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Published: unpublished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 crystal structure of materials localized to a surface with preferred crystal orientation, such as uniaxially aligned thin films (Steele et al., 2023). Often area detectors are used to measure the resulting interference pattern, which requires images to be transformed such that the axes represent reciprocal space. X-ray image analysis software often assumes that the sample is a powder. However, for grazing incidence X-ray experiments, if crystallites in the film have a preferred orientation, the image manipulation requires additional considerations.

Statement of need

There currently exists 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.

The Journal of Open Source Software

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.

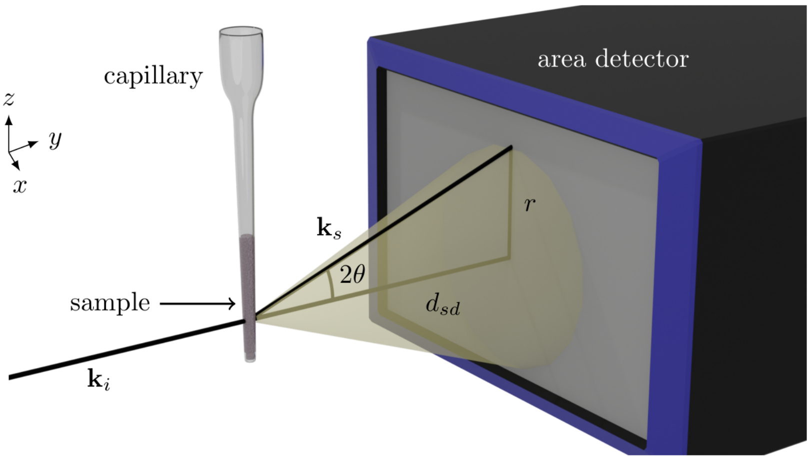

Figure 1: A typical WAXS/SAXS experiment. A powder sample is exposed to an incident beam, resulting in Debye-Scherrer cones of constructive interference. An area detector is used to intersect the cones to detect rings.

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

$$q = \frac{4\pi}{\lambda} \sin\theta, \tag{1}$$

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}:

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

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

$$q = \frac{4\pi}{\lambda} \sin\left[\frac{1}{2}\tan^{-1}\left(\frac{r}{d_{sd}}\right)\right]. \tag{3}$$

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). This means that 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 defines the PONI as the distance from the bottom-left corner of the detector (consistent with pyFAI), as seen in Figure 3. Transforming between $\mathbf{r}_{\text{poni}_{i,j}}$ and $\mathbf{r}_{\text{poni}}$ can be done with the following relation:

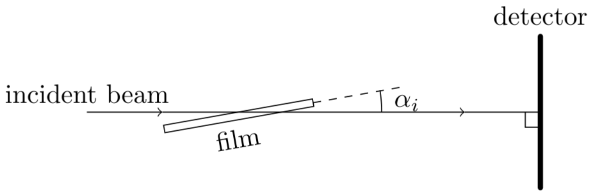

Figure 2: The supported experimental geometry has the detector positioned surface normal to the incident beam, and the grazing angle is set by rotating the sample by α_i relative to the beam.

$$\text{poni}_1 = \left(R - i_{\text{poni}} - \frac{1}{2}\right) p_z \quad (4)$$

$$\text{poni}_2 = \left(j_{\text{poni}} + \frac{1}{2}\right) p_x, \quad (5)$$

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.

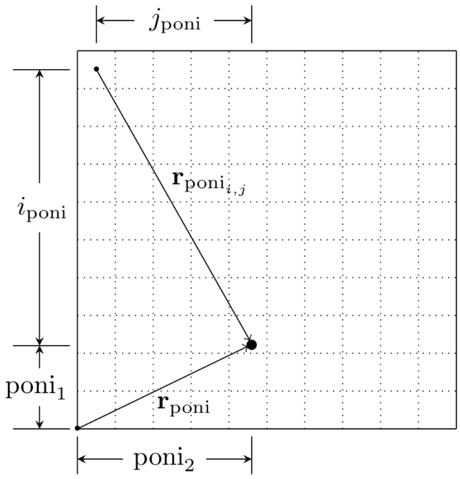

Figure 3: An example detector with 10×10 pixels. The PONI is described by the distance (in meters) from the bottom left corner. A user can convert a PONI in the $(i_{\text{poni}}, j_{poni})$ format using the `gixpy.poni.convert_to()` function.

