure factor that is returned.

`y_calc_sfx_shift(qy)`

**Overview:**

Used internally to solve the structure factor for a given qy value in the qy direction. This solution applies a 0.5 phase shift to the wave solution which can be combined with the unshifted solution to give a solution to the scattering from a hexagonal lattice.

**Parameters:**

*qy*: (float) a qy value.

**Note:**

•This method is used in conjunction with the integration call to obtain the structure factor that is returned

`class osrefl.model.sample_prep.RoundedPzPip(SLD, dim, center=None, None, None, curve=0.0, Ms=0.0)`

Bases: `osrefl.model.sample_prep.Shape`

**Overview:** It is rarely the case that a sample has totally sharp corners. This shape allows the user to determine the extent to which the corners are rounded.

**Parameters( *\_\_init\_\_* ):**

*SLD*: (float/langstroms<sup>2</sup>) The scattering length density of the sphere.

*dim*: (float,3/langstroms) x, y and z dimensions of the feature.

*center*: (float,3/langstroms) The x, y, and z component of the central point of the sphere. In the case that the center is set to [None, None, None] the shape will be put in the bottom corner of the unit cell (the bounding box will start at (0,0,0).

*Ms*: (float/langstroms) The magnetic SLD of the material for this shape.

**describe**(*x\_points*, *y\_points*, *z\_points*, *cell\_to\_fill*, *mag\_to\_fill*)

**Overview:**

This module takes in x, y, and z points and fills the matrix array with the SLD of the shape for the points that fall within the shape

**Parameters:**

*x\_points* (float/langstroms) an array of x points to be determined if they fall within the parallellapped.

*y\_points* (float/langstroms) an array of y points to be determined if they fall within the parallellapped.

*z\_points* (float/langstroms) an array of z points to be determined if they fall within the parallellapped.

## 2.1. Model Creation

29

`cell_to_fill`(float,array/langstroms) This is the SLD matrix of the unit cell. It is filled by the render function.

`mag_to_fill`(float,array/langstroms) This is the Ms matrix of the unit cell. It is filled by the render function.

`height`()

**Overview:**

Returns the total height of the parallellapped. This differs from thickness which only describes the thickness of the individual sphere whereas this method returns the maximum z-value of the shape in the unit cell.

`is_core_of`(*element*, *offset*=(0, 0, 0))

**Overview:**

This method is used to place the Shape 'self' into the center of the Shape element. This creates a core shell type geometry.

**Parameter:**

*element*: (Shape) The Shape object that is being embedded into the selected Shape object 'self' (The shapes who's center value is being altered).

**Note:**

•In the case of use with a Layer Object, the Shape object will only be placed in the center in the z-direction because it's location in the x-y plane does not make a difference.

`length`()

**Overview:**

Returns the maximum length of the parallellapped (x direction)

`on_top_of`(*element*)

**Overview:** This method alters the center value of a shape object so that the Shape who's `on_top_of` method was called is located on top of the Shape 'element'.

**Parameter:**

*element*: (Shape) The Shape object that is being put on top of the selected Shape object 'self' (The shapes who's z-component center value is being altered).

`thickness`()

**Overview:**

Returns the total thickness of the parallellapped.

`width`()

**Overview:**

Returns the maximum width of the parallellapped (y direction)

`class osrefl.model.sample_prep.Scene`(*shapes*=[[]])

Bases: object

**Overview:**

This class is used to aggregate the different Shape objects that form a complete unit cell. It is used primarily by GeomUnit as a queue of objects that can be rendered into the unit cell array.

**Parameters:**

`shapes`:list(Shape[]) = a list of Shape objects to be put into the scene.

30

**Note:**

- The Shapes may be entered as a list or as individual items.

**add\_element** (*element*)

**Overview:**

Adds a shape object to the Scene object. This may be useful in the case where the Scene has been created and the user just wants to add one more Shape to it.

**Parameters:**

*element* A Shape object to add to a scene.

**query\_center** (*limit= 'min'*)

**Overview:**

This method returns the vertical component of the center points of the Shape objects in the Scene.

**Parameters:**

*limit: (string)* Allows the user to filter through the vertical center values of the Shape objects to return only the desired result.

**Returns**

- 'max' returns the highest center value in the scene.
- 'min' returns the lowest center value in the scene. This is the location of the vertical component of the center values in real space.
- 'all' returns an array of all of the centers of all of the shapes.

**query\_height** (*limit= 'max'*)

**Overview:**

This method determines the maximum height of the Scene object (with an option to return the minimum if limit is given a value). This is not a thickness measurement so the highest Shape in the scene is not necessarily the thickest.

**Parameters:**

*limit: (string)* Allows the user to filter through the heights of the

**Returns**

Shape objects to return only the desired result.

- 'max' returns the highest shape in the scene.
- 'min' returns the lowest shape in the scene. This is the location of the top of the shape who's top is lowest in real space.
- 'all' returns an array of all of the heights of all of the shapes.

**Note:**

•Each shape in a Scene has its own height method. When this module is called, it searches through the heights of all of the Shape objects in Scene to determine what the height of the unit cell should be. It then records this value in Dxyz as the z value.

**class** `osrefl.model.sample_prep.Shape`

Bases: `object`

**Overview:****2.1. Model Creation**

31

*Abstract Class:* The different possible shape descriptions that can be used to build the unit cell. This class allows for the definition of shapes for a unit cell with different properties to be treated, on a fundamental level, as a geometric structure that can be added to a unit cell.

**Note:**

•Although all shapes are different, they all have the notion of a 'bounding box', which is a Parallellapped shape that can encompass the shape. Methods that treat the bounding box of a shape rather than the individual attributes that make the shape are held in this class.

**is\_core\_of** (*element, offset=(0, 0, 0)*)

**Overview:**

This method is used to place the Shape 'self' into the center of the Shape element. This creates a core shell type geometry.

**Parameter:**

*element: (Shape)* The Shape object that is being embedded into the selected Shape object 'self' (The shapes who's center value is being altered).

**Note:**

•In the case of use with a Layer Object, the Shape object will only be placed in the center in the z-direction because it's location in the x-y plane does not make a difference.

**on\_top\_of** (*element*)

**Overview:** This method alters the center value of a shape object so that the Shape who's `on_top_of` method was called is located on top of the Shape 'element'.

**Parameter:**

*element: (Shape)* The Shape object that is being put on top of the selected Shape object 'self' (The shapes who's z-component center value is being altered).

**class** `osrefl.model.sample_prep.Space`

Bases: `object`

**Overview:**

*Abstract Class:* - This is an object that holds the information about the space that the theory function is being calculated for.

**class** `osrefl.model.sample_prep.Sphere` (*SLD, r=1/0, center=(None, None, None), Ms=0/0*)

Bases: `osrefl.model.sample_prep.Shape`

**Overview:**

Uses the generic formula for a sphere to create a sphere object.

**Parameters**(\_\_init\_\_):

*SLD: ((floatangstroms^-2)* The scattering length density of the sphere.

*r: ((floatangstroms)* The radius of the sphere.

*center: ((float,3)langstroms)* The x, y, and z component of the central point of the sphere. In the case that the center is set to [None, None, None] the shape will be put in the bottom corner of the unit cell (the bounding box will start at (0,0,0).

*Ms: ((floatangstroms)* The magnetic SLD of the material for this shape.

**discritize** (*x\_points, y\_points, z\_points, cell\_to\_fill, mag\_to\_fill*)

**Overview:**

32

This module takes in x,y, and z points and fills the matrix array with the SLD of the shape for the points that fall within the shape

**Parameters:**

*x\_points: (floatlangstroms)* an array of x points to be determined if they fall within the sphere.

