X-ray Diffraction GUI Manual[*]

Joshua Lande (joshualande@gmail.com)

February 9, 2008

Table of Contents

Table of Contents **1**

1 Tips and Tricks **2**
Calibration 2
Masking 4
Cake 4
Integrate 5
Macro 5

2 Examples **6**

3 Diffraction Theory and Detector Geometries **6**
The Three Tilt Angels 8
The R Rotation 9
The β Tilt 9
The α Roll 11
Inverting the Equations 13
Q, 2θ, and χ 13
Implementation In Code 14

4 Calibration **17**
The Calibration Algorithm 18
The Fitting 18
Performing Image Calibration 18
The Q File Format 18

5 Caking **18**

[*]The red text in this paper are links to other parts of the document or other webpages.

6 Intensity Integration **18**

7 Pixel Masking **18**
Threshold Masking 19
Polygon Masking 20

8 Macros **24**
Looping Over Diffraction Data 25
A More Interesting Example 27
Little Tidbits 28
Macro Commands 29
What You Can't Do With Macros 33

9 Software Licensing **33**

References **34**

Index **35**

1 Tips and Tricks

Calibration

The Calibration tab lets you load into the program x-ray diffraction data and it lets you calibrate this data to determine the experimental parameters that characterize the diffraction experiment. You can load in diffraction data using the `"Data File:"` input. The program recognizes `"mar2300"`, `"mar3450"`, `"mccd"`, and `"tiff"` data. To load multiple files at once, and work the with sum image, you need to go into the file part of the menu and select `"Open Multiple Files"`.

Image Calibration is used to infer the experimental setup by analyzing standard diffraction data. This program characterizes a diffraction experiment with several parameters:

- `"xc:"`, `"yc:"` - The x and y coordinates on the detector where the incoming x-ray beam would have hit the detector were there no sample in the way (in pixels).
- `"d:"` - Distance from sample to detector (in mm).
- `"E:"` - Energy of the incoming beam (in eV).
- `"alpha:"`, `"beta:"` 2 orthogonal tilt parameters of the detector (in degrees).
- `"R:"` - The rotation of the detector around the center (in degrees).
- `"pl:"` - The pixel length of the image. The width of one pixel (in microns).
- `"ph:"` - The pixel height of the image. The height of one pixel (in microns).

Before calibrating an image, three things must be done. First, you must load data taken of a standard calibration sample. Then, you must load a Q data file with Q values of peaks corresponding to the sample. Finally, the program needs an initial guess at the calibration values. This can be done with the `"Parameters"` inputs. You can sometimes find some of the calibration data in the header of a diffraction file. If the header contains any of this data, you can push the `"Get From Header"` button to put the header values into the inputs. Once an initial guess is given, the `"Do Fit"` button will refine the calibration parameters and get the best guess at the real experimental parameters.

2θ is the angle of scatter of a particular beam of light which left the crystal and arrived at the detector at a certain point. 2θ is a property of all points on the detector dependent on exactly how the diffraction experiment is set up. Q is defined as $Q = 4\pi \times \sin(2\theta/2)/\lambda$, the wavelength of the incoming beam. D is defined as $2\pi/Q$.

If you prefer to deal with a photons' wavelength instead of its energy, you can go into the Calibration menu and switch from `"Work in eV"` to `"Work in Lambda"`. If you do so, the calibration parameter `"E:"` will be replaced with "λ:" and the current value will be converted. The conversion between these is $E = hc/\lambda$.

The `"Q Data:"` input lets you load in Q data. The Format of a Q data file is a 2 column space or tab separated plain text file. The first line must be `"Q dQ"`. The rest of the file must be lines of Q values followed by a ΔQ range. The ΔQ range gives the program a range in the image where it can find the real diffraction peaks. You can give a list of D values instead. The first line must be of the form `"D dD"` and this must be followed by lines of D and ΔD. For the fit to work, no other diffraction peaks in the image can fall within a Q range. This program stores many standard Q files. To get at them, click the `"Calibration"` option in the menu, then the `"Standard Q"` option, and pick the crystal that you are using.

You can modify the fit in a couple of ways. To do the fit, the calibration algorithm will look a certain number of places in the image to find diffraction peak. It does by running from the center of the image out and looking for peaks in the ΔQ ranges. Once the program finds these peaks, it can store the peak list and use the same one when fitting again. This option can be changed using the `"Use Old Peak List (if possible)?"` selection. If you want to change the number of peaks that the program tries to find, you can use the `"Number of Chi?"` input. This tells the program how many of these radial slices from the center should be done. The `"Stddev?"` input tell the program what ratio higher the peak must be then the standard deviation of the background near the peak in order for the peak to be considered real. The higher the value, the more picky the program is about finding legitimate peaks.

If you know some of the experimental parameters exactly, you can push the `"Fixed?"` check box associated with that variable and it will be not refined when fitting. The pixel length and pixel height are never refined.

To see how good the current calibration parameters are at characterizing the loaded data, you can use the `"Draw Q Lines?"` check box to make the program draw on the diffraction image the lines of constant Q given by the Q data file. You can also draw the Q range specified in the Q file using the `"Draw dQ Lines?"` check box. To see what peaks the program found while doing the fit, you can select the `"Draw Peaks?"` check box and the peaks will be displayed on top of the diffraction image.

You can zoom into the diffraction image by left clicking in the image, dragging your mouse, and then releasing. You can zoom out by right clicking on the image. You can pan across the image by shift clicking on the image and then dragging. You can make the image bigger or smaller by resizing the window.

In the file menu, you can use the `"Save Image"` option to save the current diffraction file in several popular image formats. The image will be saved with the current zoom level and any Q lines, ΔQ lines, or peaks drawn on top of it.

Masking

You can make the program ignore certain pixels when doing diffraction analysis. This is all done on the `"Masking"` tab. To have the program ignore pixels above or below a certain value, use the threshold masking option. You can have the program ignore all pixels that are larger then a certain value by selecting `"Do Greater Than Mask?"` and putting the particular value into the `"(Pixels Can't Be) Greater Than Mask:"` input. You can have the program ignore all pixels less than a certain value with the corresponding `"Do Less Than Mask?"` and the `"(Pixel's Can't Be) Less Than Mask:"` input. Once you set the threshold, these any pixel either too large or too small will be displayed on the diffraction and cake displays as a different color. That color is user specified by the color inputs next to the checkboxes. When you do an intensity integration, these pixels are not used in the averages.

You can also have the program mask certain whole areas of the image if those pixels are undesirable for some reason. If you wish to do so, select the `"Do Polygon Mask?"` option. To draw the image on the screen that you do not want, click `"Add Polygon"` and then then left click on the diffraction image for each of the vertices of the polygon that you want to mask out. To draw the final vertex and finish the polygon, just use right click. To remove any polygon, push the `"Remove Polygon"` button and click on the polygon that you want to get rid of. You can remove all the polygons in the program using the `"Clear Mask"` button, save all the polygons in the program to a file using the `"Save Mask"` button, and load saved polygons using the `"Load Mask"` button.

Cake

A caked image is a plot of diffraction data in Q, χ space. χ is a measure of the angle around what would be the incoming x-ray beam if the beam was not scattered. By convention, χ is equal to 0 degrees to the right of the pixel center of the image and increases while going counterclockwise. To cake the data, you need to give the program a range and bin size in both Q and χ. When you push `"Do Cake"`, the program will present a new window with the caked data in it. You can interact with the cake image just like with the diffraction image described above. If you have one of the `"Draw Q Lines"`, or `"Draw dQ Lines"` or `"Draw Peaks"` boxes selected from the `"Calibration"` tab, these will also be drawn on the cake image. In particular, the Q and ΔQ lines are just vertical lines on the caked image. You can save the caked data as plain text using the `"Save Data"` button. You can save the caked image as a popular image format using the `"Save Image"` button. The image will have the Q lines and peaks saved in the image if that option is selected when saving.

You can also apply a polarization correction to the caked data using the `"Do Polarization Correction?"` check box and the `"P?"` input. The formula to calculate a polarization correction is:

$$\begin{aligned} I &= Im/PF & (1) \\ PF &= P(1-(\sin(2\theta)\sin(\chi-90))^2)+(1-P)(1-(\sin(2\theta)\cos(\chi-90))^2) & (2) \end{aligned}$$

with Im the measured intensity. There is a convenient feature called AutoCake which picks a range of Q and χ values and then cakes that range. AutoCake will pick the smallest cake range so that the whole image shows up in the cake. It will pick the bin size so that each pixel displayed on the screen corresponds to a single bin.

Integrate

Intensity integration lets you calculate average intensity as a function of Q, χ, or 2θ. By default, the option is to integrate in Q or χ. If you want to integrate in 2θ, you must click on the integrate option of the menu and select `"Work in 2theta"` instead of `"Work in Q"`. To integrate intensity, you need to give a lower value, an upper value, and a bin size. With these inputs, the `"Integrate"` button will perform the integration. This data will be plotted in a new window. By default, the integration will be over all possible values of the other independent variable. If you integrate in Q, it will be for all χ. This can be changed by using the constrain checkboxes. For example, to integrate in Q only for the χ values between 0 and 90, you would select the `"Constraint With Range on Right?"` option and the put in the `"Chi Lower?"` input the value 0 and the `"Chi Upper?"` input the value 90. When you then pushed the integrate button, the program would apply the proper constraints.

You can apply the same sort of polarization correcting when integrating. You can also save integration data using the `"Save Data"` button. The output file will be many rows of two column plain text data.

Macro

If you want to perform a complicated task many times, the easiest thing to do is to record a macro of all the commands and then run the macro instead of doing everything by hand. To record a macro, go into the Macro part of the menu and select `"Start Record Macro"`. After doing the tasks you want recorded, push `"Stop Record Macro"` and can save these macro commands to a text file. To run a macro file, you can go into the Macro part of the menu and select `"Run Saved Macro"`.

