The Journal of Open Source Software

GixPy: A Python package for transforming grazing incidence X-ray scattering images

Edward Tortorici[1] **and Charles T. Rogers**[1]

1 Department of Physics, University of Colorado Boulder

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Grazing incidence X-ray scattering techniques are used to investigate the atomic structure of materials localized on a flat surface. Such materials often grow with a preferred crystal orientation, such as uniaxially aligned thin films (Steele et al., 2023). Often area detectors are used to image angular scattering. These images are then transformed such that the axes represent Fourier components of the spatial scattering, i.e. "reciprocal space". X-ray image analysis software often assumes that the sample is a randomaly oriented powder. However, for grazing incidence X-ray experiments, crystallites in the film often have a preferred orientation, and the image manipulation requires additional considerations.

X-ray scattering experiments frequently use a parallel beam of X-rays with incident wavevector $\mathbf{k}_i$, with magnitude $2\pi/\lambda$, where λ is the X-ray wavelength. The beam is directed at a sample (an example is shown in Figure 1). X-rays scattered from electrons in the sample have a new wavevector $\mathbf{k}_s$. Here, we consider the case of elastic scattering where the magnitude of $\mathbf{k}_s$ is $2\pi/\lambda$. Scattered X-rays are collected on a planar imaging array of X-ray detecting pixels (Rowlands & Yorkston, 2000). The measured values at a pixel location determine $\mathbf{k}_s$, and along with the known $\mathbf{k}_i$, the so-called reciprocal space can be built from the scattering vector $\mathbf{q}$:

$$\mathbf{q} \equiv \mathbf{k}_s - \mathbf{k}_i. \tag{1}$$

X-ray scattering theory shows that the scattering vectors are the Fourier components of the spatial electron density in the sample and are the principal information yielded by X-ray scattering experiments.

Statement of need

There currently exist many tools for transforming wide-angle X-ray scattering (WAXS) and small-angle X-ray scattering (SAXS) images into reciprocal space, including pyFAI (Kieffer & Ashiotis, 2013) and Nika (Ilavsky, 2012). However, these tools lack the capability of processing raw images from grazing incidence wide/small-angle X-ray scattering (GIWAXS/GISAXS) experiments. Here we refer to both GIWAXS and GISAXS as grazing incidence X-ray scattering (GIXS). An existing Python package, pygix, is capable of processing GIWAXS and GISAXS images into reciprocal space. However, it lacks transparency, in that, the documentation is sparse, and it utilizes look-up tables to perform the transformation, making the source code difficult to parse. Furthermore, researchers interested in utilizing GIXS experiments likely already do powder X-ray experiments, and have a preferred suite of analysis tools, and pygix lacks the ability to be an intermediary step for non-Python tools.

GixPy seeks transparency in order to serve not only as a useful tool, but also an educational tool for those who are less experienced with grazing incidence experiments. This goal is achieved by maintaining well documented and commented code that utilizes direct computation, and is written with source-code readability in mind. This is intended to allow students and researchers to have an accessible resource, with examples, that helps them learn how to process GIXS images and understand the necessity of this procedure.

Furthermore, GixPy is workflow agnostic, allowing it to be utilized as an intermediary step for anyone who already has a preferred WAXS/SAXS image processing software. This allows users to not need to learn an entirely new system to do their analysis in, and can simply use GixPy to pre-process an image before giving it to their preferred environment for analysis. However, since GixPy is built as a Python tool, it has been built to seamlessly integrate with pyFAI to serve as a complete processing tool.

Powder transformation

Existing tools, such as Nika and pyFAI transform images with the assumption that samples are a powder, such that the scattering results in Debye-Scherrer cones (Cullity & Stock, 2014). A typical experimental setup is exemplified in Figure 1. An area detector is used to intersect the Debye-Scherrer cones to detect rings of constructive interference.

The scattering angle 2θ can be related to reciprocal space through Bragg's law:

$$q = \frac{4\pi}{\lambda}\sin\theta, \tag{2}$$

where λ is the wavelength of the scattered X-rays. The scattering angle can be determined from the radius of the ring on the detector r and the sample-detector distance d_{sd}:

$$\tan 2\theta = \frac{r}{d_{sd}}, \tag{3}$$

so a powder image transformation calculates q from the ring radii using

$$q = \frac{4\pi}{\lambda}\sin\left[\frac{1}{2}\tan^{-1}\left(\frac{r}{d_{sd}}\right)\right]. \tag{4}$$