*y\_points: (floatlangstroms)* an array of y points to be determined if they fall within the sphere.

*z\_points: (floatlangstroms)* an array of z points to be determined if they fall within the sphere.

*cell\_to\_fill: (floatarraylangstroms)* This is the SLD matrix of the unit cell. It is filled by the render function.

*mag\_to\_fill (floatarraylangstroms)* This is the Ms matrix of the unit cell. It is filled by the render function.

**height ()**

Overview:

Returns the total height of the sphere. This differs from thickness which only describes the thickness of the individual sphere whereas this method returns the maximum z-value of the shape in the unit cell.

**is\_core\_of (element, offset=(0, 0, 0))**

Overview:

This method is used to place the Shape 'self' into the center of the Shape element. This creates a core shell type geometry.

**Parameter:**

*element: (Shape)* The Shape object that is being embedded into the selected Shape object 'self' (The shapes who's center value is being altered).

**Note:**

•In the case of use with a Layer Object, the Shape object will only be placed in the center in the z-direction because it's location in the x-y plane does not make a difference.

**length ()**

Overview:

Returns the maximum length of the sphere (x direction)

**on\_top\_of (element)**

Overview: This method alters the center value of a shape object so that the Shape who's on\_top\_of method was called is located on top of the Shape element .

**Parameter:**

*element: (Shape)* The Shape object that is being put on top of the selected Shape object 'self' (The shapes who's z-component center value is being altered).

**thickness ()**

Overview:

Returns the total thickness of the sphere.

**width ()**

Overview:

Returns the maximum width of the sphere (y direction)

`class osrefl.model.sample_prep.Theta_space (minimums, maximums, points)`

Bases: `osrefl.model.sample_prep.Space`

**Overview:**

In some cases, it may be desirable to calculate the scattering from a model in theta space. This object acts like a Q\_space object but the calculations are carried out in real space not reciprocal space.

**Parameters( \_\_init\_\_ )::**

*minimums: (float(2)langstroms)* The minimum theta values that the user would like solved. The data is in the form:  $[\theta_{min}^x, \theta_{min}^y]$

*maximums: (float(2)langstroms)* The maximum theta values that the user would like solved. The data is in the form:  $[\theta_{max}^x, \theta_{max}^y]$

*points: (float(2)langstroms)* The number of points that the user would like to calculate for. (defined by the minimums and maximums) split into. The data is in the form:  $[\theta_{in}^x, \theta_{out}^x]$

**Parameters(Class):**

*theta\_step: (float(2)degrees)* Step size in  $\theta_{in}$  and  $\theta_{out}$

*theta\_list: (float(2)array[degrees])* The total list of values being solved for in  $\theta_{in}$  and  $\theta_{out}$ .

**q\_calc (w/l)**

Overview:

This calculates the total Q vector based on the given theta values

**Parameters**

*w/l: (floatlangstroms)* The wavelength of the probing beam.

**Return:**

*q\_vector: (arraylangstroms^\*1)* An array of Q vectors calculated for the combination of  $\theta_{in}$  and  $\theta_{out}$  values for this object.

**vectorize (type='float', unit='deg')**

Overview:

Turns the theta information given by a theta\_space object into vectors to allow for vector math. Uses the numpy reshape functionality.

**Parameters:**

*type(str):* Allows the user to define the type of the numbers that theta is. (eg: float, complex)

```
class osrefl.model.sample_prep.Unit_Cell (Dy%: n, unit, value_list, step, mag_unit=None,
magVec=[None, None, None], inc_sub=[None,
None], rawUnit=None)
```

Bases: object

**Overview:**

Contains the information for the processed unit cell information. This class makes the implementation of scattering calculations easier by creating one structure that only contains information necessary for the theory function calculation, regardless of how that information was obtained.

**Producer Classes:**

*GeomUnit*

Uses geometric shape parameters to calculate the discretized unit cell. Magnetic support is included in the form of a unit cell which contains the magnetic SLD values.

*OOMMUnit*

Uses the *omf* output format produced by the Object Oriented MicroMagnetic Framework software. To created both the structure and magnetic scattering arrays. The *Unit\_Cell* objected created by this class will contain the structural SLD as well as a list of arrays which contain the x,y, and z magnetic components. This can be used by the magnetic approximations to calculate the four magnetic cross-sections.

*K3DUnit*

Loads a raw data fill which is exported by the K3d modeling software K-3D software. This type of creation does not support magnetic representations.

*GratingUnit*

Loads a png image and turns it into a 3D SLD array. It extends the image in the y direction where the sensitivity to scattering is low.

**add\_media ()**

**Overview:**

Adds a top and bottom layer to be the SLD of the incident medium and the substrate.

**Note:**

•This addition is important for the DWBA modeling where the calculation assumes that the top and bottom layer are semi-infinite.

**generateMIE (mifData=None)**

**mag\_view ()**

**Overview:**

Outputs a 3D rendered viewing of the magnetic unit cell array using MayaVI.

**repeat (xy\_repeat)**

**Overview:**

Creates copies of the single unit cell array in the x-y direction as specified by the user.

**Parameters:**

*xy\_repeat: (int,[2]count)* The number of times the unit cell is repeated in the x and y direction (including the original unit).

**view ()**

**Overview:**

Outputs a 3D rendered viewing of the unit cell array using MayaVI.

**viewSlice ()**

## 2.2 Calculations

### 2.2.1 osrefl.theory.scatter

**class osrefl.theory.scatter.Calculator (lattice, probe, space, feature, omf=None)**

**Bases:** object

**Overview:**

This holds all of the information for calculation of scattering for reflectometry. This allows a user to build a sample, request an output and based on an approximation choice, produce scattering.

### 2.2. Calculations

**Parameters:**

*lattice: (Lattice)* see *Lattice* for more information.

*probe: (Beam)* see *Beam* for more information.

*space (space)* see *space* or *-sample\_prep.thru.space* for more information.

*feature: (Unit\_Cell)* see *Unit\_Cell* for more information.

**omf: (OmF)** This is an object which holds the magnetic moment information about the sample. It contains three arrays of the the same size as the unit cell which hold each of the x, y, and z components of the magnetic moment.

**BA ()**

**Overview:**

This Born Approximation calculation is written entirely in Python and assumes that the scattered beam is so small that the transmitted beam is essentially =1. This makes for a simple calculation, however, it does not allow for the capturing of the dynamically effects seen in real scattering. Because of the simplistic nature of this calculation, some tricks can be used to speed up the calculation. This version of the BA calculation uses a chirp-z transform (CZT) to solve the Form Factor. The chirp-z is essentially a FFT which allows for solving the transform anywhere on the sphere. With this, we can solve for any Q range without wasting any resources calculating for areas we don't need.

The Form Factor calculation is:

$$FF = abs \left( \frac{-i}{4\pi yz} * (1.0 - exp^{i(q_x y + \Delta d_x y)}) * CZT\{omf\} \right)^2$$

It is also normalized by the surface area:

$$NormFactor = \left( \frac{4 * \pi}{q_x y} * D_{x,y} \right)^2$$

For the formalism to the structure factor see *Rectilinear ()* or *Hexagonal ()*

**DWBA (refract=True)**

**Overview:**

**SMBA ()**

**Overview:**

This is a Python implementation of the *cudaSMBA ()*. It is significantly slower and was only really used for testing and validation purposes. Still, it may be useful in the future and is available in this package.