By editing macro files by hand, you can make them much more useful. Most macro commands are just the name of the GUI item possibly followed by whatever that thing would want (like a filename or number). The macro command to load a diffraction file is `"Data`

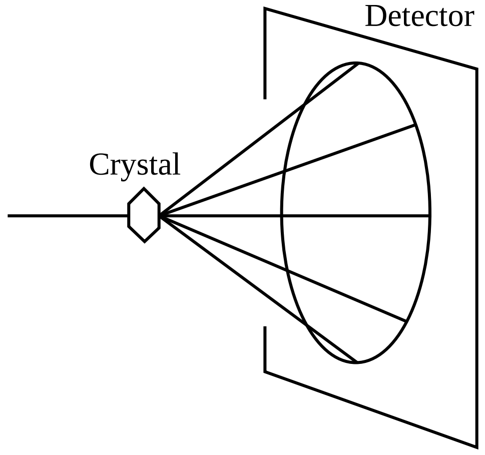


Figure 1: An X-Ray diffraction setup. X-rays scatter from a 3-D sample and are captured by a 2-D detector. In this setup, the detector is perpendicular to the incoming x-ray beam.

`File:"` followed by a line with a filename such as `"C:/somefolder/thefile.mar3450"`. The filename line can be replaced by a list of filenames, a directory containing diffraction data, or some combination of both. The program will run the subsequent macro lines on every file given in the list and on all diffraction files found in any folders given in the list. The loop over diffraction files will end with a subsequent `"Data File:"` line, a `"END LOOP"` line, or the end of the macro file.

When lopping over diffraction files, there is a special markup which makes it easy to save files into the right place. This can be done with the special markup names `"BASENAME"` and `"FILENAME"`. Whenever the program runs across `"BASENAME"` in the macro file, that string will be replaced with the current base of the diffraction file that has been loaded. Similarly, `"FILENAME"` will get replaced with the filename of the current diffraction file. In our previous example, `"BASENAME"` would get replaced with `"C:/somefolder"` and `"PATHNAME"` would get replaced with `"thefile"`. You could recreate the original file name with `"PATHNAME/FILENAME.mar3450"`. This markup can be used to save things with useful names. For example, the macro line `"Save Integration Data"` followed by the line `"PATHNAME/FILENAME_q_i_integration.dat"` would, for our previously loaded file, save the integrated intensity data as `"C:/somefolder/thefile_q_i_integrat`
For all other diffraction data being looped over, the diffraction data would get saved next to the corresponding data file.

2 Examples

3 Diffraction Theory and Detector Geometries

X-ray diffraction is the process of shining a beam of high energy x-rays at a regular crystal. This can be modeled simply as in figure 1. What we have is a series of cones of light that emminate preferentially at certain angles normal to the outgoing beam. These cones of light are then detected by a flat detector, as is shown in the diagram. Usually, the interesting thing to measure when doing x-ray diffraction are the angles of these cones of light. By convention, this angle is called 2θ. Were the detector to be perfectly perpendicular to the incomming

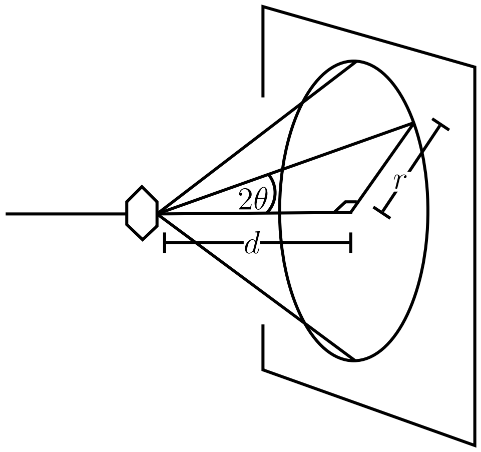


Figure 2: The same setup as in figure 1. Here, 2θ is the scattering angle, d is the distance from the crystal to the detector, and r is the distance from the center of the detector to some particular point (which 2θ is associated with). Note that by center of the detector, we mean the point on the detector where the beam would hit if did not interact with the crystal.

beam, as in figure 1, our cones of light would be detected as circles of high intensity. Were we to know certain experimental parameters such as the distance from the sample to the detector and the distance from the center of the detecor to a particular ring (or really any point on the detector), we could easily calculate the scattering angle of light that ended up at that particular spot on the detector. Suppose that the distance from the crystal to the detector is d and the distance from the center of the detector to our particular point on the detector is r, as is shown in figure 2. Then the scattering angle will be calculated as

$$\tan 2\theta = \frac{r}{d} \tag{3}$$

Unfortunnatley, life is not always as simple as this and sometimes things get a little bit more complicated. In particular, the assumption taht the detector is exactly perpendicular to the incomming beam in unphysical. It will unphysical because perfect alignment can never be acheived and the detector will always in practice be slightly offset with respect to the incomming beam, even if only slightly. Failing to account for this tilt introduce a systematic error in measuring 2θ and doing any other sort of data analysis. It is also unphysical because there are often good reasons to collect diffraction data from detectors at significant tilts . The main reason why this is done is to collect data at more extreme angles without needing to user larger detectors. This rational is made intuitive in figure 3. So there is a need to analyze detectors with tilted geometires. I will here present this theory of tilted detectors as it was developed by Abhik Kumar in [3]. I could simply refer you to this paper for the results that I will cite, but this paper is, in my oppinion, almost absolutely undecipherable. So to save you the misery of pooring through it yourself, I will do my best to cleanly work through his findings.

What we are interested in is mathematically describing position coordinates on a tilted detector by relating them to more theoretically motivated quantities such as the scattering angles that would lead to a beam hitting that particular point on the detector. This will be the equation that is needed to do the diffraction analysis. In order to do this, we must first work out the transformation of points on a tilted detector to points on an untilted detector. This is to say that we want to figure out where on an untilted detector the beam would have

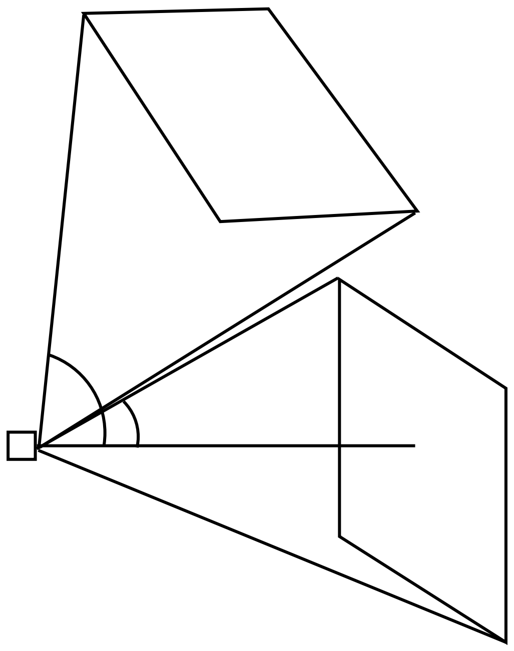


Figure 3: This diagram illustrates how tilted geometries allow for the collection of diffraction data at more extreme angles without the need for a larger detector.

hit were it to hit that untilted detector instead of the tilted detector. We will call the point on the untilted detector (x, y) as it is measured on the untilted detector and the point on the tilted detector (x''', y''') as measured by the tilted detector. The notation will become obvious shortly. This is shown schematically in figure 4.

What we are interested in figuring out is a way of relating this point to some other point (x, y) which would be on an imagined on a plane perpendicular to the incomming beam who's center coincides in 3 dimensional space with the center of the tilted detector.

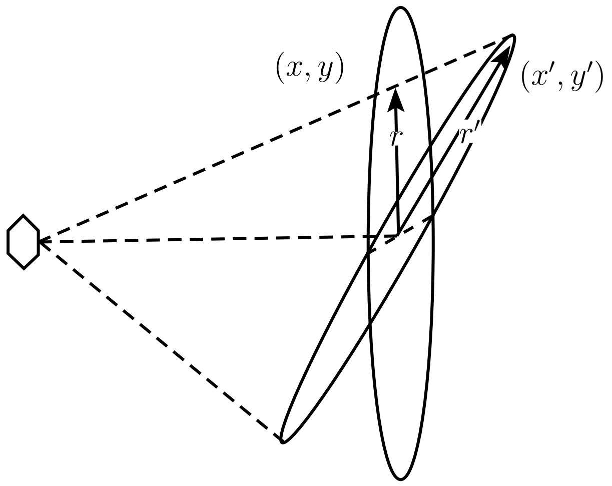


Figure 4: The setup of the experiment. Here, the detector is titled by some arbitrary angle with respect to the incoming beam. We will call some arbitrary point on the tilted detector as (x''', y'''). We are interested in relating this point to the point (x, y) on some imagined untilted detector where a scattered beam would have hit were that tilted detector set up instead of the tilted detector.

The Three Tilt Angels

In order to relate these points, we need to find a way to describe some arbitrary tilt. To do so, we will need to have 3 tilt parameters. We will characterize these tilts by two orthogonal rotations about the x and y axis and one rotation about the center of the detector. These three angles are shown in action in figure 5.

In order to find this relationship, we will apply each of these rotations separately, one after another. And the combination of each of the three rotations will yield the total transformation.