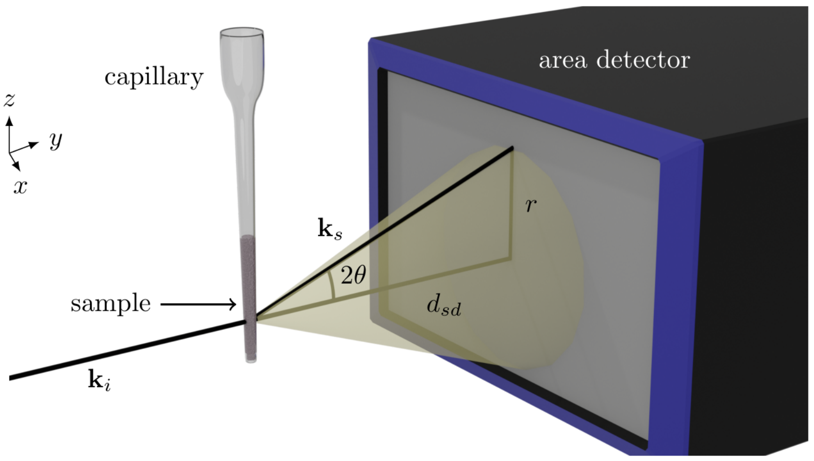

Figure 1: A typical WAXS/SAXS experiment. A powder sample is exposed to an incident beam, resulting in Debye-Scherrer cones of constructive interference. An area detector is used to intersect the cones to detect rings.

A GixPy transformation processes an image, such that a processed image can be transformed assuming powder symmetry will produce correct results.

Geometric assumptions

GixPy supports geometries where the incident beam is perpendicular to the detector and the sample is brought into the beam path (see Figure 2). Not only does this geometry lead to simplifications that allow for more transparent calculations, but it is also consistent with many laboratory-scale GIXS systems, such as the Xenox Xuess. In this geometry, the point of normal incidence (PONI) on the detector and where the incident beam hits the detector (the beam center) are the same locations on the detector.

The top-left pixel of the detector is the origin of the data array and the PONI is defined as the distance from the bottom-left corner of the detector (consistent with pyFAI), as seen in Figure 3. Other software, including Nika, define the PONI by the row-column index $(i_{\text{poni}}, j_{\text{poni}})$. Transforming between these two conventions can be done with the following relation:

$$\text{poni}_1 = \left(R - i_{\text{poni}} - \frac{1}{2}\right) p_z \tag{5}$$

$$\text{poni}_2 = \left(j_{\text{poni}} + \frac{1}{2}\right) p_x, \tag{6}$$

where R is the number of rows in the image and p_x and p_z are the horizontal and vertical widths of a pixel respectively. This transformation can be done with

```
gixpy.poni.convert_to(poni_ij, pixel_widths, image_shape)
```

and reversed with

```
gixpy.poni.convert_from(poni, pixel_widths, image_shape)
```

Where each input can be a tuple, list, or NumPy array, with the first element being the vertical value and the second element being the horizontal value.

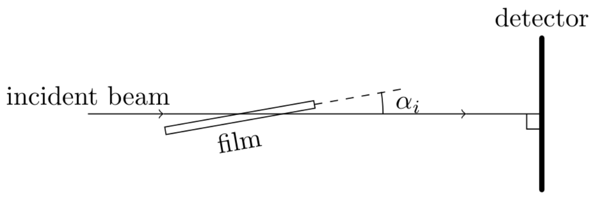

Figure 2: The supported experimental geometry has the detector positioned surface normal to the incident beam, and the grazing angle is set by rotating the sample by α_i relative to the beam.

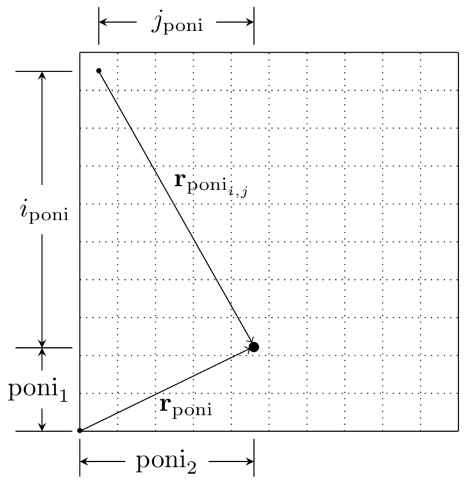

Figure 3: An example detector with 10×10 pixels. The PONI is described by the distance (in meters) from the bottom left corner. A user can convert a PONI in the $(i_{\text{poni}}, j_{poni})$ format using the `gixpy.poni.convert_to()` function.