**SMBAfft (precision='float32', refract=True)**

**Overview:**

**cudaBA (precision='float32', refract=True)**

**Overview:**

This version of the Born Approximation (BA) uses a scattering kernel that is written in C++ and was developed for solving the Substrate Modified Born Approximation(SMBA) (see *cudaSMBA ()*). This kernel normally takes in a set of incoming and outgoing wave functions to perturb the probing wave with. Because the BA assumes that the wavefunction does not change as a function of sample penetration. The incoming and outgoing wavefunctions are set so that  $t = 1$ , effectively solving the long hand version of the BA (see *longBA ()*).

The advantage of this method is that it can distribute the calculation across multiple GPU devices solving the problem significantly faster.

This form factor is also *normalized*

For the formalism to the structure factor see `Rectilinear()` or `Hexagonal()`

#### Parameters *precision*: (striprecision)

This parameters allows the user to toggle between `float32` and `float64` precision. For most nvidia graphics cards, the `float32` is handled better and makes for significantly faster calculations.

*refract*: (bool)

This parameters toggles the refractive shift calculation. Generally, the refractive index of the substrate of a sample cause a shift in effective Q below the horizons, setting `refract` to `TRUE` will cause a shift of:

$$q_x + \lambda * P_{\text{substrate}}$$

at  $-q_x$  values and:

$$q_x - \lambda * P_{\text{substrate}}$$

`cudaMBA` (*precision*=`'float32'`, *refract*=`True`)

`cudaSMB` (*precision*=`'float32'`, *refract*=`True`)

#### Overview

The Substrate Modified Born Approximation (SMBA) is a variation of the Born Approximation (BA) where by the scattering is perturbed by the wavefunction of the income and outgoing wavefunction as it interacts with the incident media/substrate interface. This perturbation gives rise to the horizons of the sample where the beam enters directly from the side face of the substrate.

This calculation uses C++ calculation kernels on the nvidia GPUs. It uses binders from pyCuda to simplify the kernel parallelization. There are two C++ calculation kernels used for this calculation.

The first kernel can be found in `wavefunction_kernel.cc`, it solves for the wavefunction for a wave interacting with the incident media/substrate interface, it first calculates the scattering vector:

$$k_j = n_j k_0 = \sqrt{1 - \frac{4\pi\rho_j}{k_0}}$$

Once this is solved for, the reflection and transmission is calculated for the substrate/incident media stack. This is a matrix equation:

$$\overline{M}(\Delta z) = \begin{pmatrix} \cos(k_i \Delta z) & \frac{1}{k_i} \sin(k_i \Delta z) \\ -k_i \sin(k_i \Delta z) & \cos(k_i \Delta z) \end{pmatrix}$$

and the reflection is:

$$r = \frac{M_{1,1} + (i * n_0 * M_{0,1}) + \frac{n_2^2}{n_j} * (-M_{1,0} - i * n_0 * M_{0,0})}{-M_{1,1} + i * n_0 * M_{0,1} + \frac{n_2^2}{n_j} (M_{1,0} - i * n_0 * M_{0,0})}$$

and the transmitted beam is:

$$t = 1.0 + r$$

Now the wavefunction for the perturbation is solved for. The wave function used for the perturbation is dependent on the direction of the incoming and outgoing beam:

For  $k_{in} < 0.0$ :

$$\Psi_{in1} = t * \exp^{-ik_i \Delta r_z}$$

$$\Psi_{in2} = 0.0$$

For  $k_{in} > 0.0$ :

$$\Psi_{in1} = 1 * \exp^{-ik_i \Delta r_z}$$

$$\Psi_{in2} = r * \exp^{-ik_i \Delta r_z}$$

For  $k_{out} < 0.0$ :

$$\Psi_{out1} = 1 * \exp^{-ik_i \Delta r_z}$$

$$\Psi_{out2} = r * \exp^{-ik_i \Delta r_z}$$

For  $k_{out} > 0.0$ :

$$\Psi_{out1} = t * \exp^{-ik_i \Delta r_z}$$

$$\Psi_{out2} = 0.0$$

With these pieces of information, the SMBA can be solved for. The final form factor is:

$$FF = \left( \frac{-i}{(q_x q_z)} * (1.0 - \exp^{i q_x q_z * \Delta r_x q_z}) * \left[ \Psi_{in} * \sum_{n=0}^{D_{z, n1z}} \{ r h_{0, n1z} * \exp^{i Q_{z, n1z} * D_{z, n1z}} \} * \Psi_{out} \right] \right)^2$$

This form factor is also *normalized*

For the formalism to the structure factor see `Rectilinear()` or `Hexagonal()`

Parameters *precision*: (striprecision)

This parameters allows the user to toggle between float32 and float64 precision. For most nvidia graphics cards, the float32 is handled better and makes for significantly faster calculations.

*refract*: (bool)

This parameters toggles the refractive shift calculation. Generally, the refractive index of the substrate of a sample cause a shift in effective Q below the horizons, setting `refract` to `TRUE` will cause a shift of:

$$q_x + \lambda * P_{\text{substrate}}$$

at  $-q_x$  values and:

$$q_x - \lambda * P_{\text{substrate}}$$

**fitCompare** (*other*: *extraCompare=None*, *niles=[ 'other', 'self' ]*)

**Overview:**

This method plots two different data sets on the sample window for an easy visual comparison of the data.

**Warning:** This method is obsolete!

**generalCompare** (*otherData*, *niles*)

**longBA** ()

**Overview:**

For testing and validation, it can be handy to have a long-hand version of the Born Approximation. This BA is written entirely in Python and solves the form factor using an explicit sum instead of the FFT or CZT modules. It is slow! The form factor is:

$$F F = a b s \left( \frac{-i}{q_x q_z} * (1.0 - \exp^{i(q_x a_{0,0} + \Delta d_{0,0})}) * \sum_{n=0}^{D_{\text{max}}} \{ r h o_{0,n} * \exp^{i(q_x a_{0,0} + D_{0,n})} \} \right)^2$$

This form factor is also *normalized*

For the formalism to the structure factor see `Rectilinear ()` or `Hexagonal ()`

**magneticBA** ()

**Overview:**

This calculation solves the Born Approximation for a magnetic sample using an `OmF`.

For any magnetic scattering, four cross-sections must be solved for. First, the magnetic scattering length density must be obtained. This can be found

**Parameters:**

*struc\_cell*: (**float:3D array**)`langstroms^2`) The structural scattering potential of the feature being scattered off of.

*Q*: (**q\_space**) A `Q`-space object that holds all of the information about the desired q space output.

*lattice*: (**Lattice**) A lattice object that holds all of the information needed to solve the structure factor of the scattering.

*space*: (**Space**) Holds all of the information about the experimental beam needed to apply beam dependent corrections to the data.

*omf*: (**OmF**) This is an object which holds the magnetic moment information about the sample. It contains three arrays of the the same size as the unit cell which hold each of the x, y, and z components of the magnetic moment.

**partial\_magnetic\_BA** ()

**Overview:**

This calculation does the magnetic born approximation but assumes that the contribution to the magnetic SLD from the qx and qy components of the magnetic moment are negligible and the whole system can be estimated as only containing magnetic contribution in the qz direction.

**Warning:** This method is not accurate for magnetic moments aligned in the q directions and should not be used!

**Parameters:**

*struc\_cell*: (**float:3D array**)`langstroms^2`) The structural scattering potential of the feature being scattered off of.

*mag\_cell*: (**float:3D array**)`langstroms^2`) The magnetic scattering potential of the feature being scattered off of.

*Q*: (**q\_space**) A `Q`-space object that holds all of the information about the desired q space output.

*lattice*: (**Lattice**) A lattice object that holds all of the information needed to solve the structure factor of the scattering.