The R Rotation

We will first deal with the physical rotation by angle R of the plane. What we have to deal with is the point (x, y) on a plane being superimposed onto the point (x', y') on another plane which is rotated by an angle R. This is shown schematically in figure 6. The rotation of the plane is mathematically equivalent to rotating the coordinates and the equation describing the rotation of a plane is:

$$x = x' \cos R + y' cos R \tag{4}$$
$$y = y' \cos R - x' cos R \tag{5}$$

The β Tilt

We will now take the point (x', y') and on the rotated plane and see what the coordinates of the corresponding point (x'', y'') would be if the point was projected onto another plane titled by an angle β around $\hat{y}$. This is to say that we will figure out where on the tilted plane a beam would hit were it not to hit that detector instead of the un-tilted detector. This relationship is diagrammed in figure 7. We can use the geometry of these diagrams to figure out the relationships between these coordinates. Using the property of similar triangles, we see that

$$\frac{x'}{d} = \frac{x'' \cos \beta}{d + x'' \sin \beta}. \tag{6}$$

From this it follows that

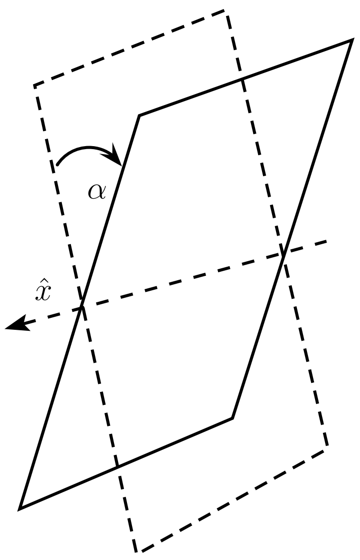
$$\boxed{x' = \frac{dx'' \cos \beta}{d + x'' \sin \beta}.} \tag{7}$$

Using similar triangles again, we see that

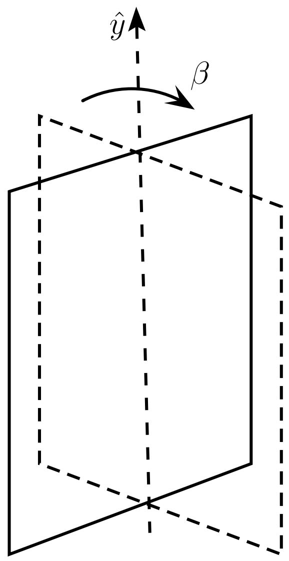
$$\frac{y'}{a} = \frac{y''}{a + b} \tag{8}$$
$$\frac{d}{a} = \frac{d + x'' \sin \beta}{a + b}. \tag{9}$$

from which it follows that

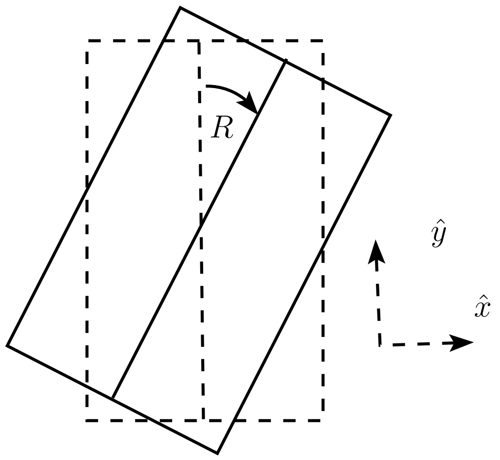
$$\boxed{y' = \frac{dy''}{d + x'' \sin \beta}.} \tag{10}$$



(a) The tilt angle α.



(b) The tilt angle β.



(c) The rotation angle R.

Figure 5: Any detector rotation can be characterized as a rotation by both α and β.

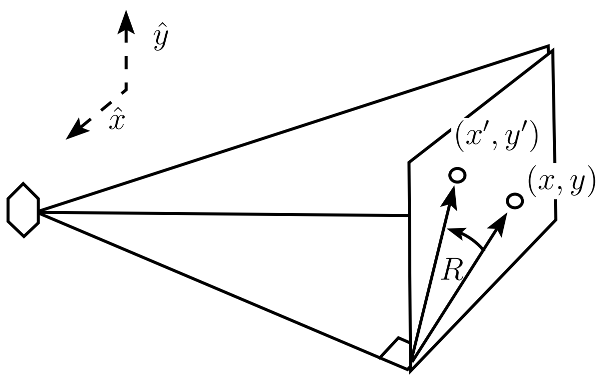


Figure 6: This figure shows how points on a detector rotated by angle R relate to the unrotated points. Remember that a rotation of the plane is equivalent to the rotation of the coordinates on the plane (which is what I have drawn).

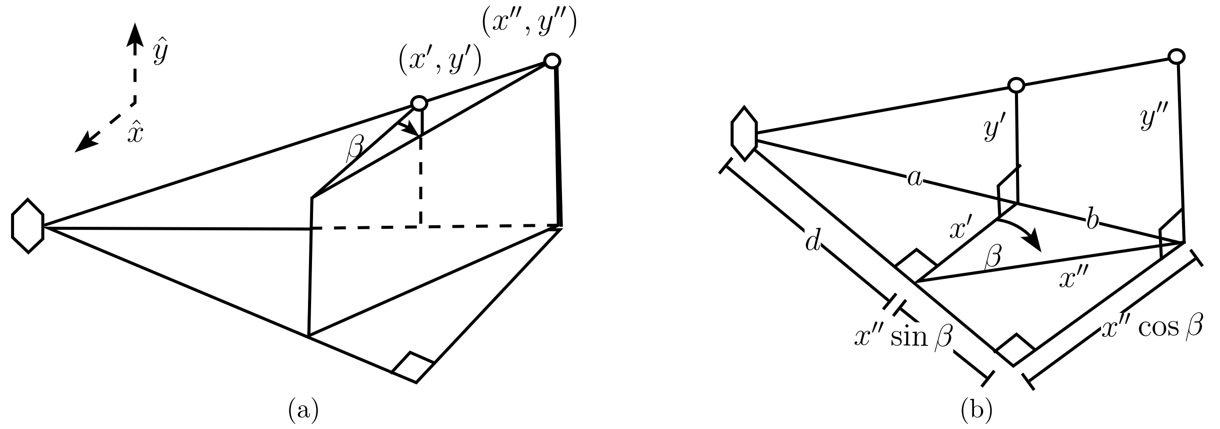


Figure 7: A diagram of the situation depicted in figure 4 where only the β rotation has been applied.

So, equation 7 and 10 give us the proper geometrical equations for relating a point on the un-tilted plane (x', y') to the corresponding point (x'', y'') on the tilted plane.

The α Roll

We can now take this point (x'', y'') on the tilted plane and project it onto another plane which has both a tilt and roll by an angle α around $\hat{x}$ applied to it. This is to say that we can determine where beam that hit a point on the tilted detector would have hit were it to instead hit a tilted and rolled detector. This is shown schematically in figure 8. A more geometric view of the situation can be seen in figure 9 and a cross section of the x axis of this figure can be seen in figure 10.

The final equations relating (x, y) to (x''', y''') are

$$x = \frac{dx'' \cos(\beta)}{d + y'' \sin(\alpha) + x'' \sin(\beta)} \tag{11}$$

and

$$y = \frac{dy'' \cos(\alpha)}{d + y'' \sin(\alpha) + x'' \sin(\beta)}. \tag{12}$$

(x''', y''') is suppose to represent what we actually measure on a real detector. Unfortunately, things are not quite so easy. We do not actually measure these values. The whole formalism assumes that we are measuring these distances from the point on the detector where the beam would hit were it not to be diffracted. Unfortunately, it is not at all clear where this is. A discussion of finding the center will be given in section 4, but for now lets simply note that there is some pixel on the detector that is the center and worry about how to find it later. Lets call it (x_c, y_c) We are now interested in some other pixel reading on the detector which corresponds to the point (x''', y'''). Lets call it (x_d, y_d). Finally, there is some material property of the detector describing the distance between each pixel (e.g.

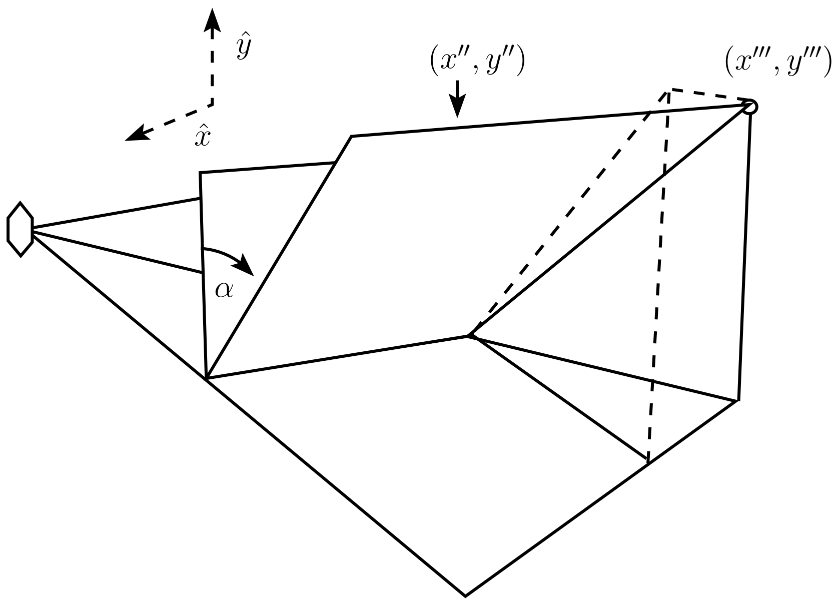


Figure 8: In this figure, we see a tilted detector and then a detector which is both tilted and rolled by an angle α around $\hat{x}$.

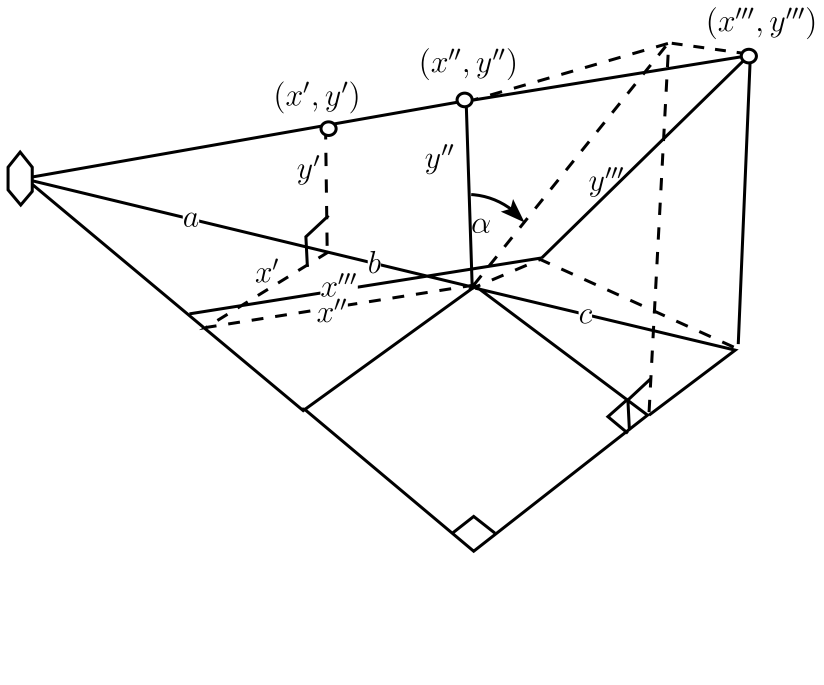


Figure 9:

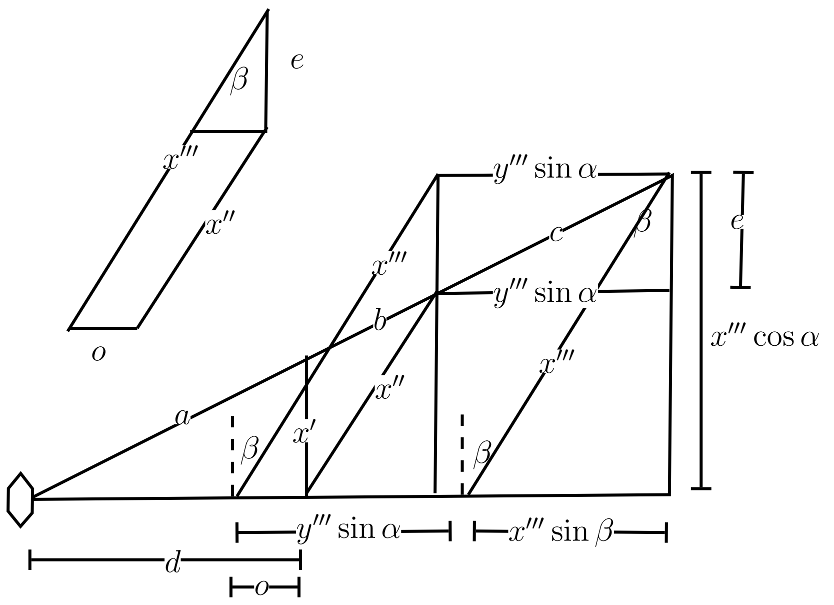


Figure 10: A cross section of the x axis of figure 8. This figure helps bring out the three dimensional aspects of that figure.

1000 mm/pixel). We will call this width ps. We can relate these quantities using:

$$x''' = (x_d - x_c) \times ps \qquad\qquad y''' = (y_d - y_c) \times ps \tag{13}$$

This means that, in terms of (x_c, y_c) and ps, we can relate (x, y) and (x_d, y_d) which are directly measurable experimental quantities.

Inverting the Equations

We can invert these formula to get

$$x'' = \frac{yd}{d\cos(\beta) - x\sin(\beta) - y\cos(\beta)\tan(\alpha)} \tag{14}$$

and

$$y'' = \frac{xd\cos(\beta)/\cos(\alpha)}{d\cos(\beta) - x\sin(\beta) - y\cos(\beta)\tan(\alpha)}. \tag{15}$$

Q, 2θ, and χ

We now have a way of relating (x''', y'''), a point on a detector with a pitch β, tilt α, and a roll R applied to it, to a point on an an untilted detector (x, y) where a beam of light would have interesected were it not to hit the tilted detector. With this relationship, we can now relate these quantities to theoretically motivated quantities. In particular, the angle of scattering of a beam is by convention called 2θ and a quanity measuring the scattering angle around the incomming beam is called χ. These quantities are shown in figure 11. We can see that the relationship between (x, y) and 2θ and χ is

$$\tan 2\theta = \frac{r}{d} = \frac{x^2 + y^2}{d} \tag{16}$$

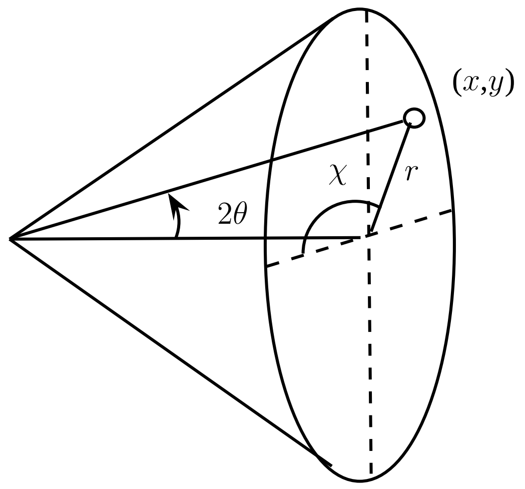


Figure 11: For a particular point (x, y), we alwasy associate two quantities: 2θ and χ. 2θ is the angle of scattering of the beam, or the angle that an incomming beam is difflected by when it diffracts off the crystal. χ is a measure of the azimuthal angle around the beam. It tells you in what direction radially outwards (with respect to the undeflected beam) the outgoing beam was was scattered.

and

$$\tan\chi = \frac{y}{x} \tag{17}$$

The quantity Q is often used instead of 2θ. they are related by

$$Q = \frac{4\pi\sin(2\theta/2)}{\lambda} \tag{18}$$

The reason for using Q instead of 2θ is because diffraction theory shows that the Q values of preferential scattering of a crystal is a material property independent of the experimental setup (such as d and λ).

Alternately, energy could be used in this formula. To do so, energy can be related to wavelength using the De Broglie's formula

$$E = \frac{hc}{\lambda} \tag{19}$$

Finally, sometimes people use the quantity D instead. D is related to Q by

$$D = \frac{2\pi}{Q} \tag{20}$$

Using equation 13, we now have a way of relating pixel coordiantes (x_d, y_d) read directly off of a detector to the theoretically motivated coordinates (Q, χ). In order to do this conversion, we must use the values x_c, y_c, ps, d, λ, α, β, and R. A discussion of how these values can be determined so that this transformation can in practice be done will be given in section 4

Implementation In Code

For reference, this section present the C source code used in the computer program to perform the transformation from pixel values on a tilted detector to (Q, χ) and back again. Listing 1 presents the function "`getTwoThetaChi()`" which converts real pixel coordaintes

(x_d, y_d), called "xPixel" and "yPixel", into the values 2θ and χ, called "twoTheta" and "chi" using the transformation described above. In order to perform this transformation, the program must be given the values x_c, y_c, ps, d, α, β, and R. For x_c and y_c, the function takes in the variables *xCenter* and *yCenter*. The program takes in "pixelLength" and "pixelHeight" for ps to identify the width and height of each pixel (which should be the same). The program takes in the sines and cosines of α, β, and R to allow for increased efficiency. These variables are called " cos_beta" "sin_beta","cos_alpha","sin_alpha", "cos_rotation", and "sin_rotation". Note that this function calculates 2θ instead of Q or D but it takes a trivial amount of work to the additional conversion.

Listing 1: The Code to convert pixel coordinates on a real detector into (Q, χ) coordinates

```
void getTwoThetaChi(double xCenter,double yCenter,double distance,
        double xPixel,double yPixel,
        double pixelLength,double pixelHeight,double rotation,
        double cos_beta,double sin_beta,double cos_alpha,double sin_
        double cos_rotation,double sin_rotation,
        double *twoTheta,double *chi) {

    double pixelLength_mm,pixelHeight_mm;
    double xMeasured,yMeasured;
    double bottom;
    double xPhysical,yPhysical;

    // pixellength comes in in units of micron
    //convert pixelLength & pixelHeight into mm units so that
    //they are comparable with distance (in units of mm)
    pixelLength_mm = pixelLength/1000.0;
    pixelHeight_mm = pixelHeight/1000.0;

    xMeasured = (xPixel-xCenter)*pixelLength_mm;
    yMeasured = (yPixel-yCenter)*pixelHeight_mm;

    bottom = distance+(yMeasured*cos_rotation-xMeasured*sin_rotation
            (xMeasured*cos_rotation+yMeasured*sin_rotation)*sin_beta

    // calculate the x y cordinates on the imaginary detector using
    xPhysical = distance*(xMeasured*cos_rotation+yMeasured*sin_rotat
    yPhysical = distance*(yMeasured*cos_rotation-xMeasured*sin_rotat

    *twoTheta = atan2(sqrt(xPhysical*xPhysical+yPhysical*yPhysical),
    // Convert to radians
    *twoTheta = *twoTheta * 180.0/PI;

    // explicitly convert chi to degrees
    *chi=atan2(yPhysical,xPhysical)*180.0/PI;

    // then add rotation to it so that chi alwasy points to the righ
    // Also, we have to mulitply chi by -1 because we have been defi
    // angles the invers of the way they should be. There is a proba
    // way to do this if I really thought through exactly how chi is
    // For the moment, through, this does exactly the right thing.
    *chi = (*chi + rotation)*(-1);

    // make sure that chi is b/n 0 and 360
    *chi = mod(*chi, 360.0);
}
```

Listing 2 presents the function "getXY()" which does the inverse transformation of function 1. It uses the same terminology for parameters as that function.

Listing 2: The Code to convert (Q, χ) values into pixel coordinates on a real detector

```
void getXY(double xCenter,double yCenter,double distance,double energ
        double q,double chi,double pixelLength,double pixelHeight,
        double rotation,
        double cos_beta,double sin_beta,double cos_alpha,double sin_a
        double cos_rotation,double sin_rotation,
        double * xPixel,double * yPixel) {

    double wavelength;
    double twoTheta;
    double xPhysical,yPhysical;
    double bottom;
    double pixelLength_mm,pixelHeight_mm;
    double xMeasured,yMeasured;

    double tan_chi;

    wavelength = 12398.4172/energy;

    // rotate chi back to point whatever way it is supposed to point

    chi = chi*(-1) - rotation;

    chi = mod(chi, 360.0);

    // explicitly convert chi to radians
    chi*=PI/180.0;

    twoTheta = 2.0*asin(wavelength*q/(4.0*PI));

    tan_chi = tan(chi);
    xPhysical = fabs(distance*tan(twoTheta)/sqrt(1.0+tan_chi*tan_chi
    // one must determine explicitly the sign of xPhysical by inspec
    // This is b/c at one point we take a sqrt in our derivation.
    if (chi>PI/2.0 && chi<3.0*PI/2.0)
        xPhysical = -1.0*xPhysical;

    yPhysical=fabs(xPhysical*tan_chi);

    // set the sign of y explicitly
    if (chi > PI && chi < 2.0*PI)
        yPhysical = -1.0*fabs(yPhysical);

    // I should worry about cos_alpha being 0
    bottom = distance*cos_beta-xPhysical*sin_beta-yPhysical*cos_beta

    xMeasured = (xPhysical*distance*cos_rotation-yPhysical*distance*
    yMeasured =  (xPhysical*distance*sin_rotation+yPhysical*distance

    // convert pixelLength & pixelHeight into mm units so that
    // they are comparable with distance (in units of mm)
    pixelLength_mm = pixelLength/1000.0;
    pixelHeight_mm = pixelHeight/1000.0;

    *xPixel = xMeasured/pixelLength_mm + xCenter;
    *yPixel = yMeasured/pixelHeight_mm + yCenter;
}
```

4 Calibration

One of the most common types of analysis of diffraction data is to perform an intensity integration in Q. This will create a plot of average intensity as a function of Q. Since powder diffraction procedures cones of light, this means that the intensity should be uniformly large for some Q values and uniformly low for others, leading to Q values where the intensity sharply peaks. The Q values that lead to these peaks can be used to learn structural information about the crystals that are being diffracted. So in principle, using the transformations just described, it should be easy to convert all of the pixel coordinates (x_d, y_d) into Q values and then plot average intensity as a function of Q. The only problem we would face is that in order to do the transformation, we would need to know the values of the the parameters that characterize an experiment. These are x_c, y_c, d, λ, α, β, and R.[1] Calibration then is the process used to find what we will now call the calibration values.