Scattering geometry

In grazing incidence X-ray scattering, there is a very small angle, called the grazing angle or incident angle, between the incident beam and the plane of the substrate the sample lies on. The incident beam, with wavelength λ, has a wavevector $\mathbf{k}_i$ with magnitude $2\pi/\lambda$. Elastic scattering, due to bound electrons in the film, will result in a scattered ray with wavevector $\mathbf{k}_s$ with the same magnitude. In the sample frame (Figure 4a), the axes are oriented along orthogonal basis vectors $\hat{x}$, $\hat{y}$, and $\hat{z}$. $\hat{z}$ is surface-normal to the substrate, $\hat{y}$ is orthogonal to $\hat{z}$ and in a direction such that $\mathbf{k}_i$ lies in the $\hat{y}$-$\hat{z}$ plane, and $\hat{x}$ is orthogonal to $\hat{y}$ and $\hat{z}$. In the sample frame, the direction of the scattered ray can be described by rotations from the $\hat{y}$-direction:

$$\mathbf{k}_s = \frac{2\pi}{\lambda}\overleftrightarrow{R}_{\hat{x}}(\alpha_s)\overleftrightarrow{R}_{\hat{z}}(\phi_s)\ \hat{y} = \frac{2\pi}{\lambda}\overleftrightarrow{R}_{\hat{x}}(\alpha_s)\overleftrightarrow{R}_{\hat{z}}(\phi_s)\begin{bmatrix}0\\1\\0\end{bmatrix} = \frac{2\pi}{\lambda}\begin{bmatrix}-\sin\phi_s\\ \cos\alpha_s\cos\phi_s\\ \sin\alpha_s\cos\phi_s\end{bmatrix}, \tag{7}$$

where $\overleftrightarrow{R}_{\hat{x}}(\theta)$ and $\overleftrightarrow{R}_{\hat{z}}(\theta)$ are rotation matrix operators:

$$\overleftrightarrow{R}_{\hat{x}}(\theta) = \begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta & -\sin\theta\\ 0 & \sin\theta & \cos\theta\end{bmatrix} \tag{8}$$

$$\overleftrightarrow{R}_{\hat{y}}(\theta) = \begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix} \tag{9}$$

$$\overleftrightarrow{R}_{\hat{z}}(\theta) = \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}, \tag{10}$$

and the column vectors are in the $(\hat{x},\ \hat{y},\ \hat{z})$ basis. Typically rotations are applied in the opposite order (Steele et al., 2023), but here, we choose this non-conventional order of operations because it leads to simplifications in the calculations detailed in this paper.

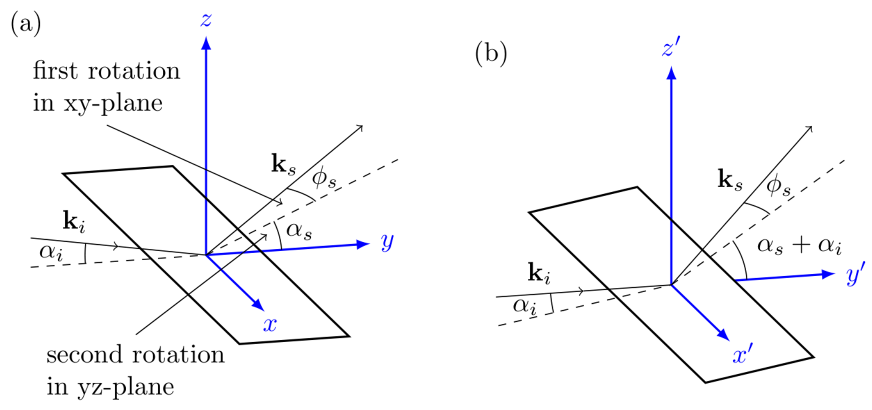

Figure 4: (a) Coordinates in the sample frame. (b) Coordinates in the lab frame.

In the lab frame (see Figure 4b), the axes are denoted by the basis vectors $\hat{x}'$, $\hat{y}'$, and $\hat{z}'$, and column vectors in this basis are denoted with a prime ($'$). The lab frame basis vectors are determined by rotating the sample frame basis vectors with $\overleftrightarrow{R}_{\hat{x}'}(\alpha_i)$. Therefore, a vector can be transfromed from the sample frame basis to the lab frame basis with this rotation operator. Therefore,

For example,

$$\mathbf{k}_i = \frac{2\pi}{\lambda}\begin{bmatrix}0\\ \cos\alpha_i\\ -\sin\alpha_i\end{bmatrix} = \frac{2\pi}{\lambda}\overleftrightarrow{R}_{\hat{x}'}(\alpha_i)\begin{bmatrix}0\\ \cos\alpha_i\\ -\sin\alpha_i\end{bmatrix}' = \frac{2\pi}{\lambda}\begin{bmatrix}0\\1\\0\end{bmatrix}'. \tag{11}$$