*beam*: (**Beam**) Holds all of the information about the experimental beam needed to apply beam dependent corrections to the data.

**partial\_magnetic\_BA\_long** ()

**qz\_slice** (*qz=0.0*)

**Overview:**

This plots a slice in the y direction designated by the *qz* in the case where there is corrected resolution data, it also will give the *qz* slice from the resolution corrected data.

**Parameter:**

*qz*: (**float**)`langstroms^-1`) The *qz* value that is being sliced over. This does not average over a range. It will choose the closest *qz* value that was solved for by the calculation and produce a log plot of the results.

**resolution\_correction** ()

**Overview:**

Applies a resolution correction to the data using the beam information from a `sample_prep.Beam` included in `scatter.Calculator` class. It applies a gaussian correction for the beam's angular divergence and the divergence in beam energy.

**Returns**

*self.corrected\_data* (**float**)`array`)`langstroms^-2`)

Fills in the the values for this attribute of the `scatter.Calculator`. If this method is not run, the value of this attribute is `None`.

**viewCor** ()

**Overview:**

Uses the `magPlotSlicer` module to view the resolution corrected models. This module includes tools for:

- Slice averaging for the data vertically and horizontally
- Viewing linear and log plots of both 2D slices and 3D image plots
- Altering of the color axis scale

`viewCorrector ()`

**Overview:**

Uses the magPlotSlicer.py module to view both the resolution corrected and uncorrected models.

This module includes tools for:

- Slice averaging for the data vertically and horizontally
- Viewing linear and log plots of both 2D slices and 3D image plots
- Altering of the color axis scale

`viewDcor ()`

**Overview:**

Uses the magPlotSlicer.py module to view the uncorrected models. This module includes tools for:

- Slice averaging for the data vertically and horizontally
- Viewing linear and log plots of both 2D slices and 3D image plots
- Altering of the color axis scale

`view_corrected (bl=None, vmin=None, vmax=None)`

**Overview:**

This plots the resulting scattering with the resolution correction. The user should make sure they have run the `scatter.resolution_correction ()` method before using this method.

**Parameters:**

*bl:* (str) This parameter can be used to change the title of the plot

*vmin:* (float)Angstroms<sup>-2</sup>) This is the minimum intensity value plotted on the 2D plot. Any intensity value below this value is plotted as the minimum.

*vmax:* (float)Angstroms<sup>-2</sup>) This is the maximum intensity value plotted on the 2D plot. Any intensity value below this value is plotted as the maximum.

`view_linear ()`

**Overview:**

Generally used for testing purposes, this view plots the intensity on a linear scale rather than a log scale. This can be useful when troubleshooting a calculation.

`view_uncorrected (bl=None, vmin=None, vmax=None)`

**Overview:**

This plots the resulting scattering without any resolution correction applied to the scattering calculation. This can be useful for studying the effects that the resolution has on the data measured from the instrument.

**Parameters:**

*bl:* (str) This parameter can be used to change the title of the plot.

*vmin:* (float)Angstroms<sup>-2</sup>) This is the minimum intensity value plotted on the 2D plot. Any intensity value below this value is plotted as the minimum.

*vmax:* (float)Angstroms<sup>-2</sup>) This is the maximum intensity value plotted on the 2D plot. Any intensity value below this value is plotted as the maximum.

## 2.3 Data Load

### 2.3.1 osrefl\_loaders.andr\_load

`class osrefl_loaders.andr_load.Data`

Base: object

**Overview:**

This is a data loader for .cgl files which is converted into a format that can be understood by the rest of software infrastructure.

`view ()`

**Overview:**

This module plots out the data for viewing.

## 2.4 .omf File Loader

### 2.4.1 osrefl\_model.omf\_loader

`class osrefl_model.omf_loader.Omf (filename=None)`

Base: object

**Overview:**

When the Object Oriented Micro Magnetic Framework (OOMMF) solves the magnetic minimization, it saves the results in a .omf file. This class allows the user to load the information from a .omf file about the magnetic moments in the sample and save them as a python array.

It also works for the oommf124pre-20080627 version of OOMMF

**Parameters**

*M* (array,float)angstrom): The total magnetic moment vector of the scattering.

*mx* (array,float)angstrom): The x component of the magnetic moment vector.

*my* (array,float)angstrom): The y component of the magnetic moment vector.

*mz* (array,float)angstrom): The z component of the magnetic moment vector.

*parameters* (dictionary,str) Holds a dictionary which is generated from the header of the .omf file.

This is useful for obtaining other information about the model run.

**Note:**

•This class contains other attributes which are not generally used for calculation purposes. The user should look in the code for information on these attributes.

**Warning:** The .omf file loaded by this module MUST be created from the mmDisplay screen.

Creating a .omf file for loading

- Run oommf.cel

- Select the appropriate server for processing the calculations(this is the local machine for non-distributed calculations.)
- Select oxssi from the mmlaunch box.
- In the Oxssi window select File->load and select the .nif file created by the OsRefl software. (see `sample_prep_Unit_Cell_generateMIF ()`)
- Run the magnetic minimization by pressing the "Run" button
- Add a mmlDisp from the mmlaunch menu

**Selection Input:**

Output	Destination	Schedule
Magnetization Output	mmlDisp<object for output>	Send Button

- In File->> Save As.. create a .omf file

**Note:**

- The omf loader supports *Text*, *Binary-4*, and *Binary-8* formats

**ConvertRho ()**

**Overview:**

There is a factor  $C'$  which is used to convert the magnetic moment to a magnetic scattering length density. Because the OOMMF software allows for different units, the  $C'$  must be chosen based on the OOMMF model.

**Returns** (array[3]|angstroms^-2)

**downsample** (*down\_factor=10*)

**Overview:**

This method resamples  $x, y$  data into bigger boxes. It does this by averaging the surrounding moments and assigning this weighted average to the rest of the data.

**Note:**

- $Q_x, Q_y$  resolution is typically much worse than exchange length  $Q_z$  resolution is pretty good, so this method does not resample in the  $z$  direction.

**Warning:** This module requires file `*rebin_simple.py*` which is not a common package.

**generate\_coordinates ()**

**Overview:**

Calculates the  $x, y$  and  $z$  values for each of the discretized units in the model from the information obtained from the header file.

**generate\_normalized\_m ()**

**Overview:**

The moments given in this file are the absolute magnitudes. This method normalizes the data by the total moment.

**view ()**

**view1xkdz** (*plot\_title=None, z\_layer=0*)

**Overview:**

This method shows a color plot of the angle between  $m_x, m_y$ .