The Calibration Algorithm

Although in principle all the calibration values could be experimentally measured, in practice they can not be directly measured to an acceptable level of precision. Instead, a standard calibration procedure is used to infer these values from real diffraction data. The trick to doing this calibration is to image a standard while performing the diffraction analysis of an unknown sample. Assuming that the diffraction machine was not changed between the collection of the standard crystal and the diffraction of the unknown sample, the calibration data corresponding to the two images will be the same. So, if we can figure out the calibration values of the standard crystal, we can use these values when analyzing the unknown crystal. This is exactly what is done in practice.

What it means to use a standard crystal is to know the particular Q values for which the crystal preferentially scatters light. With this information, and the calibration values for some particular experiment, we could in principle figure out exactly what diffraction pattern we should find. This do this, we could, for each Q value, vary χ and calculate the (x_d, y_d) coordinate corresponding to that (Q, χ) pair. After using enough χ values, we would be able to fill in the rings as they would show up on the detector.

In fact, my program can do just this. If you load in a set of Q values (more about this in section 4) and then put into the program some calibration values, and then push the `"Draw Q Values?"` check box, you can then see what the particular diffraction image would have shown up on the detector. (NEED TO SHOW EXAMPLE HERE)

Being able to do this still leaves us with a hard problem to solve. For particular calibration values, we can eaisly calcualte what the defraction pattern should look like. But what we really know is what the calibration values are for the known diffraction pattern of a standard crystal. In order to perform the real calibration, then, we can vary the calibration values until they make the mattern that can be calculated to show up to match the pattern that was actually captured. The process of image calibration then is a procedure to 'fit' the calibraiton values to a diffraciton patter with known Q values.

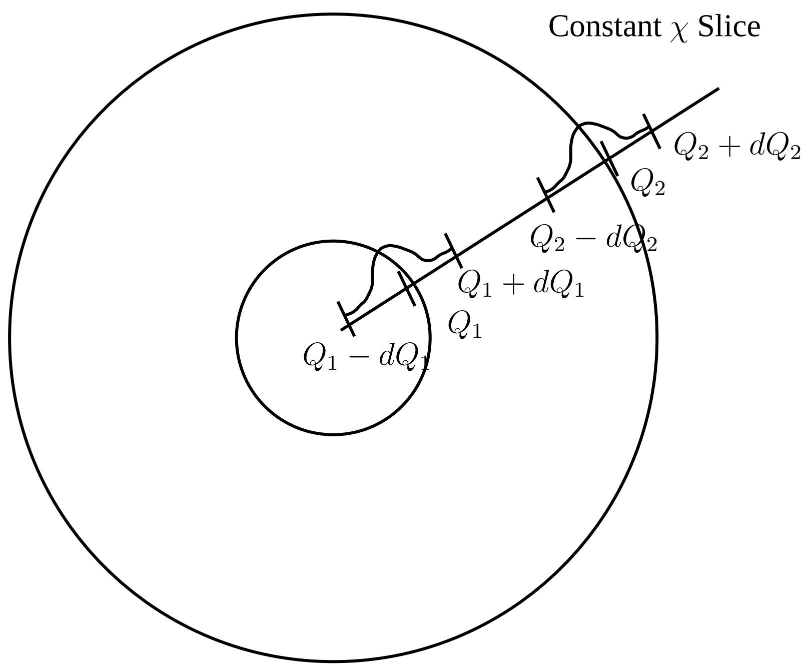


Figure 12: Here is a schematic diagram of the peak finding algorithm. For a particular χ slice, my code fits Gaussian along the line to find the peaks.

The Fitting

Performing Image Calibration

The Q File Format

5 Caking

6 Intensity Integration

7 Pixel Masking

When analyzing diffraction data using this program, not all of the pixels in an image have to be used in the data analysis. In order to make the program ignore certain pixels when doing the analysis, this program allows for two types of pixel masking: threshold masking and polygon masking. You can apply a mask to a diffraction image by going to the `"Masking"` tab of the GUI. There is a screen shot in figure 13 of this tab.

Threshold Masking

The top half of the `"Masking"` tab is devoted to threshold masking. Threshold masking allows all pixels, either above a certain intensity value or below a certain intensity value, to be ignored when doing diffraction analysis. If you want to apply a mask that will cause

[1] The pixel scale *ps* is usually know in advance as a uniform property of the detector being used.

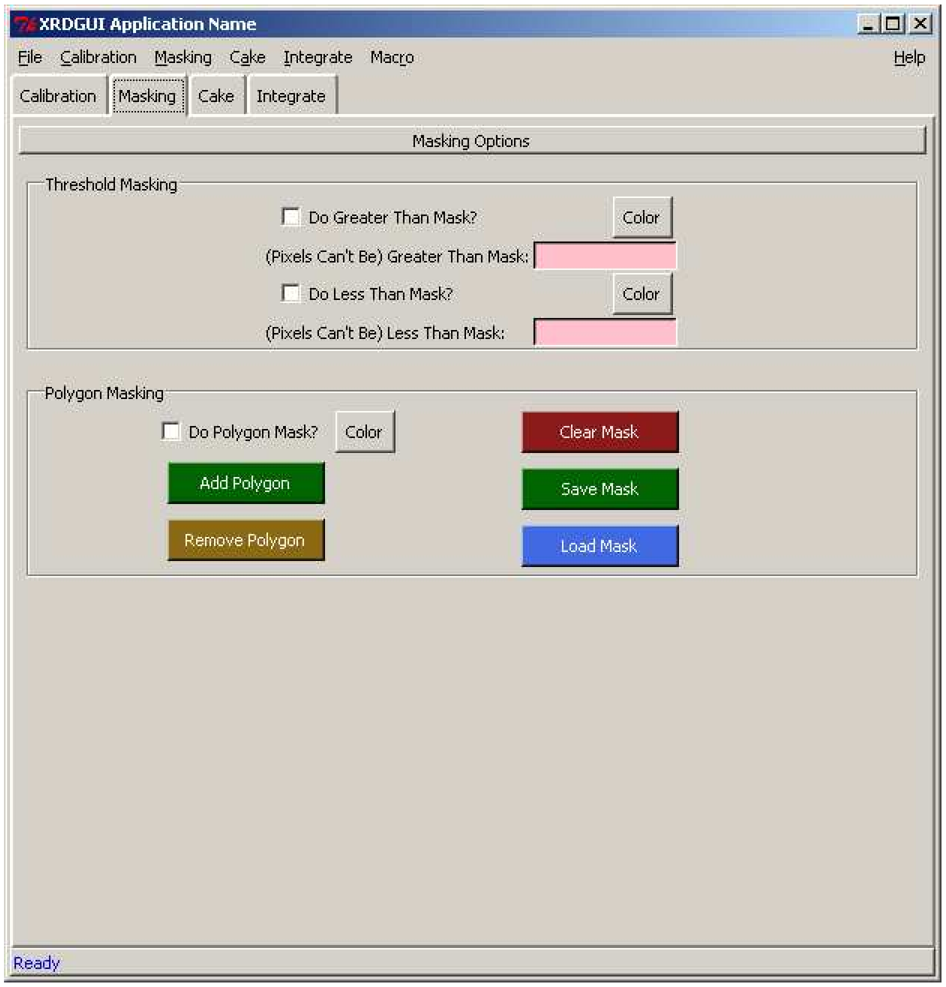
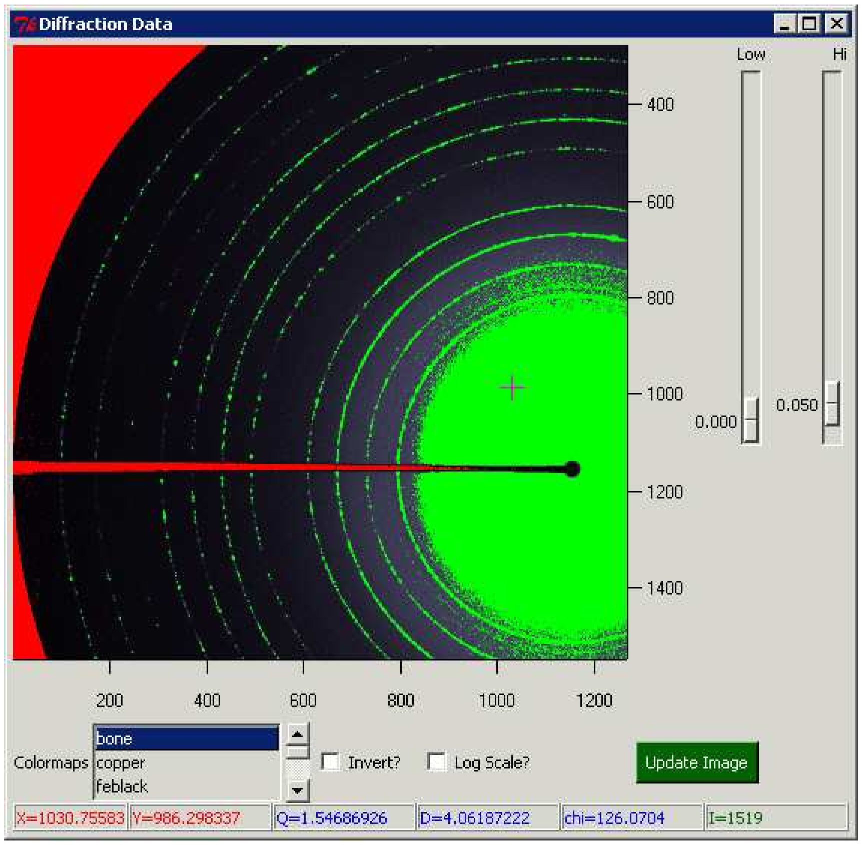


Figure 13: The page of the GUI for pixel masking. This page allows for threshold masking and polygon masking. Threshold masking can be used to have the program ignore all pixels with intensity above or below a certain value when doing data analysis. Polygon masking can be used to have the programme ignore all pixels in certain areas of the image when doing data analysis.

all pixels greater than a certain value to be ignored, you can select the `"Do Greater Than Mask?"` option. The `"(Pixel's Can't Be) Greater Than Mask"` input lets you specify what the maximum pixel should be. Correspondingly, the `"Do Less Than Mask"` check box allows you to require that the program ignores all pixels below a certain value when doing diffraction analysis. The lowest value can be specified by the `"Less Than Mask"` input.