Therefore

$$\begin{aligned}\mathbf{k}_s &= \frac{2\pi}{\lambda}\overleftrightarrow{R}_{\hat{x}'}(\alpha_i)\begin{bmatrix}-\sin\phi_s\\ \cos\alpha_s\cos\phi_s\\ \sin\alpha_s\cos\phi_s\end{bmatrix}'\\ &= \frac{2\pi}{\lambda}\overleftrightarrow{R}_{\hat{x}'}(\alpha_i)\overleftrightarrow{R}_{\hat{x}'}(\alpha_s)\overleftrightarrow{R}_{\hat{z}'}(\phi_s)\ \hat{y}'\\ &= \frac{2\pi}{\lambda}\overleftrightarrow{R}_{\hat{x}'}(\alpha_s+\alpha_i)\overleftrightarrow{R}_{\hat{z}'}(\phi_s)\ \hat{y}'\\ &= \frac{2\pi}{\lambda}\begin{bmatrix}-\sin\phi_s\\ \cos(\alpha_s+\alpha_i)\cos\phi_s\\ \sin(\alpha_s+\alpha_i)\cos\phi_s\end{bmatrix}'.\end{aligned} \tag{12}$$

Furthermore, the PONI frame is a two dimensional space in the plane of the detector with basis vectors $\hat{x}''$ and $\hat{z}''$ and with an origin at the PONI. The $\hat{z}''$-direction is the same as the $\hat{z}'$-direction, but the $\hat{x}''$-direction is in the opposite direction of $\hat{x}'$, so that positive ϕ_s correspond to positive x''. A pixel location can be expressed as $\mathbf{r}''$, where

$$\mathbf{r}'' = x''\ \hat{x}'' + z''\ \hat{z}''. \tag{13}$$

The scattering angles, ϕ_s and α_s can then be related to $\mathbf{r}''$ as seen in Figure 5:

$$z'' = d_{sd} \tan(\alpha_s + \alpha_i) \tag{14}$$
$$x'' = \sqrt{d_{sd}^2 + z''^2} \tan(\phi_s), \tag{15}$$

where d_{sd} is the sample detector distance.

Row i and column j coordinates can be related to $\mathbf{r}$ through the equations

$$x'' = (j_{\text{poni}} - j)p_x \tag{16}$$
$$z'' = (i_{\text{poni}} - i)p_z, \tag{17}$$

where i_{poni} and j_{poni} are the row and column index of the PONI respectively, and p_x and p_z are the horizontal and vertical widths of a rectangular pixel.

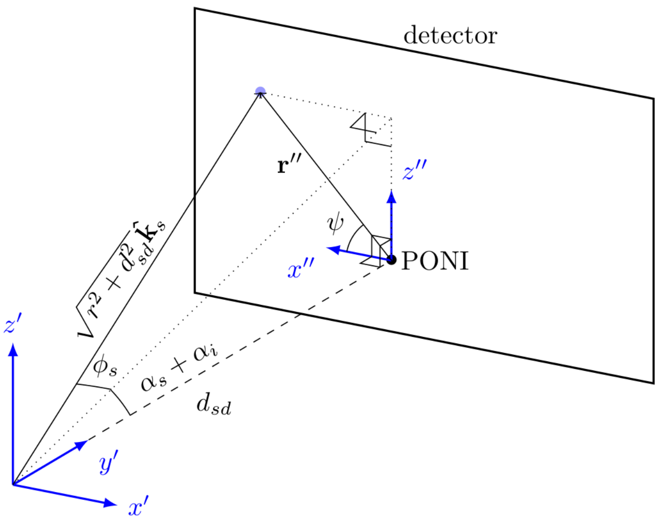

Figure 5: In the lab frame, the scattering angles can be related to coordinates (x'' and z'') on the detector relative to the PONI.

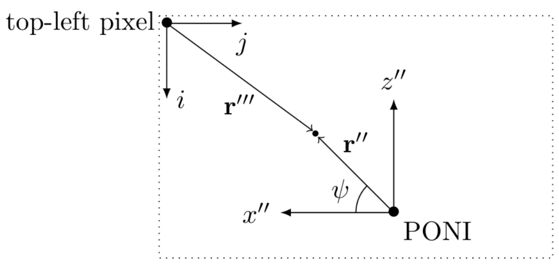

Figure 6: The detector origin is the center of the top-left pixel, and i and j are the row and column indices respectively. Distances from the PONI $\mathbf{r} = x\,\hat{x}'' + z\,\hat{z}''$ can also be described by their magnitude $r = \sqrt{x^2 + z^2}$ and azimuthal angle ψ.