CHAPTER  
THREE

---

## INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

## PYTHON MODULE INDEX

### 0

osref1.loaders.andr\_load, 42  
osref1.model.omf\_loader, 42  
osref1.model.sample\_prep, 9  
osref1.theory.scatter, 35

# INDEX

**A**  
add\_element() (osrefl.model.sample\_prep.Scene method), 31  
add\_medial() (osrefl.model.sample\_prep.Unit\_Cell method), 35  
**B**  
BA() (osrefl.theory.scatter.Calculator method), 36  
Beam (class in osrefl.model.sample\_prep), 9  
buildUnit() (osrefl.model.sample\_prep.GeomUnit method), 15  
**C**  
Calculator (class in osrefl.theory.scatter), 35  
Cone (class in osrefl.model.sample\_prep), 9  
ConvertRho() (osrefl.model.onf\_loader.Onf method), 43  
cutdAB() (osrefl.theory.scatter.Calculator method), 36  
cutdMagBA() (osrefl.theory.scatter.Calculator method), 37  
cutdMBA() (osrefl.theory.scatter.Calculator method), 37  
**D**  
Data (class in osrefl.loaders.andr\_load), 42  
discriuze() (osrefl.model.sample\_prep.Cone method), 10  
discriuze() (osrefl.model.sample\_prep.Ellipse method), 11  
discriuze() (osrefl.model.sample\_prep.Ellipsoid method), 13  
discriuze() (osrefl.model.sample\_prep.Pyramid method), 19  
discriuze() (osrefl.model.sample\_prep.Parallelepiped method), 21  
discriuze() (osrefl.model.sample\_prep.Parallelepiped method), 23  
discriuze() (osrefl.model.sample\_prep.Pyramid method), 24  
discriuze() (osrefl.model.sample\_prep.RoundedParPip method), 29  
discriuze() (osrefl.model.sample\_prep.Sphere method), 32  
discriuze() (osrefl.model.sample\_prep.Layer method), 33  
**E**  
Ellipse (class in osrefl.model.sample\_prep), 11  
Ellipsoid (class in osrefl.model.sample\_prep), 12  
**F**  
fitCompare() (osrefl.theory.scatter.Calculator method), 39  
**G**  
gauss\_normalize() (osrefl.model.sample\_prep.Hexagonal method), 16  
gauss\_normalize() (osrefl.model.sample\_prep.Lattice method), 19  
gauss\_normalize() (osrefl.model.sample\_prep.Rectilinear method), 27  
gauss\_sinc\_calc() (osrefl.model.sample\_prep.Rectilinear method), 27  
generalCompare() (osrefl.theory.scatter.Calculator method), 39  
generate\_coordinates() (osrefl.model.onf\_loader.Onf method), 43  
generate\_normalize\_m() (osrefl.model.onf\_loader.Onf method), 43  
generateMFF() (osrefl.model.sample\_prep.Unit\_Cell method), 35  
GeomUnit (class in osrefl.model.sample\_prep), 14  
getEvent() (osrefl.model.sample\_prep.Q\_space method), 26  
getKSpace() (osrefl.model.sample\_prep.Q\_space method), 26  
GrayImgUnit (class in osrefl.model.sample\_prep), 15  
**H**  
height() (osrefl.model.sample\_prep.Cone method), 10  
height() (osrefl.model.sample\_prep.Ellipse method), 12  
height() (osrefl.model.sample\_prep.Ellipsoid method), 13  
height() (osrefl.model.sample\_prep.K3DUnit method), 19  
height() (osrefl.model.sample\_prep.Layer method), 22

49

height() (osrefl.model.sample\_prep.Parallelepiped method), 23  
height() (osrefl.model.sample\_prep.Pyramid method), 25  
height() (osrefl.model.sample\_prep.RoundedParPip method), 30  
height() (osrefl.model.sample\_prep.Sphere method), 33  
Hexagonal (class in osrefl.model.sample\_prep), 16  
**I**  
is\_core\_of() (osrefl.model.sample\_prep.Cone method), 10  
is\_core\_of() (osrefl.model.sample\_prep.Ellipse method), 12  
is\_core\_of() (osrefl.model.sample\_prep.Ellipsoid method), 13  
is\_core\_of() (osrefl.model.sample\_prep.Layer method), 22  
is\_core\_of() (osrefl.model.sample\_prep.Parallelepiped method), 23  
is\_core\_of() (osrefl.model.sample\_prep.Pyramid method), 25  
is\_core\_of() (osrefl.model.sample\_prep.RoundedParPip method), 30  
is\_core\_of() (osrefl.model.sample\_prep.Shape method), 32  
is\_core\_of() (osrefl.model.sample\_prep.Sphere method), 33  
**K**  
K3d\_Isiform() (osrefl.model.sample\_prep.K3DUnit method), 19  
K3DUnit (class in osrefl.model.sample\_prep), 18  
K3DUnit.K3D\_Shape (class in osrefl.model.sample\_prep), 18  
K3DUnit.K3D\_Shape\_Collection (class in osrefl.model.sample\_prep), 18  
**L**  
Lattice (class in osrefl.model.sample\_prep), 19  
Layer (class in osrefl.model.sample\_prep), 21  
length() (osrefl.model.sample\_prep.Cone method), 10  
length() (osrefl.model.sample\_prep.Ellipse method), 12  
length() (osrefl.model.sample\_prep.Parallelepiped method), 24  
length() (osrefl.model.sample\_prep.Pyramid method), 25  
length() (osrefl.model.sample\_prep.RoundedParPip method), 30  
length() (osrefl.model.sample\_prep.Sphere method), 33  
longBA() (osrefl.theory.scatter.Calculator method), 39  
**M**  
mag\_view() (osrefl.model.sample\_prep.Unit\_Cell method), 35  
**50**

magneticBA() (osrefl.theory.scatter.Calculator method), 39  
**N**  
normalized() (osrefl.model.sample\_prep.Q\_space method), 26  
**O**  
Onf (class in osrefl.model.onf\_loader), 42  
on\_top\_of() (osrefl.model.sample\_prep.Cone method), 11  
on\_top\_of() (osrefl.model.sample\_prep.Ellipse method), 12  
on\_top\_of() (osrefl.model.sample\_prep.Ellipsoid method), 13  
on\_top\_of() (osrefl.model.sample\_prep.Layer method), 22  
on\_top\_of() (osrefl.model.sample\_prep.Parallelepiped method), 24  
on\_top\_of() (osrefl.model.sample\_prep.Pyramid method), 25  
on\_top\_of() (osrefl.model.sample\_prep.RoundedParPip method), 30  
on\_top\_of() (osrefl.model.sample\_prep.Shape method), 32  
on\_top\_of() (osrefl.model.sample\_prep.Sphere method), 33  
OOMFFUnit (class in osrefl.model.sample\_prep), 22  
osrefl.loaders.andr\_load (module), 42  
osrefl.model.onf\_loader (module), 42  
osrefl.model.sample\_prep (module), 9  
osrefl.theory.scatter (module), 35  
**P**  
Parallelepiped (class in osrefl.model.sample\_prep), 23  
partial\_magnetic\_BA() (osrefl.theory.scatter.Calculator method), 40  
partial\_magnetic\_BA\_Jong() (osrefl.theory.scatter.Calculator method), 40  
phase\_shift() (osrefl.model.sample\_prep.Hexagonal method), 16  
phase\_shift() (osrefl.model.sample\_prep.Pyramid method), 20  
phase\_shift() (osrefl.model.sample\_prep.Rectilinear method), 27  
Pyramid (class in osrefl.model.sample\_prep), 24  
q\_calc() (osrefl.model.sample\_prep.Theta\_space method), 34  
Q\_space (class in osrefl.model.sample\_prep), 25  
query\_center() (osrefl.model.sample\_prep.Scene method), 31  
query\_height() (osrefl.model.sample\_prep.Scene method), 31

