OsRefl Documentation

Release 1.1.1

Christopher Metting

June 15, 2011

CONTENTS

Contents:

CHAPTER

ONE

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 simultaniously by going to Link pythonxy and installing their product. In addition, if a Cuda compatable Nvidia GPU device is available, pyCuda must also be installed which may be downloaded at Link pycuda.

Once the package is obtained from the repository and the previous prerequisites are installed, there is a small set of C code that must be compiled. 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 (first) 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 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.

1.1.3 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 GrayImgUnit. 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(SLD = 4.506842e-6,dim=[3.75e4,3.75e4,700.0])
Cr = Layer(SLD = 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 discritization 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 = GeoUnit.buildUnit()
```

GrayImagUnit

A unit cell can also be created using the GrayImagUnit. 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, discretizeation 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, ie. 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 pares 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 modelling 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 dicritized system with an simple way to produce magnetic structures.

1.1.4 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([-.0001,-0.001,0.00002],[.0001,0.001,0.04],[200,50,200])
```

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

```
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)
```

1.1.5 Theory Function

Now that the software has everything it needs to calculate off-specular scattering, a modelling formalism must be chosen. The option here can be found elsewhere in the documentation but the modelling 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()
```

1.1.6 Viewing

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

```
sample.view_uncorrected()
```

to view the uncorrected scattering or:

```
sample.view_corrected()
```

for the corrected data. Because there is no set convention for what the user will want to view, the script must have:

```
show()
```

to view the output plots.

CHAPTER

TWO

REFERENCE

2.1 Model Creation

2.1.1 osrefl.model.sample_prep

class osrefl.model.sample_prep.**Beam**(*wavelength=None*, *angular_div=None*, *background=None*, *wavelength_div=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.

angluar_div: **(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^2)** 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.

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|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: **(float|angstroms)** 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: **(float|angstroms)** The total thickness of the cone object (absolute thickness).

width()

Overview:

Returns the maximum width of the cone (y direction).

Returns:

width: **(float|angstroms)** 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: $(x^2/a^2) + (y^2/b^2) = 1$. The 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(__init__):

SLD: **(float|angstroms^2)** The scattering length density of the Ellipse.

dim: **(float,[3]|angstroms)** 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]|angstroms)** 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: **(float|angstroms)** 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.

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|angstroms)** an array of x points to be determined if they fall within the Ellipse.

y_points: **(float|angstroms)** an array of y points to be determined if they fall within the Ellipse.

z_points: **(float|angstroms)** an array of z points to be determined if they fall within the Ellipse.

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.

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|angstroms^2)** The scattering length density of the Ellipsoid.

dim: **(float,[3]|angstroms)** 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]|angstroms)** 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).

Ms: **(float|angstroms)** The magnetic SLD of the material for this shape.

Note:

•This is a lenticular shaped object.

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|angstroms)** an array of x points to be determined if they fall within the Ellipsoid.

y_points: **(float|angstroms)** an array of y points to be determined if they fall within the Ellipsoid.

z_points: **(float|angstroms)** an array of z points to be determined if they fall within the Ellipsoid.

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.

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.

width()

Overview: Returns the maximum width of the Pyramid (y direction)

class osrefl.model.sample_prep.**GeomUnit**(*Dxyz=[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:

Dxyz: **(float,[3])|(angstroms)** 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 course/fine 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 renderer 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)|angstroms)

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|angstroms^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]|angstroms)

This is the reals 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.

buildUnit()

Overview:

Producer method: This method produces a Unit_Cell object from the rendered geomUnit object. Because this is 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.**GrayImgUnit**(*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.

- 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 OOMMFUnit 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 rectilinear structure factor for two phases of repeating units and adding these phases together.

0 = feature phase 1 O = feature phase 2 ~ = spacing

Hexagonal:

.0~0~0~0~0

O~O~O~O~O~O

.0~0~0~0~0

O~O~O~O~O~O

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 uses methods from Lattice.

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 course 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]*)

Bases: 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([]) 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[x] is an object of type Shape

description_list a list of objects of type shape that holds the edges, vertices and Sof all of the features

K3DUnit.**discritize**(*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.sample_prep.**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.

0|0|0|0|0

0|0|0|0|0

0|0|0|0|0

Hexagonal: A lattice structure that is packed in a hexagonal ordering.

_0|0|0|0|0|0

0|0|0|0|0|0|0

_0|0|0|0|0|0

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 course 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: **(float|angstroms^2)** The scattering length density of the Pyramid.

thickness_value: **(float|angstroms)** The thickness of the layer.

center: **(float,[3]|angstroms)** 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: **(float|angstroms)** 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|angstroms)** an array of x points to be determined if they fall within the layer.

y_points: **(float|angstroms)** an array of y points to be determined if they fall within the layer.

z_points: **(float|angstroms)** an array of z points to be determined if they fall within the layer.

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.

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.**OOMMFUnit**

Bases: object

Overview:

This class is used to create a Unit_Cell object from a .omf 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: **(float|angstroms^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.**Parallelapiped**(*SLD, dim, center=[None, None, None], Ms=0.0*)

Bases: osrefl.model.sample_prep.Shape

Overview: Uses the generic formula for a parallelapiped feature to create a parallelapiped object

Parameters(__init__):

SLD: **(float|angstroms^2)** The scattering length density of the sphere.

dim: **(float,[3]|angstroms)** x, y and z dimensions of the feature.

center: **(float,[3]|angstroms)** 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|angstroms)** 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|angstroms)** an array of x points to be determined if they fall within the parallelapiped.

y_points **(float|angstroms)** an array of y points to be determined if they fall within the parallelapiped.

z_points(float|angstroms) an array of z points to be determined if they fall within the parallelapiped.

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.

height()

Overview:

Returns the total height of the parallelapiped. 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 parallelapiped (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 parallelapiped.

width()

Overview:

Returns the maximum width of the parallelapiped (y direction)

class osrefl.model.sample_prep.**Pyrimid**(*SLD, dim, stub=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: **(float|angstroms^2)** The scattering length density of the Pyramid.

dim: **(float,[3]|angstroms)** 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.

stub: **(float|angstroms)** 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 stub parameter fixed.

center: **(float,[3]|angstroms)** 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: **(float|angstroms)** 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|angstroms)** an array of x points to be determined if they fall within the Pyramid.

y_points: **(float|angstroms)** an array of y points to be determined if they fall within the Pyramid.

z_points: **(float|angstroms)** an array of z points to be determined if they fall within the Pyramid.

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.

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(__init__):

minimums: **(float,[3]|angstroms)** 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]|angstroms)** 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]|angstroms)** 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]|angstroms^-1)** The reciprocal space step size for the x,y and z dimensions.

q_list: **(float,(3)[array]|angstroms^-1)** The total list of values being solved for in the x, y and z directions.

q_refract: **(float,[array]|angstroms^-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]|angstroms)** This is the equivelent 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|angstroms^-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|angstroms)

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|angstroms^-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:

type(str): 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 girded 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 uses methods from Lattice.

gauss_normalize(*args*)

gauss_struc_calc(*Q*, *strucRefract=False*)

Overview:

This structure calculation applies a gaussian convelution to the delta function diffraction peaks produced by the structure factor to produce a more accurate theory function. The convelution 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(*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 course 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:

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_struc_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.**RoundedParPip**(*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|angstroms^2)** The scattering length density of the sphere.

dim: **(float,[3]|angstroms)** x, y and z dimensions of the feature.

center: **(float,[3]|angstroms)** 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|angstroms)** 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|angstroms)** an array of x points to be determined if they fall within the parallelapiped.

y_points **(float|angstroms)** an array of y points to be determined if they fall within the parallelapiped.

z_points(float|angstroms) an array of z points to be determined if they fall within the parallelapiped.

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.

height()

Overview:

Returns the total height of the parallelapiped. 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 parallelapiped (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 parallelapiped.

width()

Overview:

Returns the maximum width of the parallelapiped (y direction)

class osrefl.model.sample_prep.**Scene**(*shapelist*=[ ])

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:

shapelist:(Shape|[]) = a list of Shape objects to be put into the scene.

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:

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 Parallelapiped 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 a 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: **(float|angstroms^2)** The scattering length density of the sphere.

r: **(float|angstroms)** The radius of the sphere.

center: **(float,[3]|angstroms)** 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|angstroms)** 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|angstroms)** an array of x points to be determined if they fall within the sphere.

y_points: **(float|angstroms)** an array of y points to be determined if they fall within the sphere.

z_points: **(float|angstroms)** an array of z points to be determined if they fall within the sphere.

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.

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 reciprical space.

Parameters(__init__):

minimums: **(float,[2]|angstroms)** The minimum theta values that the user would like solved. The data is in the form: [θ_{in}^{min}, θ_{out}^{min}]

maximums: **(float,[2]|angstroms)** The maximum theta values that the user would like solved. The data is in the form: [θ_{in}^{max}, θ_{out}^{max}]

points: **(float,[2]|angstroms)** 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: [θ_{in}, θ_{out}]

Parameters(Class):

theta_step: **(float,[2]|degrees)** Step size in θ_{in} and θ_{out}

theta_list: **(float,(2)[array]|degrees)** The total list of values being solved for in θ_{in} and θ_{out}.

q_calc(*wl*)

Overview:

This calculates the total Q vector based on the given theta values

Parameters

wl: **(float|angstroms)** The wavelength of the probing beam.

Return:

q_vector: **(array|angstroms^-1)** An array of Q vectors calculated for the combination of θ_{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**(*Dxyz*, *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 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.

OOMMFUnit

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 calculated 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.

GrayImgUnit

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.

generateMIF(*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.

Parameters:

lattice: **(Lattice)** see `Lattice` for more information.

probe: **(Beam)** see `Beam` for more information.

space **(space)** see `Q_space` or *~sample_prep.theta_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 t=1. This makes for a simple calculation, however, it does not allows 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:

$$FF = abs\left(\frac{-i}{q_{x,y,z}} * (1.0 - exp^{i*q_{x,y,z}*\Delta d_{x,y,z}}) * CZT\{\rho_{unit}\}\right)^2$$

It is also normalized by the surface area:

$$Normfactor = \left(\frac{4*\pi}{q_z * M_{x,y} * D_{x,y}}\right)^2$$

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:* (str|precision)

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:

$$q_x + \lambda * \rho_{substrate}$$

at $-q_x$ values and:

$$q_x - \lambda * \rho_{substrate}$$

cudaMagBA (*precision='float32'*, *refract=True*)

cudaSMBA (*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:

$$\overline{M_l(\Delta z)} = \begin{pmatrix} cos(k_l \Delta z) & \frac{1}{k_l} sin(k_l \Delta z) \\ -k_l sin(k_l \Delta z) & cos(k_l \Delta z) \end{pmatrix}$$

and the reflection is:

$$r = \frac{M_{1,1} + (i * n_0 * M_{0,1}) + \frac{-i}{n_f} * (-M_{1,0} - i * n_0 * M_{0,0})}{-M_{1,1} + i * n_0 * M_{0,1} \frac{-i}{n_f} (M_{1,0} - i * n_0 * M_{0,0})}$$

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 $k_{in} < 0.0$:

$$\Psi_{in_1} = t * \exp^{-ik_{\parallel}\Delta q_z}$$

$$\Psi_{in_2} = 0.0$$

For $k_{in} > 0.0$:

$$\Psi_{in_1} = 1 * \exp^{-ik_{\parallel}\Delta q_z}$$

$$\Psi_{in_2} = r * \exp^{-ik_{\parallel}\Delta q_z}$$

For $k_{out} < 0.0$:

$$\Psi_{out_1} = 1 * \exp^{-ik_{\parallel}\Delta q_z}$$

$$\Psi_{out_2} = r * \exp^{-ik_{\parallel}\Delta q_z}$$

For $k_{out} > 0.0$:

$$\Psi_{out_1} = t * \exp^{-ik_{\parallel}\Delta q_z}$$

$$\Psi_{out_2} = 0.0$$

With these pieces of information, the SMBA can be solved for. The final form factor is:

$$FF = \left(\frac{-i}{q_{x,y,z}} * (1.0 - exp^{i*q_{x,y,z}*\Delta d_{x,y,z}}) * \left[\Psi_{in} * \sum_{n=0}^{D_{x,y,z}} \{ rho_{unit} * exp^{i*Q_{x,y,z}*D_{x,y,z}} \} * \Psi_{out} \right] \right)^2$$

This form factor is also *normalized*

For the formalism to the structure factor see `Rectilinear()` or `Hexagonal()`

Parameters *precision:* (str|precision)

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:

$$q_x + \lambda * \rho_{substrate}$$

at $-q_x$ values and:

$$q_x - \lambda * \rho_{substrate}$$

fitCompare(*other, extraCompare=None, titles=['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*, *titles*)

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:

$$FF = abs\left(\frac{-i}{q_{x,y,z}} * (1.0 - exp^{i*q_{x,y,z}*\Delta d_{x,y,z}}) * \sum_{n=0}^{D_{x,y,z}} \{rho_{unit} * exp^{i*Q_{x,y,z}*D_{x,y,z}}\}\right)^2$$

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|angstroms^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|angstroms^2)** The structural scattering potential of the feature being scattered off of.

mag_cell: **(float:3D array|angstroms^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,angstroms^-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|angstroms^-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.py 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

viewCorUncor()

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

viewUncor()

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(*lbl=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:

lbl: **(str)** This parameter can be used to change the title of the plot.

vmin: **(float|Angstroms^-2)** This is the minimum intensity value plotted on the 2D plot. Any intensity value below this value is plotted as the minimum.

vmin: **(float|Angstroms^-2)** 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 then a log scale. This can be useful when troubleshooting a calculation.

view_uncorrected(*lbl=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:

lbl: **(str)** This parameter can be used to change the title of the plot.

vmin: **(float|Angstroms^-2)** This is the minimum intensity value plotted on the 2D plot. Any intensity value below this value is plotted as the minimum.

vmin: **(float|Angstroms^-2)** 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**
Bases: object

Overview:

This is a data loader for .cg1 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*)
Bases: 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 oommf12a4pre-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.tcl

•Select the appropriate server for processing the calculations(this is the local machine for non-distributed calculations.)

•Select oxsii from the mmLaunch box.

•In the Oxsii window select File>>load and select the .mif file created by the OsRefl software. (see `sample_prep.Unit_Cell.generateMIF()`)

•Run the magnetic minimization by pressing the "Run" button

•Add a mmDisp from the mmLaunch menu

Selection Input:

Output	Destination	Schedule
Magnetization Output	mmDisp<*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:

•Qx, Qy resolution is typically much worse than exchange length Qz 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()

viewFixedZ(*plot_title=None*, *z_layer=0*)

Overview:

This method shows a color plot of the angle between mx, my.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

o

INDEX

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y