When you apply a threshold mask, the pixels over this threshold will be colored differently on the diffraction and cake image. You can specify what color you want these masked pixels to show up as by with the `"Color"` selector button next to the greater than and less then masks. Figure 14 shows what the diffraction image looks like when all the pixels with intensity above 5000 were masked and colored green and all pixels below 30 were masked and colored red.

If you apply a greater than mask and then save the cake data to a file, any of the pixels that are larger than the greater than mask are outputted with value -2. If you apply a less than mask and then save the cake data, any of the pixels that are smaller than the less than mask are saved as -3. If you need to analyze caked data further on, you need to make sure to account for this. When you apply a threshold mask and then perform an intensity



X=1030.75583 Y=986.298337 Q=1.54686926 D=4.06187222 chi=126.0704 I=1519

Figure 14: Here is an example of diffraction with a greater than mask and less than mask applied. All pixels in this image with intensity greater than 5000 have been colored green. All pixels in this image with intensity less than 30 have been colored red. Applying an intensity mask can be a useful way to see if a detector's pixels have been overloaded and it can also be used to ensure that no overloaded pixels are used in subsequent data analysis.

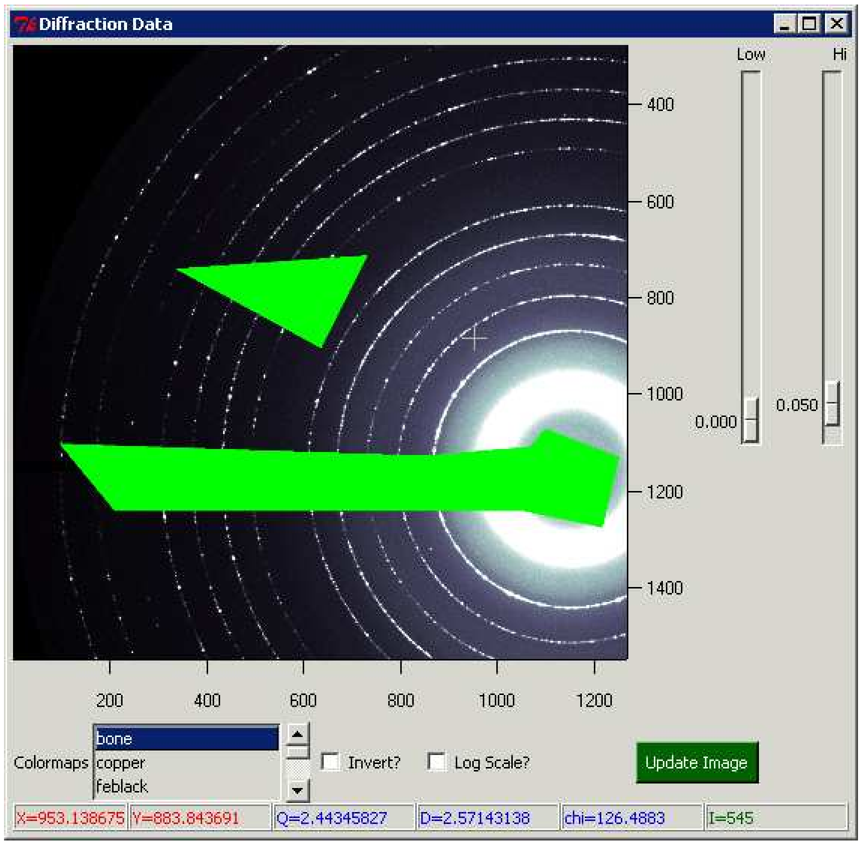
integration, any of the too high or too low pixels are simply ignored when calculating average intensity.

Polygon Masking

Sometimes, large areas of a diffraction image are bad and should not be used in any data analysis. For example, often a beam stop blocks part of the diffraction data and the pixels which were blocked by the beam stop should be ignored. Also, sometimes parts of the experimental setup can block parts of the detector and those areas of the data must be ignored. This is called shadowing. To allow for this sort of masking, the program has a polygon masking feature. You can draw polygons around certain parts of the diffraction image and those parts of the image will not be used in any diffraction analysis. The program can handle as many separate polygons as you want to use.

Once polygons have been added to the image (as is described below), you can have these polygons used when performing data analysis by checking the `"Do Polygon Mask?"` check box. Once you do that, these polygons will be displayed on the diffraction and cake image. This is to say that any pixel in the diffraction or cake image that is inside of one of the polygons will be displayed with a different color. An example of polygons on a diffraction image are shown in figure 15.

You can change this color using the `"Color"` button next to the `"Do Polygon Mask?"`



X=953.138675 Y=883.843691 Q=2.44345827 D=2.57143138 chi=126.4883 I=545

Figure 15: In this screen shot of a diffraction image, we see two polygon masks that have been added to the image. One of them is blocking the beam stop and the other is just there for show. Note that this program can use multiple polygons.

check box. When you save out cake data, any pixels inside of a polygon mask will be given an intensity value of -4. When you perform an intensity integration, any of the masked pixels will simply be ignored when calculating average intensities.

To add a polygon mask to the image, you have to push the `"Add Polygon"` button. When you push it, this button will stay stuck. So pushing it enters you into polygon drawing mode. When you are in this mode, the diffraction image will behave differently. You will not be able to do any zooming or panning on the image. Instead, when you left click on the diffraction image, you will begin drawing the polygon. The first left click on the image will add the first vertex of the polygon. Each success left click will add another vertex to the polygon. When you want to finish drawing the polygon (by adding one final vertex to it), you right click where you want that final polygon to be. Once you right click, the program will exit the `"Add Polygon"` mode and the regular zooming features of the diffraction image will take over. That polygon will then be added to the program. You can add as many polygons to the program as you wish. An example of drawing a polygon is shown in figure 16. If you are are part way through drawing a polygon and decide that you don't like it any more, you can simply unpush the `"Add Polygon"` button and the polygon in progress will be removed and the regular bindings will take over.

If you want to remove a polygon that is already in the program, you can push the `"Remove Polygon"` button. Like the `"Add Polygon"` button, this button will stay pushed and change