Index

- qz\_slice() (osrefl.theory.scatter.Calculator method), 40
- R**
  - rec\_LfH() (osrefl.model.sample\_prep.Hexagonal method), 17
  - rec\_LfH() (osrefl.model.sample\_prep.Lattice method), 20
  - rec\_LfH() (osrefl.model.sample\_prep.Rectilinear method), 27
  - Rectilinear (class in osrefl.model.sample\_prep), 27
  - render() (osrefl.model.sample\_prep.GeomUnit method), 15
  - repeat() (osrefl.model.sample\_prep.Unit\_Cell method), 35
  - resolution\_correction() (osrefl.theory.scatter.Calculator method), 40
  - RoundedParPip (class in osrefl.model.sample\_prep), 29
- S**
  - Scene (class in osrefl.model.sample\_prep), 30
  - Shape (class in osrefl.model.sample\_prep), 31
  - SMBAB() (osrefl.theory.scatter.Calculator method), 36
  - SMBAlfH() (osrefl.theory.scatter.Calculator method), 36
  - Space (class in osrefl.model.sample\_prep), 32
  - Sphere (class in osrefl.model.sample\_prep), 32
  - struc\_calc() (osrefl.model.sample\_prep.Hexagonal method), 17
  - struc\_calc() (osrefl.model.sample\_prep.Rectilinear method), 28
- T**
  - Theta\_space (class in osrefl.model.sample\_prep), 33
  - theta\_struc\_calc() (osrefl.model.sample\_prep.Rectilinear method), 28
  - thickness() (osrefl.model.sample\_prep.Cone method), 11
  - thickness() (osrefl.model.sample\_prep.Ellipse method), 12
  - thickness() (osrefl.model.sample\_prep.Ellipsoid method), 14
  - thickness() (osrefl.model.sample\_prep.Lattice method), 22
  - thickness() (osrefl.model.sample\_prep.Parallelapiped method), 24
  - thickness() (osrefl.model.sample\_prep.Pyramid method), 25
  - thickness() (osrefl.model.sample\_prep.RoundedParPip method), 30
  - thickness() (osrefl.model.sample\_prep.Sphere method), 33
- U**
  - Unit\_Cell (class in osrefl.model.sample\_prep), 34
  - unitBuild() (osrefl.model.sample\_prep.GrayImageUnit method), 16
  - unitBuild() (osrefl.model.sample\_prep.OOMMFUnit method), 22
- V**
  - value\_extend() (osrefl.model.sample\_prep.GeomUnit method), 15
  - vectorize() (osrefl.model.sample\_prep.Q\_space method), 26
  - vectorize() (osrefl.model.sample\_prep.Theia\_space method), 34
  - view() (osrefl.loaders.andr\_load.Data method), 42
  - view() (osrefl.model.omf\_loader.Omf method), 43
  - view() (osrefl.model.sample\_prep.Unit\_Cell method), 35
  - view\_corrected() (osrefl.theory.scatter.Calculator method), 41
  - view\_linear() (osrefl.theory.scatter.Calculator method), 41
  - view\_uncorrected() (osrefl.theory.scatter.Calculator method), 41
  - viewCor() (osrefl.theory.scatter.Calculator method), 40
  - viewCorUncor() (osrefl.theory.scatter.Calculator method), 41
  - viewFixedZ() (osrefl.model.omf\_loader.Omf method), 43
  - viewSlice() (osrefl.model.sample\_prep.Unit\_Cell method), 35
  - viewUncor() (osrefl.theory.scatter.Calculator method), 41
- W**
  - width() (osrefl.model.sample\_prep.Cone method), 11
  - width() (osrefl.model.sample\_prep.Ellipse method), 12
  - width() (osrefl.model.sample\_prep.Ellipsoid method), 14
  - width() (osrefl.model.sample\_prep.Parallelapiped method), 24
  - width() (osrefl.model.sample\_prep.Pyramid method), 25
  - width() (osrefl.model.sample\_prep.RoundedParPip method), 30
  - width() (osrefl.model.sample\_prep.Sphere method), 33
- X**
  - x\_calc\_sfx() (osrefl.model.sample\_prep.Hexagonal method), 17
  - x\_calc\_sfx() (osrefl.model.sample\_prep.Lattice method), 20
  - x\_calc\_sfx() (osrefl.model.sample\_prep.Rectilinear method), 28
  - x\_calc\_sfx() (osrefl.model.sample\_prep.Hexagonal method), 17
  - x\_calc\_sfx\_shift() (osrefl.model.sample\_prep.Lattice method), 20
  - x\_calc\_sfx\_shift() (osrefl.model.sample\_prep.Rectilinear method), 28
  - x\_gauss\_sfx() (osrefl.model.sample\_prep.Hexagonal method), 18
  - x\_gauss\_sfx() (osrefl.model.sample\_prep.Lattice method), 20
  - x\_gauss\_sfx() (osrefl.model.sample\_prep.Rectilinear method), 28
- Y**
  - y\_calc\_sfx() (osrefl.model.sample\_prep.Hexagonal method), 18
  - y\_calc\_sfx() (osrefl.model.sample\_prep.Lattice method), 20
  - y\_calc\_sfx() (osrefl.model.sample\_prep.Rectilinear method), 28
  - y\_calc\_sfx\_shift() (osrefl.model.sample\_prep.Hexagonal method), 18
  - y\_calc\_sfx\_shift() (osrefl.model.sample\_prep.Lattice method), 21
  - y\_calc\_sfx\_shift() (osrefl.model.sample\_prep.Rectilinear method), 29

## Appendix E

### Software Diagram

This appendix is a diagram of the software layout. It is Unified Modeling Language(UML) diagram of the main class structure involved in this software. It can be used as a reference for the software flow. For a more detailed explanation of what each class does, refer to the software instruction manual in appendix D. The diagram is split into 3 different section because of space limitations but they are all related by the red relation arrows. Figure E.1 illustrates a majority of the classes used to calculate the the scattered data. Figure E.2 illustrates the possible unit building utilities and how they related to the calculations. The most complicated of these unit building options is the GeomUnit class which is comprised of a Scene of Shape objects. The possible Shape objects are illustrated in figure E.3.