The Journal of Open Source Software

Scattering geometry

In grazing incidence X-ray scattering, there is a very small angle, called the grazing angle or incident angle, between the plane of the film and the incident beam. 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 such that the z-direction is surface-normal to the film plane, and the x-direction is perpendicular to the direction of the incident beam. In the sample frame, the direction of the scattered ray can be described by rotations from the y-direction:

$$\mathbf{k}_s = \frac{2\pi}{\lambda} R_x(\alpha_s) R_z(\phi_s)\ \hat{y}, \tag{6}$$

where $\hat{y}$ is the y-direction in the sample frame. This is a non-conventional order of operations, but it leads to simplifications in the calculations. In the lab frame (see Figure 4), the axes are denoted $\hat{x}'$, $\hat{y}'$, and $\hat{z}'$, and the $\hat{y}'$-direction is in the direction of the beam. A $R_x(\alpha_i)$ rotation will move from the sample frame to the lab frame, so in the lab frame, the scattered wavevector is

$$\begin{aligned} \mathbf{k}_s &= \frac{2\pi}{\lambda} R_x(\alpha_i) R_x(\alpha_s) R_z(\phi_s)\ \hat{y}' \\ &= \frac{2\pi}{\lambda} R_x(\alpha_s + \alpha_i) R_z(\phi_s)\ \hat{y}'. \end{aligned} \tag{7}$$

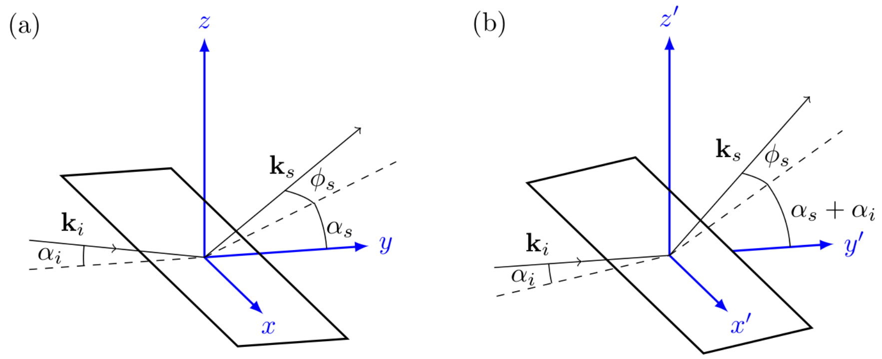

Figure 4: (a) Coordinates in the sample frame. (b) Coordinates in the lab frame.

The scattering angles can then be related to coordinates on the detector as seen in Figure 5:

$$z'' = d_{sd} \tan(\alpha_s + \alpha_i) \tag{8}$$

$$x'' = \sqrt{d_{sd}^2 + z^2} \tan(\phi_s), \tag{9}$$

where z'' and x'' are coordinates on the detector with respect to the x''-z''-plane. Note: the z''-direction is the same as the z'-direction, but has its origin at the PONI instead of the sample, but the x''-direction is reversed from the x'-direction.

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

$$x'' = (j_{\text{poni}} - j)p_x \tag{10}$$
$$z'' = (i_{\text{poni}} - i)p_z, \tag{11}$$

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.

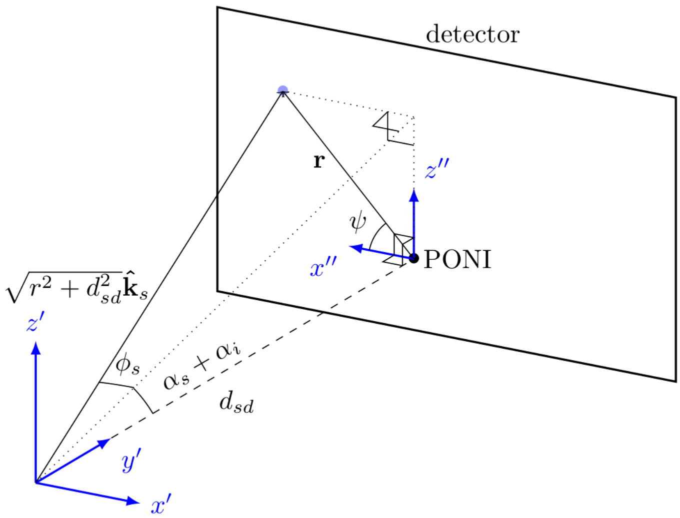

Figure 5: In the lab frame, the scattering angles can be related to coordinates (x and z) on the detector relative to the PONI.