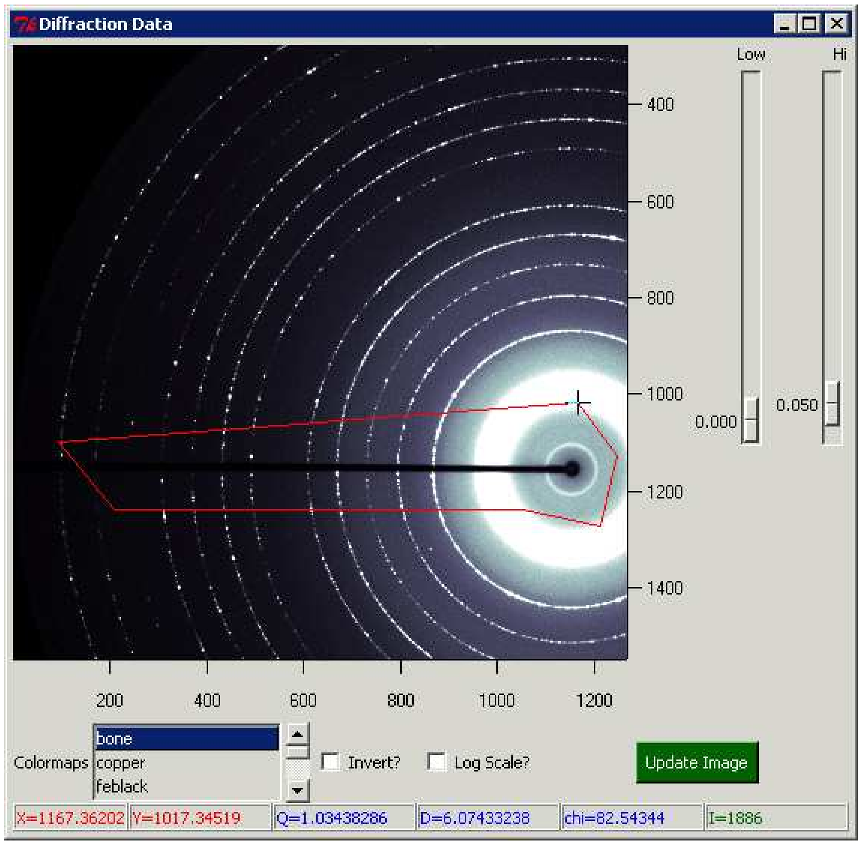


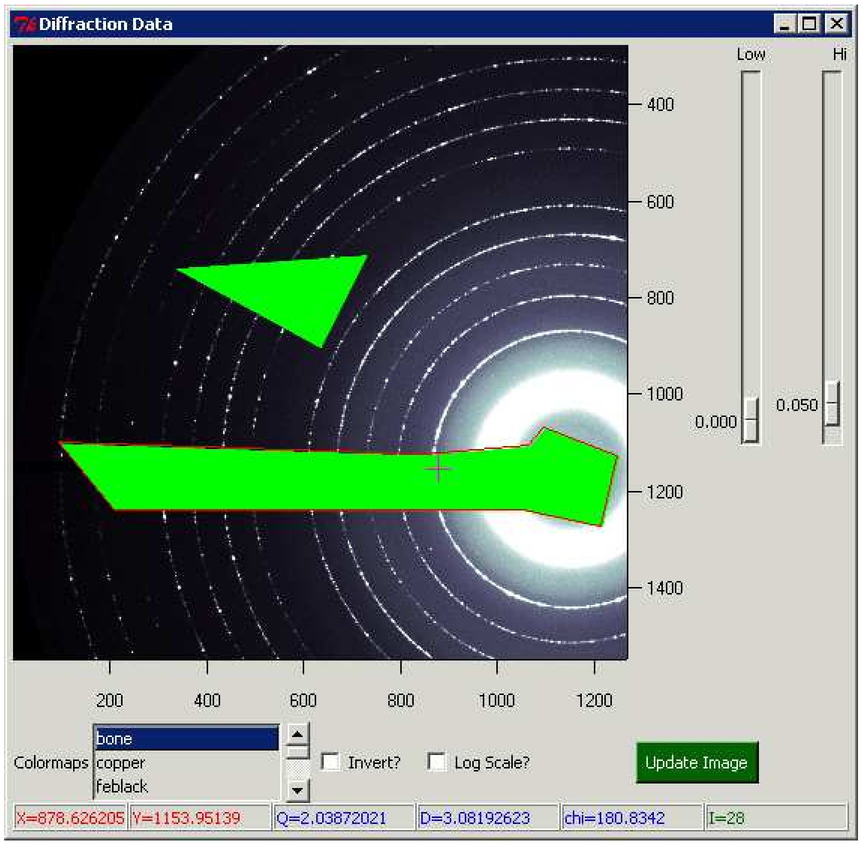
Figure 16: Here is a screen shot of the diffraction image where the user is adding a new polygon mask that will be added into the program. This mask will completely cover up the beam stop and ensure that those values are not later used when performing an intensity integration.

the bindings of the diffraction image. When you now click on the diffraction image while the mouse is on top of a polygon, that polygon will be removed from the file. After deleting a polygon, the program will leave the `"Remove Polygon"` mode and diffraction image will regain the regular zoom features. An example of removing a polygon is shown in figure 17 If you have a change of heart after pushing the `"Remove Polygon"` button and no longer want to remove a polygon, you can simply unpush the button and the program will leave the `"Remove Polygon"` mode. A gotcha that can easily happen when you are using the diffraction image is that you push the `"Add Polygon"` or `"Remove Polygon"` button and then forget that you pushed it. If you do so, the diffraction image will start behaving as previously described. Don't forget that you can always unpush the button to return the program to its normal state.

If you want to remove all the polygons in one go, you can use the `"Clear Mask"` button. If you want to save all the polygons inside of the program to a file, you can use the `"Save Mask"` button. If you want to load all of the polygons from a file into the program, you can use the `"Load Mask"` button. The file format for polygon mask files is very simple. For example, saving out the polygons shown in figure 15 creates the file:

Listing 3: 'A demonstrative polygon mask file'

```
# Polygon(s) drawn on Thu Feb 07 00:00:21 2008
93.140587183     1098.06704199
```



X=878.626205 Y=1153.95139 Q=2.03872021 D=3.08192623 chi=180.8342 I=28

Figure 17: Here is a screen shot of the diffraction image where the user is in the process of removing a polygon. To remove a polygon, you have to push the "`Remove Polygon`" button and then click on the polygon in the diffraction image that you want to remove. When you mouse over a polygon, the program will display a border around it so that you know that that particular polygon is the one that will be deleted.

```
208.013978042    1237.77792276
1052.48863517    1237.77792276
1213.93231962    1271.92947139
1248.08386825    1126.00921814
1095.95424252    1067.02017959
1064.90738013    1104.27641447
847.579343365    1122.9045319

332.201427619    737.923438212
633.355992844    902.471808902
729.601266267    709.981262058
```

The format for these files is to have each line have an (x,y) coordinate for one of the nodes in the polygon. Multiple polygons are separated by newlines and there should be no other formatting. Note that as in figure 15, one of the polygons has only three vertices and is a triangle while the other has many vertices and is a more complicated shape. As always, comment lines beginning with # are ignored except when they are between coordinates in which case they also signify the division of separate polygons.

8 Macros

The macro file format is pretty simple. A macro file contains many commands which automates the GUI. Each command in the GUI gets its own line. Each command has a fairly straightforward style. If you want to do something to the GUI, you look at the name on top of the thing or near the thing and macro command is exactly that string. This sounds confusing but it is not too bad. If you want to get the calibration data from the header of the image, the macro line is `"Get From Header"`. If you want to fit the calibration data, the macro line is `"Do Fit"`.

Things get more interesting when what you want to do requires doing more then just pushing a button. For example, if you want to deselect the `"Draw Q Data?"` checkbox, your macro needs to specify that you deselect this option instead of selecting it. To dos this, the first part of the macro command is the name of the checkbox `"Cake"` and the next line tells you what you want to do with it. Your macro would contain these lines:

Listing 4: 'Draw the Q Lines on the Display'

```
Draw Q Data?
    Select
# Or, to not display them:
Draw Q Data?
    Deselect
```

The exact same format is used for user input. To change the calibration values, your macro file would look like:

Listing 5: 'Input a Number'

```
xc:
    1752.3
beta:
    5.23
```

Finally, some of the Gui commands require a file name. For example, you must provide a file name to load a diffraction file or save a caked image. Whenever the command has a required file, the next line in the macro file should be that file name. For example, if you wantd to save the cake image, your macro file would contain the line:

Listing 6: 'Save the Caked Image'

```
Save Caked Image
    C:/data/cake_output.jpg
```

If you look at the first page, there are three inputs: `"Get From Header:"`, `"dark current:"`, and `"Q data:"`. The macro command to load any of these is a little bit ambiguous. When using the acutal GUI, you would, at least in principle, type in the name of a file and then press load. But there is no reason to make the GUI so redundant. So to load in any of these using a macro command, all you have to do is give the name of the input and then the filename. It will automatically load the file without you explicitly giving the "load" line. So, for example, to load in the Q data, you would include the following lines:

Listing 7: 'Load the Q Data'

```
Q Data:
    C:/data/q_data.dat
```

Looping Over Diffraction Data

Suppose you want to do the same general analysis to many diffraction files at the same time. There is an easy way to do this. To analize one file at a time, you would simply issue the command

Listing 8: 'Load the Diffraction Data'

```
Load Data:
    C:/data/first.mar3450
Integrate Q Lower?
    .25
Integrate Q-I
# ...
```

To loop over multiple diffraction images at once, you could simply give more files after the first one. You can even give it whole directories. When you give it a directory to loop over, the program will (non-recursively) look for all the diffraction files in that directory and include them in the list. Just make sure to put all the files on the same line. The loop will end when one of the 3 things in the macro file happens. Either, a subsequent line in the macro file reads "END LOOP", more diffraction data is loaded using the command "Load Data:", or the macro file ends. For example, if we look at this macro file.

Listing 9: 'Loop Over Diffraction Data'

```
Load Data:
    C:/data/first.mar3450 C:/data/second_file.mar3450
Integrate Q Lower?
    .25
Integrate Q-I
END_LOOP
Draw Q Lines?
    Select
# ...
```

We see that it would get evaluated exactly like the following macro file:

Listing 10: 'An Equivalent Macro'

```
Load Data:
    C:/data/first.mar3450
Integrate Q Lower?
    .25
Integrate Q-I
Load Data:
    C:/data/second.mar3450
Integrate Q Lower?
    .25
Integrate Q-I
Draw Q Lines?
    Select
# ...
```

Finally, there is a convenience markup which can help you make fancy macros. Whenever you have loaded data in, you can refer to the part name of the current diffraction file that is loaded using the string `"PATHNAME"` and you can refer to the file name itself using the string `"FILENAME"`. So, in our previous example, if we had loaded the file `"C:/data/second_file.mar3450"`, `"PATHNAME"` would get chaned into `"C:/data"` and `"PATHNAME"` would get evaluated to `"second_file"` without the extension. In effect, you can imagine building back the full name from `"PATHNAME"` and `"FILENAME"` using an equation line

`C:/data/second_file.mar3450=FILENAME/PATHNAME.mar3450`

These commands are useful because they allow you to loop over many files at once but still save things in useful places and with useful names. It would be easy, for example, to save the intensity data you calculate for each file being looped over using the macro command:

Listing 11: 'Using the FILENAME and PATHNAME Markup'

```
Save Integration Data
    FILENAME/PATHNAME\_intensity.dat
```

This would save, for example, `"C:/data/first.mar3450"`'s intensity data to `"C:/data/first_intensity.`
`"C:/data/second.mar3450"`'s intensity data to `"C:/data/second_intensity.dat"`, and the same for all the others. This feature lets you have the macro to save each of the files to the right place and give it a useful name.

A More Interesting Example

Listing 12: 'A Non-Trivial Macro'

```
# Macro file to calibrate from one file and
# then analyze many others
```

```
Data File:
    C:\data\calibration\cal.mar3450
Get From Header
# Fix the energy before doing the fit
E Fixed
    Select
Number of chi?
    150
stddev?
    8
# This would be standard Q values for the sample
Q Data:
    S:\data\calibration\Q_data.dat
Do Fit
Save Calibration
    C:\data\calibration\cal_values.dat
Draw Q lines?
    Select
Draw Peaks?
    Select
AutoCake
Cake Data Hi
    0.03
Save Caked Image
    C:\data\calibration\cal_cake.jpg
# Load in a directory of data to analyze
Data File:
    S:\data\to_analyze\
Integrate Q Lower?
    .25
Integrate Q Upper?
    4.5
Integrate Number of Q?
    500
Integrate Q-I data
Save Integration Data
    PATHNAME\FILENAME_Q_I_integration.dat
```

The macro first moves the GUI to the calibration tab. It then loads in a calibration image, Q data, and uses as the initial guess for the calibration values number in the header of the file. It then does the fit and saves the fit calibration values to a file. It then cakes the data and saves a cake of this calibration with the Q lines and Peaks so that later visual inspection can show how good the fit was. Afterwards, it loops over a directory of diffraction dat and, using the same calibration data, sets the Q range of the intensity integration, does the intensity integration, and saves integrated data next to the corresponding diffraction files. It should now be apparent that macro are both fun and easy.