Reciprocal space

The scattering vector, defined in Equation (1), is directly related to the Fourier components of the lattice vectors that describe the crystal. This Fourier space is referred to as reciprocal space, and the magnitude of the scattering vector can be related to the Bragg angle θ through Bragg's law: Equation (2). The magnitude of the scattering vector is also related to a lattice plane spacing d via

$$d = \frac{2\pi}{q}. \tag{18}$$

In the sample frame (Figure 4a), the scattering vector can be written using Equations (7) and (11):

$$\mathbf{q} = \mathbf{k}_s - \mathbf{k}_i = \frac{2\pi}{\lambda} \begin{bmatrix} -\sin\phi_s \\ \cos\alpha_s \cos\phi_s - \cos\alpha_i \\ \sin\alpha_s \cos\phi_s + \sin\alpha_i \end{bmatrix}. \tag{19}$$

Many thin films have cylindrical symmetry, in that individual crystallites have a preferred orientation of a lattice vector in the z-direction (normal to the substrate surface), but are disordered in rotations on the surface of the substrate (Breiby et al., 2008). The cylindrical symmetry of crystallite orientations leads to cylindrical symmetry in reciprocal space, where $q_{xy} = \sqrt{q_x^2 + q_y^2}$ represents the radial axis. A grazing incidence X-ray image transformation into reciprocal space then requires the following calculations:

$$q_{xy} = \frac{2\pi}{\lambda}(\sin^2\phi_s + (\cos\alpha_s \cos\phi_s - \cos\alpha_i)^2) \tag{20}$$

$$q_z = \frac{2\pi}{\lambda}(\sin\alpha_s \cos\phi_s + \sin\alpha_i). \tag{21}$$

Equations (20) and (21) can be calculated using α_s, α_i, $\cos\phi_s$, and $\sin\phi_s$ as determined by the detector coordinates x'' and z'' and the sample-detector distance d_{sd} (Figure 5):

$$\alpha_s = \tan^{-1}\left(\frac{z''}{d_{sd}}\right) - \alpha_i \tag{22}$$

$$\cos\phi_s = \sqrt{\frac{z''^2 + d_{sd}^2}{x''^2 + z''^2 + d_{sd}^2}} \tag{23}$$

$$\sin\phi_s = \frac{x''}{\sqrt{x''^2 + z''^2 + d_{sd}^2}} \tag{24}$$

Reverse transform

In order for GixPy to be compatible with existing X-ray image analysis software (accomplishing the agnosticism goal), after q_{xy} and q_z are calculated for each pixel location, they are related to r_{xy} and r_z such that a powder transformation (utilizing Equation (4)) will produce the correct results. This is done by reversing the powder transformation:

$$r = d_{sd} \tan\left[2\sin^{-1}\left(\frac{\lambda q}{4\pi}\right)\right], \tag{25}$$

where $q = \sqrt{q_{xy}^2 + q_z^2}$. The following trig identities (Spiegel et al., 2012):

$$\tan 2u = \frac{2\tan u}{1 - \tan^2 u} \tag{26}$$

$$\tan\left[\sin^{-1}\left(\frac{u}{2}\right)\right] = \frac{u}{\sqrt{4-u^2}}, \tag{27}$$

can be used to show

$$r = d_{sd} q' \frac{\sqrt{4 - q'^2}}{2 - q'^2}, \tag{28}$$

where $q' = \lambda q / 4\pi$.

The azimuthal angle ψ (as seen in Figures 5 and 6) is related to both r and q in the same way:

$$\cos\psi = \frac{r_{xy}}{r} = \frac{q_{xy}}{q} \tag{29}$$

$$\sin\psi = \frac{r_z}{r} = \frac{q_z}{q}, \tag{30}$$

so

$$r_{xy} = d_{sd} q'_{xy} \frac{\sqrt{4 - q'^2_{xy} - q'^2_z}}{2 - q'^2_{xy} - q'^2_z} \tag{31}$$

$$r_z = d_{sd} q'_z \frac{\sqrt{4 - q'^2_{xy} - q'^2_z}}{2 - q'^2_{xy} - q'^2_z}, \tag{32}$$

where $q'_{xy} = \lambda q_{xy}/4\pi$ and $q'_z = \lambda q_z / 4\pi$.

Seeding the transformed image

For every pixel's location relative to the PONI, GixPy calculates an r_{xy} and r_z using Equations (31) and (32) and then creates a new image where all the counts from each pixel are moved to a location corresponding to r_{xy} and r_z for that pixel. As illustrated in Figure 7, the new image will have a PONI corresponding to the maximum value of r_{xy} and r_z of all the pixels:

$$i^T_{\text{poni}} = \max(r_z)/p_z \tag{33}$$

$$j^T_{\text{poni}} = \max(r_{xy})/p_x, \tag{34}$$

where p_z and p_x are the vertical and horizontal widths of a pixel respectively. r_{xy} and r_z, for each pixel, correspond to row i^T and column j^T in the transformed image according to

$$i^T = \max(r_z) - r_z \tag{35}$$

$$j^T = \max(r_{xy}) - r_{xy}. \tag{36}$$

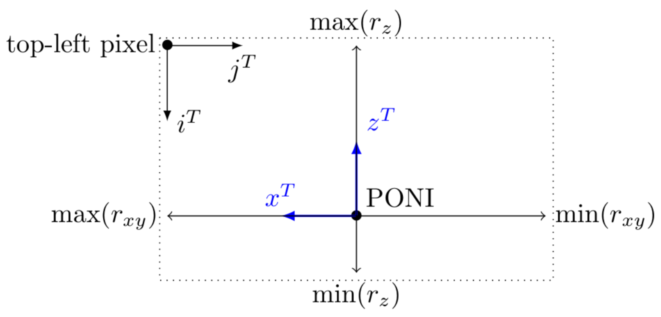

Figure 7: The transformed image's PONI and shape can be determined by the minimums and maximuns of the r_{xy} and r_z found in the transformation calculation.