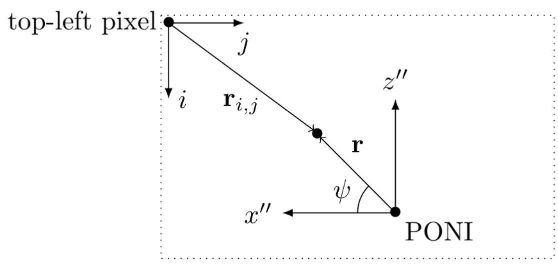

Figure 6: The detector origin is the center of the top-left pixel, and i and j are the row and column indices respectively. Distances from the PONI $\mathbf{r} = x\,\hat{x}'' + z\,\hat{z}''$ can also be described by their magnitude $r = \sqrt{x^2 + z^2}$ and azimuthal angle ψ.

Reciprocal space

The scattering vector $\mathbf{q}$ is defined as

$$\mathbf{q} = \mathbf{k}_s - \mathbf{k}_i, \tag{12}$$

and the magnitude of the scattering vector can be related to the Bragg angle θ through Bragg's law: Equation (1). The magnitude of the scattering vector is also related to a lattice plane spacing d via

$$d = \frac{2\pi}{q}. \tag{13}$$

In the sample frame (Figure 4a),

$$\mathbf{k}_i = \frac{2\pi}{\lambda}\begin{bmatrix} 0 \\ \cos\alpha_i \\ -\sin\alpha_i \end{bmatrix}, \tag{14}$$

$$\mathbf{k}_s = \frac{2\pi}{\lambda}\begin{bmatrix} -\sin\phi_s \\ \cos\alpha_s\cos\phi_s \\ \sin\alpha_s\cos\phi_s \end{bmatrix}, \tag{15}$$

so in the sample frame, the scattering vector can be written

$$\mathbf{q} = \mathbf{k}_s - \mathbf{k}_i = \frac{2\pi}{\lambda}\begin{bmatrix} -\sin\phi_s \\ \cos\alpha_s\cos\phi_s - \cos\alpha_i \\ \sin\alpha_s\cos\phi_s + \sin\alpha_i \end{bmatrix}. \tag{16}$$

Many thin films have cylindrical symmetry, in that individual crystallites have a preferred orientation of a lattice vector in the z'-direction, but are disordered in rotations on the surface of the substrate (Breiby et al., 2008). The cylindrical symmetry of the crystallites 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:

$$q_{xy} = \frac{2\pi}{\lambda}(\sin^2\phi_s + (\cos\alpha_s\cos\phi_s - \cos\alpha_i)^2) \tag{17}$$

$$q_z = \frac{2\pi}{\lambda}(\sin\alpha_s\cos\phi_s + \sin\alpha_i). \tag{18}$$

Equations (17) and (18) 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):

$$\alpha_s = \tan^{-1}\left(\frac{z''}{d_{sd}}\right) - \alpha_i \tag{19}$$

$$\cos\phi_s = \sqrt{\frac{z''^2 + d_{sd}^2}{x''^2 + z''^2 + d_{sd}^2}} \tag{20}$$

$$\sin\phi_s = \frac{x''}{\sqrt{x''^2 + z''^2 + d_{sd}^2}} \tag{21}$$

Reverse transform

In order to suffice the agnosticism goal, after GixPy calculates q_{xy} and q_z for each pixel location, it then relates these to r_{xy} and r_z such that a powder transformation (utilizing Equation (3)) will produce the correct results. This is done by reversing the powder transformation:

$$r = d_{sd}\tan\left[2\sin^{-1}\left(\frac{\lambda q}{4\pi}\right)\right], \tag{22}$$

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

$$\tan 2u = \frac{2\tan u}{1 - \tan^2 u} \tag{23}$$

$$\tan\left[\sin^{-1}\left(\frac{u}{2}\right)\right] = \frac{u}{\sqrt{4-u^2}}, \tag{24}$$

can be used to show

$$r = d_{sd} q' \frac{\sqrt{4 - q'^2}}{2 - q'^2}, \tag{25}$$

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:

$$\cos\psi = \frac{r_{xy}}{r} = \frac{q_{xy}}{q} \tag{26}$$

$$\sin\psi = \frac{r_z}{r} = \frac{q_z}{q}, \tag{27}$$

so

$$r_{xy} = d_{sd} q'_{xy} \frac{\sqrt{4 - q'^2_{xy} - q'^2_z}}{2 - q'^2_{xy} - q'^2_z} \tag{28}$$

$$r_z = d_{sd} q'_z \frac{\sqrt{4 - q'^2_{xy} - q'^2_z}}{2 - q'^2_{xy} - q'^2_z}, \tag{29}$$