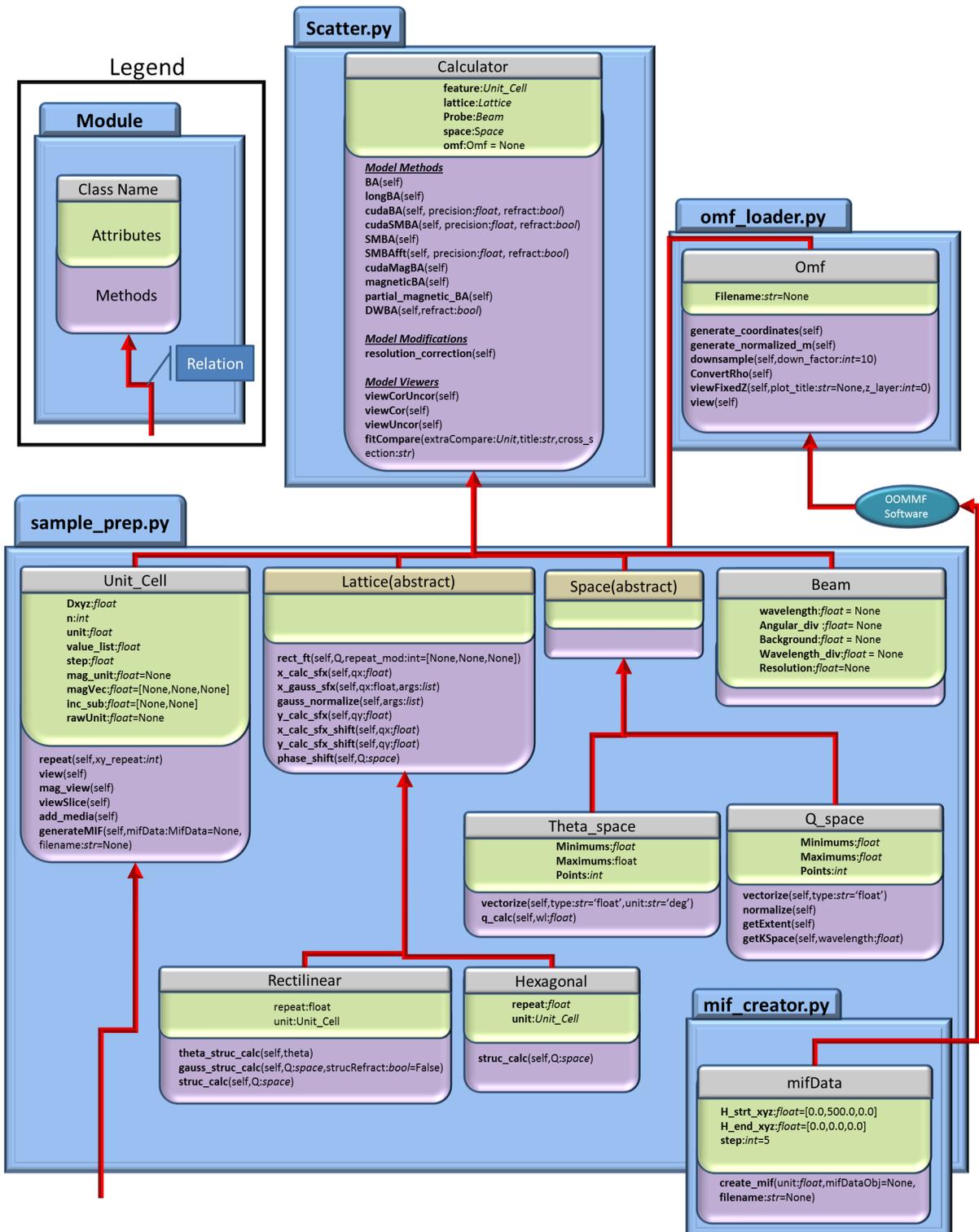


Figure E.1: UML diagram of the main calculation components involved in the theory function calculation.

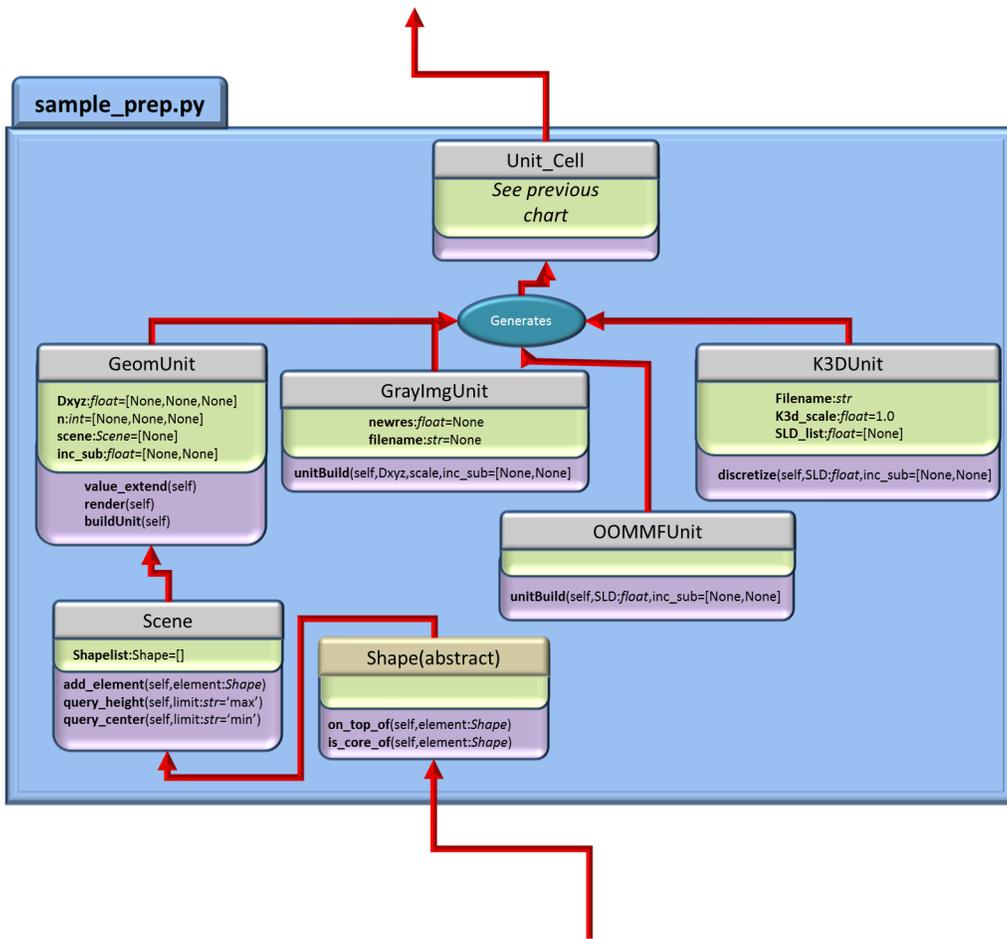


Figure E.2: UML diagram of the unit building classes. These classes can all be used to build the finite element models from which scattering may be calculated.