The transformed image will have rows R^T and columns C^T as determined by

$$R^T = \text{ceil}(\text{max}(r_z) - \text{min}(r_z)) + 1 \tag{37}$$
$$C^T = \text{ceil}(\text{max}(r_{xy}) - \text{min}(r_{xy})) + 1, \tag{38}$$

where the minimums are negatively valued if the PONI is on the detector, $\text{ceil}(x)$ is the ceiling function, and the extra 1 is padding to guarantee that there is room for the pixel splitting step. The transformed image is seeded by creating a NumPy array of zeros with shape $(R^T,\ C^T)$. To account for how many pixels are moved to a new pixel location, a second NumPy array, referred to as the transformed flat field is also created.

Pixel splitting

A pixel index is determined by flooring i^T and j^T, and the counts are split amongst that pixel's neighbors, as seen in Figure 8. Remainders ρ are determined by

$$\rho_i = i^T - \text{floor}(i^T) \tag{39}$$
$$\rho_j = j^T - \text{floor}(j^T), \tag{40}$$

and the counts get distributed according to following weights

$$w_{\text{current pixel}} = (1-\rho_i)(1-\rho_j) \tag{41}$$
$$w_{\text{column neighbor}} = (1-\rho_i)\rho_j \tag{42}$$
$$w_{\text{row neighbor}} = \rho_i(1-\rho_j) \tag{43}$$
$$w_{\text{diagonal neighbor}} = \rho_i\rho_j, \tag{44}$$

where the sum of the weights adds to 1. It is clear that when the remainders are zero, then the "current pixel" gets all the counts, and when both remainders are 0.5, all the pixels get 1/4 the counts.

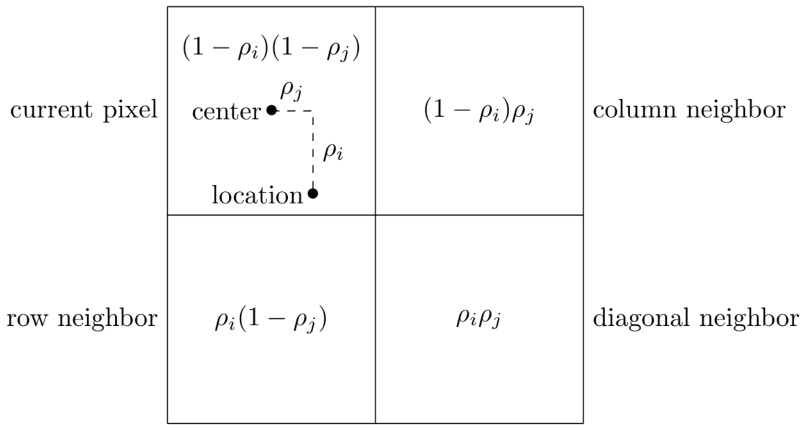

Figure 8: The counts are split amongst neighboring pixels.

Moving pixels

Every pixel in the original image is looped over, and the new row and column indices $(i^T,\ j^T)$ are determined using Equations (35) and (36) by first calculating scattering angles using Equations (22) to (24). Then q_{xy} and q_z are computed with Equations (20) and (21), r_{xy} and r_z with Equations (31) and (32), and the new PONI and image shape with Equations (33), (34), (37), and (38). The weights are calculated for each pixel using Equations (41) to (44), and the counts in pixel $(i,\ j)$ from the original image are added to the counts in pixel $(i^T,\ j^T)$ and its neighbors according to the pixel splitting weights. This is executed by compiled C code written in gixpy.c, but a Pythonic version of this step would look like:

```python
new_image = np.zeros((R_T, C_T))  # as determined by Eq (34) and (35)
new_flatfield = np.zeros((R_T, C_T))
for i in range(image.shape[0]):      # loop over rows of the original image
    for j in range(image.shape[1]):  # loop over columns of the original image
        new_i = int(i_T[i, j])       # floor of i^T, as calculated by Eq (32)
        new_j = int(j_T[i, j])       # floor of j^T, as calculated by Eq (33)

        # calculate weights
        remainder_i = i_T[i, j] - new_i  # Eq (36)
        remainder_j = j_T[i, j] - new_j  # Eq (37)
        w_current_pixel = (1 - remainder_i) * (1 - remainder_j)  # Eq (38)
        w_column_neighbor = (1 - remainder_i) * remainder_j      # Eq (39)
        w_row_neighbor = remainder_i * (1 - remainder_j)         # Eq (40)
        w_diagonal_neighbor = remainder_i * remainder_j          # Eq (41)

        # split pixel
        new_image[new_i, new_j] += image[i, j] * w_current_pixel
        new_image[new_i + 1, new_j] += image[i, j] * w_row_neighbor
        new_image[new_i, new_j + 1] += image[i, j] * w_column_neighbor
        new_image[new_i + 1, new_j + 1] += image[i, j] * w_diagonal_neighbor

        # account for pixel movement
        new_flatfield[new_i, new_j] += w_current_pixel
        new_flatfield[new_i + 1, new_j] += w_row_neighbor
        new_flatfield[new_i, new_j + 1] += w_column_neighbor
        new_flatfield[new_i + 1, new_j + 1] += w_diagonal_neighbor
```