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 (28) and (29) and then creates a new image where all the counts from each pixel is 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:

$$i^T_{\text{poni}} = \max(r_z)/p_z \tag{30}$$

$$j^T_{\text{poni}} = \max(r_{xy})/p_x, \tag{31}$$

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

$$i^T = \max(r_z) - r_z \tag{32}$$

$$j^T = \max(r_{xy}) - r_{xy}. \tag{33}$$

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

$$R^T = \mathsf{ceil}(\mathsf{max}(r_z) - \mathsf{min}(r_z)) + 1 \tag{34}$$
$$C^T = \mathsf{ceil}(\mathsf{max}(r_{xy}) - \mathsf{min}(r_{xy})) + 1, \tag{35}$$

where the minimums are negatively valued if the PONI is on the detector, 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.

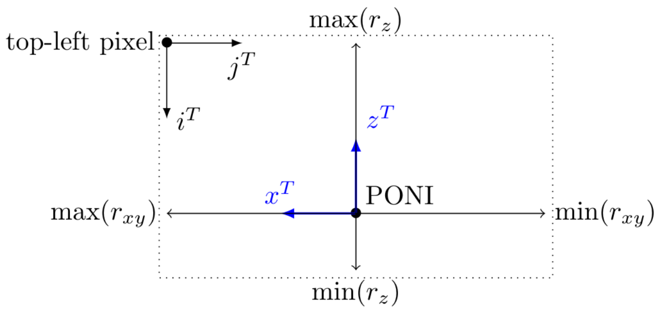

Figure 7: The transformed image's PONI and shape can be determined by the minimums and maximuns of the r_{xy} and r_z found in the transformation calculation.

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

$$\rho_i = i^T - \mathsf{floor}(i^T) \tag{36}$$
$$\rho_j = j^T - \mathsf{floor}(j^T), \tag{37}$$

and the counts get distributed according to following weights

$$w_{\text{current pixel}} = (1 - \rho_i)(1 - \rho_j) \tag{38}$$
$$w_{\text{column neighbor}} = (1 - \rho_i)\rho_j \tag{39}$$
$$w_{\text{row neighbor}} = \rho_i(1 - \rho_j) \tag{40}$$
$$w_{\text{diagonal neighbor}} = \rho_i\rho_j, \tag{41}$$

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.

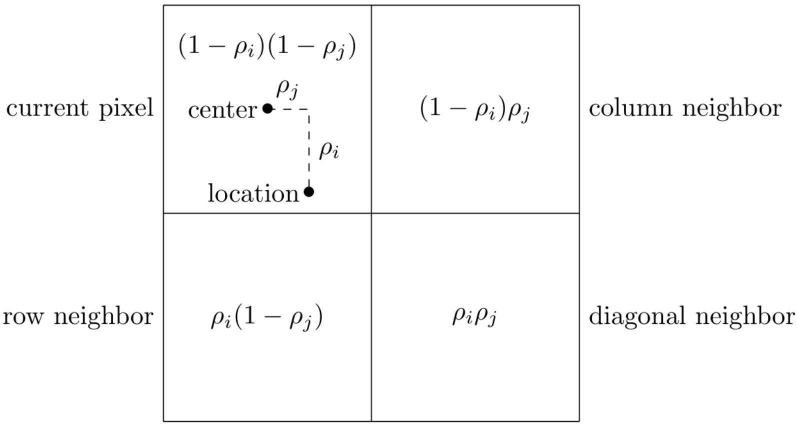

Figure 8: The counts are split amongst neighboring pixels.

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 (32) and (33) by first calculating scattering angles using Equations (19) to (21). Then q_{xy} and q_z are computed with Equations (17) and (18), r_{xy} and r_z with Equations (28) and (29), and the new PONI and image shape with Equations (30), (31), (34), and (35). The weights are calculated for each pixel using Equations (38) to (41), 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:

```
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
```

The Journal of Open Source Software

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:

$$C = \frac{R}{F}. \tag{42}$$

A flat field should be 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. For example. 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.