Little Tidbits

- Any of the macro commands themselves are case insensitive. The command `"GeT fRoM hEaDeR"` is just as valid as the command `"gET fROM hEADER"` and `"Get From Header"`. You don't have to sweat it.
- White spaces at the beginning and end of the line are ignored. In the preceding examples, the spaces separating macro commands from input values such as file names are there only to increase readability. You don't need them if you don't want.
- New lines are ignored
- comment lines of the form `"# This would be a comment"` are ignored.
- You don't have to worry about explicitly moving from tab to tab in the computer program. The computer program will move to the right automatically before performs the action.
- When you issue the macro command `"E:"` or `"E Fixed"`, the computer program will automatically set the GUI to `"Work in eV"`. If you issue the command or `"lambda:"` or `"lambda fixed:"` then the comptuer program will set the GUI to `"Work in Lambda"`. You can also explicitly set the GUI to either mode using the commadn `"Work in eV"` or `"Work in Lambda"`.
- I need to figure out what colors are allowed.

Macro Commands

Table 1: Macro Commands

Command	Followed By	Effect
Calibration Values		
`Data File:`	Filename(s) and/or Directory(s)	Loops over loading in each file
`Dark Current:`	Filename	Loads in the Dark Current
`Q Data:`	Filename	Load in the Q data
`Get From Header:`	None	Sets the calibration data to the value stored in the image header.
		Continued on next page

Table 1 – continued from previous page

Command	Followed By	Effect
`Load From File:`	Filename	Loads a calibration data file.
`Previous Values`	None	Loads the previously stored calibration values.
`Save To File`	Filename	Saves the calibration data to a file.
`xc:`	Number	Sets the x center.
`xc Fixed:`	`Select` or `Deselect`	Sets whether or not to fix the x center while doing the fit.
`yc:`	Number	Set the y center.
`yc Fixed:`	`Select` or `Deselect`	Sets whether or not to fix the y center while doing the fit.
`d:`	Number	Set the distance.
`d Fixed:`	`Select` or `Deselect`	Sets whether or not to fix the distance while doing the fit.
`E:`	Number	Sets the energy.
`E Fixed:`	`Select` or `Deselect`	Sets whether or not to fix the energy while doing the fit.
`lambda:`	Number	Sets the wavelength.
`lambda Fixed:`	`Select` or `Deselect`	Sets whether or not to fix the wavelength while doing the fit.
`alpha:`	Number	Sets the α angle.
`alpha Fixed:`	`Select` or `Deselect`	Sets whether or not to fix the α angle while doing the fit.
`beta:`	Number	Sets the β angle.
`beta Fixed:`	`Select` or `Deselect`	Sets whether or not to fix the β angle while doing the fit.
`R:`	Number	Sets the rotation angle.
`R Fixed:`	`Select` or `Deselect`	Sets whether or not to fix the rotation angle while doing the fit.
`Draw Q Lines?`	`Select` or `Deselect`	Sets wether or not to draw constant Q lines on the screen.
`Draw Q Lines Color?`	A color	Sets the color of the constant Q lines
`Draw dQ Lines?`	`Select` or `Deselect`	Draw the delta Q lines on the diffraction image
`Draw dQ Lines Color?`	A color	Change the color of the delta Q lines
`Draw Peaks?`	`Select` or `Deselect`	Draw the fit peaks on the diffraction and cake image

Continued on next page

Table 1 – continued from previous page

Command	Followed By	Effect
`Draw Peaks Color?`	A color	Change the color of the peaks
`Update`	None	Update the diffraction image
`Do Fit`	None	Fit the calibration values to a loaded diffraction image
`Make/Save Peak List`	Filename	Creates a peak list just as happens when doing the fit, but instead of acutally doing the fit it saves the peaks as an ASCII file for later use.
`Use Old Peak List (if possible)?`	`Select` or `Deselect`	Uses the previously found peak list again when doing the fit.
`Number of Chi?`	Value	The number of χ slices around the diffraction image to pick and use when doing the calibration
`Stddev`	Value	The σ threshold for allowing a peak.
Diffraction Display Options		
`Diffraction Data Colormaps`	A colormap name	Select the color map to use for the diffraction image.
`Diffraction Data Invert?`	`Select` or `Deselect`	Invert the color map that is being used
`Diffraction Data Log Scale?`	`Select` or `Deselect`	Take the log of all the data points before displaying them.
`Diffraction Data Low?`	Value from 0 to 1	The normalized intensity value which will be scaled to %0 of the image brightness when displaying the diffraction image.
`Diffraction Data Hi?`	Value from 0 to 1	The normalized intensity value which will be scaled to %100 of the image brightness when displaying the diffraction image.
`Save Diffraction Image`	Filename	Save the diffraction image to a file (possibly including Q lines and peaks.
`Work in eV`	`Select` or `Deselect`	Change the GUI to deal with energy in units of eV
		Continued on next page

Table 1 – continued from previous page

Command	Followed By	Effect
`Work in Lambda`	`Select` or `Deselect`	Change the GUI to deal with energy in units of λ using the conversion $E = hc/\lambda$.
Cake Macro Commands		
`AutoCake`	None	Make the computer pick a nice Q and χ range and Cake the data.
`Cake Q Lower?`	Value	The lower Q value of the caked data.
`Cake Q Upper?`	Value	The upper Q value of the caked data.
`Cake Number of Q?`	Value	The number of Q bins to use while caking the data.
`Cake Chi Lower?`	Value	The lower χ value of the caked data.
`Cake Chi Upper?`	Value	The upper χ value of the caked data.
`Cake Number of Chi?`	Value	The number of χ bins to use while caking the data.
`Do Cake`	None	Cake the data.
`Last Cake`	None	Go back to the previous cake values.
`Save Caked Image`	Filename	
`Save Caked Data`	Filename	
Cake Display Options		
`Cake Data Colormaps:`	Colormap	
`Cake Data Invert?`	`Select` or `Deselect`	
`Cake Data Log Scale?`	`Select` or `Deselect`	
`Cake Data Low`		
`Cake Data Hi`		
Intensity Integration Macro Commands		
`Integrate Q Lower?`		
`Integrate Q Upper?`		
`Integrate Number of Q?`		

Continued on next page

Table 1 – continued from previous page

Command	Followed By	Effect
`Integrate Chi Lower?`		
`Integrate Chi Upper?`		
`Integrate Number of Chi?`		
`Integrate Q-I Data`		
`Integrate Chi-I Data`		
`Save Integration Data`		

What You Can't Do With Macros

Just to be clear:

- There is no way with a macro to zoom into the diffraction data, the cake data, or the intensity integrated data
- You can't draw individual polygon masks and you can't remove individual polygon masks. All you can do is load in polygon's from file and save all the current polygons to a file.
- Currently, you cannot load in and add together multiple images using the macro. I indent to code up this feature at some point.

9 Software Licensing

This program is released under the GNU General Public License (GPL) version 2. To read about the GPL and see what you rights to this program are, you read about the GPL at `http://www.gnu.org/licenses/old-licenses/gpl-2.0.html`. For the most part, you can freely use and distribute this software and you can freely make any modifications to it with the condition that any modifications are clearly stated and that your modificates are released under the same license as this program.

This program uses the software package levmar for performing Levenberg-Marquardt nonlinear least squares minimization. It is also released under the GPL. That package can be found at `http://www.ics.forth.gr/~lourakis/levmar/`.[4]

This program also uses the function get_pck() from the CCP4 package DiffractionImage to uncompress Mar2300 and Mar3450 data. This code was written by Dr. Claudio Klein. This package is also released under the GPL and can be found at `http://www.ccp4.ac.uk/ccp4bin/viewcv`

This program also uses W. Randolph Frankin's pnpoly() function for performing a point inclusion in polygon test. This code can be found at `http://www.ecse.rpi.edu/Homepages/wrf/Research`
We comply with his software license which is reproduced below[1]:

Copyright (c) 1970-2003, Wm. Randolph Franklin

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimers. Redistributions in binary form must reproduce the above copyright notice in the documentation and/or other materials provided with the distribution. The name of W. Randolph Franklin may not be used to endorse or promote products derived from this Software without specific prior written permission. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

References

[1] W. Randolph Franklin. Pnpoly - point inclusion in polygon test. [web page] `http://www.ecse.rpi.edu/Homepages/wrf/Research/Short_Notes/pnpoly.html`, Oct. 2005. [Accessed on 06 Feb. 2008.].

[2] Dr. Claudio Klein. pck.c. [web page] `http://www.ccp4.ac.uk/ccp4bin/viewcvs/ccp4/lib/Diffract`
Oct. 1995. [Accessed on 06 Feb. 2008.].

[3] Abhik Kumar. Analysis Strategy of Powder Diffraction Data with 2-D Detector. Tech. Rep., Office of Science, SULI Program, Stanford Linear Accelerator Center, Menlo Park, CA, December 2005.

[4] M.I.A. Lourakis. levmar: Levenberg-marquardt nonlinear least squares algorithms in C/C++. [web page] http://www.ics.forth.gr/~lourakis/levmar/, Jul. 2004. [Accessed on 06 Feb. 2008.].

Index

Q Values, 17
α, 8
β, 8

Abhik Kumar, 6

Calibration, 16
calibration, 17
CCP4, 33

Detector Geometries, 5
Diffraction Theory, 5

Examples, 5

Fitting, 32

get_pck(), 33
GNU, 32
GPL, 32

Intensity Integration, 18

Least Squares Minimization, 32
Levenberg-Marquardt, 32

Macros, 24
Mar2300, 33
Mar3450, 33

Pixel Masking, 18
Polygon Inclusion Testing, 33

Rotation, 8

Software Licensing, 32

Tips and Tricks, 1