Flat-field correction

A flat-field correction is used to compensate for relative gains of each pixel (Rowlands & Yorkston, 2000). A corrected image C is computed from the raw data R and a flat-field image F, where the flat-field values represent the relative sensitivity of each pixel:

$$C = \frac{R}{F}. \tag{45}$$

A flat field image is used to correct for pixels that are more or less sensitive than the average pixel, and/or if mulitple images are stitched together such that there are regions of the stitch that have more or less exposure time than average. Regardless of whether or not a flat-field correction is needed for the original image, a flat-field correction will always be needed for the GIXS transformation.

As can be seen in Figure 9, the transformation results in a *missing wedge* (Baker et al., 2010). Pixels moved out of the missing wedge disportionately move to the edge of the wedge. This results in these pixels, in the transformation, being more sensitive than pixels not near the edge of the wedge.

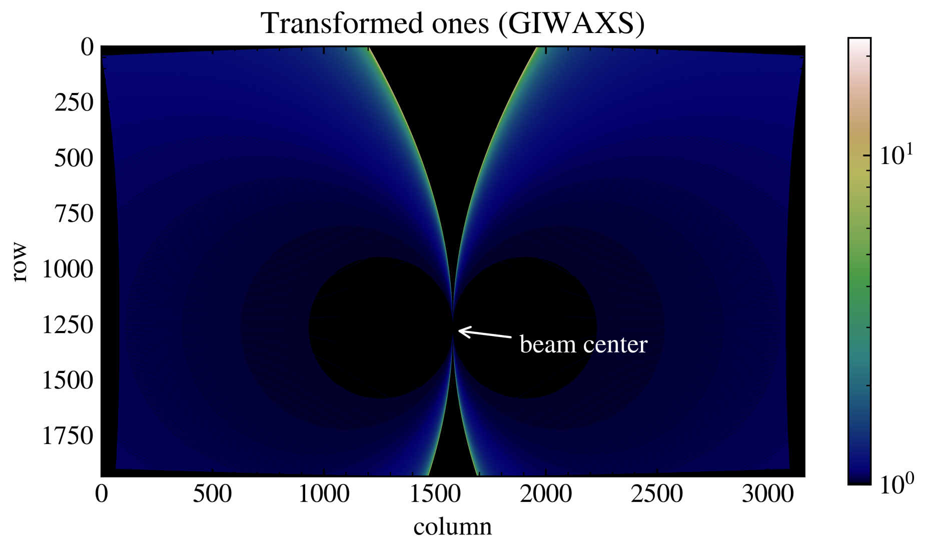

Figure 9: This image was generated by transforming an array of ones with shape $(2000,\ 3000)$, using 75×75 μm pixels, a detector distance of 150 mm, and a grazing-incidence angle of $0.3°$.

The extra brightness along this edge is corrected by also transforming the original image's flat field. An array of ones will represent the flat-field image for an image that needs no correction. The result of a GIXS transform will then yield both an array for the data image and for the flat-field image, where the transformed flat-field image can be used to correct for the edge brightness.

Solid-angle correction

X-rays generated by X-ray tube sources lose intensity according to the inverse square law. Since a flat area detector is used to detect the scattered rays, rays that are detected further from the PONI will lose more intensity than those detected near the PONI. A solid-angle correction adjusts the intensity of pixels to the amount of counts the detector would see if its surface wrapped a sphere around the sample. This is often desired to compare to data that would be collected by a diffractometer.