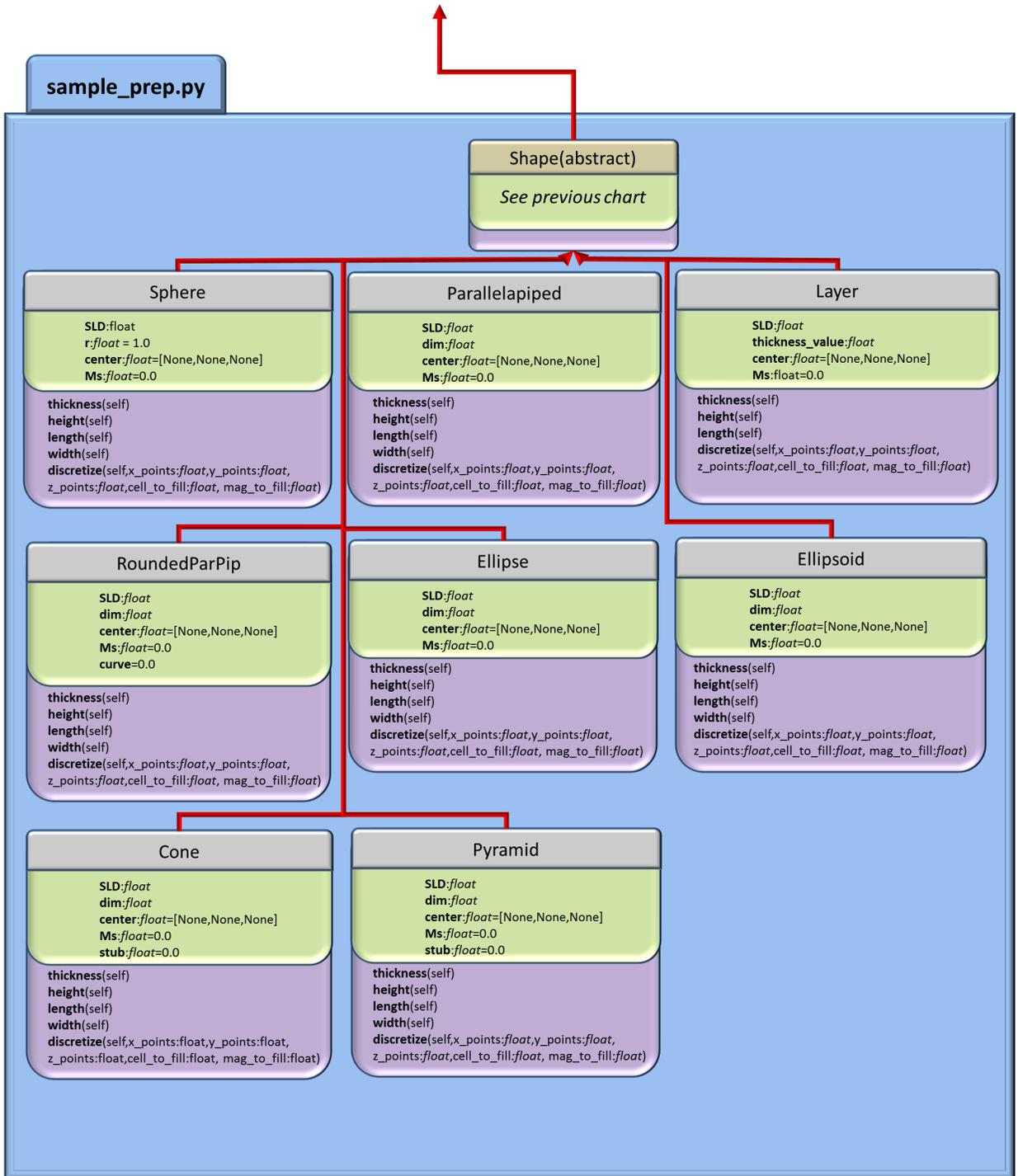


Figure E.3: The Shape classes available to build a Scene for modeling.

## Bibliography

- [1] Ng1 reflectometer website. <http://ncnr.nist.gov/instruments/ng1refl/design.html>. Accessed August 15, 2011.
- [2] McGreevy R. L. Bilheux H. Z. Anderson, I. S. *Neutron Imaging and Applications*. Springer Science, 2009.
- [3] Dupre N. Lee P. L. Proffen T. Parise J. B. Grey C. P. Breger, J. Short- and long-range ordering in the positive electrode material,  $\text{Li}(\text{NiMn})_{0.5}\text{O}_2$ : A joint x-ray and neutron diffraction pair distribution function analysis and nmr study. *Journal of the American Chemical Society*, 127:18096–18104, 2005.
- [4] T. Chatterji. *Neutron scattering from magnetic materials*. 2006.
- [5] A. Daillant, J.; Gibaud. *X-Ray and Neutron Reflectivity: Principles and Applications*. Springer, 1999.
- [6] Porter D. Donahue, M. Object oriented micromagnetic framework. <http://math.nist.gov/oommf/>. Accessed May 7, 2011.
- [7] J. A. Dura, D. J. Pierce, C. F. Majkrzak, N. C. Maliszewskyj, D. J. McGillivray, M. Losche, K. V. O'Donovan, M. Mihailescu, U. Perez-Salas, D. L. Worcester, and S. H. White. And/r: Advanced neutron diffractometer/reflectometer for investigation of thin films and multilayers for the life sciences. *Review of Scientific Instruments*, 77(7), 2006.
- [8] F. Filges, D.; Goldenbaum. *Handbook of Spallation Research*. Wiley-VCH, 2009.
- [9] Gutberlet T. Katsaras J. Fitter, J. *Neutron scattering in biology*. 2006.
- [10] Majkrzak C. F. Fitzsimmons, M. R. *Modern Techniques for Characterizing Magnetic Materials*. Kluwer, 2005.
- [11] Association for Computing Machinery. Special Interest Group on Automata, Computability, Society for Industrial, and Applied Mathematics. Activity Group on Discrete Mathematics. *Discrete Algorithms*. ACM, 1991.
- [12] John D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science and Engineering*, 9(3):90–95, 2007.
- [13] J. B. Jomass, H. W. Schlenoff. Salt-induced polyelectrolyte interdiffusion in multilayered films: a neutron reflectivity study. *Macromolecules*, 38:8473–8480, 2005.

- [14] E. Kentzinger, U. Ruecker, B. Toperverg, F. Ott, and T. Bruckel. Depth-resolved investigation of the lateral magnetic correlations in a gradient nanocrystalline multilayer. *Physical Review B*, 77(10), 2008.
- [15] J. W. Kiel, M. E. Mackay, B. J. Kirby, B. B. Maranville, and C. F. Majkrzak. Phase-sensitive neutron reflectometry measurements applied in the study of photovoltaic films. *Journal of Chemical Physics*, 133(7), 2010.
- [16] Watson S. M. Davies J. E. Zimanyi G. T. Liu K. Shull R. D. Kirby, B. J. Direct observation of magnetic gradient in co/pd pressure-graded media. *Journal of Applied Physics*, 105:07C929, 2009.
- [17] Leroy F. Renaud G. Lazzari, R. Grazing-incidence small angle x-ray scattering from dense packing of islands on surfaces: Development of distorted wave born approximation and correlation between particle size and spacing. *Physical Review B*, 76:125411, 2007.
- [18] G. Ljungdahl and S. W. Lovesey. Surface scattering near grazing angles: The distorted wave born approximation for rough surfaces. *Physica Scripta*, 53(6):734–748, 1996.
- [19] Travis E. Oliphant. Python for scientific computing. *Computing in Science and Engineering*, 9(3):10–20, 2007.
- [20] open source community. 3d modelling software. <http://www.k-3d.org/>. Accessed April 24, 2011.
- [21] open source community. Python gui toolbox. <http://www.wxpython.org/>. Accessed April 24, 2011.
- [22] open source community. Python programming language. <http://www.python.org/>. Accessed April 24, 2011.
- [23] Edward Prince. Neutron scattering instrumentation: A tutorial review. *Applied Spectroscopy Review*, 2004.
- [24] LR RABINER, RW SCHAFER, and CM RADER. Chirp z-transform algorithm. *IEEE TRANSACTIONS ON AUDIO AND ELECTROACOUSTICS*, AU17(2):86, 1969.
- [25] M. Rauscher, H. Reichert, S. Engemann, and H. Dosch. Local density profiles in thin films and multilayers from diffuse x-ray and neutron scattering. *Physical Review B*, 72(20), 2005.
- [26] S. K. Sinha, E. B. Sirota, S. Garoff, and H. B. Stanley. X-ray and neutron-scattering from rough surfaces. *Physical Review B*, 38(4):2297–2311, 1988.

- [27] Brown C. M. Herm Z. R. Chavan S. Bordiga S. Long J. R. Sumida, K. Hydrogen storage properties and neutron scattering studies of mg-2(dobdc)-a metal-organic framework with open mg<sup>2+</sup> adsorption sites. *Chemical Communications*, 47:1157–1159, 2011.
- [28] B. Toperverg, V. Lauter-Pasyuk, H. Lauter, O. Nikonov, D. Ausserre, and Y. Gallot. Morphology of off-specular neutron scattering pattern from islands on a lamellar film. *Physica B*, 283(1-3):60–64, 2000.
- [29] B. Toperverg, V. Lauter-Pasyuk, H. Lauter, O. Nikonov, D. Ausserre, and Y. Gallot. Off-specular neutron scattering from islands on a lamellar film. *Physica B*, 276:355–356, 2000.
- [30] B. P. Toperverg. Specular reflection and off-specular scattering of polarized neutrons. *Physica B-Condensed Matter*, 297(1-4):160–168, 2001.
- [31] G.H. Vineyard. Grazing-incidence diffraction and the distorted-wave approximation for the study of surfaces. *Physical Review B*, 26(8):4146, 1982.
- [32] Langridge S. Webster, J. Applications of index matching in reflectometry, sans and brewster angle microscopy. *Current opinion in colloid and interface science*, 4:186–189, 1999.
- [33] Chlistunoff J. Majewski J. Borup R. L. Wood, D. L. Nafion structural phenomena at platinum carbon interfaces. *Journal of the American Chemical Society*, 131:18096–18104, 2009.