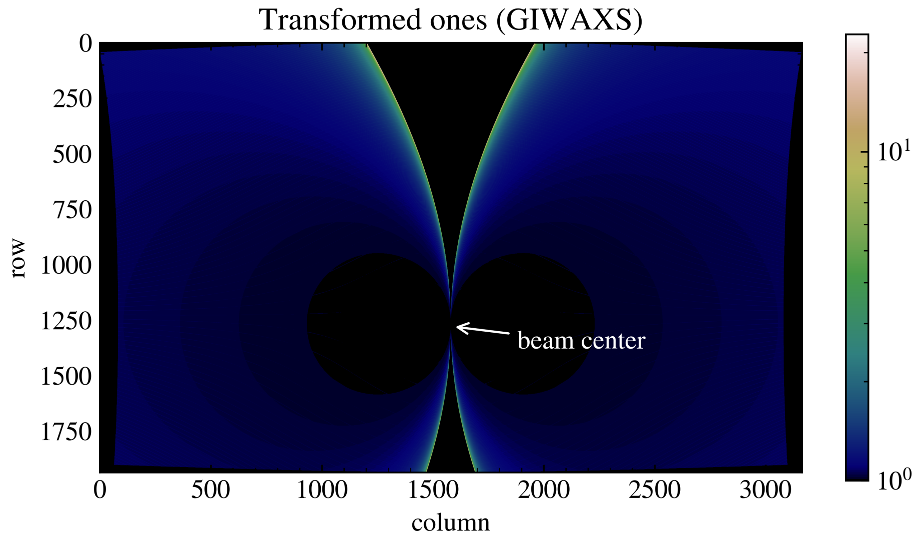

Figure 9: This image was generated by transforming an array of ones with shape $(2000,\ 3000)$, using 75×75 μm pixels, a detector distance of 150 mm, and a grazing-incidence angle of 0.3°.

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 beam center will lose more intensity than those detected near the beam center. 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).

$$d_{sd} = d_{\text{ray}} \cos 2\theta. \tag{43}$$

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

$$\frac{I(d_{\text{ray}})}{I(d_{sd})} = \left(\frac{d_{sd}}{d_{\text{ray}}}\right)^2 = \cos^2 2\theta. \tag{44}$$

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

$$\Omega = \sec^3 2\theta. \tag{45}$$

The solid-angle correction will then adjust 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.

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.

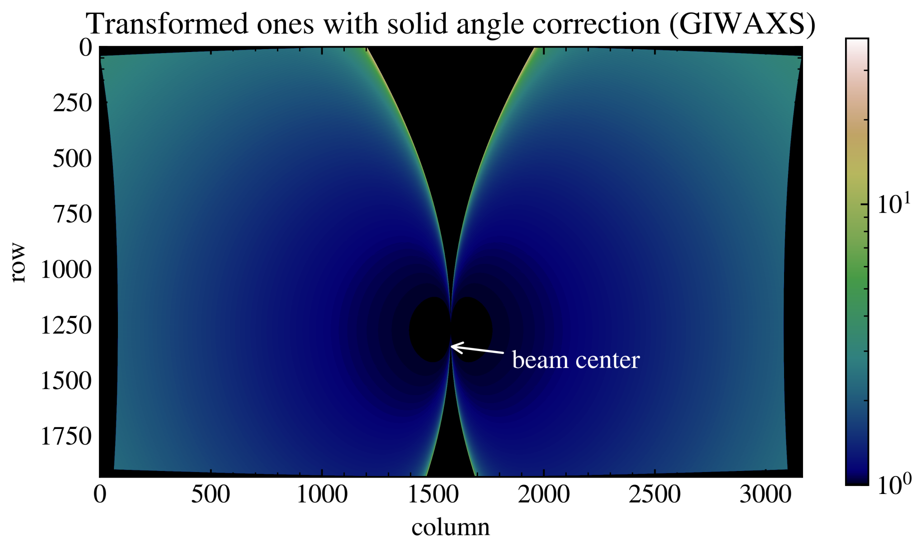

Figure 10: The solid-angle correction increases the intensity of pixels as a function of scattering angle to compensate for the inverse square law and the angle of incedence of a pixel.

Refernces

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