The distance a ray travels d_{ray} to the detector is determined by the sample-detector distance d_{sd} and the scattering angle 2θ (as seen in Figure 1).

$$d_{sd} = d_{\text{ray}} \cos 2\theta. \tag{46}$$

The intensity of a ray that travels a distance d_{ray} relative to its intensity after traveling a distance d_{sd} is then

$$\frac{I(d_{\text{ray}})}{I(d_{sd})} = \left(\frac{d_{sd}}{d_{\text{ray}}}\right)^2 = \cos^2 2\theta. \tag{47}$$

Furthermore, the angle of incidence of the ray makes with the detector will be the same as the scattering angle and will result in a further attenuation of $\cos 2\theta$. Therefore, rays that hit the detector will lose intensity according to $\cos^3 2\theta$. A solid angle correction reverses this attentuation by multiplying the counts in pixels by Ω, where

$$\Omega = \sec^3 2\theta. \tag{48}$$

Since the solid-angle correction is relative to the geometry of the original image, it is best to apply the solid-angle correction during the transformation, and it should *NOT* be applied to the transformed image.

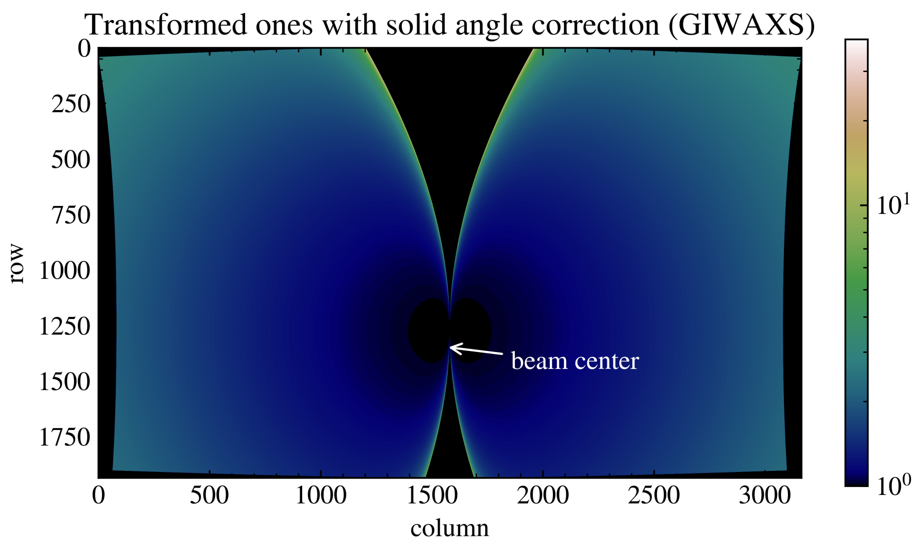

Figure 10: The solid-angle correction increases the intensity of pixels as a function of scattering angle to compensate for the inverse square law and the angle of incedence of a pixel.

Acknowledgements

The authors gratefully acknowledge use of the Materials Research X-Ray Diffraction Facility at the University of Colorado Boulder (RRID: SCR_019304), with instrumentation supported by NSF MRSEC Grant DMR-1420736.

Refrences

Baker, J. L., Jimison, L. H., Mannsfeld, S., Volkman, S., Yin, S., Subramanian, V., Salleo, A., Alivisatos, A. P., & Toney, M. F. (2010). Quantification of thin film crystallographic orientation using x-ray diffraction with an area detector. *Langmuir*, *26*, 9146–9151. https://doi.org/10.1021/la904840q

Breiby, D. W., Bunk, O., Andreasen, J. W., Lemke, H. T., & Nielsen, M. M. (2008). Simulating x-ray diffraction of textured films. *Journal of Applied Crystallography*, *41*, 262–271. https://doi.org/10.1107/S0021889808001064

Cullity, B. D., & Stock, S. R. (2014). *Elements of x-ray diffraction* (3rd ed.). Pearson Education Limited. ISBN: 1269374508

Ilavsky, J. (2012). Nika: Software for two-dimensional data reduction. *Journal of Applied Crystallography*, *45*(2), 324–328. https://doi.org/10.1107/S0021889812004037

Kieffer, J., & Ashiotis, G. (2013). PyFAI: A python library for high performance azimuthal integration on GPU. *Powder Diffraction*. http://arxiv.org/abs/1412.6367

Rowlands, J. A., & Yorkston, J. (2000). Flat panel detectors for digital radiography. In R. L. V. Metter, J. Beutel, & H. L. Kundel (Eds.), *Handbook of Medical Imaging* (Vol. 1, pp. 223–328). SPIE. http://spiedl.org/terms

Spiegel, M. R., Lipschutz, S., & Liu, J. (2012). *Mathematical handbook of formulas and tables* (4th ed.). McGraw-Hill Education.

Steele, J. A., Solano, E., Hardy, D., Dayton, D., Ladd, D., White, K., Chen, P., Hou, J., Huang, H., Saha, R. A., Wang, L., Gao, F., Hofkens, J., Roeffaers, M. B. J., Chernyshov, D., & Toney, M. F. (2023). How to GIWAXS: Grazing incidence wide angle x-ray scattering applied to metal halide perovskite thin films. *Advanced Energy Materials*, *13*. https://doi.org/10.1002/aenm.202300760